BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

METHODS FOR INTELLIGENT NETWORK FORENSICS

INTELIGENTNÍ SÍŤOVÉ FORENZNÍ METODY

PHD THESIS
DISERTAČNÍ PRÁCE

AUTOR PRÁCE
AUTHOR JAN PLUSKAL

SUPERVISOR doc. Ing. ONDŘEJ RYŠAVÝ, Ph.D.
ŠKOLITEL

BRNO 2022

Abstract

This dissertation is a collection of the author's peer-reviewed papers, with a common topic of computer network forensic analysis, published in journals and conferences in computer science, digital forensics. In contrast to understanding network forensics as a discipline of network security monitoring, this work's merit is to aid law enforcement agency (LEA) officers in conducting network forensic investigations. The distinction lies in putting emphasis on extracting evidence from illicit activities rather than detecting network attacks or security incidents.

This work revisits methods used for the forensic investigation of captured network traffic by critically analyzing tools commonly used by LEA investigators. The objective is to identify weaknesses, design solutions, and propose new approaches. Particular interest is given to processing incomplete network communication that typically occurs in low-quality interception provided by Internet Service Providers (ISPs). The proposed method involves omitting missing parts and intelligently rewinding the protocol parsers to pass the missing segments using information from transport and internet layers. This process allowed the creation of novel features for the application protocol identification, thus additionally enabling application protocol identification and finer-grained application identification. Subsequent research analyzed the performance characteristics of single-machine captured network communication and designed, implemented, and evaluated a linearly scalable architecture for distributed computation. Lastly, the problem of overlay and tunneled communication was tackled by thoroughly analyzing Generic Stream Encapsulation (GSE).

The presented research is publicly available, except for the limitations enforced by the publishing houses. When applicable, methods have been implemented into the open source network forensic investigation and analysis tool, Netfox Detective, and verified using enclosed datasets. All data sets and results are available and referenced in their respective publications.

Keywords

network forensic analysis, application protocol identification, captured network traffic processing

Klíčová slova

síťová forenzní analýza, identifikace aplikačního protoklu, zpracování zachycených dat

Reference

PLUSKAL, Jan. *Methods for Intelligent Network Forensics.* Brno, 2022. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Ondřej Ryšavý, Ph.D.

Rozšířený abstrakt

Tato disertační práce je souborem vybraných recenzovaných prací autora spojených tématem forenzní analýzy počítačových sítí. Práce byly publikovány v posledním desetiletí v časopisech a konferencích zaměřujících se na oblast informatiky se specializací na digitální forenzní analýzu. Tato práce se nedívá na síťovou forenzní analýzu jako disciplínu monitorování síťové bezpečnosti, ale zajímá se o pomoc při forenzním vyšetřování kriminalisty z policejních složek (LEA). Rozdíl spočívá spíše v zaměření se na získávání důkazů o nezákonných činnostech než na odhalování síťových útoků nebo bezpečnostních incidentů.

Následující text uvádí přehled přiložených článků (v chronologickém pořadí) v této práci. Článek VII popisuje první iteraci našeho síťového forenzního nástroje Netfox Detective. Je uvedena prezentace jeho architektury určené pro jedno uživatelské prostředí pracovní stanice. Dokument popisuje výzvy a problémy, které bylo nutné pro úspěšný návrh a implementaci síťového forenzního nástroje vyřešit. Nejdůležitějším přínosem tohoto výzkumu bylo zjištění, že kvalita vstupních dat je mnohdy nízká a je třeba, aby byl tento fakt zohledněn již v počátku návrhu síťového forenzního systému.

Z tohoto důvodu jsme se zaměřili na problematiku zpracování neúplných dat a naše závěry publikovali v článku VI. Neúplnost zachycené síťové komunikace může být způsobena několika vlivy. Nejčastějšími jsou zahození packetu z důvodu přetečení vyrovnávací paměti sondy, odposlech na lince, kde je aplikováno asymetrické směrování, vyrovnávání zátěže mezi několik serverů (load balancing). Naším řešením je postavit heuristiky s využitím informací ze síťové a transportní vrstvy a provést aproximaci určení aplikačních zpráv.

Navržené heuristiky pro detekci začátku a konce aplikační zprávy je možné využít i pro zpřesnění identifikace aplikačních protokolů. Článek V popisuje náš výzkum z této oblasti, kde jsme porovnali tři ML algoritmy využívající náš framework pro zpracování síťové komunikace spolu s extraktory vlastností založených jak na standardních identifikátorech, tak na námi nově navržených identifikátorech s využitím výše zmíněných heuristik. Zkoumali jsme vzájemné závislosti vlastností, které jsou odlišné pro jednotlivé aplikační protokoly, a navrhli využít eliminaci takto korelovaných vlastností. Výzkum jsme uzavřeli návrhem statistické metody, která zakomponovala výše zmíněné poznatky.

Článek IV je zaměřený na vyšetřovatele, kteří působí v terénu a získávají důkazy přímým přístupem do zabezpečených Wi-Fi sítí. Tímto způsobem je možné odhalit přímo připojená zařízení, interagovat s nimi a obstarat data, která jsou detailnější nežli běžný záchyt na úrovni poskytovatele připojení. Cílem uvedeného článku je prozkoumat možnosti provedení automatizované penetrace bezdrátové sítě a zprostředkování přímého přístupu ke komunikaci (Man-in-the-Middle) i pro vyšetřovatele, kteří nemají dostatečné technické vzdělání v IT oboru.

Rostoucí množství dat přenášených po síti zvyšuje výpočetní nároky na výpočetní prvky analyzující zachycenou komunikaci. Vzhledem k tomu, že vertikální škálování není udržitelný proces, může být na čase zaměřit se na změnu přístupu a prozkoumat možnosti škálování do šířky, tedy provedení analýzy zachycené komunikace na více výpočetních prvcích než jeden pracovní stroj vyšetřovatele. V článku III jsme se zaměřili na návrh takového systému, který respektuje dříve zmíněné způsoby zpracování poškození komunikace a zároveň umožňuje zpracovat síťovou komunikaci na clusteru výpočetních prvků.

Poslední identifikovanou výzvou, na kterou se v této práci zaměříme, je zpracování tunelovaného provozu. Článek II popisuje tunelovací protokoly, se kterými se mohou LEA vyšetřovatelé běžně setkat. Vybrali jsme jeden z komplexnější protokolů, Generic Stream Encapsulation (GSE,) na kterém ukazujeme možnosti integrace podpory zpracování do našeho síťového forenzního nástroje.

Poslední přiložený článek I shrnuje tuto disertační práci popisem našeho síťového forenzního nástroje Netfox Detective, jakožto demonstračního prostředí metod a konceptů, které tato práce a přiložené články popisují. S využitím tohoto nástroje ověřujeme představené metody v praxi.

Prezentovaný výzkum je volně dostupný vyjma článků s omezeným přístupem. Tam, kde to bylo možné, byly metody implementovány do našeho nástroje pro forenzní vyšetřování a analýzu sítě s otevřeným zdrojovým kódem a jsou plně dostupné komunitě. Metody byly ověřeny pomocí přiložených datových sad. Všechny datové sady a výsledky jsou volně dostupné a odkazované v příslušných publikacích.

Methods for Intelligent Network Forensics

Declaration

I hereby declare that this PhD thesis was prepared as an original work by the author under the supervision of doc. Ing. Onřej Ryšavý, Ph.D. I have listed all the literary sources, publications, and other sources, which were used during the preparation of this thesis.

. . . .
Jan Pluskal
August 15, 2022

Acknowledgments

I want to express my deepest thanks to my supervisor Ondřej, thank you for the opportunity to become part of the NES@FIT research group, to study and work under your supervision, and to my colleagues – the members of our research group – guys, it has been my pleasure to work with you!

My gratitude belongs to my lifelong love and companion, whom I respect deeply, Galina; thank you for your support and willingness to stand by my side and always take care of everything that is needed when I am not there to support you and help you out. Clara, our daughter, even though you don't realize it yet, you are my inspiration to overcome all hardships, to finish my duties just to get back home to you, and overall to become a better version of myself. If you ever happen to read this, let me know how I managed it.

My parents and grandparents, thank you for raising me this way and supporting me throughout my life. Even though not all of you are with me today, your wisdom guided me throughout my life, and your support was and continues to be invaluable to me.

Vladimir, here goes my third recipe for your students' cookbook. I hope you will really publish it! This one is for muffins, which are Clara's favorite.

Muffins

- 400 g plain flour
- 2 pcs baking powder
- 250 ml semi-skimmed milk
- 250 g sugar
- 200 g chocolatey, both dark and white combined
- 2 pcs banana
- 250 g roast fat

The instructions are simple – mix all ingredients, fill the pucks, bake it for 45 minutes on 200 °C and enjoy!

Contents

Chapter 1

Introduction

The presented dissertation is a collection of peer-reviewed papers with a common topic of computer network forensics. The paper's target audience are law enforcement investigators, specialists, and programmers of network forensic tools. These papers reflect the author's journey of gaining experience in computer network forensics, forensic investigation, and law enforcement investigators' daily work. Struggling to improve the current state, I was looking for solutions to the open questions proposed by the *Law Enforcement Agencies* (LEA) practitioners.

1.1 Motivation

Traditional *Network Forensics*, as the state-of-the-art Chapter 2 indicates, is focused on incident detection and response (i.e., IDS/IPS system) in the scope of network administration intended for small businesses, corporations, and critical infrastructure networks.

This thesis aims to address the problem from the point of view of LEA investigators whose modus operandi differs from those of network administrators. Of course, there is related work focused on the LEA investigators' needs, but, as it appears, the current state does not meet their demands, suggested by the constant innovation supported by national grants such as VG20102015022, VI20172020062, VH20192021043.

This research dates back to 2014 (and continues onward), just after Snowden's leak, when penetration of encryption on the public Internet services was not considered a "big issue." As history proves, eight years after, we can still encounter some services that do not use encryption, e.g., some email transfer services, low-energy IoT communication, and plain DNS. Some of them may leak metadata even when the actual content is encrypted (like pre-TLS 1.3 leaked service name identifiers (SNI)).

Network forensic tools are becoming, as is usually the fate of an open source when it is no longer maintained, inadequate for the task. That is because, more often than not, they are developed as academic research and supported by a grant project. After the project ends and the tool has not been mass-publicized, not at the science conferences but among the actual end-users, the project tends to be abandoned and no longer maintained. Due to the rapid evolution of communication, it is adrift and no longer entirely usable. Additionally, these tools require expert knowledge because, in the majority, they are single-purpose solutions controlled by a command-line interface or a simple graphical user interface. LEA officers without adequate training and deep domain knowledge will likely not use them or miss crucial evidence. Also, there is the issue of visibility of such tools because LEA investigators tend to "do the job," not the research of methodologies and tools.

In contrast, commercial products developed by teams of hundreds of employees with proper founding can keep up with the evolution and keep the tools entirely usable and production ready. They typically use a user-friendly interface to process data sources and provide the investigator with an easily understandable reports. Because of that, commercial tools are de facto becoming standards admissible in a court of law. These tools are intended to be massively used by LEA, but because they are closed source, ever-suspicious investigators are wary of using them. Therefore, there is pressure for applied academic research to produce open source, customizable, and reliable tools. An additional motivation to create open source network forensic tools is that LEA investigators have approaches that need to be kept confidential. Revealing them to a private company that supplied a commercial tool is either inconceivable, or the cost for customization of such tool is economically unreasonable. Therefore, the open source nature allows for such approaches' easy extendability while maintaining the low cost of such modifications. Furthermore, these changes can be rebased on the tool mainstream, thus allowing for almost effortless tool upgrades.

Here we are, between open source single-purpose tools that are trustworthy but hard to use and commercial ones "to fit all them easily usable tools" that are not customizable and may not be trusted by some LEA investigators.

This research tries to help LEA investigators combat cybercrime by providing advanced state-of-the-art methods for network forensic investigation packed in a graphical application that validates the results and allows practical applications.

In simple terms, this dissertation can be considered a cookbook on how to write your network forensic investigation tool that is customizable to fit various use cases, the challenges you will face, and the approach you can take to conquer them.

1.2 Problem Statement

The connection speed to the Internet in households, small businesses, and virtual private servers (VPS) rentable in data centers is experiencing unprecedented growth. Illegal activities carried out on devices connected to the Internet pose various challenges to LEA investigations.

First, the amount of criminal activities conducted over the network increases with the penetration of new digital technologies amongst the population, implicating the increase of cases for digital forensic investigators to solve. This fact creates a problem because there is a shortage of IT specialists in the LEA officers' lines.

Second, the increase in communication speed generates more data to be processed by forensic specialists, which requires increased computing power to process the data and, furthermore, the introduction of appropriate methods to utilize the added resources to scale well.

Due to the expert shortage, the solution seems to be the research of novel approaches, the use of more sophisticated methodologies, and modern tools to be used by the investigators; otherwise, processing the ever-increasing amount of data appears to be unsustainable.

1.3 Research Goal and Objectives

> *The goal is to revisit network forensic methods to improve their capabilities and reliability for processing network communication and extracting evidence, enabling the implementation of efficient tools for LEA investigators.*

Therefore, the incentive of this dissertation is to help LEA investigators deal with *Network Forensic* investigations in their daily work.

During the past eight years, I have been in contact with multiple LEA investigators, LEA executives, and commercial vendors during private meetings, scientific conferences, or closed business conferences organized directly for LEAs. During this time, I realized that as long as there is demand, there is a vendor that has a solution to LEA problems. But LEAs are bureaucratic organizations with a fixed budget that requires planning, and adaptation to novel problems occurs slowly.

To mitigate this conundrum, this dissertation states the following research objectives to collect, update, or propose novel approaches to problems of network forensic investigation; see the following description and Figure 1.1 for the visual representation of the linkage between the objectives and the selected papers.

Capturing and processing in/complete network data is a fundamental step for network forensic investigation. Without the ability to robustly deal with missing parts of the communication, application protocol parsers must stop on the first occurrence of a missing piece of the communication, no matter how small or significant it is for the investigation.

Application protocol and finer-grained application identification are necessary steps before using an application protocol parser to dissect the communication. The decision must be made to identify which application protocol is used in the particular application flow. Taking the identification further, we may also deduce some valuable meta-information, such as which application was used by the user. Furthermore, the classifier should not expect that the communication is entirely captured.

What should be the **architecture of a network forensic tool / scale or not to scale** are questions many ask. Is it better to run the tool on a single machine / workstation environment, be centralized on dedicated server(s), or scale up past a single computation unit and utilize spare resources on a cluster? Is the achievable speed improvement worth the cost of the computation hardware?

Tunneled and overlay networks have been used to interconnect geographically separated networks or computer systems to allow end-to-end connectivity and possibly add a security layer using encryption. For forensic investigators, such technology is a significant factor, even though the data transported may not be fully read due to encryption.

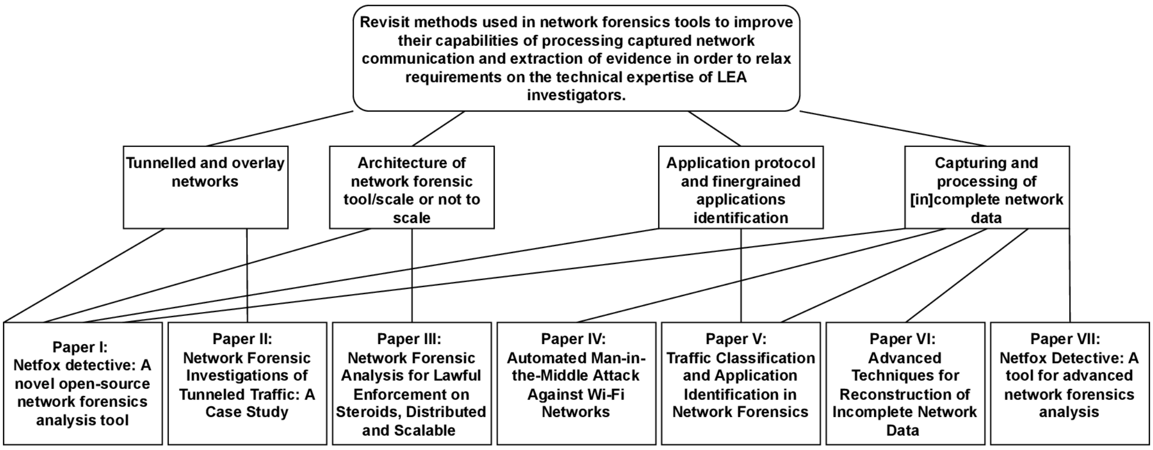

Revisit methods used in network forensics tools to improve their capabilities of processing captured network communication and extraction of evidence in order to relax requirements on the technical expertise of LEA investigators.
Tunnelled and overlay networks
Architecture of network forensic tool/scale or not to scale
Application protocol and finergrained applications identification
Capturing and processing of [in]complete network data
Paper I: Netfox detective: A novel open-source network forensics analysis tool
Paper II: Network Forensic Investigations of Tunneled Traffic: A Case Study
Paper III: Network Forensic Analysis for Lawful Enforcement on Steroids, Distributed and Scalable
Paper IV: Automated Man-in-the-Middle Attack Against Wi-Fi Networks
Paper V: Traffic Classification and Application Identification in Network Forensics
Paper VI: Advanced Techniques for Reconstruction of Incomplete Network Data
Paper VII: Netfox Detective: A tool for advanced network forensics analysis

Figure 1.1: This figure depicts relations between research goal, objectives and selected papers.

These objectives and their solutions are implemented primarily in the Netfox Detective network forensic tool, which demonstrates the validity and verifies the usability of this research in real-world applications.

1.4 Structure of the Dissertation Thesis

The dissertation is a composition of the selected conference and journal publications of the author accompanied by a rational introduction part. The seven selected peer-reviewed papers summarize the contribution of this dissertation. All papers are attached in their original publicized form.

The dissertation is organized as follows. The first chapter provides an introduction, including motivation, a problem statement, and a brief description of the goal and research objectives. The second chapter places the research in its appropriate place by defining the related work. The third chapter summarizes the author's research and contributions. The last chapter discusses the results, highlights contributions, and concludes this thesis.

Chapter 2

State of the Art

Related work is a fundamental part of every research, so let us define the context for this dissertation. This chapter is divided into six sections concerning the research objective stated in Section 1.3.

First, let us clarify the meaning of network forensics and designate this dissertation in this subdiscipline of computer forensics. Section 2.1 brings a broader introduction to network forensics, as seen by various renowned authors in this field. Multiple theoretical frameworks are discussed to describe the recommended forensic approaches.

To develop advanced methods for forensic investigation of networks, it is necessary to study existing tools, identify weaknesses, and design improvements. Section 2.2 provides a brief overview of these tools. The distinction between multiple categories is made, and various taxonomies are presented. Notice that taxonomies classify tools into numerous categories that may not correspond to each other. In other words, the authors may not agree on the classification.

To do the advanced, we need to define the basics. Section 2.3 covers the approach to processing network data. Distinctions are made between Network Security and Monitoring (NSM) tools and Network Forensic and Analysis Tools (NFAT). The crucial role of packet loss intercepting is underscored by some suggestions for achieving it.

Before we can parse the network traffic, knowing which application protocol carried the data is crucial. Section 2.4 covers related research for the identification of application protocols. This information is essential because we need to know the application protocol used to apply an appropriate parser to extract valuable information. In some cases, when data are encrypted, extraction of metadata, such as *SNI*[1] in the case of TLS/SSL, or categorization of application and content types, such as voice or text, could be essential.

Not all communication is encapsulated in the traditional way *(Ethernet | Wi-Fi) / (IPv4 | IPv6) / (TCP | UDP)*, but a significant amount of it is tunneled in overlay protocols such as 6in4, Teredo, GSE, etc. Section 2.5 describes the processing capabilities of overlay networks, mainly used in NFATs.

Lastly, because the amount of communication required for forensic investigations today increases, Section 2.6 introduces related research on parallel processing, scalability, and distributability of network forensic tools.

[1] Service Name Identifier reveals service hostname

2.1 Network Forensics

Let's assume that the term *network forensics* might be slightly confusing. The common understanding is that network forensics is a cross-over between digital forensics and computer security [65]. It is concerned with the capture, recording, and analysis of network communication for for detecting and investigating incidents [53]. For simplicity, network forensics deals with data traces acquired by passive or active network devices. The main goal of network forensics is to investigate network evidence to determine whether there was a security incident or other anomaly, provide evidence, and document their investigation [31].

According to Palmer, network forensics is the use of scientifically proven techniques to collect, fuse, identify, examine, correlate, analyze, and document digital evidence from multiple digital sources. The objective is to uncover facts related to planned intent or successful measurement of unauthorized activities that are intended to disrupt, corrupt, or compromise system components. Information gained during the investigation can be used to respond to or recover from these illicit activities [52].

Network forensics can be described using various process models. The first, proposed by Palmer [52] in DFRWS 2001, is a linear model that contains these steps: identification, preservation, collection, examination, analysis, presentation, and decision. Emanuel Pilli has updated this waterfall-like model with fast iteration shortcuts and called it a generic process model [53]. The alternative to this model is the OSCAR process model, which contains these steps: obtain information, strategize, collect evidence, analyze, and report. Davidoff, who proposed it in his book "Network forensics: tracking hackers through cyberspace" [15], simplified it and made it linear again.

An inherent part of network forensics is its techniques (NFT). We can study them based on forensic process models, forensic tools, and forensic frameworks. Khan, in his "Network forensics: Review, taxonomy, and open challenges" [34], reviews fundamentals such as traceback-based NFT, converge network-based NFT, attack graph-based NFT, distributive-based NFT, and NFT using IDS.

2.2 Network Forensic Tools

Network forensics aims to make sense of volatile network communication. Interpreting low-level network protocols requires expert knowledge to see the bigger picture [10]. Specific network forensic tools can be used to relax the requirement of expert knowledge and make the network forensic investigation accessible to more investigators [22, 25]. These tools should support the summarization, clustering, and highlighting of relevant information [7], such as extracting the content of transmitted files and user credentials or performing additional analysis and visualization in an easily understandable form. Many single-purpose network forensic tools are available (see more in the upcoming sections), but their capabilities, functionality, and usability are lacking behind traditional forensic toolkits [10] such as EnCase or Autopsy.

Network forensic tools are best described using taxonomies that categorize them according to their properties. By studying multiple taxonomies, we can observe that authors may disagree on the tool classification, i.e., one author classifies a tool as network monitoring and another as forensic analysis.

One of the first taxonomies [23] proposed in 2002 by Simon Garfinkel is based on monitoring and recording network data; see Table 2.1. The first approach, *catch it as you can*, tries to capture all data that pass through the network in real-time and analyze them in batches later. This approach requires a large amount of data storage [42] but may produce better results because multiple tools can be used to analyze these captured data. The second approach, *stop, look and listen*, uses filtration to limit the amount of data captured to deal with situations where it is not legal to record information unless for some specific reason, such as a court order [23].

“catch it as you can”	
Commercial	NetVCR
	NetIntercept
Open source	tcpdump
	windump

“stop, look, and listen”
Network Flight Recorder (NFR)
SillentRunner
snort intrusion detection system
NetWitness
"Carnivore" Internet wiretapping system

Table 2.1: Taxonomy of network forensic tools according to Simson Garfinkel, 2002, Table source [77].

Meghanathan et al. in 2009 proposed in their taxonomy [45] to divide tools according to their focus, emphasizing the growing interest in domain-specific tools; see Table 2.2. The authors claim that the essential categories are email, web forensics, and packet sniffers.

Email forensics	Packet sniffers	Web forensics
emailTrackerPro	AirPcap	Index.dat analyzer
SmartWhoIs	Ethereal	Web Historian
	WinPcap	

Table 2.2: Taxonomy of network forensic tools according to Meghanathan et al., 2009, Table source [77].

One of the most complex taxonomies was indisputably proposed by Pilli and Joshi [53] in 2010 in the Digital Investigation journal; see Table 2.3. They introduce the term *Network Forensic Analytical Tools* as opposed to *Network Security Monitoring tools*. This distinction is crucial in distinguishing tools designed for forensic investigators (NFAT) from those intended for network administrators (NSM). This taxonomy was updated in 2016 by the same authors [30]; see Table 2.4, resulting in more up-to-date categorization. Note that the subcategories have changed slightly and that proprietary/commercial NFATs have been reduced. This reduction in commercial tools is probably due to the increased secrecy around them[2].

NFATs	
Open source	NetworkMiner
	PyFlag
	Xplico
Proprietary	DeepSee
	InfiniStream
	Iris
	NetDetector
	NetIntercept
	NetWitness
	OmniPeek
	SilentRunner

NSM tools	
Fingerprinting	Nmap
	P0f
IDS	Bro
	Snort
Manipulation	TCPReplay
	SiLK
Packet capture	Argus
	flow-tools
	NfDump
	Nessus
	PADS
	Sebek
	TCPDump
	TCPFlow
	Wireshark
Pattern matching	Ngrep
	TCPXtract
Statistic	NetFlow
	Ntop
	TCPDstat
	TCPStat
	TCPTrace

Table 2.3: Taxonomy of network forensic tools according to Pilli and Joshi, 2010, Table source [77].

[2] Authors observation obtained while discussing the problem with tool vendors on ISS World conference during the past decade.

NFATs	
Open source	PyFlag Xplico
Proprietary	NetDetector NetIntercept OmniPeek

NSM tools	
Intrusion detection systems (IDS)	Bro Snort
Network monitoring tools	IPTraf Ntop TCPStat VisualRoute
Network scanning tools	Angry IP Scanner Nmap Wireless Network Watcher
Network sniffers and packet analyzing tools	Aircrack-ng eMailTrackerPro Kismet NetworkMiner ngrep WebScarab Wireshark
Vulnerability assessment tools	Acunetix WVS Metasploit Nessus Nikto Yersinia Wikto

Table 2.4: Network forensic tools updated taxonomy according to Pilli and Joshi, 2016, Table source [77].

Davidoff and Ham proposed a taxonomy [15] in 2012 based on the tool's functionality and the investigation phase in which the tool can be used; see Table 2.5.

WAP discovery tools	
Open source	KisMAC
	Kismet
	NetStumbler
Proprietary	Skyhook

IDS/IPS	
Open source	Bro
	Snort
Proprietary	CheckPoint IPS-1
	Cisco IPS
	Corero Network Security
	Enterasys IPS
	HP TippingPoint IPS
	IBM Security NIPS
	Sourcefire 3D System

Traffic acquisition
dumpcap
libpcap
tcpdump
tshark
winpcap
Wireshark

Packet analysis	
Protocol analysis tools	tshark Wireshark
Packet analysis tools	Bless ngrep tshark Wireshark
Flow analysis tools	pcapcat tcpflow tcpXtract tshark Wireshark
Higher-layer traffic analysis tools	findsmtpinfo.py NetworkMiner oftcat smtpdump

Statistical flow analysis	
Sensors	Argus softflowd yaf
Flow record export protocols	IPFIX NetFlow sFlow
Collection systems	Argus flow-tools nfdump NfSen SiLK (flowcap, rwflowpack)
Flow record analysis tools	Argus Client Tools *(ra, racluster, ragraph, ragrep, rahisto, rasort)* EtherApe FlowTraq flow-tools nfdump NfSen SiLK *(PySiLK, rwcount, rwcut, rwfilter, rwidsquery, rwpmatch, rwstats, rwuniq)*

Table 2.5: Taxonomy of network forensic tools according to Davidoff and Ham, 2012, Table source [77].

Complementary categorization can also be based on how the investigator interacts with the tool. Lubis and Siahaan proposed to divide tools into *console* and *GUI* categories; see Table 2.6.

Console-based tools	GUI-based tools
ARP	E-detective
Gnetcast - GNU	Netcat
ifconfig	Wireshark/Ethereal
Network Mapper (Nmap)	
ping	
snoop	
TCP dump	
Xplico	

Table 2.6: Taxonomy of network forensic tools according to Lubis and Siahaan, 2016, Table source [77].

The European Union Cyber Security Agency (ENISA) developed a handbook [17] — *Introduction to Network Forensics* based on the experience of the CSIRT community. Their categorization is similar to Davidoff and Ham [15] based on the intended tool usage; see Table 2.7.

Flow capture & analysis tools	Full-state analysis tools	IDS
Argus	WireShark	Snort

Packet capturing tools	Pattern matching tools
tcpdump	ngrep
dumpcap	

Table 2.7: Taxonomy of network forensic tools according to ENISA, 2019, Table source [77].

Studying the taxonomies, we may observe that each author group focused on different aspects. Garfinkel [23] concentrated on the volatility of the data and the granularity that can be achieved with limited computation resources. In contrast, Meghanathan et al. [45] showed concern for the application domain. Pilli and Joshi [53, 30] extended categorization by focusing on forensic investigators and network administrators. Davidoff and Ham [15] created detailed categorization of NSM tools. Lubis and Siahaan [40] and ENISA [17] also focused on NSM tools. Based on the presented taxonomies, we may conclude that the development of generally usable open source NFAT tools have been put aside.

2.3 Capturing and Processing of In/complete Network Data

Network traffic is the most common data source for NFATs [10, 22, 25, 66]. Although there are tools, mainly NSM, allowing online analysis, like Wireshark and TCPDump, this approach is generally discouraged for forensics [12, 8] because of its bottom-up approach that requires a large amount of manual labor. Forensic science involves repeatability of the investigation process [12], thus rendering these live NSM tools usable in preliminary investigation phases as a control mechanism functional for validation of deployment of Lawful Interception (LI) probes.

Capturing data using LI probes is a complex problem. Due to the volatility of network data, what is not captured is lost forever. This fact poses a challenge for the capturing. The capturing appliances are software [4] or hardware [49, 32, 59] based.

Software-based appliances utilize the kernel functionality of the operating system to capture packets not intended for the particular interface that the interception is running on (e.g., using promiscuous mode [4]). A *naive* approach may be to use plain TCPDump, or Wireshark to capture the network traffic. As several studies have shown [4, 16, 37, 3, 61, 5], this approach leads to severe packet loss. A *sophisticated* approach is to use a kernel module-based tool, e.g., TCPDump compiled with PF_RING, or a commercial solution like ntop's n2disk^{TM} that is optimized for the task and is already based on PF_RING[3]. Empirical experience has shown that, as the vendor claim, n2disk^{TM} can store network traffic up to 10 Gbps. Additionally, using FPGA-based NIC, n2disk^{TM} can store up to 40 Gbps[4].

Hardware-based appliances are typically advanced solutions developed in general by private companies. Their detailed specification and additional functionalities are not publicly available. Some vendors publish the specification in the form of white paper, e.g., *NetQuest* that announces up to 100 Gbps capabilities [49]. Another rare occurrence is research done by Czech's NREN CESNET on hardware-accelerated traffic processing on 100 Gbps networks [32, 59]. Other major players on the market providing not only *packet interception* but also *deep packet inspection* (DPI) for LEA are Sandvine, ENEA Cosmos Division, and XCI, according to the ISS World Training conference [50].

Regarding the state-of-the-art interception appliances, empirical observation shows that not a negligible portion of intercepted network traffic provided to the LEA by ISPs is not without packet loss. A commonly used approach is to utilize port mirroring, i.e., *SPAN port* on a switch that may introduce packet loss under a load [78]. Determination of packet loss on capturing probe is challenging by itself. The TCP reassembling can be used to prove that some part of data transmitted over a network was missing from the packet trace. Still, it does not necessarily prove that the capturing appliance is at fault because of other possibilities like asymmetrical routing. To determine packet loss of protocols on UDP, additional analysis and understanding of application protocol are required (providing that application protocol carries identifiers that can be used).

The practice has shown that network forensic practitioners need tools tolerant of packet loss. These tools have to use application protocol parsers that do not stop on the first invalid data but contain a robust parsing engine that allows for rewinding the invalid portion of data streams.

2.4 Application Protocol Identification

The application protocol identification is an inherent part of network forensics. Without the precise knowledge of the application protocol in question, the NFAT or NSM tool cannot extract crucial information carried by the protocol because the tool would not know which application protocol parser to use.

[3] `https://www.ntop.org/products/packet-capture/pf_ring/`

[4] `https://www.ntop.org/products/traffic-recording-replay/n2disk/`

In digital forensics, particularly storage device or mobile device forensics, the artifacts are identified by matching the hashes of investigated files to well-known ones stored in databases like VirusTotal[5]. This approach filters the system or otherwise uninteresting files and allows the investigator to focus only on individual files that most likely contain the digital evidence.

This approach works very well for static / constant files but is not directly applicable to computer networks because of their entropy. Transferring the same static file using the same application protocol may result in data streams with different characteristics. The data stream checksum would differ because of variable internet, transport, and application protocol fields. Additionally, external aspects dependent on the transfer media type, network utilization, and quality of the service (retransmissions) may also differ.

The most straightforward method of application protocol identification is to use well-known protocol port numbers. This method utilizes port numbers present in the transport protocols, either TCP or UDP. The accuracy of this method is about 60–80 % [47, 6] and hugely depends on a particular sample of applications in question. Services may use random protocol ports usually defined by a service administrator or used implicitly for services like multimedia streaming, multiplayer games, or various types of traffic tunneling.

Because LEA is traditionally focused on extracting as much information/meta information as possible, we need to go deeper and improve the accuracy. Traditionally, there are several directions we may take [51, 33, 48, 71, 76, 64].

Supervised machine learning [28] tackles the problem with learning by example. The classification model is created using annotated data sets. Usually, application protocols contained in the data set are classified with reasonable accuracy. Protocols that were not part of the training set are often miss-classified into one of the known categories.

Unsupervised machine learning [19] is a technique that implicitly expects that there are unknown application protocols in the data set. This method does not require a data set to be trained on. Categorization is done on the data during the classification process. Similar samples of application protocols are assigned to the same category.

Semi-supervised machine learning [18] is a combination of the approaches above. The sample application communication is categorized using the clustering/unsupervised methods, and by applying the supervised method/s, we may infer a correct label for otherwise unclassified samples.

Machine learning methods require data preprocessing that is concluded with feature extraction. In this domain, we recognize the following feature categories with respective extraction methods [51, 33, 48, 71, 48]:

Payload analysis extracts features from the packet contents (payload/s). This method works well for unencrypted / plain-text application protocols but poorly for encrypted ones.

Statistical methods [28, 35, 24] do not look into the data but use metadata, such as information about packet size, inter-packet delays, etc. This method also works for encrypted application protocols.

[5] `https://www.virustotal.com/`

Hybrid methods combine payload analysis with statistical models. Utilized features may be a combination of the methods mentioned above. These methods work well for encrypted and unencrypted application protocols [9, 41].

Additional sources summarising advances in application protocol identification / classification are surveys by Nguyen and Armitage [51], Namdev et al. [48], and Velan et al. [71], who focused on encrypted traffic. Recently, JA3 emerged as a defacto standard for fingerprinting clients and JA3S for fingerprinting service providers (servers) [1]. The best results were achieved by pairing JA3 and JA3S to identify client and server applications / services. Note that this identifies particular implementation of client/service that may change in time due to service updates [2]. Mercury by McGrew et al. (Cisco) [44] is similar, based on a broader feature set. Even though it may seem that fingerprinting of SSL/TLS has been solved (using JA3 or Mercury), Hejcman's bachelor thesis [26] shows that it may be further refined.

The contribution of this thesis is based on the previous work of the following authors. Erik Hjelmvik's SPID [27, 28] statistical-based algorithm, further improved by Kohnen [36], is a very lightweight algorithm capable of application protocol identification on the fly from the beginning of the flows. Foroushani and Zincir-Heywood [20] have shown in graphical details possibilities of separation of different encrypted application protocols using statistical information extracted from the flows. Dai et al. [14] and Miskovic et al. [46] studied communication-based fingerprinting of mobile applications. Erman et al. [18] described a flow-based semi-supervised classification method that can accommodate known and unknown applications.

Due to a significant investment required to create and maintain traditional application protocol identification methods, current research is exploring additional paths. A survey done by Wang et al. [75], who summarized possibilities achievable by applying Deep Learning, shows promising results. Compared to the aforementioned traditional methods, Deep Learning may ease maintainability and overcome limitations posed by time-consuming, costly handcrafted features and frequent feature updates.

Nowadays, a need for fine-grained classification arises. Fu et al. [21] evaluated their system CUMMA for classifying mobile messaging app service usage by jointly modeling user behavioral patterns, network traffic characteristics, and temporal dependencies. Using a statistical-based approach, they can segregate messages into classes such as text, audio notes, pictures, stream voice calls, location sharing, and short videos. They showed that this segregation is possible without decryption keys and deep packet inspection of contents.

2.5 Overlay and Tunneling Network Protocols

Overlay networks are becoming popular for creating virtual / logical networks over physical infrastructure. Overlays are no longer a domain of traditional VPN protocols like PPTP, GRE, L2TP, and OpenVPN. Novel, encrypted by design, protocols such as Hamachi, ZeroTier, and WireGuard are increasing their popularity. Additionally, the rise of anonymization networks like Freenet[6], the Tor Project[7], and the Internet Invisibility Project (I2P)[8] complicated forensic investigation even further.

[6] `https://github.com/freenet/fred`

[7] `https://www.torproject.org/`

[8] `https://geti2p.net/en/`

Support for some overlay or tunneling protocols in NFAT tools is rare. *Pyflag* does not have any support. *Xplico* supports L2TP, VLAN, and PPP. More comprehensive support for encapsulation is implemented in NSM tools, particularly Wireshark and TShark. *NetworkMiner* supports GRE, 802.1Q, PPPoE, VXLAN, OpenFlow, SOCKS, MPLS, EoMPLS, and ERSPAN.

The reason is most likely that NFATs dissect protocols to extract information. Encrypted overlay networks are not particularly interesting in this regard. On the other hand, their presence may be a piece of helpful information for the investigator [67]. Their detection and meta-information about an encapsulated content extraction extend the topic of application protocol identification; see Section 2.4. This lack of overlay protocol support in NFATs opens up novel research opportunities in this field.

2.6 Network Forensics of Big Data

Network forensic investigation is no longer a domain that deals with small packet traces of a few hundred megabytes [34]. The penetration of high-speed internet connection for small businesses, residents, and even mobile devices is more significant than ever. The boom of multimedia consumerism in teen generations [11] pushes network infrastructures to unprecedented growth, and naturally, we would expect NFATs to keep up.

Contradictory, the scientific community's interest in developing parallel or even better scalable and distributed (so as not to confuse with cloud forensics) NFATs has not increased in the last decade or so. Vallentin has done thorough state-of-the-art research in this area in his dissertation [69] under the supervision of Vern Paxson, who covered the years 2005-2015. Duplicating a detailed overview of this period in this work would be wasteful. Vallentin concludes that "The academic treatment of large-scale network forensics is strikingly thin." and that "The last decade of research on network forensics paints a fragmented picture: only occasional interest, even in security-centric venues." Vallentin concluded his research in the publication called "{VAST}: A Unified Platform for Interactive Network Forensics" [70]. Recently, VAST was used by other research groups [60] as a backend for distributed computation.

Since Vallentin's time, several other authors have researched the acceleration possibilities of distributed network forensics. D'Alessandro investigated options of scalable network traffic classification using distributed support vector machines [13] which are a crossover with Section 2.4. Ryšavý, Rychlý and Jeřábek [63, 29, 62] used Apache based technologies, namely Hadoop, Spark, Kafka, Ignite. Their research focused on identification and clustering in big data network flow traces.

Chapter 3

Research Summary

This chapter summarizes the research included in this dissertation and the related contributions of mine. The research presented can be commonly classified into computer science, computer security, digital forensics, and network forensics. Section 3.1 summarizes the work and highlights motivations and contributions for the presented research. For better navigation, see Figure 1.1, which shows the relationships between research objectives and selected papers.

Section 3.2 compose a list of seven selected papers included in this dissertation. A brief overview of each publication is provided in a summary form containing motivation and related contributions. Additional information is included as abstract, original citation form, and references to publications that cited the paper. In case other publications preceded the paper, its citations are also referenced.

Section 3.3 contains a list of my other publications related to this dissertation but not included. The list consists of technical reports and student conference publications that were published before my doctoral studies but bear witness to my interest in this field since I was an undergraduate student.

Section 3.4 lists national and international projects in which I have participated. Section 3.5 enumerates the software and specimens that I have contributed. Section 3.6 contains a list of presentations, posters, and invited speeches I have given in the last decade concerning the research covered by this dissertation. Section 3.7 and Section 3.8 list related / unrelated bachelor and master theses I have supervised.

3.1 Overview

This dissertation aims to help law enforcement agency (LEA) investigators conducting a criminal investigation to be more efficient in their work and lower the requirements for their preliminary understanding of technical details, allowing them to focus on the investigative side. Using Network Forensic and Analysis Tools along the lines of regular investigators provides them with the means to process captured network communication directly and extract information instead of waiting for dedicated IT professionals to preprocess the data for them. To achieve this goal, I have participated in several grant projects that focused on the needs of network forensic investigators, researched various problems, and helped develop the tools mentioned earlier.

Along the way, we have faced several difficulties that posed exciting research challenges beyond engineering and required a rigorous scientific approach to be solved. We were asked to develop a network forensic investigation tool to consume the captured network communication and extract information from the application messages. With this information, the investigator can build a case.

The first challenge was to design the application architecture. The *paper VII*, describes the first iteration of the Netfox Detective application. We tried to separate it into layers with well-defined interfaces focusing only on their respective concerns. This paper elaborates on several challenges that we have encountered in this implementation. The most severe one was a recognition that the quality of the input data varies, and forensic application needs to incorporate this into their design.

Therefore, *paper VI* investigates the steps to take to fully overcome Internet service providers' inability to capture network communication without packet loss. Or face other factors like asymmetric routing, load-balancing, etc., that causes the captured data to be incomplete. We have developed several heuristics that utilize information from lower than application protocol layers to create abstractions of application messages that application protocol parsers can parse to extract the contents to form the evidence.

Furthermore, we realized that these heuristics and application protocol abstractions might contribute to more precise application protocol identification and maybe allow us to identify the exact applications that generated the communication. *Paper V* describes our achievements in this regard. The result was an implementation of a framework that could extract features from application data flow and conversations to allow us to benchmark several approaches and machine learning algorithms. Once again, all of this was implemented in the Netfox Detective NFAT application.

Paper IV targets in-field LEA operatives that need to gather evidence directly from a wireless local area network (WLAN). Allowing the investigator to access the LAN directly, compared with an investigation of captured network communication on the Internet service provider level, introduces several benefits discussed in the paper. Automating this kind of Man-in-the-Middle attack complies with our goal of allowing regular investigators to obtain the needed evidence without waiting for an IT specialist to get it for them.

The increasing amount of data transmitted over the network required investigating new processing methods. The clear choice to increase the performance of something embarrassingly parallel is to scale up the computation. *Paper III* describes our attempts to design and implement a scalable framework for network forensics. Previous papers VII, VI, and V show our approach to network data processing in a single process on a single machine. This paper investigates these methods and scales the processing linearly while maintaining the same robust incomplete data processing.

The last challenge we identified was the processing of tunneled traffic. *Paper II* describes the common tunneling protocols that LEA investigators can encounter. We chose one of the most complex protocols, the *Generic Stream Encapsulation*, to demonstrate how a complex tunneling protocol can be dissected by our processing framework while maintaining its properties of robustness for incomplete data processing.

The final *Paper I* concludes this dissertation. This paper describes all the methods and principles we have designed to overcome the challenges posed by the incomplete data processing for network forensic analysis on a single machine. This paper introduces the final version of the *Netfox Detective* tool as a Proof-of-concept platform to demonstrate the correctness and usability of the methods designed for network forensic investigation. Additionally, a crossover to Network Security Monitoring is shown by using the platform to visualize SIP Fraud attacks in cooperation with Czech NREN CESNET.

3.2 Papers Included in this Dissertation

This section provides an overview of selected papers included in this dissertation. An explanation of its motivation is included in each paper, and direct contributions are mentioned. The author's participation in creating the publication is noted with the conference / journal ranking or impact factor.

3.2.1 Paper I

Jan Pluskal, Frank Breitinger, and Ondřej Ryšavý. "Netfox detective: A novel open-source network forensics analysis tool". In: *Forensic Science International: Digital Investigation* 35 (2020), p. 301019. ISSN: 2666-2817

Author's participation: 50 %[1]
Impact factor: 1.805 (Q1)

Motivation and contributions

This is the most recently published paper that summarizes my research in the area of network forensics and the development of an open source network forensics and analysis tool — Netfox Detective[2]. This tool served as a Proof-of-Concept platform that demonstrated the functionality of each feature described in the previous papers (except for scalability and research related to WiFi) and therefore demonstrated its correctness.

This paper described Netfox Detective, a novel, easy-to-use, powerful network forensic platform for top-down investigations. The tool covered the forensic model's examination, analysis, and investigation phases as defined by Pilli [55]. The following contributions are provided in detail:

1. Introduction of the investigation profiles that contained all necessary data for sharing the case between multiple investigators.

2. The new method to reassemble the TCP stream based on heuristics (the method itself was previously published [45], but the tool contains an improved version).

3. Improved identification of application-level sessions within TCP streams; the system could identify more application sessions compared to other tools.

4. Support for analysis of traffic encapsulated in the GSE protocol; to the best of my knowledge, Netfox Detective had been the only open source NFAT that supported GSE.

5. A novel web page reconstruction approach; compared to other tools, the tool not only extracted objects from HTTP communication but also reconstructed the page entirely (rewriting sources of all intercepted objects like CSS, pictures, video streams, etc.). Pages were stored as MAFF archives, including snapshots showing how the page changed over time. JavaScript was interpreted, and particular API calls were mocked to be injected with intercepted ones, like REST API calls. The reconstruction of a web page required analysis and correlation of multiple L7 conversations because a page usually references (includes) data from various domains.

[1] Author participation states the contribution index used for publication submission into the Czech national database of research, development and innovations (RIV).

[2] `https://github.com/nesfit/NetfoxDetective`

The paper has been cited in:

- Kousik Barik, Saptarshi Das, Karabi Konar, Bipasha Chakrabarti Banik, and Archita Banerjee. “Exploring user requirements of network forensic tools”. In: *Global Transitions Proceedings* 2.2 (2021), pp. 350–354

Abstract

Network forensics is a significant sub-discipline of digital forensics, which has become increasingly important in an age where everything is connected. To deal with the amounts of data and other network challenges, practitioners require powerful tools that support them. This paper highlights a novel open source network forensic tool named Netfox Detective that outperforms existing tools such as Wireshark or NetworkMiner in certain areas. For instance, it provides a heuristically based engine for traffic processing that can be easily extended. Our application tolerates malformed or missing conversation segments using robust parsers (we rely not solely on the RFC description but heuristics). Besides outlining the tool’s architecture and basic processing concepts, we also explain how it can be extended. Lastly, a comparison with similar tools is presented, and a real-world scenario is discussed.

3.2.2 Paper II

Jan Pluskal, Michal Koutenský, Martin Vondráček, and Ondřej Ryšavý. “Network Forensic Investigations of Tunneled Traffic: A Case Study”. In: *Revue roumaine des sciences techniques. Série Électrotechnique et Énergétique* 64.4 (2019), pp. 429–434. ISSN: 0035-4066

Author’s participation: 25 %

Impact factor: 0.76 (Q3)

Motivation and contributions

The present paper provided an overview of the expected points in the network topology that law enforcement agencies (LEA) can use to conduct lawful interception. We summarized the most used tunneling protocols and discussed their features concerning digital forensic analysis. For each protocol, the possibility of content extraction was explained. Also, a brief overview of methods for encapsulated traffic classification was provided. The problem of connection recovery from tunneled communication was demonstrated using the GSE protocol as an example.

Abstract

The increasing importance of network forensics in the investigations conducted by Law Enforcement Agencies is indisputable. Today’s Internet does not carry ordinary TCP/IP traffic but utilizes many other encapsulations and tunneling protocols. This paper provides an overview of the most used tunneling protocols and their features with regard to digital forensic analysis. A generic stream encapsulation case study describes how the investigator can obtain encapsulated application data from within.

Preceding related paper: Jan Pluskal, Martin Vondráček, and Ondřej Ryšavý. "Network Forensics in GSE Overlay Networks". In: *Proceedings of the 6th Conference on the Engineering of Computer Based Systems.* ACM. 2019. ISBN: 9781450376365

Author's participation: 60 %
Conference ranking: N/A

3.2.3 Paper III

Viliam Letavay, Jan Pluskal, and Ondřej Ryšavý. "Network Forensic Analysis for Lawful Enforcement on Steroids, Distributed and Scalable". In: *Proceedings of the 6th Conference on the Engineering of Computer Based Systems.* ACM. 2019. ISBN: 9781450376365

Author's participation: 30 %
Conference ranking: N/A

Motivation and contributions

This paper described a scalable architecture design for processing network packet traces at that time work in progress. According to our previous research, the processing speed of Netfox Detective, which was around 100 Mbps, seemed too slow, and we were looking for acceleration possibilities. Because the task of conversation tracking and consequence transport protocol processing (creating abstractions of application messages, possibly using TCP reassembling and heuristics) is embarrassingly parallel, we realized that we could scale the job on multiple devices instead of one processing unit.

The contribution of this paper lies in the design, performance, and properties discussion of a new Network Forensic and Analysis Tool (NFAT) – Network Traffic Processing & Analysis Cluster (NTPAC). This specimen utilizes distributed computing architecture to improve the performance of network traffic analysis while being less demanding on hardware requirements than related systems.

To extract evidence from network packets, we must thoroughly analyze them, performing several consecutive operations such as packet dissecting, flow identification, network stream composition, application protocol identification, and message parsing and artifact extraction. Unlike the other NFAT tools, NTPAC could correctly process captured malformed traffic without yielding misleading evidence. NTPAC performed a forensic analysis of network traffic in high-speed networks. The system design used a scalable approach to run the tool on a single machine and a computing cluster. Compared to other NFAT tools, NTPAC was an order of magnitude faster and was scaling linearly.

The paper has been cited in:

- Daniel Gustavsson. *Molnforensik: En litteraturstudie om tekniska utmaningar och möjligheter inom IT-forensik mot molnet.* 2020

Abstract

Forensic analysis of intercepted network traffic focuses on finding and extracting communication evidence, such as instant messaging, email, VoIP calls, localization information, documents, and images. Due to the amount of data captured, this process is time-consuming and complicated. Most commonly used forensic network analysis tools have limited capabilities for extensive data processing. In this paper, we are introducing a new tool that

achieves better data processing performance using available computing resources through distributed processing. Thanks to the technology used, this tool can be used on commodity hardware in a local area network, in a dedicated computing cluster, or cloud environment.

Preceding related paper: Viliam Letavay, Jan Pluskal, and Ondřej Ryšavý. "A Scalable Architecture for Network Traffic Forensics". In: *The Fifteenth International Conference on Networking and Services ICNS 2019.* Athens, GR: The International Academy, Research and Industry Association, 2019, pp. 32–36. ISBN: 9781612087115

Author's participation: 30 %
Conference ranking: B3 (Qualis)

The paper has been cited in:

- Kousik Barik, Saptarshi Das, Karabi Konar, Bipasha Chakrabarti Banik, and Archita Banerjee. "Exploring user requirements of network forensic tools". In: *Global Transitions Proceedings* 2.2 (2021), pp. 350–354

3.2.4 Paper IV

Martin Vondráček, Jan Pluskal, and Ondřej Ryšavý. "Automated Man-in-the-Middle Attack Against Wi-Fi Networks". In: *The Journal of Digital Forensics, Security and Law: JDFSL* 13.1 (2018), pp. 59–80. ISSN: 1558-7215

Author's participation: 30 %
Impact factor: N/A

Motivation and contributions

This paper is based on Martin Vondracek's bachelor thesis [105], deals with the automation of MitM attack on Wi-Fi networks and is also supported by software [42]. Due to its wireless nature, Wi-Fi networks constitute an ideal data source for LEA investigation. Capturing traces from local Wi-Fi may bring new information because local services (non-routable on the public internet) tend to be poorly secured. The additional benefit of being connected to the local network is the more offensive possibility of conducting MitM attacks. Various commercial vendors developed and sold tactical solutions to support this use case.

The contribution of this research was gathering state-of-the-art tools and approaches for penetration of wireless networks and developing an overlay application that allowed for a regular, non-technical person to operate it. In this way, field LEA operators could gather evidence from wireless networks without the complex knowledge of an IT professional.

Additionally, we focused on the detection possibilities of wireless attacks on devices intended for home use. The analysis showed that even without enterprise-level monitoring and logging, an attack on these low-power devices introduces a noticeable increase in latency that can be monitored, and an alert can be raised.

The paper has been cited in:

- Tina Wu, Frank Breitinger, and Stephen O'Shaughnessy. "Digital forensic tools: Recent advances and enhancing the status quo". In: *Forensic Science International: Digital Investigation* 34 (2020), p. 300999
- Mohamed Amine Khelif, Jordane Lorandel, Olivier Romain, Matthieu Regnery, Denis Baheux, and Guillaume Barbu. "Toward a Hardware Man-in-the-Middle Attack on PCIe Bus for Smart Data Replay". In: *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE. 2019, pp. 230–237
- Mohamed Amine Khelif, Jordane Lorandel, Olivier Romain, Matthieu Regnery, Denis Baheux, and Guillaume Barbu. "Toward a hardware man-in-the-middle attack on pcie bus". In: *Microprocessors and Microsystems* 77 (2020), p. 103198
- Crispin R Jose. "Exploring Security Process Improvements for Integrating Security Tools within a Software Application Development Methodology". PhD thesis. Colorado Technical University, 2020
- Cynthia Valeria Maza Gonzalez and Fabián Gustavo Rochina Manobanda. "Estado del arte utilizando mapeo sistemático de seguridad de redes domésticas WIFI en familias ecuatorianas". B.S. thesis. 2021

Abstract

Currently used wireless communication technologies suffer security weaknesses that can be exploited, allowing eavesdropping or spoofing of network communication. This paper presents a practical tool that can automate the attack on wireless security. The package developed, wifimitm, provides functionality to automate MitM attacks in a wireless environment. The package combines several existing tools and attack strategies to bypass wireless security mechanisms, such as WEP, WPA, and WPS. The tool presented can be integrated into a solution for automated penetration testing. Also, a popularization of the fact that such attacks can be easily automated should raise public awareness of the state of wireless security.

Preceding related paper: Martin Vondráček, Jan Pluskal, and Ondřej Ryšavý. "Automation of MitM Attack on Wi-Fi Networks". In: *9th International Conference on Digital Forensics & Cyber Crime*. Vol. 2018. 216. Springer International Publishing, 2017, pp. 207–220. ISBN: 9783319736969

Author's participation: 30 %
Conference ranking: N/A

The paper has been cited in:

- Tina Wu. "Digital forensic investigation of IoT devices: tools and methods". PhD thesis. University of Oxford, 2020
- Duc Le Tran, Thong Trung Tran, Khanh Quoc Dang, Reem Alkanhel, and Ammar Muthanna. "Malware Spreading Model for Routers in Wi-Fi Networks". In: *IEEE Access* 10 (2022). All Open Access, Gold Open Access, pp. 61873–61891. DOI: 10.1109/ACCESS.2022.3182243

3.2.5 Paper V

Jan Pluskal, Ondrej Lichtner, and Ondřej Ryšavý. "Traffic Classification and Application Identification in Network Forensics". In: *Fourteenth Annual IFIP WG 11.9 International Conference on Digital Forensics.* Ed. by Gilbert Peterson and Sujeet Shenoi. New Delhi, IN: Springer International Publishing, 2018, pp. 161–181. ISBN: 9783319992778

Author's participation: 40 %
Conference ranking: B5 (Qualis)

Motivation and contributions

This paper introduced a novel approach to application protocol identification and application (that generated the communication) classification. The identification/classification of the application protocol is necessary for any Network Security Monitoring tool or Network Forensic Analysis Tool to extract any useful information from the application layer. Tools use application parsers to extract this information, but without the knowledge of the application protocol, the tool is unaware of which application parser to use. Various application protocol parsers may consume any data; therefore, their acceptance of the application data stream cannot be used for identification purposes. The ability to also identify an application that generated the communication yields additional value to a forensic investigator.

The contributions of this paper are presented in the following points:

Testbed that implemented three classification methods, namely Bayesian Network, Random Forests, and Enhanced Statistical Probability Identification, was presented. Additionally, feature extraction was implemented as a modular framework allowing users to create and experiment with new features. The entire testbed used GUI for experimenting with feature elimination, classification, and visual result analysis. The analysis allowed for various feature comparisons and visualization of the feature correlation matrix. The user could iterate and experiment with the testbed to proceed with the hyperparameter tuning.

Dataset created in laboratory environment in cooperation with various students simulating real activity / work on staged computers, over multiple days containing 19,5 GB of annotated captured network communication in the form of enhanced PCAP files. Capturing traffic using Namon[3] [115, 23], we have created a unique, annotated, captured network trace dataset that has been publicly available since publication.

Feature elimination as an automated process that allowed us to create optimal classifiers that omit correlated features for a particular application protocol or application communication.

Classification of applications as a finer-grained complement to the identification of the application protocol was described.

Bayesian Network classifier enhanced with automated feature elimination was created and trained on the aforementioned dataset.

Random Forests classifier enhanced with automated feature elimination was created and trained on the aforementioned dataset.

[3] `https://jzlka.github.io/namon/`

Enhanced Statistical Probability Identification method was developed, benchmarked, and compared to a baseline formed by Bayesian Network and Random Forests classifiers. In comparison, this method did not embed explicit feature elimination because it is an inherent part of it.

The paper has been cited in:

- Hilmand Khan, Sarmad Hanif, and Bakht Muhammad. "A survey of machine learning applications in digital forensics". In: *Trends in Computer Science and Information Technology* 6.1 (2021), pp. 020–024
- Kousik Barik, A Abirami, Karabi Konar, and Saptarshi Das. "Research Perspective on Digital Forensic Tools and Investigation Process". In: *Illumination of Artificial Intelligence in Cybersecurity and Forensics*. Springer, 2022, pp. 71–95

Abstract

Network traffic classification is essential for network monitoring, security analyses, and digital forensics. Without an accurate traffic classification, the computational demands imposed by analyzing all IP traffic flows are enormous. Classification can also reduce the number of flows that must be examined and prioritized for analysis in forensic investigations.

This chapter presents an automated feature elimination method based on a feature correlation matrix. Additionally, it proposes an enhanced statistical protocol identification method compared to Bayesian network and random forests classification methods that offer high accuracy and acceptable performance. Each classification method is used with a subset of features that best suit the method. Methods are evaluated based on their ability to identify the application layer protocols and the applications themselves. Experiments demonstrate that the random forests classifier yields the most promising results, while the proposed enhanced statistical protocol identification method provides an interesting trade-off between higher performance and slightly lower accuracy.

3.2.6 Paper VI

Petr Matoušek, Jan Pluskal, Ondřej Ryšavý, Vladimír Veselý, Martin Kmeť, Filip Karpíšek, and Martin Vymlátil. "Advanced Techniques for Reconstruction of Incomplete Network Data". In: *Digital Forensics and Cyber Crime*. Ed. by Joshua I. James and Frank Breitinger. Cham: Springer International Publishing, 2015, pp. 69–84. ISBN: 9783319255125

Author's participation: 20 %
Conference ranking: N/A

Motivation and contributions

This paper deals with the reconstruction of incomplete network data and thus answers the research question proposed by the previous publication [72]. Because network data is volatile and what is not captured is lost forever, we need to create a robust framework supporting these robust application protocol parsers to extract as much information as possible from the application conversation. This paper advances Netfox Framework, originally developed as a part of my master thesis [60], and adds additional functionalities, namely support for the processing of encrypted communication implemented by Miroslav Slivka [92], and improved by Viliam Letavay [34].

The main contribution of this paper is the robust algorithm for reassembling potentially incomplete network data, its heuristics, and its ability to signal this information to the application protocol parsers. This way, conversation tracking is not only using data from Internet (L3) and Transport (L4) layers but also embeds the L4 reassembling. *L7PDUs* are introduced as abstractions of application messages.

The analysis showed that without this incorporation of reassembling into the conversation tracking, the other NFATs provided incorrect conversation tracking in case particular parts of TCP signaling were missing, thus giving the investigators incorrect results. The possibility of joining multiple TCP flows into one may lead to false evidence.

Furthermore, this paper presented a novel approach to *Web Mail* analysis. It used multiple HTTP decoders to process webmail communication and search for patterns commonly used in that communication. Using this approach, we were able to extract the contents of webmails from captured traces of several online email services.

The precondition for this webmail and other analyses was implementing SSL/TLS decryption support into the tool. With this module activated, it was possible to run application protocol parsing modules (Snoopers) on decrypted traffic under one of the following conditions. Either a private server key was included with the packet traces in case RSA (non-ephemeral) key negotiation was used. Or pre-master secrets from MITM proxy were included.

Lastly, the paper discussed the possibilities of Bitcoin traffic detection and metadata extraction. This functionality was tested in a real-world investigation and helped provide evidence for a criminal investigation of foreign (EU) LEA.

The paper has been cited in:

- Yanchao Wang, Zhongqian Su, and Dayi Song. “File Fragment Type Identification with Convolutional Neural Networks”. In: *Proceedings of the 2018 International Conference on Machine Learning Technologies*. ACM. 2018, pp. 41–47

- David Muelas, Jorge E López de Vergara, Javier Ramos, José Luis García-Dorado, and Javier Aracil. “On the impact of TCP segmentation: Experience in VoIP monitoring”. In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE. 2017, pp. 708–713

- Haidong Ge, Ning Zheng, Lin Cai, Ming Xu, Tong Qiao, Tao Yang, Jinkai Sun, and Sudeng Hu. “Adaptive Carving Method for Live FLV Streaming”. In: *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer. 2017, pp. 554–566

- David Muelas Recuenco. “Flexible Network Monitoring and Traffic Analysis Techniques for the Future Internet”. PhD thesis. Universidad Autonoma de Madrid, 2019

Abstract

Network forensics is a method of obtaining and analyzing digital evidence from network sources. Network forensics includes data acquisition, selection, processing, analysis and presentation to investigators. Due to the large volumes of transmitted data, acquired information can be incomplete, corrupted, or disordered, making further reconstruction difficult. In this paper, we address the issue of advanced parsing and reconstruction of

incomplete, corrupted, or disordered data packets. We introduce a technique that recovers TCP or UDP conversations so that application parsers can further analyze them. The presented method is implemented in a new network forensic tool called Netfox Detective. We also discuss current challenges in parsing webmail communication, SSL decryption, and Bitcoin detection.

3.2.7 Paper VII

Jan Pluskal, Petr Matoušek, Ondřej Ryšavý, Martin Kmeť, Vladimir Veselý, Filip Karpíšek, and Martin Vymlátil. "Netfox Detective: A tool for advanced network forensics analysis". In: *Proceedings of Security and Protection of Information (SPI) 2015*. Brno, CZ: Brno University of Defence, 2015, pp. 147–163. ISBN: 9788072319978

Author's participation: 15 %
Conference ranking: N/A

Motivation and contributions

This paper describes the first iteration of the implementation of the Netfox Detective tool. The focus is given on the Netfox Framework's architecture, that is, the implementation of business logic and Netfox Detective, which stood for the implementation of the GUI.

The contribution of this work was the composition of several diploma theses and related research projects. My master thesis [60] produced the Netfox Framework, supported by publications at student conferences [73, 65], which also contained a re-implementation of my bachelor thesis [56], supported by publication at student conferences [57]. Martin Mares's master thesis [41] developed the GUI – Netfox Detective. Martin Kmet's master thesis [28] dealt with detecting of RTP traffic without signaling information obtained from SIP. Vladimir Vesely's *PmLib* [98] implemented logic to open PCAP files and parsed packets up to the transport layer.

This paper proposed a research question regarding the importance of correct processing of incomplete network data. The concrete method that allows the extraction of VoIP communication without signaling from SIP [28], even when a portion of the communication is missing, is presented with a more generic solution described in the following paper [45].

The paper has been cited in:

- Beatriz Parra de Gallo. "Advances in the application of Ontologies in the area of Digital Forensic Electronic Mail". In: *IEEE Latin America Transactions* 17.10 (2019), pp. 1694–1705
- Caroline Wanjira Macharia. "Maintaining a bitcoin address repository through focused web crawling". MA thesis. Strathmore University, 2017

Abstract

Network forensics is a process of capturing, collecting, and analyzing network data for information gathering, legal evidence, or intrusion detection. The new Internet generation opens novel opportunities for cybercrime activities and security incidents using network applications. Security administrators and LEA (Law Enforcement Agency) officers are challenged to use advanced tools and techniques to detect unlawful or unauthorized activities. In case of grave suspicion of criminal activity, network forensic tools and techniques are used to find

legal evidence in a captured network communication that proves or disproves the suspect's participation in that activity.

Today, various commercial or free tools for network forensic analysis are available, e.g., Wireshark, Network Miner, NetWitness, Xplico, NetIntercept, or PacketScan. Many of these tools fail to successfully reconstruct communication when using incomplete, duplicated, or corrupted input data. Investigators also require advanced automatic processing of application data that helps them to see the actual content of the conversation, including chats, VoIP talks, file transmission, email exchange, etc.

Our research focuses on designing and implementing a modular framework for network forensics with advanced possibilities for application reconstruction. The proposed architecture consists of (i) input packet processing, (ii) an advanced reconstruction of L7 conversations, and (iii) application-based analysis and presentation of L7 conversations. Our approach employs various advanced reconstruction techniques and heuristics that work even with corrupted or incomplete data, e.g., one-directional flows, missing synchronization, unbounded conversations, etc.

The proposed framework was implemented in a tool called Netfox Detective developed by our research group. This paper shows its architecture from a functional and logical point of view and its application in the reconstruction of webmail traffic, VoIP, and RTP transmissions.

3.3 Relevant Publications not Included in this Dissertation

1. Jan Pluskal. *Netfox Detective 2.0 - Nástroj pro síťovou forenzní analýzu.* Czech. Tech. rep. FIT-TR-2017-06, CZ, 2017, p. 16

   Author's participation: 100 %

2. Jan Pluskal, Ondrej Lichtner, and Ondřej Ryšavý. *Netfox Detective - Identifikace aplikačních protokolů pomocí algoritmů strojového učení.* Czech. Tech. rep. FIT-TR-2017-05, CZ, 2017, p. 19

   Author's participation: 90 %

3. Jan Pluskal, Ondřej Ryšavý, and Vladimir Veselý. "NetFox - The network forensic extandable analysis tool". In: *6th AFCEA Student Conference Future of Information and Communication Technology.* Bucharest, RO: University Politehnica of Bucharest, 2014, pp. 68–71. ISBN: 9786065510470

   Author's participation: 90 %

4. Jan Pluskal. "NetFox.Framework - The network forensic extandable analysis tool". In: *Proceedings of the 20th Conference STUDENT EEICT 2014 Volume 2.* Brno, CZ: Brno University of Technology, 2014, pp. 280–282. ISBN: 9788021449237

   Author's participation: 100 %

5. Jan Pluskal. "Analýza a rekonstrukce komunikace typu instant messaging (YMSG a ICQ)". Czech. In: *Proceedings of the 18th Conference Student EEICT 2012 Volume 1.* Brno, CZ: Faculty of Information Technology BUT, 2012, pp. 176–178. ISBN: 9788021444607

   Author's participation: 100 %

3.4 Research Projects and Grants

1. **TENACITy: Travelling intelligENce Against CrIme and Terrorism**, team member, *EC EU - HORIZON EUROPE*, 101074048, 2022-2025

2. **Development of Decoder for IP Traffic**, deputy team leader, team member, VH20192021043, *Ministry of the interior of the Czech Republic*, 2019-2021

3. **Modern and Open Study Techniques**, team member, OP VVV PO2 ESF, *Ministry of Education, Youth and Sports Czech Republic*, 2015-2020

4. **Integrated Platform for Analysis of Digital Data from Security Incidents**, team member, VI20172020062, *Ministry of the interior of the Czech Republic*, 2017-2018

5. **Using Network Analysis Techniques to Prevent Data Loss**, research leader, Safetica, *MPO*, 2017

6. **Research and application of advanced methods in ICT**, team member, FIT-S-14-2299, *Brno University of Technology*, 2014-2016

7. **Modern tools for detection and mitigation of cyber criminality on the New Generation Internet**, team member, VG20102015022, *Ministry of the interior of the Czech Republic*, 2010-2015

3.5 Software and Specimen

1. Letavay Viliam, Pluskal Jan, Veselý Vladimír, and Grégr Matěj. *HTTP Keylogger - tool for web activity monitoring,* [Computer Software]. 2019

2. Letavay Viliam, Pluskal Jan, and Jeřábek Kamil. *Banana Pi BPI-R2 Cluster Prototype.* [Specimen]. 2018

3. Pluskal Jan. *SupportApp.* [Computer Software]. 2018

4. Pluskal Jan. *Netfox Detective 2.0 - Nástroj pro síťovou forenzní analýzu.* [Computer Software]. 2017

5. Zuzelka Josef, Pluskal Jan, Ryšavý Ondřej, and Matoušek Petr. *Modul pro zpracování zapouzdřeného síťového provozu.* [Computer Software]. 2017

6. Janeček Vít, Pluskal Jan, Ryšavý Ondřej, and Matoušek Petr. *Modul pro zpracování zapouzdřeného síťového provozu.* [Computer Software]. 2017

7. Pluskal Jan. *AppIdent - Tool for Network Application Protocols Identification.* [Computer Software]. 2017

8. Vondráček Martin, Pluskal Jan, Ryšavý Ondřej, and Matoušek Petr. *Automation of MitM Attack on WiFi Networks.* [Computer Software]. 2016

9. Marušic Marek, Pluskal Jan, Ryšavý Ondřej, and Matoušek Petr. *Automatization of MitM Attack for SSL/TLS Decryption, software.* [Computer Software]. 2016

10. Hvězda Matěj, Pluskal Jan, Ryšavý Ondřej, and Matoušek Petr. *Network Forensics Distrubuted Platform.* [Computer Software]. 2016

11. Letavay Viliam, Pluskal Jan, Ryšavý Ondřej, and Matoušek Petr. *Reconstruction of Captured Communication on iOS Platform.* [Computer Software]. 2016

12. Janeček Vít, Pluskal Jan, Ryšavý Ondřej, and Matoušek Petr. *Web Traffic Data Export to MAFF.* [Computer Software]. 2016

13. Pluskal Jan, Kmeť Martin, Karpíšek Filip, Ryšavý Ondřej, Veselý Vladimír, and Matoušek Petr. *Netfox Detective - a network forensics tool for analyzing network traffic.* [Computer Software]. 2015

14. Pluskal Jan, Veselý Vladimír, Ryšavý Ondřej, and Matoušek Petr. *Netfox.Framework - Network traffic decoder and content analyzer.* [Computer Software]. 2013

3.6 Invited Speeches, Presentations and Posters

1. Jan Pluskal. *Workshop on Correlating Blockchain Activity with Real-Life Events and Users.* [Invited speech]. ISS World Asia, Dubai, United Arab Emirates, 2022

2. Jan Pluskal. *Intercepting and Collecting Web Evidence in the Times of TLS1.3 and HTTP 3.0.* [Invited speech]. ISS World Asia, Dubai, United Arab Emirates, 2022

3. Jan Pluskal. *Intercepting and Collecting Web Evidence in the Times of TLS1.3 and HTTP 3.0.* [Invited speech]. ISS World Europe, Prague, Czech Republic, 2021

4. Jan Pluskal and Veselý Vladimír. *Intercepting and Collecting Web Evidence in the Times of TLS1.3 and HTTP 3.0.* [Invited speech]. ISS World Asia, Dubai, United Arab Emirates, 2021

5. Jan Pluskal. *Cryptocurrency Investigation Workshop.* [Invited speech]. ISS World Asia, Dubai, United Arab Emirates, 2020

6. Jan Pluskal. *Towards Fully Automated Infinitely Scalable and Maximally Effective Password Cracking of Encrypted Documents.* [Invited speech]. ISS World Asia, Dubai, United Arab Emirates, 2020

7. Jan Pluskal. *ISS MEA 2020 - SSL/TLS Interception Workshop (TLS 1.3 Edition).* [Invited speech]. ISS World MEA, Dubai, United Arab Emirates, 2020

8. Jan Pluskal and Veselý Vladimír. *TLS/SSL Decryption Workshop.* [Invited speech]. ISS World Asia, Kuala Lumpur, Malaysia, 2019

9. Jan Pluskal and Veselý Vladimír. *TLS/SSL Decryption Workshop.* [Invited speech]. ISS World MEA, Dubai, United Arab Emirates, 2019

10. Jan Pluskal and Veselý Vladimír. *TLS/SSL Decryption Workshop.* [Invited speech]. ISS World Europe, Prague, Czech Republic, 2019

11. Jan Pluskal and Veselý Vladimír. *TLS/SSL Decryption Workshop.* [Invited speech]. ISS World Asia, Kuala Lumpur, Malaysia, 2018

12. Jan Pluskal and Veselý Vladimír. *TLS/SSL Decryption Workshop.* [Invited speech]. ISS World Europe, Prague, Czech Republic, 2018

13. Jan Pluskal, Ondřej Ryšavý, and Matoušek Petr. *Detection, and Analysis of SIP Fraud Attack on 100Gb Ethernet with NEMEA System.* [Invited speech]. Cybersecurity and Privacy, Pristina, Kosovo, 2017

14. Jan Pluskal. *Detection, and Analysis of SIP Fraud Attack on 100Gb Ethernet with NEMEA System.* [Presentation]. IRTF NMGR Workshop, Berlin, 2016

15. Jan Pluskal, Ondřej Ryšavý, and Petr Matoušek. "On the Identification of Applications from Captured Network Traffic". In: *8th International Conference on Digital Forensics & Cyber Crime.* [Poster]. New York, 2016. URL: `https://prezi.com/wnxlghgkocti`

16. Jan Pluskal and Ondřej Ryšavý. *Network Forensic Tool Netfox Detective.* [Invited speech]. Cybersecurity and Privacy, Pristina, Kosovo, 2016

17. Jan Pluskal, Vladimír Veselý, Matěj Grégr, and Ondřej Ryšavý. *TLS/SSL Decryption Workshop.* [Invited speech]. ISS World Europe, Prague, Czech Republic, 2016

18. Jan Pluskal and Ondřej Ryšavý. *Concepts of Intercepted Communication Processing with Netfox Detective.* [Invited speech]. ISS World Europe, Prague, Czech Republic, 2015

3.7 Selected Relevant Supervised Theses

1. Šimon Pokorný. "Migrace a refaktorizace Netfox Detective na .NET 5". Czech. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2021. URL: `https://www.fit.vut.cz/study/thesis/22857/`

2. Richard Stehlík. "Útok na WiFi síť s využitím ESP32/8266". Czech. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2021. URL: `https://www.fit.vut.cz/study/thesis/23435/`

3. Martina Zembjaková. "Network Forensics Tools Survey and Taxonomy". Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2021. URL: `https://www.fit.vut.cz/study/thesis/23022/`

4. Tomáš Čikel. "Bezpečnostní analýza domácí IoT sítě". Slovak. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2020. URL: `https://www.fit.vut.cz/study/thesis/23135/`

5. Juraj Kubiš. "SS7 Honeypoty - proaktivní ochrana proti podvodům v mobilních sítích". Czech. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2020. URL: `https://www.fit.vut.cz/study/thesis/23130/`

6. Jozef Zuzelka. "Control of External Devices on macOS to Prevent Data Leaks". Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2020. URL: `https://www.fit.vut.cz/study/thesis/22637/`

7. Tomáš Ambrož. "Analytické webové prostředí pro zpracování síťové komunikace". Czech. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: https://www.fit.vut.cz/study/thesis/22049/

8. Daniel Dušek. "Web Application Penetration Testing Automation". Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: https://www.fit.vut.cz/study/thesis/21678/

9. Lukáš Petrovič. "Analýza anomálií v uživatelském chování". Czech. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: https://www.fit.vut.cz/study/thesis/21474/

10. Martin Vondráček. "Security Analysis of Immersive Virtual Reality and Its Implications". Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: https://www.fit.vut.cz/study/thesis/22158/

11. Šimon Podlesný. "Automatizace MITM útoků za použití jednodeskového počítače". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: https://www.fit.vut.cz/study/thesis/22142/

12. Viliam Letavay. "Zpracování síťové komunikace v distribuovaném prostředí". Czech. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2018. URL: https://www.fit.vut.cz/study/thesis/20432/

13. Hana Slámová. "Refaktorizace síťového forenzního nástroje Netfox Detective". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2018. URL: https://www.fit.vut.cz/study/thesis/20380/

14. Tomáš Chomo. "Identifikace aplikačních protokolů". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2017. URL: https://www.fit.vut.cz/study/thesis/20191/

15. Jozef Zuzelka. "Nástroj pro zachycení síťového provozu s aplikačním tagem". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2017. URL: https://www.fit.vut.cz/study/thesis/20013/

16. Jindřich Dudek. "Rekonstrukce zachycené komunikace ze sociálních sítí". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2016. URL: https://www.fit.vut.cz/study/thesis/18433/

17. Matěj Hvězda. "Distribuované zpracování zachycené síťové komunikace". Czech. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2016. URL: https://www.fit.vut.cz/study/thesis/18434/

18. Viliam Letavay. "Rekonstrukce zachycené komunikace na platformě iOS". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2016. URL: https://www.fit.vut.cz/study/thesis/18557/

19. Marek Marušic. "Automatizace MitM útoku pro dešifrování SSL/TLS". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2016. URL: https://www.fit.vut.cz/study/thesis/18593/

20. Martin Vondráček. "Automation of MitM Attack on WiFi Networks". Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2016. URL: `https://www.fit.vut.cz/study/thesis/18596/`

21. Tomáš Bruckner. "Rekonstrukce zachycené komunikace a dolování dat na sociální síti Facebook". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2015. URL: `https://www.fit.vut.cz/study/thesis/17307/`

22. Zbyněk Lazárek. "Automatizovaná detekce transportního protokolu v zachycené síťové komunikaci". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2015. URL: `https://www.fit.vut.cz/study/thesis/17384/`

3.8 Other Supervised Theses

1. Petr Urbánek. "Vytvoření brány pro chytrá zařízení Xiaomi Aqara". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2021. URL: `https://www.fit.vut.cz/study/thesis/23856/`

2. Dávid Špavor. "Generování Blazor komponent z C# tříd". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2021. URL: `https://www.fit.vut.cz/study/thesis/23588/`

3. Michal Orlíček. "Systém na správu programovacích konvencí v projektu". Czech. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2021. URL: `https://www.fit.vut.cz/study/thesis/23487/`

4. Jan Lorenc. "Detekce anomálií na základě stavu RQA systému". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2021. URL: `https://www.fit.vut.cz/study/thesis/23935/`

5. Tomáš Kolarčík. "Detekce přítomnosti osob pomocí IoT senzorů". Czech. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2021. URL: `https://www.fit.vut.cz/study/thesis/21429/`

6. Petr Drábek. "Multiplatformní mobilní včelařský deník". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2020. URL: `https://www.fit.vut.cz/study/thesis/22789/`

7. Milan Múčka. "Nízkoenergetický GPS lokátor s použitím LoRa sítě". Czech. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2020. URL: `https://www.fit.vut.cz/study/thesis/22819/`

8. Daniel Šerý. "Optimalizace LINQ pro .NET". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2020. URL: `https://www.fit.vut.cz/study/thesis/23085/`

9. Michal Tichý. "Překlad DotVVM stránek do Accelerated Mobile Pages". Czech. Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2020. URL: `https://www.fit.vut.cz/study/thesis/22954/`

10. Tomáš Žigo. “Rest API pro dotazy nad sítí Bitcoin”. Slovak. Bachelor’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2020. URL: `https://www.fit.vut.cz/study/thesis/22670/`

11. Radim Blaha. “Jak udělat chytrou domácnost pomocí open-source IoT?”. Czech. Bachelor’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: `https://www.fit.vut.cz/study/thesis/21809/`

12. Gabriela Chmelařová. “Návrh úsporných IoT senzorů a sítě chytré domácnosti”. Czech. Bachelor’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: `https://www.fit.vut.cz/study/thesis/21338/`

13. Filip Kalous. “Nástroj pro usnadnění testování GUI webových aplikací”. Czech. Master’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: `https://www.fit.vut.cz/study/thesis/21918/`

14. Lukáš Kúšik. “Open-source komponenty pro inteligentní dům”. Czech. Bachelor’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: `https://www.fit.vut.cz/study/thesis/21784/`

15. Michal Motyčka. “Porovnání softwarových architektur”. Czech. Bachelor’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: `https://www.fit.vut.cz/study/thesis/21783/`

16. Michal Orlíček. “Návrh technologického IT kurzu pro interní vzdělávání”. Czech. Bachelor’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: `https://www.fit.vut.cz/study/thesis/21582/`

17. David Průdek. “Detekce přítomnosti osob v místnosti”. Czech. Bachelor’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: `https://www.fit.vut.cz/study/thesis/21585/`

18. Martin Tomovič. “Návrh chytré domácnosti za pomoci open-source IoT”. Czech. Master’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: `https://www.fit.vut.cz/study/thesis/21337/`

19. Jakub Víšek. “Automatizace výdeje a účtování kávy”. Czech. Bachelor’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2019. URL: `https://www.fit.vut.cz/study/thesis/21472/`

20. Michal Mrnuštík. “Mezijazykový překladač C#-JavaScript pro DotVVM”. Czech. Bachelor’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2018. URL: `https://www.fit.vut.cz/study/thesis/20958/`

21. Tomáš Chovanec. “Rezervace vstupenek pomocí botů na chatovacích platformách”. Czech. Bachelor’s thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2017. URL: `https://www.fit.vut.cz/study/thesis/19343/`

Chapter 4

Conclusions

This chapter summarizes the research presented in this dissertation. The approach followed is outlined, and the results obtained are discussed. Future research directions are proposed based on the experience gained.

4.1 The Research Approach

This dissertation does not consist of basic research as is customary but describes the advances in my applied research.

My research began around 2012, with my bachelor thesis focused on the reconstruction of YMSG and OSCAR instant communication protocols. I have realized that creating single-purpose tools may bring unnecessary overhead considering its maintainability and extensibility to cover additional network protocols. In my master thesis, I developed a framework for reconstructing captured network communication that required abstracted data preprocessing steps and provided a unified interface for application protocol parsers to improve this state. To my shame, I realized that I had not conducted rigorous state-of-the-art research to compare the capabilities of existing network forensic tools, identify their weaknesses, choose the research area, and improve the state-of-the-art.

At the beginning of my doctoral studies, in 2014, I started to experiment with the most advanced open source network forensic and analysis tools and network security monitoring tools (according to Pilli and Joshi [53]; see Table 2.3) at that time, namely *Wireshark, Network Monitor, Xplico, Network Miner, and PyFlag.* With these experiments, I gained an understanding of the usability of these tools and also their capabilities. Using these experiments combined with the experience gained in my previous work, I created a list containing the four primary research objectives (see Section 1.3) I wanted to improve.

During the literature review, I realized that not a negligible number of research papers do not allow for reproducibility of their results by lacking either a description or better concrete implementation of the methods they describe. This observation has convinced me to explain my experiments, input data, and results precisely and to attach a concrete implementation with the datasets I used. Furthermore, my long-term goal was to create a tool to help LEA investigators in their daily work. I have used this opportunity to utilize this tool as a base framework for my research experiments and have extended it to most of my research results.

4.2 Contributions

This section summarizes my research contributions, while a complete and detailed description is provided along with the attached papers in Section 3.2. The most significant contribution of this work was the identification of research objectives, see Section 1.3, which could enrich the state-of-the-art.

Contributions to the **capturing and processing of in/complete network data** consist of identifying suboptimal mechanisms used in several NFAT and NSM tools [43, 54]. This may result in *inaccurate L4 and application conversation tracking*, which yields fewer conversations than actually occurred. This applies only in the case where the captured communication was incomplete. Consequently, these tools will extract artifacts from the contents of the application protocol and assign them to these inaccurately tracked conversations. Each L4 and application protocol conversation is assigned to an entity / identity responsible for the communication. Inaccurate conversation tracking merges multiple conversations into one, which may assign artifacts from the *missed* conversation to the previously identified one, resulting in the creation of *false evidence.* Furthermore, we have determined that some NFAT and NSM tools [43, 54] are not fully implementing TCP reassembling and cannot extract the content of an application message if the *TCP sequence number overflows.* Furthermore, concerning reassembling incomplete TCP conversations, some tools [43, 54] stop the artifact extraction process after the first missing data occurs. This approach may omit crucial evidence of activity that occurred in the communication after the first missing data. In the publications mentioned above, we proposed a method to remedy this situation using heuristics based on information from transport protocols. As a preliminary step, we have considered the possibility of capturing local network traffic using MitM proxies that are intrusively deployed on Wi-Fi networks [73, 74].

Contributions to **application protocol and finer-grained application identification** are described in detail in the enclosed papers [56, 54]. The major contribution of this research is to show that not only can application protocols be identified using ML algorithms, but we can also identify, with a lower probability, applications that were used to generate that communication. Secondary contributions are open source publicly available datasets for research verification and open source implementations of multiple classification algorithms that may serve as a playground for further research related to feature engineering and hyperparameter tuning. Additionally, we revisited commonly used features for application protocol classification and proposed adding new features based on information gained by reassembling application messages. This approach may eliminate certain noise introduced by IP fragmentation and TCP segmentation. Our additional contribution to feature engineering for traditional ML algorithms was the introduction of automated feature elimination based on feature correlation computed from our annotated dataset. The last contribution was the proposition of a novel statistical-based method that inherently contained feature elimination and did not require this additional pre-training step.

Contributions to what should be the **architecture of network forensic tool / scale or not to scale** research questions were addressed in the publications [39, 38]. In this research, we were looking for possibilities to increase the throughput of capture traffic network processing using horizontal scalability. Inspired by Valentin's [70] usage of the actor model, we have designed and implemented a framework capable of linear scalability while respecting advanced processing features for heuristical handling of incomplete data described in other enclosed publications [43, 56]. The overall contribution is a practical demonstration supported by rigorous measurements that show the feasibility of horizontal scalability for increasing the performance of NFATs. The secondary contribution is the creation of a PoC specimen [72] composed of low-cost / low-power computers on a single board.

Contributions to the processing of **tunneled and overlay networks** in network forensic analysis lie in identifying the need to address the underlying network encapsulation [55, 58, 54] correctly. Omitting, for example, VLAN tags may mix up unrelated flows, similarly to incorrect TCP reassembling of incomplete communication. Our contribution to this topic is the analysis of *Generic Stream Encapsulation (GSE)* and the creation of its PoC processing unit incorporated into our NFAT Netfox Detective tool, while being the only NFAT tool that supports it.

This work laid the theoretical ground for a research project sponsored by the Czech Ministry of Interior (VH20192021043).

4.3 Future Work

Considering the experience gained in the field of Network Forensic Analysis, I would like to outline future research directions that seem promising:

- Investigation of possible data sources for forensic investigation. A lawful interception at the Internet Service Provider level is de facto standard, but additional points in the network infrastructure may also be beneficial. Richard Stehlik's master thesis [68] introduces one of the promising directions.
- A deeper analysis of application communication patterns and metadata extraction is the key to fighting omnipresent encryption. Identifying not only an application protocol but also an application and type of communication such as text, voice message, interactive call, etc., may allow the use of standardized analytical approaches for Call Detail Records (CDRs) from the telecommunication world in the network forensic investigation.
- A correlation of patterns observed from network traffic captured on multiple points in the network to prove that entities were in contact, e.g., VoIP call routed through a third-party proxy may have the same characteristics on both sides of a broker; thus, the correlation may identify calling parties.

4.4 Final Notes

The presented dissertation outlined research conducted in the Networks and Distributed Systems Research Group (NES@FIT), at the Faculty of Information Technology, under the Brno University of Technology in the field of Network Forensic Analysis, which I have participated in in the last decade. The goal of this research is consistent with the needs of the Czech Law Enforcement Agencies that supported the selected research objectives presented in this work. The results of this research have been given to end users along the lines of LEA investigators. To the best of my knowledge, they are being used or considered for practical applications.

Bibliography

[1] John Althouse, Jeff Atkinson, and Josh Atkins. *JA3 - A method for profiling SSL/TLS Clients*. `https://github.com/salesforce/ja3/`. 2017.

[2] Blake Anderson and David McGrew. “TLS Beyond the Browser: Combining End Host and Network Data to Understand Application Behavior”. In: IMC ’19. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 379–392. ISBN: 9781450369480. DOI: `10.1145/3355369.3355601`. URL: `https://doi.org/10.1145/3355369.3355601`.

[3] E Anderson and M Arlitt. “Full packet capture and offline analysis on 1 and 10 gb networks”. In: (2006).

[4] Sabeel Ansari, SG Rajeev, and HS Chandrashekar. “Packet sniffing: a brief introduction”. In: *IEEE potentials* 21.5 (2003), pp. 17–19.

[5] Pallavi Asrodia and Vishal Sharma. “Network monitoring and analysis by packet sniffing method”. In: *International Journal of Engineering Trends and Technology (IJETT)* 4.5 (2013), pp. 2133–2135.

[6] Tom Auld, Andrew W Moore, and Stephen F Gull. “Bayesian neural networks for internet traffic classification”. In: *IEEE Transactions on neural networks* 18.1 (2007), pp. 223–239.

[7] Nicole Beebe. “Digital forensic research: The good, the bad and the unaddressed”. In: *IFIP International Conference on Digital Forensics*. Springer. 2009, pp. 17–36.

[8] Joshua Broadway, Benjamin Turnbull, and Jill Slay. “Improving the Analysis of Lawfully Intercepted Network Packet Data Captured for Forensic Analysis”. In: *2008 Third International Conference on Availability, Reliability and Security*. 2008, pp. 1361–1368. DOI: `10.1109/ARES.2008.122`.

[9] Elie Bursztein. “Probabilistic identification for hard to classify protocol”. In: *IFIP International Workshop on Information Security Theory and Practices*. Springer. 2008, pp. 49–63.

[10] Eoghan Casey. “Network traffic as a source of evidence: tool strengths, weaknesses, and future needs”. In: *Digital Investigation* 1.1 (2004), pp. 28–43. ISSN: 1742-2876.

[11] Clement Chau. “YouTube as a participatory culture”. In: *New directions for youth development* 2010.128 (2010), pp. 65–74.

[12] Vicka Corey, Charles Peterman, Sybil Shearin, Michael S Greenberg, and James Van Bokkelen. “Network forensics analysis”. In: *IEEE Internet Computing* 6.6 (2002), pp. 60–66.

[13] Valerio D'Alessandro, Byungchul Park, Luigi Romano, Christof Fetzer, et al. "Scalable network traffic classification using distributed support vector machines". In: *2015 IEEE 8th International Conference on Cloud Computing*. IEEE. 2015, pp. 1008–1012.

[14] Shuaifu Dai, Alok Tongaonkar, Xiaoyin Wang, Antonio Nucci, and Dawn Song. "NetworkProfiler: Towards automatic fingerprinting of Android apps". In: *Proceedings - IEEE INFOCOM* (2013), pp. 809–817. ISSN: 0743166X. DOI: `10.1109/INFCOM.2013.6566868`.

[15] Sherri Davidoff and Jonathan Ham. *Network forensics: tracking hackers through cyberspace*. Vol. 2014. Prentice hall Upper Saddle River, 2012.

[16] Luca Deri et al. "Improving passive packet capture: Beyond device polling". In: *Proceedings of SANE*. Vol. 2004. Amsterdam, Netherlands. 2004, pp. 85–93.

[17] ENISA. *Introduction to Network Forensics*. Tech. rep. European Union Agency for Cybersecurity (ENISA), 2019. DOI: `10.2824/995110`.

[18] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. "Offline/realtime traffic classification using semi-supervised learning". In: *Performance Evaluation* 64.9 (2007), pp. 1194–1213.

[19] Alessandro Finamore, Marco Mellia, and Michela Meo. "Mining unclassified traffic using automatic clustering techniques". In: *International Workshop on Traffic Monitoring and Analysis*. Springer. 2011, pp. 150–163.

[20] Vahid Aghaei Foroushani and A Nur Zincir-Heywood. "Investigating application behavior in network traffic traces". In: *Computational Intelligence for Security and Defense Applications (CISDA), 2013 IEEE Symposium on*. IEEE. 2013, pp. 72–79.

[21] Yanjie Fu, Hui Xiong, Xinjiang Lu, Jin Yang, and Can Chen. "Service usage classification with encrypted internet traffic in mobile messaging apps". In: *IEEE Transactions on Mobile Computing* 15.11 (2016), pp. 2851–2864.

[22] Simson Garfinkel. "Digital forensics research: The next 10 years". In: *Digital Investigation* 7 (2010), S64–S73.

[23] Simson Garfinkel. "Network Forensics: Tapping the Internet". In: *O'Reilly Network* (2002).

[24] Gabriel Gómez Sena and Pablo Belzarena. "Early traffic classification using support vector machines". In: *Proceedings of the 5th International Latin American Networking Conference*. ACM. 2009, pp. 60–66.

[25] Vikram S Harichandran, Frank Breitinger, Ibrahim Baggili, and Andrew Marrington. "A cyber forensics needs analysis survey: Revisiting the domain's needs a decade later". In: *Computers & Security* 57 (2016), pp. 1–13.

[26] Lukáš Hejcman. "Fingerprinting and Identification of TLS Connections". Bachelor's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2021. URL: `https://www.fit.vut.cz/study/thesis/23922/`.

[27] Erik Hjelmvik. "The SPID algorithm-statistical protocol identification". In: *Gävle, Sweden, October* (2008).

[28] Erik Hjelmvik and Wolfgang John. "The SPID Algorithm". In: *Swedish National Computer Networking Workshop*. 2009, p. 21.

[29] Kamil Jeřábek and Ondřej Ryšavý. "Big data network flow processing using Apache Spark". In: *Proceedings of the 6th conference on the engineering of computer based systems*. 2019, pp. 1–9.

[30] R. C. Joshi and Emmanuel S. Pilli. "Network Forensic Tools". In: *Fundamentals of Network Forensics*. Ed. by A. J. Sammes. Springer, 2016. Chap. 4, pp. 71–93. ISBN: 978-1-4471-7297-0. DOI: 10.1007/978-1-4471-7299-4{_}4.

[31] R.C. Joshi and Emmanuel S. Pilli. *Fundamentals of Network Forensics*. Springer, 2016.

[32] Lukáš Kekely, Jan Kučera, Viktor Puš, Jan Kořenek, and Athanasios V. Vasilakos. "Software Defined Monitoring of Application Protocols". In: *IEEE Transactions on Computers* 65.2 (2016), pp. 615–626. DOI: 10.1109/TC.2015.2423668.

[33] Jawad Khalife, Amjad Hajjar, and Jesus Diaz-Verdejo. "A multilevel taxonomy and requirements for an optimal traffic-classification model". In: *International Journal of Network Management* 24.2 (2014), pp. 101–120.

[34] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz, and Iftikhar Ahmad. "Network forensics: Review, taxonomy, and open challenges". In: *Journal of Network and Computer Applications* 66 (2016), pp. 214–235. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2016.03.005. URL: https://www.sciencedirect.com/science/article/pii/S1084804516300121.

[35] Christopher Köhnen, Christian Überall, Florian Adamsky, Veselin Rakocevic, Muttukrishnan Rajarajan, and Rudolf Jäger. "Enhancements to Statistical Protocol IDentification (SPID) for Self-Organised QoS in LANs." In: *ICCCN*. 2010, pp. 1–6.

[36] Christopher Köhnen, Christian Überall, Florian Adamsky, Veselin Rakočević, Muttukrishnan Rajarajan, and Rudolf Jäger. "Enhancements to Statistical Protocol IDentification (SPID) for self-organised QoS in LANs". In: *Proceedings - International Conference on Computer Communications and Networks, ICCCN*. 2010.

[37] Stefan Kornexl, Vern Paxson, Holger Dreger, Anja Feldmann, and Robin Sommer. "Building a time machine for efficient recording and retrieval of high-volume network traffic". In: *5th Internet Measurement Conference*. USENIX Association. 2005, pp. 267–272.

[38] Viliam Letavay, Jan Pluskal, and Ondřej Ryšavý. "A Scalable Architecture for Network Traffic Forensics". In: *The Fifteenth International Conference on Networking and Services ICNS 2019*. Athens, GR: The International Academy, Research and Industry Association, 2019, pp. 32–36. ISBN: 9781612087115.

[39] Viliam Letavay, Jan Pluskal, and Ondřej Ryšavý. "Network Forensic Analysis for Lawful Enforcement on Steroids, Distributed and Scalable". In: *Proceedings of the 6th Conference on the Engineering of Computer Based Systems*. ACM. 2019. ISBN: 9781450376365.

[40] Akhyar Lubis and Andysah Putera Utama Siahaan. "NetworkForensic Application in General Cases". In: *IOSR Journal of Computer Engineering (IOSR-JCE)* 18.6 (2016), pp. 41–44. DOI: 10.9790/0661-1806044144.

[41] Yan Luo, Ke Xiang, and Sanping Li. "Acceleration of decision tree searching for IP traffic classification". In: *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM. 2008, pp. 40–49.

[42] Marie-Helen Maras. "Network Forensics: An Introduction". In: *Computer Forensics: Cybercriminals, Laws, and Evidence*. Second. Jones & Bartlett Learning, 2015. Chap. 12. ISBN: 978-1-4496-9222-3.

[43] Petr Matoušek, Jan Pluskal, Ondřej Ryšavý, Vladimír Veselý, Martin Kmeť, Filip Karpíšek, and Martin Vymlátil. "Advanced Techniques for Reconstruction of Incomplete Network Data". In: *Digital Forensics and Cyber Crime*. Ed. by Joshua I. James and Frank Breitinger. Cham: Springer International Publishing, 2015, pp. 69–84. ISBN: 9783319255125.

[44] David McGrew, Brandon Enright, and Blake Anderson. *Mercury: network metadata capture and analysis*. `https://github.com/cisco/mercury`. 2019.

[45] Natarajan Meghanathan, Sumanth Reddy Allam, and Loretta A Moore. "TOOLS AND TECHNIQUES FOR NETWORK FORENSICS". In: *International Journal of Network Security & Its Applications (IJNSA)* 1.1 (Apr. 2009), pp. 14–25.

[46] Stanislav Miskovic, Gene Moo Lee, Yong Liao, and Mario Baldi. "AppPrint: Automatic fingerprinting of mobile applications in network traffic". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8995. Springer Verlag, 2015, pp. 57–69.

[47] Andrew W Moore and Konstantina Papagiannaki. "Toward the accurate identification of network applications". In: *Passive and Active Network Measurement* 3431 (2005). Ed. by ConstantinosEditor Dovrolis, pp. 41–54. URL: `http://www.springerlink.com/index/re7ej0uj7eep2htl.pdf`.

[48] Neeraj Namdev, Shikha Agrawal, and Sanjay Silkari. "Recent advancement in machine learning based internet traffic classification". In: *Procedia Computer Science* 60 (2015), pp. 784–791.

[49] NetQuest. *I-9100 100/40/10G Interceptor Network Monitoring Access solutions for today's intelligent packet optical transport networks*. Jan. 2021. URL: `https://netquestcorp.com/wp-content/uploads/2021/01/NQ-I-9100-Interceptor-Datasheet-2.pdf` (visited on 05/08/2022).

[50] NetQuest. *ISS World Training - Intelligent Support Systems for Lawful Interception, Electronic Surveillance and Cyber Intelligence Gathering*. May 2022. URL: `https://www.issworldtraining.com/` (visited on 05/08/2022).

[51] Thuy TT Nguyen and Grenville Armitage. "A survey of techniques for internet traffic classification using machine learning". In: *IEEE Communications Surveys & Tutorials* 10.4 (2008), pp. 56–76.

[52] Gary Palmer. "A Framework for Digital Forensic Scrience Part of A Road Map for Digital Forensic Research". In: *Digital Forensic Research Conference (DFRWS)*. 2001, pp. 15–20.

[53] Emmanuel S. Pilli, Ramesh C. Joshi, and Rajdeep Niyogi. "Network forensic frameworks: Survey and research challenges". In: *Digital Investigation* 7.1 (2010), pp. 14–27.

[54] Jan Pluskal, Frank Breitinger, and Ondřej Ryšavý. "Netfox detective: A novel open-source network forensics analysis tool". In: *Forensic Science International: Digital Investigation* 35 (2020), p. 301019. ISSN: 2666-2817.

[55] Jan Pluskal, Michal Koutenský, Martin Vondráček, and Ondřej Ryšavý. "Network Forensic Investigations of Tunneled Traffic: A Case Study". In: *Revue roumaine des sciences techniques. Série Électrotechnique et Énergétique* 64.4 (2019), pp. 429–434. ISSN: 0035-4066.

[56] Jan Pluskal, Ondrej Lichtner, and Ondřej Ryšavý. "Traffic Classification and Application Identification in Network Forensics". In: *Fourteenth Annual IFIP WG 11.9 International Conference on Digital Forensics.* Ed. by Gilbert Peterson and Sujeet Shenoi. New Delhi, IN: Springer International Publishing, 2018, pp. 161–181. ISBN: 9783319992778.

[57] Jan Pluskal, Petr Matoušek, Ondřej Ryšavý, Martin Kmeť, Vladimir Veselý, Filip Karpíšek, and Martin Vymlátil. "Netfox Detective: A tool for advanced network forensics analysis". In: *Proceedings of Security and Protection of Information (SPI) 2015.* Brno, CZ: Brno University of Defence, 2015, pp. 147–163. ISBN: 9788072319978.

[58] Jan Pluskal, Martin Vondráček, and Ondřej Ryšavý. "Network Forensics in GSE Overlay Networks". In: *Proceedings of the 6th Conference on the Engineering of Computer Based Systems.* ACM. 2019. ISBN: 9781450376365.

[59] Zdenek Rosa, Tomáš Čejka, Martin Žádník, and Viktor Pus. "Building a feedback loop to capture evidence of network incidents". In: *2016 12th International Conference on Network and Service Management (CNSM).* 2016, pp. 292–296. DOI: `10.1109/CNSM.2016.7818435`.

[60] Minoo Rouhi, Quirin Scheitle, Oliver Gasser, Christian Wahl, Marcin Nawrocki, Matthias Wählisch, Raphael Hiesgen, and Thomas C Schmidt. "Incident Forensics in Distributed High-Speed Networks". In: ().

[61] Javier Rubio-Loyola, Dolors Sala, and Ali Ismail Ali. "Maximizing packet loss monitoring accuracy for reliable trace collections". In: *2008 16th IEEE workshop on local and metropolitan area networks.* IEEE. 2008, pp. 61–66.

[62] Marek Rychl and Ondrej Rysav. "Big data security analysis with tarzan platform". In: *Journal of Cyber Security and Mobility* (2019), pp. 165–188.

[63] Marek Rychlý and Ondřej Ryšavý. "TARZAN: An integrated platform for security analysis". In: *2017 Federated Conference on Computer Science and Information Systems (FedCSIS).* IEEE. 2017, pp. 561–567.

[64] Ola Salman, Imad H Elhajj, Ayman Kayssi, and Ali Chehab. "A review on machine learning–based approaches for Internet traffic classification". In: *Annals of Telecommunications* 75.11 (2020), pp. 673–710.

[65] Filipo Sharevski. *Mobile Network Forensics: Emerging Research and Opportunities: Emerging Research and Opportunities.* IGI Global, 2018.

[66] Leslie F. Sikos. "Packet analysis for network forensics: A comprehensive survey". In: *Forensic Science International: Digital Investigation* 32 (2020), p. 200892. ISSN: 2666-2817. DOI: `https://doi.org/10.1016/j.fsidi.2019.200892`. URL: `https://www.sciencedirect.com/science/article/pii/S1742287619302002`.

[67] Daniel Spiekermann and Tobias Eggendorfer. "Towards Digital Investigation in Virtual Networks: A Study of Challenges and Open Problems". In: *2016 11th International Conference on Availability, Reliability and Security (ARES).* 2016, pp. 406–413. DOI: `10.1109/ARES.2016.34`.

[68] Richard Stehlík. "Útok na WiFi síť s využitím ESP32/8266". Czech. Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2021. URL: `https://www.fit.vut.cz/study/thesis/23435/`.

[69] Matthias Vallentin. "Scalable Network Forensics". PhD thesis. UC Berkeley, 2016.

[70] Matthias Vallentin, Vern Paxson, and Robin Sommer. "{VAST}: A Unified Platform for Interactive Network Forensics". In: *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 2016, pp. 345–362.

[71] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. "A survey of methods for encrypted traffic classification and analysis". In: *International Journal of Network Management* 25.5 (2015), pp. 355–374.

[72] Letavay Viliam, Pluskal Jan, and Jeřábek Kamil. *Banana Pi BPI-R2 Cluster Prototype.* [Specimen]. 2018.

[73] Martin Vondráček, Jan Pluskal, and Ondřej Ryšavý. "Automated Man-in-the-Middle Attack Against Wi-Fi Networks". In: *The Journal of Digital Forensics, Security and Law: JDFSL* 13.1 (2018), pp. 59–80. ISSN: 1558-7215.

[74] Martin Vondráček, Jan Pluskal, and Ondřej Ryšavý. "Automation of MitM Attack on Wi-Fi Networks". In: *9th International Conference on Digital Forensics & Cyber Crime.* Vol. 2018. 216. Springer International Publishing, 2017, pp. 207–220. ISBN: 9783319736969.

[75] Pan Wang, Xuejiao Chen, Feng Ye, and Zhixin Sun. "A Survey of Techniques for Mobile Service Encrypted Traffic Classification Using Deep Learning". In: *IEEE Access* 7 (2019), pp. 54024–54033. DOI: `10.1109/ACCESS.2019.2912896`.

[76] YiPeng Wang, Xiaochun Yun, Yongzheng Zhang, Liwei Chen, and Tianning Zang. "Rethinking robust and accurate application protocol identification". In: *Computer Networks* 129 (2017), pp. 64–78. ISSN: 1389-1286. DOI: `https://doi.org/10.1016/j.comnet.2017.09.006`. URL: `https://www.sciencedirect.com/science/article/pii/S1389128617303572`.

[77] Martina Zembjaková. "Network Forensics Tools Survey and Taxonomy". Master's thesis. Brno, CZ: Brno University of Technology, Faculty of Information Technology, 2021. URL: `https://www.fit.vut.cz/study/thesis/23022/`.

[78] Jian Zhang and Andrew Moore. "Traffic trace artifacts due to monitoring via port mirroring". In: *2007 Workshop on End-to-End Monitoring Techniques and Services.* IEEE. 2007, pp. 1–8.

Appendix A

Included Papers

A.1 Netfox detective: A novel open-source network forensics analysis tool

Jan Pluskal, Frank Breitinger, and Ondřej Ryšavý. “Netfox detective: A novel open-source network forensics analysis tool”. In: *Forensic Science International: Digital Investigation* 35 (2020), p. 301019. ISSN: 2666-2817

Forensic Science International: Digital Investigation 35 (2020) 301019

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

Netfox detective: A novel open-source network forensics analysis tool

Jan Pluskal [a], [*], Frank Breitinger [b], Ondřej Ryšavý [a]

[a] *Brno University of Technology, Faculty of Information Technology, Božetěchova 2, Brno, Czech Republic*
[b] *Hilti Chair for Data and Application Security Institute of Information Systems, University of Liechtenstein, Fürst-Franz-Josef-Strasse, 9490, Vaduz, Liechtenstein*

ARTICLE INFO

Article history:
Received 21 January 2020
Received in revised form
16 June 2020
Accepted 18 June 2020
Available online xxx

Keywords:
Network forensics
Protocol analysis
Web forensics
Network forensic analysis tool
Lawful interception

ABSTRACT

Network forensics is a major sub-discipline of digital forensics which becomes more and more important in an age where everything is connected. In order to cope with the amounts of data and other challenges within networks, practitioners require powerful tools that support them. In this paper, we highlight a novel open-source network forensic tool named — Netfox Detective — that outperforms existing tools such as Wireshark or NetworkMiner in certain areas. For instance, it provides a heuristically based engine for traffic processing that can be easily extended. Using robust parsers (we are not solely relying on the RFC description but use heuristics), our application tolerates malformed or missing conversation segments. Besides outlining the tool's architecture and basic processing concepts, we also explain how it can be extended. Lastly, a comparison with other similar tools is presented as well as a real-world scenario is discussed.

1. Introduction

Network forensics aims to understand/reconstruct events from network communication, which often requires expert knowledge (interpreting the low-level network protocols in order to see the big picture) (Casey, 2004). To eliminate some of the complexity, adequate tools are essential (Garfinkel, 2010; Harichandran et al., 2016). Specifically, tools should support investigators by summarizing, clustering and highlighting relevant information (Beebe, 2009), e.g., provide contents of transmitted files, extract user credentials or perform analysis and visualize the data in an easily understandable form. While there are many different network forensic analysis tools (Pilli et al., 2010) out there (details discussed in the upcoming sections), their functionalities, capabilities, and usability are not keeping up with traditional forensics toolkits (Casey, 2004) such as EnCase, 2020 or The Sleuth Kit (TSK) & Autopsy, 2020.

Thematic classification: While network forensics and cloud forensics are related, the latter one is usually more complex, e.g., it may involve Software Defined Networking (SDN (McKeown et al., 2008)) which comes with additional evidence such as Logfiles from the SDN controller, compute nodes or cloud controller (Spiekermann et al., 2017). These networks also use state-of-the-art networking technology (100–400 Gbps) that cannot be monitored without hardware acceleration (typically FPGA), and even then, only selected flows can be fully captured (Kekely et al., 2016) and used for further, detailed examination. Netfox Detective is intended for network forensic analysis and visualization on a PC and does not compete with these tools, but uses them to filter and capture data.

Terms and definition: For readers not completely familiar with the network terminology, we included an overview in Appendix A.

1.1. Analysis of network communication

Two of the most popular tools for Network Security Monitoring (NSM) are Wireshark and TCPDUMP, 2020, which are commonly used by network administrators to identify problems or security incidents (Pilli et al., 2010). Wireshark provides a large number of protocol parsers, can extract the content of the communication for several application protocols and offers a detailed view of the network communication. While it is one of the most powerful tools, its bottom-up analysis approach means that finding and extracting evidence often requires (intensive) labor and expert domain knowledge. Nevertheless, Wireshark is continuously optimized,

[*] Corresponding author.
E-mail addresses: ipluskal@fit.vutbr.cz (J. Pluskal), Frank.Breitinger@uni.li (F. Breitinger), rysavy@fit.vutbr.cz (O. Ryšavý).

https://doi.org/10.1016/j.fsidi.2020.301019
2666-2817/© 2020 Elsevier Ltd. All rights reserved.

and usage of analyzers and LUA plugins eases up the investigation. Netfox Detective partially addresses this by implementing advanced features such as heuristical TCP reassembling or L7 conversation tracking or reconstruction of forensic artifacts extracted from the communication. Furthermore, Wireshark does not scale well above hundreds of megabytes of source data, and thus, data preprocessing is necessary for large inputs. TCPDUMP, 2020, on the other hand, has only a command line interface that allows admins to inspect incoming and outgoing network traffic.

There are also more specialized tools that can extract valuable forensic information, for instance, ngrep, 2020, ssldump, 2020, or tcpxtract, 2020. These tools were created to solve specific problems such as searching for a phrase in network communication, decoding encrypted communication if a private key is known, or extracting transferred files from network communication, respectively. To take advantage of all tools, an investigator is required to combine them. For repeating tasks, one may write scripts to speed up the process and thus, reduce the amount of manual labor.

Without question, there are many practitioners who prefer featureful open-source tools (Beebe, 2009; Farmer and Venema, 2009) although there is a risk that they are poorly documented, out-of-date, and even abandoned (Garfinkel, 2010).

1.2. Expected properties for network forensic tools

According to Cohen (2008), a network forensic analysis tool (NFAT) should provide a certain set of general features (listed as items 1—3 below). We further analyzed the demands and identified some more specific features yielding the following list of requirements:

1. *Efficient processing of large capture files:* Current investigations deal with a big amount of data that needs to be analyzed. Tools are required to provide at least partial results quickly.
2. *Extraction of high-level information:* Network communication can be analyzed at different levels but for digital investigation extracting artifacts from data sources is a priority.
3. *Validation of results:* Applying reliable procedures and the possibility to validate the integrity of results is a crucial requirement on all forensic tools including NFATs.
4. *Process non-standard or incomplete traffic*: Network communication should be correctly processed regardless of the acceptable deviations from the specification.
5. *Robust data decapsulation:* Even in the presence of IP fragmentation and data stream multiplexing, the tool should be able to identify and compose unique application level conversations.
6. *Support for overlay networks:* Network communication may be encapsulated using tunneling techniques, e.g., Virtual Private Networks. If possible, detection and extraction are then followed by the analysis of the encapsulated messages.
7. *Application protocol identification*: Services communicating on non-standard or dynamic ports require advanced methods for application identification. Without the correctly identified type of communicating application, it is difficult to extract any high-level information.
8. *Investigation process:* The tool should support the top-bottom investigative process and guide the user. It is essential that even non-expert personnel can operate NFAT and extract evidence to support their cases.

The presented list is not exhaustive and stems from our experience in network traffic analysis and evaluation of existing NFATs. Some requirements are conflicting, for instance, processing of large data sources and in-depth analysis of conversations to extract high-level artifacts.

1.3. Network forensic tools

Besides Network Security Monitoring (NSM) tools that are intended for packet capturing, fingerprinting, or intrusion detection, there are some network forensic analysis tools (NFAT) specifically designed to support investigators. These aim to ease analysis by automating artifacts extraction and providing intuitive user interfaces. Usually, these tools have a top-down approach which makes the analysis simpler and saves time. In the following we briefly summarize the five prominent tools (numbers in brackets related to Sec. 1.2 and show missing properties):

- NetIntercept was one of the first NFATs (Corey et al., 2002). It accepts PCAP files (no live captures), reassembles TCP flows and extracts artifacts from protocols running even on non-standard ports. Note: NetIntercept is closed source and to the best of our knowledge no longer available for download. Thus, we were unable to perform a more detailed evaluation.
- PyFlag, 2020 [1, 3, 4, 6, 7, 8] "is a general purpose, open source, forensic package which merges disk forensics, memory forensics, and network forensics" (Cohen, 2008). By using specialized scanners, PyFlag can understand several application protocols and extract the communicated contents. However, according to Forensics Wiki, the tool is deprecated.[1]
- XPlico, 2020 [1, 3, 4, 5] is open source NFAT that is preinstalled on major digital forensics distribution such as DEFT, Security Onion and even Kali. It understands about 30 application protocols and can extract the content of emails, Session Initiation Protocol (SIP) or web communication.
- NetworkMiner, 2020 [1, 3, 4, 8] is a passive network sniffer/packet capturing tool that can detect operating systems, sessions, hostnames, open ports, and more. It also allows extracting files from about a dozen commonly used application protocols. In the professional version, NetworkMinor also extracts VoIP calls, supports Geo IP localization, performs port-independent protocol identification, OS fingerprinting, and web browser tracing.
- TCPFlow, 2020 [2, 3, 4, 5, 6, 8] "captures data transmitted as part of TCP connections (flows), and stores the data in a way that is convenient for protocol analysis and debugging. Each TCP flow is stored in its own file. Thus, the typical TCP flow will be stored in two files, one for each direction. TCPFlow can also process stored 'tcpdump' packet flows". It is important to note that TCPflow does not recognize IP fragments; therefore, reassembling of such conversations will be malformed.

While these tools have different strengths, our tool provides some unique features which are pointed out in Sec. 5.

1.4. Problem description

Although many tools have been developed/exist, several tools are outdated, abandoned, or do not meet all expected properties (see Sec. 1.2). Additionally, current tools are not intuitive (require training), not (easily) expandable or can handle network traffic captures in the order of magnitude of gigabytes which were requirements/statements from the Lawful Enforcement Agency (LEA) officers. Last, existing tools are not structured along the investigative process; commonly there is no case management, the linkage between investigations, and verification of results which can be helpful during investigations.

[1] https://www.forensicswiki.org/wiki/PyFlag%20 (last accessed 2019-08-17).

Table 1
Performance of selected operations using the M57 case PCAP files. Machine configuration: CPU i7-4790, 4.00 GHz, 64 GB DDR4, Crucial MX100 SSD, Windows 10. Experiments were repeated 10-times, measured by *time* and *Perfmon* utilities.

	Operation	Backend$_{\mp}$	Frontend + Backend$_{\mp}$	Wireshark†	NetworkMiner†	tcpflow$^{\dagger,\Delta}$
1	Total time	6 m 14s, $\sigma \approx$ 15.23 *s*	9 m 36s, $\sigma \approx$ 30.12 *s*	8 m 48s, $\sigma \approx$ 17.34 *s*	41 m 23s, $\sigma \approx$ 124.43 *s*	13 m 39s, $\sigma \approx$ 64.21 *s*
2	Max RAM usage	8.3GB	8.5GB	7.1GB	20GB	243MB
3	Avg CPU usage	76%, $\sigma \approx 8$ %	66%, $\sigma \approx 18$ %	12%, $\sigma \approx 3$ %	15%, $\sigma \approx 2$ %	3%, $\sigma \approx 1$ %
4	Sessions (TCP + UDP)	118,709	118,709	98,084	49,865	93,619
5	TCP - missing	3.9%*	3.9%*	0.6%*	N/A	N/A
6	DNS - records	238,531	238,531	150,426	183,527	N/A
7	Emails	28	28	N/A	39	N/A
8	FTP	16	16	N/A	1	N/A
9	Complete Web pages	6	6	N/A	N/A	N/A
10	Speed	101.8 Mbps	66.1 Mbps	72.1 Mbps	15.3 Mbps	46.5 Mbps

(†) To measure comparable results, in-memory database has been used.
(†) The tool was downloaded as a binary release.
(Δ) The tcpflow 1.4.4 was ran with parameters *-r file.pcap -a -Fm* to do ALL post-processing and split output in 1 M directories.
(*) Netfox Detective computes TCP loss based on lost segment size (see Eq. (2)). For Wireshark, we computed it by applying the *tcp.analysis.lost_segment* filter and then utilized Eq. (1). This does not mean that the tool lost the data but they were not present in the capture, i.e., the capturing probe lost them.

1.5. Notes on legal requirements

Possible real-world usage of Netfox Detective, as well as other NFAT tools, needs to be under the frame of legal requirements and restrictions. Then conditions of the legal use of NFAT tools cannot be stated world-wide. EU countries and even states of a single country, e.g., the USA or Germany, have different laws about collecting digital traces related to user activities (see ENISA (2019), section 2.6). Network forensics necessary requires to gather IP addresses, packet captures, or log files that may contain all kinds of private data, including passwords, usernames, credit card numbers, etc. Specific laws regarding online services, protection of critical infrastructures, and cybercrime or computer crime may apply to the practice of digital investigation. Commonly they limit what data can be acquired or the way in which data can be processed. The presented tool is only technical equipment able to process captured communication. Same as in the case for other NFAT, the tool is able to extract various artifacts from network communication and it is required that investigators have to abide by the law, especially since matters may be taken to court. Often knowing what law applies to the situation may be challenging and the advice of trained legal experts is needed.

1.6. Contribution and paper structure

This paper provides Netfox Detective; a novel, easy-to-use, powerful network forensic platform for top-down investigations. Our tool covers examination, analysis, and investigation phases of the forensic model as defined by Pilli et al. (2010). In detail, we provide the following contributions:

1. Introduction of investigation profiles that contain all necessary data for sharing the case by just copying the investigation folder — Sec 3.3.
2. The new method of TCP stream reassembling based on heuristics (method itself was previously published (Matoušek et al., 2015), but the tool contains an improved version of it) — Sec 3.4 and Appendix E.
3. Improved identification of application-level sessions within TCP streams; the system can identify more application sessions compared to other tools (see Table 1) — Sec 3.4, Appendix E.
4. Seamless analysis across boundaries of multiple capture files that ensures correct processing of long-running conversations (i.e., overlapping conversations are processed correctly) — Sec 3.4. To the best of our knowledge, no NFAT or NSM tool currently has this functionality which is crucial for LEA forensic investigation. Data sources in form of PCAP files are typically split due to time or space constraints.
5. Support for analysis of traffic encapsulated in GSE protocol; to the best our knowledge, Netfox Detective is the only open-source NFAT that supports GSE — Sec 5.3.
6. Novel approach for web page reconstruction; in comparison to other tools, we do not only extract objects from HTTP communication, but we also reconstruct the page entirely (rewriting sources of all intercepted objects like CSS, pictures, video streams, etc.). Pages are stored as a MAFF, 2020 archive including snapshots that show how the page changed over time. The JavaScript is interpreted, and particular API calls are mocked to be injected with intercepted ones, like REST API calls — Sec 6.2. The reconstruction of a web-page requires analysis and correlation of multiple L7 conversations, because a page usually references (includes) data from multiple domains.

Note, the system has a modular architecture where processing engine, data-access component, and visualization subsystem can be used separately. The function related to packet capture file processing, namely, file parsing, conversation tracking, application protocol identification, application data extraction, and analysis can also be used as a standalone console tool and integrated to automated investigation procedures and combined with other existing tools.

The source code[2] is released on GitHub and under the Apache Licence 2.0. Additional information can be found on Netfox Detective's YouTube channel: https://goo.gl/fKM8Vs.

The remainder of this paper is organized as follows: Sec. 2 describes the system architecture, illustrates the frontend, and explains possibilities on how to extend Netfox Detective. Sec. 5 highlights some of the unique features of our tool as well as contains a comparison with other prominent network forensics/security tools. The last section concludes the paper.

2. Netfox Detective

Netfox Detective is a network forensic tool that was developed to support digital forensic practitioners to analyze network captures and to extract evidence from packet traces quickly. The development started off as PoC (Pluskal et al., 2015) with slower processing pipeline and storage, a limited set of application protocol support, and capabilities in general. It allows to correctly

[2] https://github.com/nesfit/NetfoxDetective.

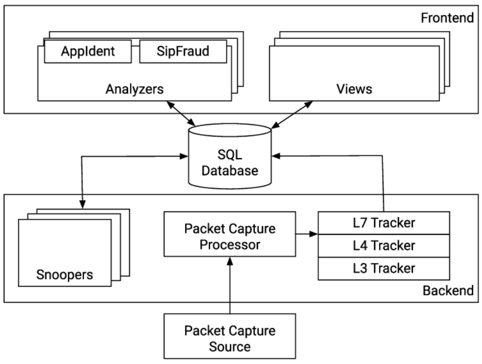


Fig. 1. The overview of Netfox Detective Architecture.

identify network conversations, parse common Internet protocols, and extract metadata as well as content from the end-to-end communication. Additionally, it is possible to extend the tool with new functionality through a well-documented API.

The tool is a Windows application relying on the.NET Platform and is available as an installation package that performs necessary deployment steps. Our implementation exploits many advantages of this platform like the rich graphical user interface provided by Windows Presentation Foundation (WPF), short development times due to a high abstraction language (C#), and availability of many libraries provided through NuGet packages. Furthermore, the implementation utilizes the Task Parallel Library (TPL) for parallel processing.

The software consists of over 140,000 lines of code[3] organized in about 110 projects. While it currently does not support live analysis, it accepts a variety of different network capture formats such as libPcap, 2020, Pcap-NG, 2020, and Network Monitor (MNM) format.

Fig. 1 describes the architecture, which is composed of two main components:

Frontend is primarily a rich visual user interface (GUI, see Fig. 2) that is built on top of the backend and contains analysis capabilities (Sharafaldin et al., 2019). *Analyzers* are frontend interfaces that allow adding new functionality. Details are outlined in Sec. 3.2.

Backend is a network traffic processing engine that performs: capture file processing, protocol parsing, traffic analysis, and metadata extraction. It is independent of the frontend (GUI) and comes with its own CLI which allows to integrate it in automated processing pipelines or to use it as a single-purpose tool. *Snoopers* are backend interfaces that allow adding new functionality. Details are outlined in Sec. 3.4.

2.1. Analyzers vs. snoopers

The tool can be extended through the implementation of snoopers or analyzers. Analyzers have more advanced functionality and different purpose than snoopers. The Analyzer API provides access to data storage as well as the user interface. An analyzer can be bound either to application or investigation scope. Thus, it is possible to integrate highly specialized analyzers for specific cases. Analyzers can create investigations, add capture files, or run any operation supported by Netfox Detective or access any data.

On the other hand, snoopers can access information from the processing pipeline through the database (metadata storage). Snoopers can extract objects from the source data but may also utilize other data such as flow records, log files, etc. Snoopers are intended to parse the application conversation protocols (L7, listed below) and extract data such as files, videos, or HTTP headers. More details about analyzers and snoopers are provided in Appendix C and Appendix B, respectively.

Note, Netfox Detective is too complex to explain every detail in this paper, and thus, we focus on some important design decisions in the next section. We plan on releasing more information/details over the years.

3. Design decisions

While we made many decisions along the way, the following subsections discuss the most important ones: GUI design, investigative process workflow, and packet processing pipeline.

3.1. No live captures

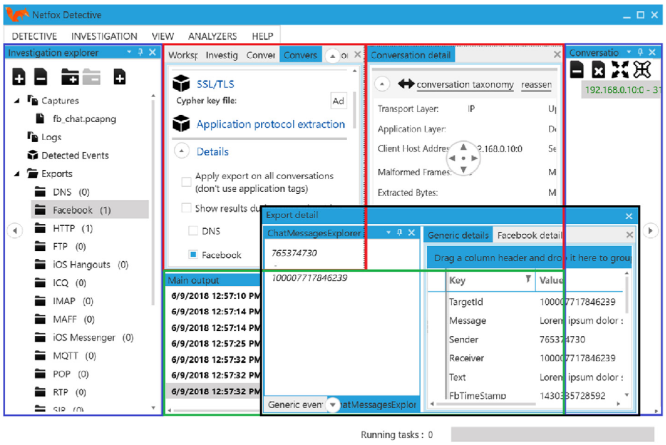
Netfox Detective does not support live captures but accepts several input formats, which had several reasons. First, lawful interception deployment contains one or more capturing probes that store data on drives locally, or on remote storage (Invea, 2020). Secondly, the analysis is often performed on more powerful equipment rather than the capturing probe. Third, this was not a requirement by LEA.

3.2. GUI design

The GUI follows the principles of Master/Detail screen layout (Microsoft Corporation, 2017) supported by the navigator panels as shown in Fig. 2. This organization is ideal for creating an efficient user experience (Scott and Neil, 2009) when the user needs to navigate between linked items (Beebe, 2009). The user interface provides a high degree of customization, utilizing a grid layout of dockable views. The application has three main areas, namely, left-hand side, upper right and lower right, that host basic visual components:

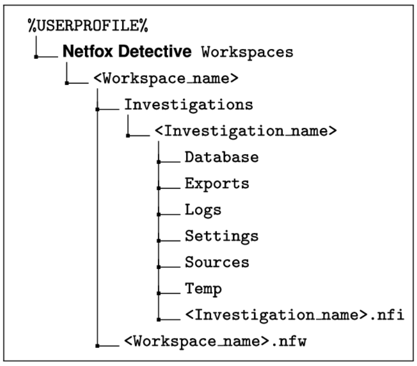
- *Investigation Explorer* is the main navigation panel of the application. It organizes Captures, Logs, Detected Events and Exported objects (see the *left blue box* in Fig. 2). More details about the structure are given in Fig. 3, and discussed in the *Investigation Explorer* paragraph.
- *Conversation View* provides a list of all tracked conversations in source capture files (see *left red box*).
- *Conversation Detail* provides information for the selected conversation. The presented content may contain links for additional data and detailed information on the target object (see *right red box*).
- *Detail View, e.g., Export Detail,* provides additional information for specific object types. The content uses links to navigate via multiple views (see *the black box at the bottom*).
- *Conversation Explorer* contains a list of conversations that were associated with investigated objects, e.g., conversation or export object (see *right blue box*).
- *Output Window* contains a list of events generated during the processing. These events may be informative, warnings or errors raised during source data processing (see the *green box*, only partially shown).

[3] Calculated by Visual Studio (code metrics) on the complete implementation; excludes white spaces, comments, usings, and third-party libraries.



Key	Value
TargetId	100007717846239
Message	Lorem ipsum dolor :
Sender	765374730
Receiver	100007717846239
Text	Lorem ipsum dolor :
FbTimeStamp	1430335728592

Fig. 2. A screenshot of the UI design of Netfox Detective with highlighted dockable locations. Each pane can be moved and docked to any dockable location inside the Netfox Detective window, or drag & dropped outside the window to materialize a new one with the same dockable properties. This way, an investigator split the application across multiple screens.



```
%USERPROFILE%
└── Netfox Detective Workspaces
    └── <Workspace_name>
        ├── Investigations
        │   └── <Investigation_name>
        │       ├── Database
        │       ├── Exports
        │       ├── Logs
        │       ├── Settings
        │       ├── Sources
        │       ├── Temp
        │       └── <Investigation_name>.nfi
        └── <Workspace_name>.nfw
```

Fig. 3. The structure of an investigation folder. All workspaces are stored under the user's profile folder. Each workspace and each investigation has its name — suffixed with GUID for uniqueness. Each investigation contains a database, exports (extracted data from traffic), logs, settings, sources (copies of source data, e.g., PCAPs), and temp (for temporary data generated by snoopers and analyzers). Metadata about the workspace and investigation is stored in **.nfw*, **.nfi* files, respectively.

3.3. Investigative process workflow

The application was designed according to already well-established concepts known from *Integrated Development Environments* that programmers use to organize complex software designs (Microsoft Corporation, 2017). With respect to digital forensics, we consider an *Investigation* to be an equivalent to a *project*; *Investigations* are combined into a *Workspace* that is equivalent to a *Solution*. An investigator can choose on which *Investigation(s)* s/he wants to work on and add data in the form of *PCAP* files or *logs*. Data is processed, and all gathered information is stored in an *Investigation's* scope; nothing is shared beyond that. In case several PCAPs are added (e.g., cause they have been split previously), across analysis is conducted (they will be treated as one PCAP internally for tracking and reconstruction of events).While data is never shared between investigations, we allow opening multiple investigations (in separate docked panes) which allow comparing data from multiple sources.

3.4. Packet processing pipeline

To master the challenges of parsing and to polish all information gathered, it consists of several interconnected implementation blocks which compose a packet processing pipeline. The pipeline (lower right-hand side of Fig. 1) performs (i) packet file loading and processing, (ii) conversation tracking, (iii) application recognition and (iv) extracted (meta)data storing. Thus, the processing pipeline handles the identification of protocols for each packet, performs defragmentation, and does stream reassembly for TCP communication (L7 Tracker). A detailed view is provided in Fig. 4. Note, the snoopers allow to extend the backend and will be discussed in Sec. Appendix C.

Packet file loading and processing. (i.e., components Packet Capture Source, Packet Capture Processor, L3-L7 Trackers, and Appldent): Source packet capture files are processed by the corresponding packet file loader depending on their file type. The processing of the frames is sequentially where each loaded frame is dissected into the low-level protocols to identify its key properties, such as a physical address, network address, or ports. The dissected packet is forwarded to the next component (i.e., L3 Tracker) which performs further processing.

Conversation Tracking. Conversation tracking is a critical component of the system as it examines each dissected packet and associates it with the corresponding conversation.[4] A conversation is considered as the basic data object for further analysis. The system identifies conversations at different network layers:

[4] Note, conversations are also called bi-flows in some literature.

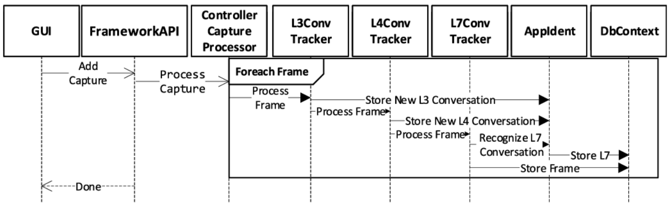


Fig. 4. Abstract capture file processing scheme with a sequential passage. Data dependencies between models are omitted. The ultimate goal is to identify and collect application level conversations. In order to accomplish this, communication at low levels need to be properly identified, messages parsed, relevant data extracted, and packet composed. This is achieved by conversation trackers.

- Packets sharing the same source and destination addresses belong to the same network layer conversation (L3). Every pair of devices shares a single L3 conversation.
- Packets with the same network source and destination addresses, transport layer source and destination ports and a specific transport protocol belong to the same transport layer conversation (L4). At this layer, the conversation mostly corresponds to a pair of TCP streams or UDP data exchanges.
- Lastly, application layer conversations (L7) are identified using various TCP heuristics we have developed previously (Matoušek et al., 2015) and improved for this article. The difference is in the handling of corner cases in TCP reassembling, namely the computation with seq. numbers, order of processing of colliding TCP sequences, and remaining sequences without introductory, or conclusive TCP flags, for details, compare Appendix E 2.e.(ii—iii), 2.h, 4, 5 and original paper. The heuristics solve the problem when dealing with incomplete data or multiple sessions that are merged into a single transport layer conversation. L7 conversations reflect a single session between a client and a server application.

Correct identification of conversations from source packets is a challenging task as several issues may arise, e.g., out of sequence packets, missing packets, fragmented packets, or missing termination packets. To succeed, we use several heuristics to identify and collect as many conversations as possible, even in corrupted or incomplete data sources. Additionally, the tool addresses the need for fast processing by using available processor cores, implementing concurrent conversations processing.

Metadata Storage (database). Extracted information, e.g., conversations at different layers, application layer data units, and other relevant information, is stored in a SQL database. The bulk insert method is used to obtain better performance. Thus integrity is not guaranteed until all data is inserted. The user interface is aware of this and handles temporally incomplete data correctly. The database is accessed through persistence providers that allow to easily add support for different databases.[5]

3.5. Security considerations

Netfox Detective is intended for a single-user environment, i.e., it runs on an investigator's desktop. Therefore, the system does not include user management, authentication, or authorization. The designated way to share investigation between multiple investigators is to export/import the workspace. This decision allows to enable the more extensive use of our tool by investigators that prefer disconnected systems to protect sensitive data against misuse. Netfox Detective, therefore, does not require a certification process to be usable inside LEA.

4. Testing

Given the complexity of our application, testing was (is) an essential part throughout the development process, where we followed a Test-Driven Development (TDD) methodology. TDD requires writing tests first, then production code that passes the tests and lastly to refactor the code to improve its structure. We utilized unit tests, which then also ensures integration/regression testing and ensures the correctness of new versions. Because our focus is very specific (network data parsing and analysis), mocking the data would be tedious (Osherove, 2015). Therefore, we omit the unit tests in favor of integration/system tests that use data loaded from PCAP files processed (in-time of the test) by our processing pipeline.

To develop and test modules (snoopers/analyzers), we started by collecting testing data first, where we either downloaded available PCAPs or created our ground of truth utilizing our private networks. In the latter case, we then filtered the captured data using Wireshark, which ensured that we only deal with one application message, action, or scenario at a time. If Wireshark supported the application protocol, we compared both results (ours and Wireshark's).

In the beginning, we also used Microsoft Network Monitor, 2020 (MNM), which allowed us to develop parsers written in *Network Parsing Language* (NPL). In other words, we created parsers for two different frameworks and compared results. Given that MNM is outdated, and this is not the most reliable method for testing, we abandoned MNM.

After carving basic events from the protocol messages worked correctly (single packet), we created more complex scenarios (e.g., a login scenario which has multiple packets) and manually verified the results. Lastly, we created a comprehensive dataset and extracted key data (e.g., the summary of extracted events) which we then used as benchmark data for new version testing to prevent regression bugs. Currently, Netfox Detective contains 1000+ tests that are automatically executed whenever new code is committed and run approximately 46min. In case that a regression bug is found, the merge is denied until the bug is fixed.

5. Evaluation

The rest of this section discusses the efficiency (see Sec. 5.1) followed by a summary of carving capabilities. In Sec. 5.3 we compare Netfox Detective to other exiting tools before we provide a real-world example. The last section explains the sec:web; a very unique feature of our application.

[5] Currently, the tool supports Microsoft SQL and in-memory data storage.

5.1. Efficiency assessment

Although Netfox Detective is an offline analysis tool, runtime/memory footprint are essential aspects. Thus, this section discusses the runtime efficiency in comparison with Wireshark and NetworkMiner. To measure the efficiency, we used the M57, 2020 M57-Patent scenario[6] PCAP files which consist of several PCAP files with a total size of 4.8GB and 5, 707, 845 frames (we combined them into a single PCAP). Note, given that each tool performs very different tasks, this is only a rough comparison.

The results are provided in Table 1. As can be seen, Netfox Framework is slightly faster than Wireshark despite the TCP reassembling of all sessions. Note, when opening the case the 2nd time, all data is extracted from the database which is completed in a matter of seconds. However, we require more memory footprint (RAM). Netfox Detective is slightly slower than Netfox Framework as it visualizes the information. NetworkMiner is about 4–7 times slower than the other tools. The average CPU usage is not reaching 100% with Netfox Framework and Netfox Detective because of the thread synchronization, I/O operations, Garbage Collection, and back pressure in the processing pipeline that balances overall performance and resource utilization. Overall, the Mbps per tool vary between 15 and 100.

Additional efficiency indicators are given in Table 2, where we focus on rows 12 and 13 (processing speed and parallel processing; remaining rows are discussed in Sec. 5.3). As shown, Netfox Detective allows parallel processing, which should make it faster than the deprecated PyFlag. On the other hand, Cohen (2008) points out that PyFlag is not intended for high-speed. Concerning XPlico, more research is needed as it also processes data in parallel, and we did not find information on processing speed.

5.2. Event carving capabilities

The next important aspect for forensics is *event carving*, i.e., restoring particular events such as an FTP Login, a DNS query or sending emails from a comprehensive stream. This section primarily focuses on NetworkMiner (NM) and Netfox Framework and their capabilities; Wireshark does not incorporate advanced forensic features such as emails or web page reconstruction as it is intended for Network Security Monitoring (Sira, 2003; Pilli et al., 2010).

For comparison, we decided to focus on detected sessions, TCP reassembling, and DNS records where the results are shown in Table 1. These properties strongly depend on how a tool was implemented. Higher numbers reflect finer granularity (this does not mean that higher (or lower) numbers are better).

Sessions: the number of TCP and UDP sessions recognized by each tool. This feature strongly depended on the mechanism handling missing fragments, see Appendix E. Ithere is no packet loss; the tools should report the same number of TCP sessions; UDP sessions can differ in case the tool uses an inactivity timeout threshold to split UDP sessions (the UDP protocol does not carry any signaling information that can be used to determine the end of a session).

TCP missing: signifies how much information is lost and cannot be recovered, e.g., capturing problems, packet loss, or storage issues. All issues are related to actions that occurred before processing of the capture file, i.e., they are not caused by Netfox Detective. There are different ways to calculate the loss as shown in Eq. (1) or Eq. (2):

$$lost\_packets \,/\, all\_packets[\%] \tag{1}$$

$$lost\_bytes \,/\, all\_bytes[\%] \tag{2}$$

Netfox Detective uses the Eq. (2) as we believe that if a sequence of packets is lost, their count is unknown and can be approximated using a heuristic based approach on MTU or previously observed segment sizes. However, we had to utilize Eq. (1) as Wireshark does not explicitly count *lost_bytes*.

DNS records: the number of events carved from DNS traffic. Netfox Detective extracts much more events compared to NM that only considers DNS response packets (Mockapetris, 1987b) and ignores query packets (Mockapetris, 1987a). NM also ignores some other record types such as PTR, SRV or MX that may carry valuable forensic information, e.g., a mapping of IP address to the domain name (PTR), a definition of the service location (the hostname and port number (SRV)), or domain names of mailing servers (MX). This additional information may be useful in case of DNS spoofing attacks/investigations (Huber et al., 2010). Lastly, NM only shows the first record from an answer section. In contrast, Netfox Framework processes all, i.e., all records from Question, Answer, Authority, Additional from both packet types (not only response).

Emails and errors: reflects the number of extracted emails. NM identifies more emails as Netfox Framework currently only considers emails sent through the SMTP protocol; NM also processes emails sent through webmail.[7]

FTP: the number of events identified in the FTP session. While NM extracts only transferred files, Netfox Detective and Wireshark show other related (meta-)information about the FTP sessions such as the login or list-command.

Web pages: the number of reconstructed web pages using our module. In total, 182 HTTP objects were found which created six MAFF Archives containing full offline web page snapshots including CSS and other downloaded objects. For additional details we refer to Sec. 6.2.

In summary: each of the tools has its strengths and weaknesses, and one has to choose the best tool for the job. For instance, Netfox Detective has focused on carving capabilities from conversations containing missing data.

5.3. Comparison to existing tools

This section compares Netfox Detective against other applications concerning capabilities, functionality, and features. A summarized overview with is provided in Table 2 and is discussed in the upcoming paragraphs.

In its current version, Netfox Detective does not allow *live data capture* or *PCAP-over-IP* and thus is not as flexible as NetworkMiner, 2020 or XPlico, 2020. However, it supports various capture file types. Note, this was a design decision: we work under the premise that data is gathered on capturing probes and uploaded for analysis after the capture ends (or parts of the ongoing capture are provided).

In terms of support for encapsulation protocols, NetworkMinor has a wide variety of supported protocols. However, to the best of our knowledge, Netfox Detective, and Wireshark are currently the only applications that support Generic Stream Encapsulation (GSE). In comparison to other protocols, GSE frequently uses multiple encapsulations, whereas other protocols usually do not. That requires a significant change in the tool's architecture.

[6] https://digitalcorpora.org/corpora/scenarios/m57-patents-scenario%20 (last accessed 2019-08-17).

[7] This was a scenario we have not considered. We will update our module in the near future.

Table 2
Netfox Detective in comparison to major open-source network forensic tools. The provided information was gathered from official sources provided by the tool authors. N/A indicates that we could not find any details regarding the particular feature. We deliberately do not add any information that is not stated by authors, such as processing speed.

	Tool	Netfox Detective	NetworkMiner	XPlico	PyFlag
	Feature				
1	**Live data capture**	NO	YES	YES	NO
2	**PCAP-over-IP**	NO	YES	YES	NO
3	**Supported file types**	libPcap, Pcap-NG, MNM	libPcap, Pcap-NG	libPcap	libPcap
4	**IPv6**	YES	YES	YES	NO
5	**Encapsulation protocols**	GRE, 802.1Q, GSE	GRE, 802.1Q, PPPoE, LLMNR, VXLAN, OpenFlow, SOCKS, MPLS and EoMPLS	L2TP, VLAN, PPP	NO
6	**Application Protocol Identification**	SPID, NBAR, ESPI, Bayessian, Random Forests	SPID, PIPI	PIPI	NO
7	**Supported application protocols**	HTTP, SSL/TLS, MAFF. XMPP, YMSG, OSCAR, Facebook Messenger, Hangouts, Twitter, XChat, IMAP, POP3, SMTP, Gmail, Yahoo, RTP, SIP, Minecraft, Warcraft, Facebook, Stratum, DNS, FTP, SPDY, MQTT	FTP, TFTP, HTTP, SMB, SMB2, SMTP, POP3, IMAP, YouTube	HTTP, POP3, SMTP, IMAP, SIP, RTP, SDP, FTP, DNS, IRC, IPP, PJL, MMS, SLL	DNS, HTTP, MSN, Gmail
8	**Applications Identification**	YES	NO	NO	NO
9	**OS Fingerprinting**	YES (using typical applications)	YES	NO	NO
10	**Credentials Extraction**	Facebook, IMAP, SMTP, POP3	SMTP, HTTP Digest Authentication	NO	NO
11	**Incomplete or malformed communication**	TCP data loss, IPv4 fragmentation	N/A	NO	NO
12	**Processing speed**	100 Mbps	11.92–18.49 Mbps	N/A	N/A
13	**Parallel processing**	YES	NO	YES	NO
14	**Advanced analytical views**	YES	NO	YES	NO
15	**Persistent storage**	MSSQL, in-memory	CSV/Excel/XML/CASE/JSON-LD	SQLite, MySQL or PostgreSQL	VFS
16	**Querying/ filtering**	3-rd party tools on SQL DB	keyword search	3-rd party tools on SQL DB	YES

Rows 6–8 deal with application and their protocols. While Netfox Detective uses a variety of different algorithms to identify the protocol, NetworkMiner and XPlico rely on SPID and PIPI. Furthermore, Netfox Detective tries to identify applications as well as application protocols, e.g., HTTPS-Firefox, HTTPS-Chrome (Pluskal et al., 2018). However, further testing is required to make a qualified decision which tool works the most reliable. Concerning supported application protocols, our tool supports a wide variety of different ones, including some unique protocols like Facebook Messenger, Hangouts, Twitter, or Warcraft. Note, since those are implemented using snoopers, there will be more in the future.

OS fingerprint (row 9) is supported by NetworkMiner and Netfox Detective. While we rely on the *AppIdent* analyzer, NM uses statistical based SPID algorithm (Hjelmvik and John, 2009).

In case that user credentials are observed in a communication, Netfox Detective, and NetworkMiner allow to extract them where the two tools focus on different protocols. Another major feature is the handling of malformed, incomplete network traffic. This is based on our previous work (Matoušek et al., 2015) where we showed that the risks of undesired association of the unrelated conversation fragments yielding twisted evidence. We could not find information for NetworkMiner; however, as shown in Table 1, NetworkMiner identifies significantly fewer sessions (maybe due to combining unrelated conversations). Advanced analytical views address visualization capabilities where Netfox Detective is very flexible due to the Analyzer API (see Sec. B), which ensures that the tool can be extended with pluggable modules. In terms of XPlico, we were unable to find detailed information; besides a reference to a PHP Framework named cake-php.[8]

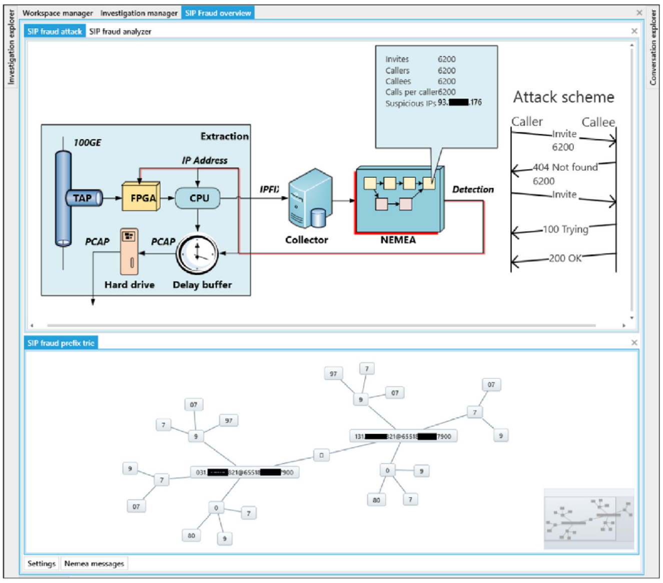
Row 16 addresses the querying/filtering capabilities of the corresponding tools. NetworkMiner, 2020, Wireshark, 2020 and PyFlag, 2020 include basic query functionality (e.g., keyword searches), XPlico and Netfox Detective require third-party tools (e.g., one may query the database using analytical third-party applications or write a new snooper). If support for hitherto application protocol is required, the advanced investigator can create a new snooper module that will be dynamically be loaded without a need of recompilation of the Netfox Detective. In comparison to Wireshark, creation of a new snooper is straightforward imperative programming based on an enriched API of a data stream that handles several types of application protocol behaviors, like request-response, asynchronous message exchange, etc., that helps to handle missing/not-captured data.

To sum it up: While there are aspects where other applications like NetworkMiner are superior, Netfox Detective has a lot of unique functionality/features and is under active development — new features can be expected. Especially the number of supported application protocols, the incomplete or malformed communication handling make and the expandability, make it a great forensics tool. Additionally, we believe that one of the major difference is usability and the amount of expertise needed (especially compared with Wireshark).

6. Example features

This section presents two of the many advanced features of Netfox Detective and have been chosen as they make Netfox Detective unique. These features have been tested in real deployments and helped LEA investigators to solve cases. Given their complexity, we also provide brief video summaries at the beginning of the corresponding sections. More videos about its capabilities can be

[8] http://wiki.xplico.org/doku.php?id = interface (last accessed 2019-08-17).



Invites 6200
Callers 6200
Callees 6200
Calls per caller6200
Suspicious IPs 93.[redacted].176

Fig. 5. This figure describes *SIP Fraud Analyzer*. The view is an interactive animation that reflects the actual state of the deployed 100GE hardware-accelerated network card with the IPFIX collector and the NEMEA system that detects network incidents based on IPFIX records. *SIP Fraud* is visualized on the upper right side with a count currently analyzed messages, i.e., 6200. At the bottom, a tree-like structure visualizes a *prefix tree* that is an interpretation of the attack. The root node in an interconnection between the same roots of telephone number attacked from different IP addresses. The path from a leaf node to the root aggregate node represents a prefix combined with a PSTN number that was tried to be called. Sensitive information, as a part of called number and IP addresses, was omitted.

found on Netfox Detective's YouTube channel.[9]

6.1. An example: SIP fraud analysis

This section reviews Netfox Detective in use based on a simulated SIP (Session Initiation Protocol) Fraud case. The SIP Fraud attack exploits a misconfiguration of the SIP server where the attacker tries to guess a secret prefix that is used to initiate a call from a VoIP network to PSTN (public switched telephone network). If the attacker finds the correct prefix, the Gateway (Callee) replies with a *200 OK* SIP message. The attacker then uses the discovered prefix to initiate a call on a premium number. The costs of the call are charged to the owner and will profit the attacker. A visual summary can be found here: http://y2u.be/P2W9uANYKyI.

To tackle the challenge, we developed the SIP Fraud Analyzer that can perform a postmortem analysis of possible SIP fraud attacks in given PCAPs. The exact procedure is best explained by Fig. 5. The upper part is an interactive animation that reflects the actual state of the system (commodity server with hardware-accelerated network card), the IPFIX collector and NEMEA system (Cejka et al., 2016) (note, this is *not* part of Netfox Detective but external equipment/software). In a nutshell, the hardware (left-hand side) captures information and forwards it to NEMEA. Once an attack (or false-positive thereof) is identified (Jansky et al., 2017), NEMEA notifies the appliance, which then captures all evidence (generates a PCAP) and stores it on the hard drive. This file then serves as input for Netfox Detective.

Knowing the workflow, we now focus on the analyzer and its responsibilities. First, NEMEA can notify Netfox Detective about its current state which allows us to update the view (e.g., the red arrow pointing from NEMEA to FPGA). Thus, an investigator can see (live) the current processing. Second, NEMEA can notify Netfox Detective when capturing is completed and trigger the analyzer to download and visualize the PCAP (the bottom pane in the figure).

Fig. 5 shows the SIP Fraud Analyzer main view that visualizes the attack pattern. The evidence has the form of prefix guessing activity represented by several SIP INVITE messages that differ by the prefix of the callee number (here it is the number 031 … @65518…. and a lot of seemingly random prefixes which reflect the attacker guessing them). In other words, if the analyzer shows a graph like this, one knows an attack occurred; if we find *200 OK* message, we know that the attack was successful.

The system was tested/developed with a confidential dataset from the National Research and Education Network (NREN). During the experimental deployment of this system, we were able to successfully extract evidence, and based on that we informed victims about their misconfiguration in SIP's PBX.[10]

6.2. Reconstruction of web pages

Another feature of the Netfox Detective is web page reconstruction which can be viewed here: http://y2u.be/CPO2rhe5Xs8. First, the *SnooperHTTP* extracts all HTTP objects and stores the

[9] https://goo.gl/fKM8Vs

[10] PBX – Private branch exchange used to relay VoIP communication to the PSTN – public switched telephone network.

contents on disk. Second, *SnooperMAFF* iterates through the HTTP objects to identify all *HTML* documents. Subsequent analysis of these documents yields all linked objects, e.g., CSS files, JavaScript scripts, media streams, and so on. Lastly, all references to web resources are rewritten (e.g., <a href = "http:// … /photo.png" will be replaced by <a href = "./photo.png"), and then the *HTML* documents including all resources are packed into Mozilla Archive Format (MAFF) archive.

The self-contained MAFF archive[11] contains all data that is related to each web page that was viewed. Experimentally, in case of the dynamic web that loads data continuously, we try to create multiple so-called snapshots that approximate how the web page may have looked. The snapshot is created with each significant change to the web page. The investigator is warned that this is experimental approximation and not an accurate replica. We do this approximation by interpreting JavaScript scripts and supplying it with resources previously extracted. Hence, we can reconstruct some dynamic pages like *webmails*, *chats*, or *video streaming* services. These approximations are stored inside the MAFF archive as additional tabs.

Note, web page reconstruction is only possible if the session is established using plain HTTP. Otherwise, it requires the investigator to get into the middle of the communication using a MitM proxy like SSLSplit, 2020 that can capture unencrypted traffic or store SSL/TLS session keys (Rescorla, 2018).

7. Conclusions

The amount of data sent over networks increases daily, and so does the number of devices connected to it. Additionally, analyzing the data becomes more complex due to encryption, the large number of different protocols or tunneling. As a consequence, forensic investigators are overwhelmed with data (possible evidence), and traditional workflows are outdated (i.e., manually combing several specialized tools like SSLSplit, 2020, TShark, 2020, or Wireshark, 2020). To cope with these challenges, it requires automated, extendable tools that support practitioners by summarizing data and providing visualization, which allows easy comprehension of the information (Beebe, 2009).

In this article, we presented Netfox Detective which is a comprehensive open-source network forensic analysis tool (NFAT) available under the Apache 2.0 License. By design, our application can be expanded by implementing new modules; backend modules are called snoopers and frontend modules (which allow more complexity) are named analyzers. To enable researchers to create new modules, we have a well-documented API including several examples. The GUI follows the principles of a Master/Detail screen layout and uses dockable views, which makes it intuitive and easy-to-use. We achieve better performance than comparable tools because of the parallel pipeline processing. As a side note: it was used by CESNET[12] for SIP Fraud Detection as mentioned in Sec. 6.1.

The evaluation and comparison with existing tools show that Netfox Detective has a good efficiency as it makes use of all cores. Additionally, it has some unique features, that cannot be found in any other NFAT, e.g., a large number of supported application protocols as listed in Table 2, support for GSE tunneling, or heuristic extraction from malformed data.

For the future, we plan on expanding Netfox Detective by creating new modules (features), e.g., finding similarities using approximate matching (Breitinger et al., 2014). We also plan on changing the mechanisms for data processing to allow computation on clusters. In terms of interoperability, we intend to add exporting capabilities into standard formats, e.g., Advanced Forensic Format (Cohen et al., 2009) or CybOX (Casey et al., 2015). Lastly, we want to create training sessions and material which will allow practitioners to become familiar with our tool.

Acknowledgement

This article has been supported by the Ministry of Education, Youth and Sports from the National Program of Sustainability (NPU II) project IT4Innovations excellence in science (no. LQ1602).

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.fsidi.2020.301019.

Appendix A. Terminology and definition

There are several definitions in the community regarding flow, conversation, etc. For this work, we used the Microsoft Network Monitor (MNM) terminology[13] which is very close to the well established terminology used by Kurose and Ross (2016).

Frame *is a data link layer (L2) protocol data unit.*

Packet is an internet layer (L3) protocol data unit.

Datagram *is a transport layer (L4) protocol data unit.*

Protocol/application message is a application layer (L7) protocol data unit (PDU). A message is a collection of one or more L7 PDUs.

L3 flow *is a sequence of packets having the same source and destination IP addresses. It represents an uni-directional transmission of packets between two network nodes.*

L3 conversation is a pair of L3 flows with mutually transposed source and destination IP addresses. It represents bi-directional transmission between two network nodes.

L4 flow *is a sequence of packets with the same source and destination IP addresses and ports, and an L4 protocol number. It represents uni-directional communication between processes, e.g., data sent by an HTTP client to an HTTP server, possibly in several L4 half sessions. An L4 flow consists of one or more L7 flows.*

L4 conversation is a pair of L4 flows with mutually transposed L3 and L4 identifiers (src/dst IP addresses and src/dst ports). It represents bi-directional communication between processes, e.g., requests and responses between an HTTP client and server. The L4 conversation may contain several L4 sessions (L7 conversations) between the same network nodes using the identical src/dst ports and the L4 protocol.

L7 flow *is a part of the L4 flow that represents a transport session, e.g., one UDP or TCP session. For TCP, an L7 flow is bounded by its initial SYN packet and its closing FIN or RST packet. For UDP, an L7 flow corresponds to an L4 flow. One L4 flow may include several L7 flows that are logically independent, e.g., several TCP sessions (HTTP requests) with the same src/dst ports and IP addresses may compose one L4 flow. A TCP L7 flow also includes starting SYN and ACK packets without any L7 payload, if present.*

L7 PDU represents an approximation of an application message, e.g., HTTP request. L7 PDU is a logical object that contains an L7 payload of one or more packets belonging to the same L7 flow. It is

[11] Note, Mozilla discontinued MAFF support in newer Firefox versions. We advise using SeaMonkey with MAFF plugin https://addons.thunderbird.net/en-us/seamonkey/addon/mozilla-archive-format/.

[12] CESNET is a developer and operator of national e-infrastructure for science, research, development, and education in Czech Republic.

[13] The terminology was determined by study of MNM manual, and blog — https://blogs.technet.microsoft.com/netmon/(last accessed 2019-08-17).

created using TCP reassembling. L7 PDU objects are processed by application parsers called L7 Snoopers. In a case of UDP, an L7 PDU is created for every L4 payload, i.e., there is an 1:1 relation between UDP payload and application message.

L7 conversation *is a pair of L7 flows. It represents logical application data that are subjected to the forensic analysis. L7 flows are interconnected to the conversation according to SYN and SYN + ACK sequence numbers in TCP. An L7 conversation includes meta data such as timestamps of the first and last PDU — selected from both directions whichever is prior and posterior, number of frames of L7 conversation, collection of virtual frames representing missing (expected) frames, type of encryption, cipher keys (for TLS decryption), collection of probable application tags (types of L7 protocol, e.g., HTTP, SMTP, etc.).*

L7 Snooper is an application data analyzer (application parser, dissector). Snooper reads L7 PDUs from L7 conversations and performs L7 processing, analysis, and visualization. L7 snoopers can co-operate with each other, e.g., SIP snooper co-operates with RTP snooper, WebMail snoopers with HTTP snooper, etc.

L7 Analyzer *is a less strict abstraction, module that encapsulates predefined behavior that applies to processed data or directly controls data processing. L7 Analyzers have full access to Netfox Detective platform and can change, extend any functionality used for processing or analysis.*

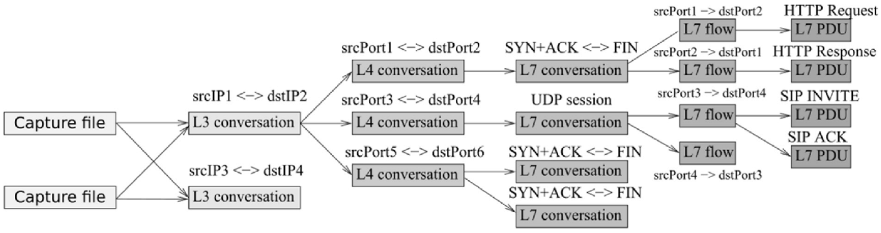


Fig. A.6. Figure describes relations between encapsulations on various levels of networking stack reflected by object hierarchy serving as containers. Data is segregated into a particular container based on common identifiers described in Section Appendix A. One *L3 Conversation* can contain frames from multiple *capture files* and have a relation *one to many* with *L4 Conversations*. The rest of graph is read similarly.

Appendix B. Analyzers (Frontend Modules)

Analyzers extend Netfox Detective with more advanced functionality that cannot be implemented as snoopers. The Analyzer API provides access to data storage as well as the user interface. An analyzer can be bound either to application or investigation scope. Thus, it is possible to integrate highly specialized analyzers for specific cases. In order to grasp the concept of analyzers, we discuss their capabilities based on the *AppIdent* — an application identification analyzer. AppIdent assigns an application protocol (or even network application) to every flow in the source data. The goal of the analyzer is to recognize applications (e.g., Google Drive, iTunes, or OneDrive) in network traffic instead of just the application layer protocol used (e.g., HTTPS).

The analyzer is implemented using machine-learning (Christodorescu et al., 2015) and statistical methods, in particular, Bayesian belief network, Random Forests, and Enhanced Statistical Probability Identification, to make the decision. Because supervised learning methods are used, there are two running modes:

The learning mode is used to build the models which required annotated data. To generate the data, we produced local network traffic and captured the communication using Microsoft Network Monitor, which automatically enriches the capture with information about the running processes.[14] Note, the training data was annotated with the application process instead of the application protocol. On the other hand, our backend engine was extended to extract the process information for learning purposes. The feature vector characterizing the application protocol was specified according to the work by Moore et al. (2013), and customized for L7 conversation-based approach instead of packet-based.

The classification mode of the analyzer is used for annotating conversation with recognized protocols and applications. It is not an easy task, and the precision varies for the classification methods and the target set of applications. For more details see Pluskal et al. (2018) who demonstrated that it is possible to distinguish between communications traces of OneDrive, Skype, iTunes, Spotify, Steam and µTorrent clients, although all of them use HTTPS.

Usually, traffic classification is a black box (e.g., in security software/hardware like IDS/IPS) and depends on the model. However, for practitioners, it may be helpful to get more insight and therefore *AppIdent* can provide additional computed results in a visual manner. In other words, we implemented views allowing the comparison of the classification results of different methods, classifier performance analysis, and hyper-parameter tuning.

Appendix C. Snoopers (Backend Modules)

The backend supports modules, called snoopers, that can access information from the processing pipeline through the database (metadata storage). Snoopers extract objects from the source data but may also utilize other data such as regular log files. Therefore, snoopers parse the application conversation protocols (L7, listed below) and extract data such as files, videos, or HTTP headers. These extracted objects are then either stored in the database or pushed to the *Investigation Explorer* (grouped by a protocol) or can be accessed from the special *Export Overview* pane where they are grouped by event type, e.g., emails, images, chat messages. The following protocols for metadata and/or content extraction are supported:

- Common internet protocols: *DNS*, *SPDY*, and *SSL/TLS*.
- Selected application protocols: *HTTP(S)*, *IMAP*, *POP3*, *SMTP*, and *FTP*.
- Email services: *Gmail*, *Yahoo*, and other *webmails*.
- Instant messaging: *XMPP*, *YMSG*, *OSCAR*, *Facebook Messenger*, *Hangouts*, and *XChat*.

[14] In detail, MNM creates a Process Info table that stores information on the socket and the process that created it.

- Social networks and gaming: *Twitter*, *Facebook*, *Minecraft*, and *Warcraft*.
- Bitcoin communication: *Stratum*.
- Voice over IP systems: *RTP* and *SIP*.
- Internet of Things communication: *MQTT*.

If the communication is not encrypted (or the server's private key is provided, and the server's configuration allows it), the snoopers can extract the communication content, e.g., transmitted files. For secured communication, only traffic metadata is available.

In order to create new snoopers, there are three abstract classes that need to be inherited:

SnooperBase can be seen as the extractor that will handle the identification of objects. The registration of a new snooper and its integration is automated as long as the snooper's *DLL* resides in the root directory of the application. Details about the snooperBase are provided at the end of this subsection.

SnooperExportObjectBase stores the actual objects. For instance, an application protocol parser will dissect the communication and create instances of domain objects. Those objects might also implement various interfaces like IChatMessage, ICall, IEMail, IPhotoMessage, etc. to automatically integrate exported objects in generic views.

SnooperExportBase encapsulates (meta-)information about the export process. For instance, the source of an L7 conversation. Additionally, it contains all extracted objects SnooperExport- Object Base.

SnooperBase. To create a new snooper, a new class that inherits from the abstract class SnooperBase including the class members, such as Name, Description, KnownApplicationPorts, CreateSnooper Export, and ProcessConversation, needs to be implemented. Additionally, the class defines multiple abstract methods that represent callback functions executed during conversation processing. An example is given in Appendix D. The functionality has to be implemented in the following methods:

- On Conversation Processing Begin – any relevant activity for creating a new object to be populated by the module.
- On Conversation Processing End – any required processing before the new object is stored in the database.
- On Before Protocol Parsing and On After Protocol Parsing – takes care of the internal state of an object and handles exceptional cases that are assigned to 'parsing state'.
- On Before Data Exporting and On After Data Exporting – takes care of the internal state of an object and handles exceptional cases that are assigned to information 'extraction state'.

Each snooper is executed on-demand, on the selected PCAP or a collection of them, according to the tool configuration. While modules can use the information provided by other modules, their basic use case is to implement extraction capabilities for application protocols. For more complex analysis, we use analyzers.

Appendix D. Abstract code for an Example Protocol snooper creation

```
public class SnooperExample : SnooperBase {
  public override string Name => "Example Protocol";
  protected override SnooperExportBase CreateSnooperExport()
      { throw new NotImplementedException(); }
  public override string Description => "Description of Example Protocol";
  public override int[] KnownApplicationPorts => new[] { 42 };

  protected override void ProcessConversation()
  {
      // we need a stream to read from
      var stream = new PDUStreamBasedProvider(this.CurrentConversation,
                                          EfcPDUProviderType.Breaked);
      // now we can create a reader for the stream
      var reader = new PDUStreamReader(stream, Encoding.GetEncoding(437), true);

      // reader will spawn messages, cycle through them
      do
      {
          this.OnBeforeProtocolParsing();

          // parse the protocol
          var message = new ExampleProtocolParseMsg(reader);
          if (!message.Valid){
              //TODO report error
              continue;
          }

          this.OnAfterProtocolParsing();
          // TODO some additional integrity checks perhaps
          this.OnBeforeDataExporting();

          var exportedObject = new SnooperExportedDataObjectExampleProtocol
                                  (this.SnooperExport){...};
          this.SnooperExport.AddExportObject(exportedObject);

          this.OnAfterDataExporting();
      } while (reader.NewMessage());
  }
}
```

Appendix E. Simplified reassembling algorithm implemented in Netfox Detective.

```
1. Select L4 flows and sort packets using their sequence numbers.
2. Process each L4 flow accordingly:
(a) Start following iteration with a SYN packet, i.e., $P_i$.
(b) Increment $Seq_i$, i.e., $Seq_i+=1$.
(c) Create a new L7 Flow to be a collection of L7 PDUs for following algorithm.
    Set $P_{init} = P_i$.
(d) Create a new L7 PDU if does not exist or if a previous L7 PDU was closed. (e) If
    $Seq_i \neq Seq_{i-1} + |P_{i-1}|$ (the expected packet is missing, check timestamps $TS$
    and sequence numbers $Seq$ as follows:
i. If $TS_i - TS_{i-1} \leq MaxTime$ and $Seq_i - Seq_{i-1} \leq MaxLost$ then a virtual packet will
    be created to replace the missing packet.
ii. If $TS_i - TS_{i-1} \geq MaxTime$ and $Seq_i - Seq_{i-1} \leq MaxLost$ then there is an
    overlapping of TCP sessions because $i$ packet, i.e., this packet, belongs to a
    different L7 flow. Skip this packet and proceed with the next one.
iii. If $Seq_i - Seq_{i-1} \geq MaxLost$ then there are too many missing data. The flow
    cannot be fully restored. Close it and proceed with next SYN packet.
(f) If $Seq_i == Seq_{i-1} + |P_{i-1}|$ the $P_i$ packet is expected, i.e., $P_i$ contains following
    data segment, add it into the L7 PDU created in 2 d.
(g) If FIN/RST/PSH flag is found or $|P| = = MaxPayload$, close the L7 PDU.
(h) If $P_{init} = = P_i$, break iteration.
(i) Increment $i$, i.e., $i+=1$ and GOTO 2 d.
3. If there remains any SYN packet in the current L4 flow, GOTO 2a
4. If the L4 flow contains any unprocessed packet, i.e., captured communication
    is incomplete and heuristic methods (2e) have to be applied.
5. Select packet $P_i$ that has maximal $Seq_i - Seq_{i-1}$ and GOTO 2c
6. Combine opposite L7 flows into an L7 conversation using corresponding SYN
    and ACK numbers.
```

P_i — represents the packet on the i-th index
$|P_i|$ — represents a payload size obtained from the packet header
Seq_i — represents sequence number of packet on i-th index
P_{init} — stores the reference to the packet that the reassembling algorithm started with
TS_i — represents time stamp of the packet on the i-th index
$MaxTime$ — variable, empirically set to 600 s
$MaxLost$ — variable, empirically set to 3800 B
$MaxPayload$ — variable, empirically set to maximal expected MTU

References

Beebe, N., 2009. Digital forensic research: the good, the bad and the unaddressed. In: IFIP International Conference on Digital Forensics. Springer, pp. 17–36.

Breitinger, F., Guttman, B., McCarrin, M., Roussev, V., White, D., 2014. Approximate Matching: Definition and Terminology. Special Publication 800-168. National Institute of Standards and Technologies. https://doi.org/10.6028/NIST.SP.800-168.

Casey, E., 2004. Network traffic as a source of evidence: tool strengths, weaknesses, and future needs. Digit. Invest. 1, 28–43.

Casey, E., Back, G., Barnum, S., 2015. Leveraging cybox™ to standardize representation and exchange of digital forensic information. Digit. Invest. 12, S102–S110.

Cejka, T., Bartos, V., Svepes, M., Rosa, Z., Kubatova, H., 2016. Nemea: a framework for network traffic analysis. In: Network and Service Management (CNSM), 2016 12th International Conference on. IEEE, pp. 195–201.

Christodorescu, M., Hu, X., Schales, D.L., Sailer, R., Stoecklin, M.P., Wang, T., White, A.M., 2015. Identification and classification of web traffic inside encrypted network tunnels. US Patent 9 (106), 536.

Cohen, M., Garfinkel, S., Schatz, B., 2009. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. Digit. Invest. 6, S57–S68.

Cohen, M.I., 2008. PyFlag - an advanced network forensic framework. Digit. Invest. 5, 112–120.

Corey, V., Peterman, C., Shearin, S., Greenberg, M.S., Van Bokkelen, J., 2002. Network forensics analysis. IEEE Internet Comput. 6, 60–66.

EnCase, 2020 (cited January 2020). https://www.guidancesoftware.com/encase-forensic.

ENISA, 2019. Introduction to network forensics (cited January 2020). https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/documents/introduction-to-network-forensics-handbook.pdf.

Farmer, D., Venema, W., 2009. Forensic Discovery, first ed. Addison-Wesley Professional.

Garfinkel, S.L., 2010. Digital forensics research: the next 10 years. Digit. Invest. 7, S64–S73.

Harichandran, V.S., Breitinger, F., Baggili, I., Marrington, A., 2016. A cyber forensics needs analysis survey: revisiting the domain's needs a decade later. Comput. Secur. 57, 1–13.

Hjelmvik, E., John, W., 2009. Statistical protocol identification with spid: preliminary results. In: Swedish National Computer Networking Workshop, pp. 399–410.

Huber, M., Mulazzani, M., Weippl, E., 2010. Who on earth is “mr. cypher”: automated friend injection attacks on social networking sites. In: Rannenberg, K., Varadharajan, V., Weber, C. (Eds.), Security and Privacy – Silver Linings in the Cloud. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 80–89.

Invea, 2020 (cited January 2020). https://www.invealawfulinterception.com.

Jansky, T., Čejka, T., Bartoš, V., 2017. Hunting sip authentication attacks efficiently. In: Tuncer, D., Koch, R., Badonnel, R., Stiller, B. (Eds.), Security of Networks and Services in an All-Connected World. Springer International Publishing, Cham, pp. 125–130.

Kekely, L., Kučera, J., Puš, V., Kořenek, J., Vasilakos, A.V., 2016. Software defined monitoring of application protocols. IEEE Trans. Comput. 65, 615–626.

Kurose, J.F., Ross, K.W., 2016. Computer Networking: a Top-Down Approach, vol. 7. Addison Wesley, Boston.

libPcap, 2020 (cited January 2020). https://www.tcpdump.org/.

M57, 2020 (cited January 2020). https://digitalcorpora.org/corpora/scenarios/m57-patents-scenario.

Maff, 2020 (cited January 2020). http://maf.mozdev.org/maff-specification.html.

Matoušek, P., Pluskal, J., Ryšavý, O., Veselý, V., Kmeť, M., Karpíšek, F., Vymlátil, M., 2015. Advanced techniques for reconstruction of incomplete network data. lecture notes of the institute for computer sciences. Soc. Info. Telecommun. Eng. 69–84, 2015.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J., 2008. Openflow: enabling innovation in campus networks. Comput. Commun. Rev. 38, 69–74.

Microsoft Corporation, 2017. Master/details - windows uwp applications (cited January 2020). https://docs.microsoft.com/en-us/windows/uwp/design/controls-and-patterns/master-details.

Microsoft Network Monitor, 2020 (cited January 2020). https://blogs.technet.microsoft.com/netmon/.

Mockapetris, P., 1987a. RFC 1034 Domain Names - Concepts and Facilities.

Mockapetris, P., 1987b. RFC 1035 Domain Names - Implementation and Specification.

Moore, A., Zuev, D., Crogan, M., 2013. Discriminators for Use in Flow-Based Classification. Qween Mary University of London. Technical Report.

NetworkMiner, 2020 (cited January 2020). https://www.netresec.com/?page=NetworkMiner.

ngrep, 2020 (cited January 2020). https://github.com/jpr5/ngrep.

Osherove, R., 2015. The Art of Unit Testing. MITP-Verlags GmbH & Co. KG.

Pcap-Ng, 2020 (cited January 2020). https://github.com/pcapng/pcapng/.

Pilli, E.S., Joshi, R.C., Niyogi, R., 2010. Network forensic frameworks: survey and research challenges. Digit. Invest. 7, 14–27.

Pluskal, J., Lichtner, O., Rysavy, O., 2018. Traffic classification and application identification in network forensics. In: Peterson, G., Shenoi, S. (Eds.), Advances in Digital Forensics XIV. Springer International Publishing, Cham, pp. 161–181.

Pluskal, J., Matoušek, P., Ryšavý, O., Kmeť, M., Veselý, V., Karpíšek, F., Vymlátil, M., 2015. Netfox detective: a tool for advanced network forensics analysis. Proceedings of Security and Protection of Information (SPI) 2015. University of Defence in Brno, pp. 147–163.

PyFlag, 2020 (cited January 2020). https://github.com/py4n6/pyflag.

Rescorla, E., 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC, p. 8446.

Scott, B., Neil, T., 2009. Designing Web Interfaces: Principles and Patterns for Rich Interactions. O'Reilly Media, Inc.

Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A., 2019. An Evaluation Framework for Network Security Visualizations. Computers & Security.

Sira, R., 2003. Network forensics analysis tools: an overview of an emerging technology. GSEC 1, 1–10 version.

Spiekermann, D., Keller, J., Eggendorfer, T., 2017. Network forensic investigation in openflow networks with forcon. Digit. Invest. 20, S66–S74. DFRWS 2017 Europe.

ssldump, 2020 (cited January 2020). http://ssldump.sourceforge.net.

SSLSplit, 2020 (cited January 2020). https://www.roe.ch/SSLsplit.

TCPDUMP, 2020 (cited January 2020). https://www.tcpdump.org/.

TCPFlow, 2020 (cited January 2020). https://github.com/simsong/tcpflow.

tcpxtract, 2020 (cited January 2020). http://tcpxtract.sourceforge.net/.

The Sleuth Kit (TSK) & Autopsy, 2020 (cited January 2020). https://www.sleuthkit.org/.

TShark, 2020 (cited January 2020). https://www.wireshark.org/docs/man-pages/tshark.html.

Wireshark, 2020 (cited January 2020). https://www.wireshark.org/.

XPlico, 2020 (cited January 2020). https://www.xplico.org/.

A.2 Network Forensic Investigations of Tunneled Traffic: A Case Study

Jan Pluskal, Michal Koutenský, Martin Vondráček, and Ondřej Ryšavý. “Network Forensic Investigations of Tunneled Traffic: A Case Study”. In: *Revue roumaine des sciences techniques. Série Électrotechnique et Énergétique* 64.4 (2019), pp. 429–434. ISSN: 0035-4066

Rev. Roum. Sci. Techn.– Électrotechn. et Énerg.
Vol. **64**, *4*, pp. 429–434, Bucarest, 2019

NETWORK FORENSIC INVESTIGATIONS OF TUNNELED TRAFFIC: A CASE STUDY

JAN PLUSKAL[1], MICHAL KOUTENSKÝ[1], MARTIN VONDRÁČEK[1], ONDŘEJ RYŠAVÝ[1]

Key words: Network traffic forensics, Generic stream encapsulation, Network forensic and analysis tool.

The increasing importance of network forensics in the investigations conducted by Law Enforcement Agencies is indisputable. Today's Internet does not carry ordinary TCP/IP traffic but utilizes many other encapsulations and tunneling protocols. In this paper, we overview the most used tunneling protocols and their features concerning digital forensic analysis. A case study of generic stream encapsulation describes how the investigator can obtain encapsulated application data from within.

1. INTRODUCTION

Internet applications use different communication protocols to exchange content. Most of network forensic analysis tools can correctly identify the communicating application and extract the content communication if encryption is not used. However, encryption is not the only obstacle for network forensic tools. Application communication can be also encapsulated in other protocols that provide an extra network layer in addition to the Internet's TCP/IP stack. These tunneling protocols are supposed to protect the encapsulated communication. It may be because the carried protocol is not compatible with the transport network technology or the additional security is necessary. Tunneling protocols are the basis for building virtual private networks. The local traffic needs to be sent over the Internet, which opens various possibilities for attackers. By using tunneling protocols, it is possible to protect the encapsulated communication with strong encryption to avoid eavesdropping and communication alteration. However, this benefit of network security represents a challenge for network forensics.

This publication extends the original paper "Network forensics in generic stream encapsulation (GSE) overlay networks" published in In 6th Conference on the Engineering of Computer Based Systems (ECBS '19), September 2–3, 2019, Bucharest, Romania [1].

1.1. PROBLEM DESCRIPTION

Network data acquisition faces many challenges. One of the complications for evidence recovery from captured network data is the use of encryption and tunneling. When end-to-end encryption was used the content of messages is protected but it is still possible to identify individual connections. In the case of tunneling protocols, multiple connections are multiplexed in the tunnel. The original design goal of tunneling protocols was to interconnect networks through possible incompatible network technology. The captured content of the tunnel can be easily extracted, and individual connections recovered. However, modern tunneling protocols include security measures by applying encryption to transferred content. Therefore, connections can only be recovered at exit points of the tunnel.

1.2. CONTRIBUTION AND PAPER STRUCTURE

The present paper provides an overview the common points in the network topology that can be used by law enforcement agencies (LEA) to conduct lawful interception. We provide a summary of most used tunneling protocols and discuss their features with respect to digital forensic analysis. For each protocol, the possibility of content extraction is explained. Also, a brief overview of methods for the classification of encapsulated traffic is provided. The issue of connection recovery from tunneled communication is demonstrated using the GSE protocol as an example.

2. LAWFUL INTERCEPTION POINTS IN NETWORK TOPOLOGY

The goal of lawful network data acquisition is to collect enough information for evidence extraction. As most of the Internet traffic is encrypted, the analysis of metadata represents the most important approach. There are many possible locations in the network topology that may be used for lawful interception and their selection depends on various circumstances. This section describes the locations and adequate techniques used to collect digital evidence out of network devices.

The end-user machine is the place where any kind of data is presented to the user, or stored. If encryption is used to protect data in transfer, this is the place where it happens. If the device can be accessed by an investigator, several techniques for obtaining the evidence via the installation of agent software that can intercept API hooks [2], capture network traffic [3], capture screen [4] or maliciously modify [5].

Internet service provider (ISP) The most typical lawful interception probe deployment occurs in the ISP network [6]. The LEA possessing a search warrant can [7] compel the ISP to reveal the retained data [6] or to intercept the suspect's communication [8] using LEA's deployed network probes [6]. Technically, there are several types of interception or traffic manipulation that can be done.

Network layer defines a physical connection between devices connected to a shared segment identified by MAC addresses that are resolved by ARP protocol. ARP can be misused to redirect the communication to an interception device [9], but it can also be error-prone [10]. The common encapsulation and tunneling protocols are VLAN, L2TP described in Section 3.

Internet layer The majority of traffic interception probes assume that traffic is redirected into them. Interception rules that are typically based on the IP address of the

[1] Brno University of Technology, Faculty of Information Technology, Božetěchova 2, 61266 Brno, Czech Republic,
E-mails: {ipluskal, koutenmi, ivondracek, rysavy}@fit.vutbr.cz.

suspect, defines which IP flows should be intercepted, *i.e.*, captured for future analysis. Interception up to 1 Gbps speeds can be done on regular PCs without additional configuration. Speeds up to 10 Gbps require that data are not copied between the kernel and userspace, *e.g.*, usage pf_ring [11], or n2disk [11]. Speeds past the 10 Gbps [12] requires custom kernel optimizations, *e.g.*, pf_ring and CPU core to NIC queue mapping. Typical encapsulations are IPsec, PPTP, IPIP, 6in4 described in Section 3.

Transport & application layer On the transport layer, we may utilize "policy-based-routing" to define rules that describe communication we are interested in to capture, or redirect to capturing probe. Typical encapsulation protocols are GRE, SSTP, Ayiya described in Section 3. On the application layer, we can go deeper and manipulate communication, *e.g.*, conduct SSL/TLS inspection, filtering, or capturing [13, 14].

Datacenter accommodates the complexity of network architecture to their size [15]. Smaller providers [16] use from common network design segmenting a network into smaller subsets on Internet layer. mid to large providers [17, 18], and cloud providers commonly use software defined networking (SDN) [19] to create virtual networks over well-designed base network layer. Customers can usually define network typologies dynamically as they create their visualized infrastructures [18]. All protocols described in Section 3 can be used.

3. ENCAPSULATION AND TUNNELING PROTOCOLS

The structure of the TCP/IP networking stack used on the Internet is quite rigid. There is a fixed number of layers, each providing certain functionality. This setup works fine for common scenarios, but occasionally the need to use a different configuration arises.

Encapsulation is a core concept for computer networks and is the basis for the layer model. As data moves downwards through the stack, from application to the physical medium, the contents get wrapped–encapsulated–at each layer in additional protocol information. When processing received data, each layer interprets its own information and forwards the encapsulated payload to the layer above.

Tunneling and encapsulation are likewise strongly related concepts. While common protocols encapsulate data of higher layer protocols, tunneling protocols may also encapsulate data of protocols of the same or lower layers.

Table 1

Summary of tunneling protocol features

Protocol	**Authentication**	**Encryption**
IPSec	Built-in	Built-in
GRE	No	No
PPTP	Using PPP	Using PPP
L2TP	Using PPP	Using PPP
SSTP	Using SSL	Using SSL
IPIP	No	No
6in4	No	No
Ayiya	No	No

This effectively allows extending the stack, repeating some layers multiple times, and can be considered a form of recursion.

Common use-cases for tunneling include transporting data over network segments with an unsupported network or data-link layer protocols or providing the illusion of being connected to a remote LAN via VPN.

3.1. COMMON TUNNELING PROTOCOLS

There exist a number of tunneling protocols varying in their application and scope. Some have very narrowly defined capabilities while others attempt to be general and extensible, see Table 1 for comparison.

IPsec is a suite of protocols that work with the IP family to provide confidentiality and integrity of transmitted data [20]. While not strictly a tunneling protocol, it can operate in a tunneling mode where the secured IP packet is encapsulated in a new packet. The operation of IPsec roughly consists of three components: security association (SA), authentication header (AH) [21] and encapsulating security payload (ESP) [22]. When a party is interested in communicating securely, it negotiates a SA which holds the necessary cryptographic parameters. Afterward, the communicating parties can include AH in their packets, which can be used to verify the integrity of the received data. AH achieves this by computing a hash from the fields of the IP header as well as the included payload and the SA. It is the last property that differentiates AH from a basic checksum and protects the data from being modified in transit. As AH protects parts of the IP header in addition to the payload, it also provides a form of authentication. The ESP can provide integrity as well as confidentiality using encryption. In transport mode, ESP only encrypts the payload; in the aforementioned tunneling mode, ESP encrypts both the IP header and the payload and encapsulates them in a new IP header.

GRE is an encapsulation protocol developed by Cisco to allow for encapsulation of link and network protocols in a generic way [23]. The protocol itself is very simple and provides no security features such as encryption or authentication. The payload packet is encapsulated in the GRE header, which is then encapsulated in the delivery protocol. The GRE header contains a protocol number identifying the encapsulated protocol. Additionally, a checksum might be present. Earlier RFCs included several other fields that specified, *e.g.*, the number of allowed recursions of encapsulation or routing information [23]. Their use has been deprecated [24].

PPTP is a tunneling protocol originally designed to carry PPP traffic over IP networks [25]. It operates on the link layer and uses a client/server model, where the server is called the PPTP network server and the client PPTP access concentrator. For encapsulation of the payload, PPTP uses an enhanced version of GRE. Each PAC-PNS pair establishes a tunnel and a control connection which runs over TCP. This control connection is used to manage both the tunnel and any user sessions using it. PPTP uses security mechanisms from PPP for authentication and encryption; the most commonly known are Password Authentication Protocol and Challenge-Handshake Authentication Protocol.

L2TP aims to tunnel PPP packets in a way that is as transparent as possible [26]. It decouples the layer 2 and PPP endpoints, allowing them to exist at different devices connected by a packet-switched network. The overall design is reminiscent of that of PPTP. The two endpoints are called the L2TP Network Server and L2TP Access Concentrator, filling similar roles as their PPTP equivalents. These two endpoints establish a tunnel which consists of a control connection and zero or more sessions. The control channel is reliable, while the channel used for transmitting data messages is not. In IP networks, the transport protocol to carry the L2TP messages is UDP, which avoids the issues brought by stacking several TCP connections on top of each other. L2TP supports the CHAP-like tunnel authentication mechanism but provides no integrity or confidentiality, leveraging features provided by PPP instead. However, it is commonly used in combination with IPsec to encrypt the payload via ESP and/or AH.

SSTP tunnels PPP frames over SSL/TLS, using TCP as its transport protocol [27]. In this case, security is provided by SSL using encryption and integrity checking. The structure of the SSTP header is quite simple, with the only interesting field being the C flag. When set, the encapsulated payload contains an SSTP control message; otherwise the higher-layer protocol.

IPIP is a protocol meant to alter the normal routing process by encapsulating the IPv4 packet in another IPv4 packet and sending it to an intermediate node [28]. The entry point of the tunnel wraps the IPv4 packet in another IPv4 header destined to the tunnel endpoint. After traversing the tunnel, the inner IPv4 packet is decapsulated and processed normally, routed and forwarded to its true destination. The protocol is simple, using no additional headers, as it is limited to one type of outer-inner protocol; most of the complexity lies in rules on how to properly handle ICMP messages. On its own, it provides no additional security features over basic IPv4.

6in4 is a transition mechanism allowing IPv6 traffic to traverse networks with only IPv4 support [29]. A tunnel is established between two devices, and the IPv6 traffic is transported by encapsulating it in IPv4. A special IP protocol number is defined for this purpose. 6in4 itself provides no security-related features such as authentication or integrity.

Ayiya attempts to solve some of the issues that transition protocols such as 6in4 have with establishing tunnels that travel through NATs. [30] These NATs need to be manually reconfigured to properly handle 6in4, which in some cases is not possible. Ayiya solves this by tunneling IP traffic not directly over IP, as is the case with 6in4 or IPIP, but over a transport layer protocol such as TCP or UDP. It aims to be general, independent of both the payload protocol and the transport protocol being used, thus the name Anything in Anything. It is even possible to tunnel the payload protocol directly over the network protocol, in the vein of IPIP, for IP over IP tunnels with minimal overhead where possible. Ayiya defines a custom header that is placed between the payload and the delivery protocol. The header contains an identity field to help determine which sender the packet has originated from, as the source port number and IP address may change arbitrarily during the connection, due to NAT, DHCP, IPv6 privacy extensions *etc.* An operation code field may specify special handling of the received packet, such as echoing it back to the sender. In addition, it contains an optional signature and authentication, providing some security features out of the box. A heartbeat message is used to keep the tunnel open, as not receiving any packets for a certain period of time results in closing the tunnel.

3.2. IDENTIFICATION OF ENCAPSULATION PROTOCOLS

To properly parse a protocol and extract information from it, it is necessary to correctly identify it. As there is no field identifying the application protocol in common transport protocol headers (TCP, UDP), and port numbers alone aren't sufficient to identify the protocol being used, several approaches have been developed to solve this issue.

Deep packet inspection (DPI) is a content-based method that attempts to identify protocols by looking inside the payload [31]. It looks for known signatures in the transmitted data to identify the data flow as a particular protocol. The signature matching process can be as simple as looking for a value in the first few bytes of the payload (application header) or complex heuristics requiring access to whole flows. DPI achieves high accuracy; the chief downside of this approach is that it needs to be able to access the data being transmitted to function properly, and it needs to inspect every packet passing through the interface. If the application uses encryption, DPI fails to provide meaningful results.

Connection patterns can be used to classify traffic into categories without inspecting the payload [32]. Sequences of flows are matched against heuristics using a set of rules. As different types of traffic (such as web or P2P) display different connection patterns over time, this information is sufficient to categorize the observed flows. While significantly simpler and less computationally intensive than DPI, this method only achieves rough categorization; it does not identify specific protocols.

Statistical methods are based on flow properties such as duration, packet size or arrival times [33]. Measurements of various protocol attributes are taken, and these are compared to existing models. It is possible to include some DPI attributes and treat them as statistical properties, resulting in a hybrid approach. Creating models by extracting fingerprints can be done manually; however, this is a very time-consuming process. Available algorithms, therefore, try to automate the process of creating new protocol models, requiring only pre-classified training data instead, utilizing machine learning [34].

As tunneling protocols work above the network or transport layer, these approaches can be used to detect encapsulated traffic as well. Few of the protocols described in the previous section provide encryption by themselves, and most offer some kind of signature available in the header that can be matched. Moreover, the accuracy of identification is of high priority, as we don't want to simply categorize the traffic to gather statistics but identify the encapsulated traffic as well. For this we need to correctly identify the protocol being used; DPI, therefore, appears to be a reasonable choice for encapsulation identification. The problem is further complicated by the possibility of IPsec being used to secure the tunneled traffic independent of the protocol being used; this is, in fact, the recommended approach by L2TP [26]. Additionally, tunneling protocols can tunnel other tunneling protocols, recursively extending the number of layers; to properly extract the application data, it is necessary to identify and decapsulate each of those protocols in turn properly.

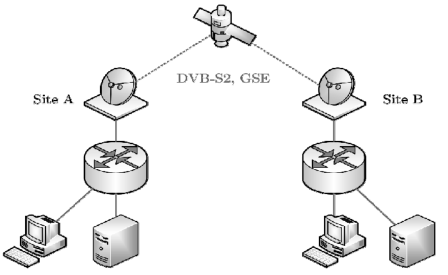


Fig. 1 – This example scenario is presenting a professional application of DVB-S2 and GSE. This architecture offers point-to-point or point-to-multipoint connections over a satellite link in both directions. Traffic between site A and site B is carried using generic stream encapsulation. The figure is based on the GSE implementation guidelines [37].

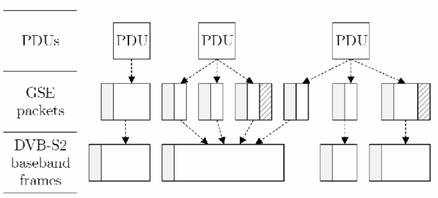


Fig. 2 – The figure shows the encapsulation of network layer PDUs into GSE packets and transmission of GSE packets inside physical layer baseband frames. GSE packets and baseband frames consist of a header (shown as a grey block) and a data field (shown as white space). GSE packet carrying the last fragment also contains CRC-32 (shown as a block with pattern). The figure is based on GSE protocol specification [35, p. 10].

4. GENERIC STREAM ENCAPSULATION (GSE) CASE STUDY

Network protocol generic stream encapsulation (GSE) was defined by the digital video broadcasting project (DVB), and it offers a way to transport IP traffic over a generic physical layer, usually over DVB physical infrastructure [35, p. 6]. GSE, as a native IP encapsulation protocol on DVB bearers, was introduced with the second-generation satellite transmission system called DVB-S2 (Fig. 1). Generic data transmission on the first generation of DVB standards was formerly possible using the multi-protocol encapsulation (MPE) on MPEGTS packets. However, MPE suffered significant overhead. GSE is also included in the Satlabs System Recommendations for DVB-RCS terminals [36].

Outline of GSE procedures operation of GSE allows transmission of variable size generic data encapsulated into baseband frames. GSE can encapsulate not only IPv4 traffic but a wide range of other protocols including IPv6, Ethernet, ATM, MPEG, and others. It supports addressing using 6-Byte MAC addresses, 3-Byte addresses, and even a MAC address-less mode [35, p. 6]. Encapsulation and decapsulation procedures performed by the DVB broadcast bearers are transparent to the rest of the network topology and the carried traffic. Shall a network layer PDU be transmitted over a satellite connection, GSE packets serve as a data link layer (Fig. 1).

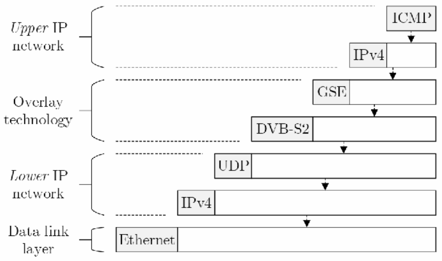


Fig. 3 – Example of IP traffic encapsulated in the GSE layer, which is carried by another IP traffic. The resulting virtual topology can be characterized as an established overlay network.

This GSE layer provides encapsulation, fragmentation, and slicing. Created GSE packets are then carried in baseband frames, *e.g.*, DVB-S2, on the physical layer (Fig. 2). The receiving side performs a reassembly process, integrity check, and a final decapsulation of transmitted PDUs [38].

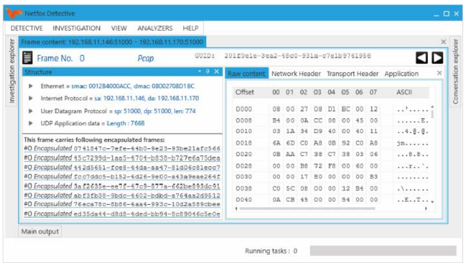
Moreover, it is also possible to transport GSE packets over, for example, standard IP network infrastructure. In this case, the DVB-S2 traffic can be carried like a generic payload on the application layer with the use of the UDP as a transport layer. Therefore, given UDP datagrams carry DVB-S2 baseband frames, which further carry GSE packets encapsulating selected protocol communication. This approach effectively establishes an overlay network infrastructure, because IP traffic can practically carry GSE packets, which can carry another layer of IP traffic. At this point, the UDP/IP layer below GSE can be considered the carrier *(encapsulating) traffic* whereas, for example, the IP layer above GSE can be described as the carried (*encapsulated*) traffic. This approach is presented in Fig. 3.

According to specifications and recommendations published by SatLabs, the implementation of a receiver with an Ethernet interface can be divided into a demodulation/decoding device, and a device focused on baseband processing. In such a case, the *L3 Mode Adaptation Receiver Header* can be prepended to received data [39, p. 10]. The receiving device would then process *DVB-S2 L3 Mode Adaptation Receiver Header*, *DVBS2 baseband frame*, and *GSE packets* to analyze transmitted communication.

Fragmentation, slicing, padding and reassembly process As noted earlier, GSE procedures can encapsulate different protocol data units in one or more GSE packets. In general, GSE packets have variable lengths, and they can be sent in different baseband frames individually or in a group. Therefore, fragmentation, slicing, padding and reassembling can occur. In this context, fragmentation refers to a situation when a PDU and extension header is fragmented into multiple GSE packets (Fig. 2). Slicing indicates a case when a GSE packet itself is divided into several contiguous baseband frames [35, p. 8]. Noted slicing, therefore, refers to physical layer fragmentation, which shall be transparent to the GSE layer [37, p. 27]. Concerning DVB-S2 applications, GSE slicing does not occur [37, p. 31].

Shall a single PDU be fragmented into several GSE packets, each packet is assigned a *fragmentation identifier (Frag ID)* label in the GSE header [35, p. 17]. Frag ID is used to match fragments belonging to the same original

PDU. This approach enables the simultaneous transmission of fragments from up to 256 different original PDUs. GSE packets carrying a complete PDU and GSE packets with PDU fragments can be distinguished using start and end flags in the GSE header. The protocol of carried PDU is indicated by protocol type/extension field in the GSE header of the first fragmented packet and every not fragmented packet. The packet with the last PDU fragment further carries a CRC-32 field used to check integrity after the reassembly process (Fig. 2). It is important to note that for example, DVB-S2 allows multiplexed transmission of multiple streams, each identified by its *input stream identifier (ISI)* [37, p. 32] in baseband header [40, p. 20]. The reassembly process has to be carried out independently for each received stream [35, p. 21]. Some of the possible GSE packet formats are presented in the technical specification [35, pp. 31–32].



Frame content: 192.168.11.146:51000 - 192.168.11.170:51000

Frame No. 0 Pcap GUID: 201f0e1e-3ea2-48c0-931a-c7e1b9761956

- Ethernet = smac: 0012B4000ACC, dmac: 08002708D1BC
- Internet Protocol = sa: 192.168.11.146, da: 192.168.11.170
- User Datagram Protocol = sp: 51000, dp: 51000, len: 774
- UDP Application data = Length : 766B

This frame carries folllowing encapsulated frames:
#0 *Encapsulated* 0741847c-7efe-44b0-9e23-93be21afc566
#0 *Encapsulated* 45c7299d-1aa5-4704-b838-b727e6a75dea
#0 *Encapsulated* 442d5651-fce8-44da-aa47-81d06c81ecc7
#0 *Encapsulated* fcc7ddc5-b152-4d26-9e00-a43a9eae264f
#0 *Encapsulated* 3af2635e-ee7f-47c9-877a-662be893dc91
#0 *Encapsulated* abf3fb38-9bdc-4602-bdbd-a764aa2d9512
#0 *Encapsulated* 76eca78c-8b86-4aa4-993c-10d2a589cbee
#0 *Encapsulated* ed35da44-d8d8-4ded-bb94-8c89046c5e0e

```
Offset  00 01 02 03 04 05 06 07   ASCII
0000    08 00 27 08 D1 BC 00 12   ..'.....
0008    B4 00 0A CC 08 00 45 00   ......E.
0010    03 1A 34 D9 40 00 40 11   ..4.@.@.
0018    6A 6D C0 A8 0B 92 C0 A8   jm......
0020    0B AA C7 38 C7 38 03 06   ...8.8..
0028    00 00 B8 72 F8 00 60 00   ...r..`.
0030    00 00 17 80 00 00 00 B3   ........
0038    C0 5C 08 00 00 12 B4 00   .\......
0040    0A CB 45 00 00 54 00 00   ..E..T..
```

Running tasks : 0

Fig. 4 – View of the frame content of the Netfox Detective presenting a frame carrying eight other encapsulated frames. It is possible to navigate between encapsulated frames using shown links labeled with GUID of the target frame.

Concerning GSE addressing modes noted earlier, an additional fourth mode called *label re-use* can be used when multiple GSE packets are carried in a single baseband frame. Shall label re-use be indicated, current GSE packet without address belongs to the same address as the last previously processed GSE packet. A more detailed analysis of GSE protocol is beyond this paper's scope. GSE packet format is defined in the protocol specification [35, p. 12]. Further information can be found in standards, recommendations, and guidelines covering GSE and DVB-S2 [35, 41, 42, 37, 43].

4.1. EVALUATION

Every layer of decapsulated traffic is subject to further network forensic analysis performed by the Netfox Detective[1]. The information is presented in the GUI. The view informs the user whether the current frame in encapsulated or not. It is also possible to navigate between views showing individual encapsulating frames (Fig. 4) and encapsulated frames. The implementation has been evaluated on publicly available dataset [2], and results (amount of correctly identified and extracted GSE communications) were comparable to the reference Wireshark implementation. A set of integration tests was implemented that verify the correct processing of GSE traffic in future releases and prohibit regression bugs from being introduced.

5. **CONCLUSIONS**

Network forensic analysis currently faces many challengesthat stems from the fact that most of the Internet traffic is encrypted. Thus, the analysis relies on the metadata of messages and the behavioral characteristics of the communication. In this paper, we have considered another issue for network forensics, namely, the use of tunneling protocols. We have identified the problem that tunneling represents for evidence extraction. Then we have presented an overview of different existing tunneling protocols and their characteristics with respect to digital forensics. Finally, we have demonstrated the case study using the GSE protocol, which allows transporting IP traffic via satellite connections. The experimental GSE protocol analyzer implements the method for full content extraction. Thus it can be used to preprocess the data for network forensic analysis tools that are unable to directly cope with tunneled communication. If tunneling protocols apply encryption to protect the encapsulated traffic, the content extraction is not possible in general. However, several approaches were proposed for the detection of application class of encapsulated communication. The paper provides a brief overview. Their adaptation for different tunneling protocols belongs to the intentions of our future work.

Received on December 1, 2019

REFERENCES

1. J. Pluskal, M. Vondráček, O. Ryšavy, *Network ` Forensics in GSE Overlay Networks*, In: Proceedings of the 6th Conference on the Engineering of Computer Based Systems, ACM, 2019.
2. V. Lifliand, A.M. Ben-Menahem, *Encrypted network traffic interception and inspection*, US Patent 8,578,486, 2013.
3. L. Temoshenko, H. Nhan, C. M Parker, R. L King, J. D. Douglas, N. A Mitchell, G. R Everhart, D. W. Currie, *System and method for intercepting packets in a pipeline network processor*, US Patent 7,046,663., 2006.
4. S. Belikovetsky, O. HaCohen, N. Lauderdale, *Deception using screen capture*, US Patent 10,425,445, 2019.
5. D. Javaheri, M. Hosseinzadeh, A. M. Rahmani, *Detection and Elimination of Spyware and Ransomware by Intercepting Kernel-Level System Routines*, IEEE Access 6 , 78321– 78332 (2018).
6. A. Imbimbo, F. Attanasio, *Apparatuses, methods, and computer program products for data retention and lawful intercept for law enforcement agencies*, US Patent 9, 204,293, 2015.
7. M. Ponsford, *The Lawful Access Fallacy: Voluntary Warrantless Disclosures, Customer Privacy, and Government Requests for Subscriber Information*, Canadian Journal of Law & Technology **15**, *1* (2017).
8. A. Muñoz, M. Urueña Pascual, R. Aparicio Morenilla, G. Rodríguez de los Santos López, *Digital Wiretap Warrant: Improving the security of ETSI Lawful Interception* (2015).
9. A. Foss, *Method for automatic traffic interception*, US Patent 7,567,573, 2009.
10. P. Branch, Ana Pavlicic, G. Armitage, *Using MAC addresses in the lawful interception of IP traffic*, In: Proc Australian Telecommunications Networks & Applications Conference (ATNAC), pp. 9– 11, 2004.
11. L. Deri *et al.*, *Improving passive packet capture: Beyond device polling*, In: Proceedings of SANE, 2004. Amsterdam, Netherlands, pp. 85–93, 2004.
12. L. Deri, A. Cardigliano, F. Fusco, *10 Gbit line rate packet-to-disk using n2disk*, In: 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 441–446, 2013.
13. S. Aryan, H. Aryan, J A. Halderman, *Internet censorship in Iran: A first look, Presented as part of the 3rd {USENIX}*, In: Workshop on Free and Open Communications on the Internet, 2013.
14. P. Winter, T. Pulls, J. Fuss, *ScrambleSuit: A Polymorph Network Protocol to Circumvent Censorship*. arXiv preprint arXiv:1305.3199 (2013).

[1] https://github.com/nesfit/NetfoxDetective
[2] https://wiki.wireshark.org/DVB-S2 (last accessed 2019-12-12).

15. D. Spiekermann, J. Keller, T. Eggendorfer, *Improving Lawful Interception in Virtual Datacenters*, In: Proceedings of the Central European Cybersecurity Conference, 2018.

16. L. Reith, *Method and system for lawful interception of packet switched network services*, US Patent 7,447,909, 2008.

17. J. Cartmell, A. Chandra, P. R Chitrapu, *Lawful interception for local selected IP traffic offload and local IP access performed at a non-core gateway*, US Patent 9,591,031, 2017.

18. G. Amato, *Lawful intercept management modules and methods for LI-configuration of an internal interception function in a cloud based network*, US Patent 9,866,435, 2018.

19. N. McKeown, *Software-defined networking*, INFOCOM keynote talk **17**, *2*, pp. 30–32 (2009).

20. Karen Seo, S. Kent, *Security Architecture for the Internet Protocol*, RFC 4301, 2005.

21. S. Kent, *IP Authentication Header*, RFC 4302, 2005.

22. S. Kent, *IP Encapsulating Security Payload (ESP)*, RFC 4303, 2005.

23. D. Farinacci, S.P. Hanks, T. Li, P.S. Traina. *1994. Generic Routing Encapsulation (GRE)*, RFC 1701.

24. T. Li, D. Farinacci, S.P. Hanks, D. Meyer, P.S. Traina, *Generic Routing Encapsulation (GRE)*, RFC 2784, 2000.

25. G. Zorn, G.-S. Pall, K. Hamzeh, *Point-toPoint Tunneling Protocol (PPTP)*, RFC 2637, 1999.

26. A.J. Valencia, G. Zorn, W. Palter, G.-S. Pall, M. Townsley, A. Rubens *Layer Two Tunneling Protocol "L2TP"*, RFC 2661, 1999.

27. Microsoft Corporation, *Secure Socket Tunneling Protocol (SSTP)* (18 ed.), 2018.

28. C. E. Perkins, *IP Encapsulation within IP*, RFC 2003, 1996..

29. R. E. Gilligan, E. Nordmark, *Basic Transition Mechanisms for IPv6 Hosts and Routers*, RFC 4213, 2005.

30. J. Massar, *AYIYA: Anything In Anything. Internet-Draft draft-massar-v6ops-ayiya-02. Internet Engineering Task Force*, https://datatracker.ietf.org/doc/html/draft-massar-v6ops-ayiya-02 Work in Progress, 2004.

31. A.W. Moore, K. Papagiannaki, *Toward the accurate identification of network applications*, In: Lecture Notes in Computer Science, **3431**, pp. 41–54, 2005.

32. W. John, S, Tafvelin, *Heuristics to classify Internet backbone traffic based on connection patterns*. 2008 International Conference on Information Networking, ICOIN (2008), pp. 1–5.

33. E. Hjelmvik, W. John, *Statistical protocol identification with spid: Preliminary results*, In: Swedish National Computer Networking Workshop, pp. 399–410, 2009.

34. S. Zander, T. Nguyen, G. Armitage, *Automated traffic classification and application identification using machine learning*. In: Proceedings - Conference on Local Computer Networks, LCN 2005, pp. 250–257, 2005.

35. ETSI, *ETSI TS 102 606-1 V1.2.1 - Digital Video Broadcasting (DVB); Generic Stream Encapsulation (GSE); Part 1: Protocol*, European Telecommunications Standards Institute, 2014.

36. SatLabs Group, *SatLabs System Recommendations Version 2.1.2*, 2010.

37. ETSI, *ETSI TS 102 771 V1.2.1 - Digital Video Broadcasting (DVB); Generic Stream Encapsulation (GSE) implementation guidelines*, European Telecommunications Standards Institute, 2011.

38. DVB Project Office, *DVB-GSE - Generic Stream Encapsulation. DVB Project Office*, https://www.dvb.org/resources/public/factsheets/dvb-gse_factsheet.pdf, 2015.

39. SatLabs Group, *Mode Adaptation Input and Output Interfaces for DVB-S2 equipment Version 1.3*, 2008.

40. ETSI, *ETSI EN 302 307-1 V1.4.1 - Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications. Part 1: DVB-S2*, European Telecommunications Standards Institute, 2014.

41. ETSI, *ETSI TS 102 606-2 V1.1.1 - Digital Video Broadcasting (DVB); Generic Stream Encapsulation (GSE); Part 2: Logical Link Control (LLC)*, European Telecommunications Standards Institute, 2014.

42. ETSI, *ETSI TS 102 606-3 V1.1.1 - Digital Video Broadcasting (DVB); Generic Stream Encapsulation (GSE); Part 3: Robust Header Compression (ROHC) for IP*, European Telecommunications Standards Institute, 2014.

43. ETSI, *ETSI TR 102 376-1 V1.2.1 - Digital Video Broadcasting (DVB); Implementation guidelines for the second generation system for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 1: DVB-S2; Part 1: DVB-S2*, European Telecommunications Standards Institute, 2015.

A.3 Network Forensics in GSE Overlay Networks

Jan Pluskal, Martin Vondráček, and Ondřej Ryšavý. “Network Forensics in GSE Overlay Networks”. In: *Proceedings of the 6th Conference on the Engineering of Computer Based Systems*. ACM. 2019. ISBN: 9781450376365

Network Forensics in GSE Overlay Networks

Jan Pluskal
Brno University of Technology
Brno, Czech Republic
ipluskal@fit.vutbr.cz

Martin Vondráček
Brno University of Technology
Brno, Czech Republic
xvondr20@stud.fit.vutbr.cz

Ondřej Ryšavý
Brno University of Technology
Brno, Czech Republic
rysavy@fit.vutbr.cz

Abstract

The importance of captured network traffic as a data-source for law enforcement crime investigation has increased because many devices are Internet-enabled and the data communication might yield crucial evidence for an investigation. There are many points in the Internet Service Provider's infrastructure where the network traffic might be captured. One of them is a satellite connection, DVB-S2, which use Generic Stream Encapsulation (GSE) protocol that carries IP traffic. Current tools for network traffic forensic analysis do not support GSE. In this paper, we describe principles of GSE, methods for GSE traffic analysis and the extension for an existing network forensic tool that performs GSE traffic processing and extraction of encapsulated communication.

CCS Concepts • **Applied computing** → **Network forensics**; • **Networks** → *Network monitoring*; *Network protocols*; Transport protocols; Application layer protocols; • **Social and professional topics** → Computer crime.

Keywords network traffic forensics, generic streaming encapsulation, network forensic and analysis tool

ACM Reference Format:
Jan Pluskal, Martin Vondráček, and Ondřej Ryšavý. 2019. Network Forensics in GSE Overlay Networks. In *6th Conference on the Engineering of Computer Based Systems (ECBS '19), September 2–3, 2019, Bucharest, Romania.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3352700.3352712

1 Introduction

The digital forensics is becoming a domain of skilled operatives employed in Law Enforcement Agencies (LEA) that are tasked to investigate crimes. Their data-sources might vary, like seized mobile phones, computers, or other storage devices. Several cases use a lawfully intercepted network traffic as a valued data-source [2].

Although the analysis of network communication was not considered the primary area of digital forensics, its importance has increased as most of the devices are Internet-enabled. Performing network forensic analysis requires adequate tool support [13, 14]. A typical network forensics analysis tool provides features that aid an investigator to reveal evidence in network communication [1]. Instead of providing network protocol details, the forensic tool is expected to extract contents of transmitted files, perform a keyword search, identify user credentials, and more [2, 19].

Many complex and functionally rich network analysis tools require expert knowledge of operators necessary to correctly pre-process the data to suit the tool. The field operatives are experienced criminal investigators but usually not computer experts. Therefore, tools they use need to be straight-forward, provide top-to-bottom analysis, and require as few expert knowledge as possible.

The overlay networks are becoming widely used by Internet Service Providers (ISPs) that are interconnecting various public places, businesses, campuses, or regular home internet connections. Technologies can be fiber-optic, metallic ethernet, 3G, 4G, 5G or satellite connection DVB-S2 that uses GSE to encapsulate IP traffic [6, 8–11].

Our motivation behind the implementation of GSE analyzer stems from the interest expressed by LEA investigators that seek a tool capable of analysis Internet communication encapsulated in various tunneling protocols. The officers prefer open-source network forensic and analysis tools (NFATs) [1, 12], even though they might be poorly documented, out-of-date, and even abandoned [13].

1.1 Problem Description

The GSE is nowadays commonly used for Internet traffic encapsulation in satellite networks. As its name suggests, it is a generic method of encapsulation and can occur on Data Link, or Application layer even recursively. The LEAs struggle to perform network forensics on data captured with GSE encapsulation, but because commonly used tools for network forensics do not process it, it is a difficult task.

1.2 Contribution and Paper Structure

This paper introduces the issues and methods of forensic analysis of the GSE protocol. In the next section, we list the most used Network Forensic Analysis Tools (NFAT) and

ECBS '19, September 2–3, 2019, Bucharest, Romania

ACM ISBN 978-1-4503-7636-5/19/09…$15.00
https://doi.org/10.1145/3352700.3352712

Network Security Monitoring (NSM) tools and their capabilities in processing tunneling traffic, in particular, GSE protocol. It is interesting that to our knowledge, none of the NFATs support GSE. Next, we provide a detailed description of Netfox Detective architecture, and atop of it, we describe the principles of GSE processing. The goal of the present work is to provide advanced information for network forensic practitioners that need to deal with GSE communication. We also implemented the GSE processing as an extension to our own NFAT making it available to the wider body of digital investigators.

2 Related Work

Network forensic practitioners commonly use two types of tools — the NSM and the NFAT [13]. This section mainly focuses on tunneling protocols support in related tools and their usability for network forensic investigation conducted by LEA officers.

NSM tools are intended for a high-level insight into the network communication. Such tools are usually fast and scalable; thus can process high volumes of network data on high-speed networks up to hundreds of gigabits per second. These tools provide information typically from lower layers, i.e., Internet and Transport, and only partially from Application, where they parse only well-known protocols; rarely they support overlay networks. Also, these tools are guided strictly by standards and usually do not include heuristics or more in-depth analysis to extract additional content. They operate online, and most cannot process malformed or incomplete communication. The incomplete communication is a typical case when interception is done on commodity hardware inside ISP infrastructure. Therefore, these tools are used mostly by network operators for measurements, accounting, and incident detection. NSM tools provide the bottom-up approach showing dissected packets and letting the investigator conduct expert analysis.

The most commonly known NSM tool is Wireshark [27] that supports the following encapsulation protocols: GSE, GRE, Ayiya, GTPv1, L2TP, SSTP, PPTP, IPIP, IPsec, 6in4, etc. It supports the broadest range of network and application protocols. Wireshark defines an API that can be used to extend its functionality by a new protocol dissector. Note that it is *the only tool supporting GSE*!

Some NSM tools can be integrated, and more sophisticated analysis can be done programmatically, like TShark [27], TCPDump [24], TCPFlow [26], NfDump [18], Suricata [23] (Teredo, GRE), Zeek [29] (Ayiya, Teredo, GTPv1, GRE), Moloch [16] (GRE) that can analyze live or intercepted communication. They can be parts of scripts that can do one or more tasks, but still can not be compared to NFAT carving and analytical capabilities.

NFAT Our focus is to provide a tool for LEA operatives to extract forensically important information mostly from the application layer of communication. This intent perfectly fits into the category of NFATs that is intended for in-depth traffic analysis, that is mainly performed *offline* on captured communication. NFATs provide the same amount of information as NSM tools but also add extra information extracted from the application layer. They conduct a thoughtful analysis of the traffic and use the extracted data to infer information that helps the investigator. The information is usually provided in a synoptic, easily navigable user interface because NFATs are intended to be used even by field operatives without specialized training.

Popular NFATs are NetworkMiner [17] (GRE, 802.1Q, PPPoE, VXLAN, OpenFlow, SOCKS, MPLS, and EoMPLS), PyFlag [3, 20], XPlico [28] (L2TP, VLAN, PPP), NetIntercept [5]. No NFAT supports GSE as far as we know.

3 Netfox Detective in Depths

In this section, we present Netfox Detective, a network analysis desktop application created for the Windows platform. We discuss the low-level network traffic processing parts to be able to explain the extension of GSE decapsulation support. The tool is composed of two parts:

- **Netfox Framework** (backend, details see Sec. 3.1) is network traffic processing engine that provides all kinds of functionality starting from capture file loading, going through traffic processing, extraction and ending with traffic analysis.
- **Netfox Detective** (frontend, details see Figs. 10, 11) is a visualization tool that depends on the backend for processing part but extending it with analytic capabilities to interpret extracted data.

For a high-level overview of the tool architecture see Fig. 1. Note, Netfox Framework is a separate set of .NET assemblies that have no dependency on Netfox Detective and can operate separately. However, the framework does not have any CLI and therefore has to be incorporated into an application. On the other hand, Netfox Detective has a direct dependency on the Netfox Framework and is compiled with it, e.g., it uses types that are defined in Netfox Framework.

3.1 Netfox Framework

Netfox Framework is the backend, and it is responsible for parsing and preparing all information gathered. For instance, it identifies used protocols, to overcome fragmentation (L3) and segmentation (L4). In its current version, it does not support live capture but can process standard input file formats such: *libPCAP, Microsoft Network Monitor cap, and PCAP-ng.*

Link Layer Once an input file is loaded, it is processed frame by frame (L2). The lowest used protocols type (e.g., LINKTYPE_ETHERNET (IEEE 802.3), LINKTYPE_IEEE802_11

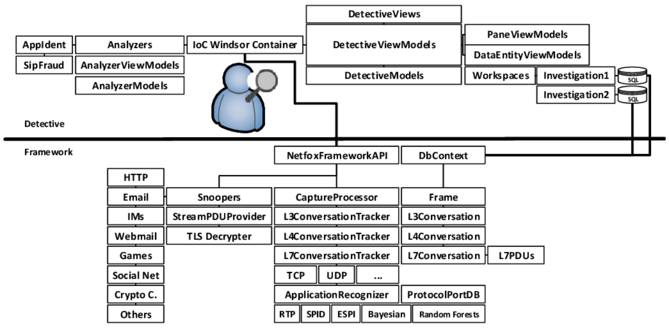


Figure 1. The figure describes the abstraction of Netfox Detective and Netfox Framework architecture. The upper part of the diagram above the line represents visual parts of the tool. Below the line, components of Netfox Framework are drawn in a hierarchical view.

(IEEE 802.11), LINKTYPE_PPP, etc.) is stored in the 'pcap_file_header' structure, and we use it to load the first protocol parser. A good overview of the Link-Layer header type values is provided by [25].

Next, we utilize the frame header and its Logical Link Controller header (LLC) where the main field is a unique identifier of the L3 protocol (e.g., IPv4, IPv6).

Notice that sometimes it might not be stored in the capture file. Link layer usually does not carry any forensically significant information; thus it is generally omitted and *LINKTYPE_RAW, LINKTYPE_NULL* link layer types are used.

Internet Layer Similarly, both IPv4 and IPv6 contain an identification of an upper layer. (Note, IPv4 names the field 'protocol'; IPv6 names it 'Next Header') which allows us to choose an appropriate L4 parser. As long as the protocol/next header is present, we can parse the communication deterministically, usually up-to the transport layer.

Transport Layer The transport layer carries no information about the subsequent protocol; therefore, the continuing application layer needs to be identified by other means to be correctly processed. We can do this identification using several methods (e.g., port-based classification, deep-packet inspection, probabilistic and statistical methods based on machine learning). Typical encapsulation with protocol examples is presented in Fig. 2.

3.2 Conversation Tracking

This section provides a comparison of ISO/OSI and TCP/IP models with denoted layer names and samples of typical protocols used on particular layers. The logical approach to process network data is to create a *forest of trees* with roots based on identifiers extracted from the lowest layer of the network encapsulation model and continue with upper encapsulation levels. This way, conversations on all levels are created, which also sets boundaries, and specific traffic can be targeted for analysis and information extraction.

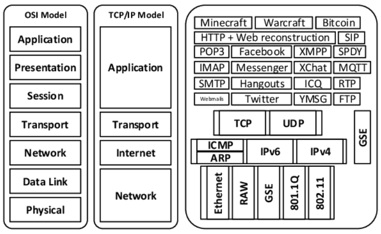


Figure 2. This figure provides the comparison of ISO/OSI and TCP/IP models with denoted layer names and samples of typical protocols used on particular layers. Netfox Detective supports all protocols that are enumerated on this figure.

Besides, each layer has its specifics that need to be taken into account before processing ongoing layer.

IPv4 (L3) fragmentation can occur, and packets need to be defragmented before further processing. Fragments are identified by *Fragment Offset* and bit *More Fragments* (MF) set in the *Flags* field. As long as MF bit is set, defragmentation process has to buffer packets and further process them in bulk, because fragments do not carry headers from upper layers, thus cannot be processed separately and in parallel.

TCP (L4) segmentation occurs regularly. Segments are agnostic to processing mechanisms, carry all required headers and can be processed in parallel. The position of a segment in transmission buffer is defined by the difference of initial sequence number (SYN packet's SEQ) and the particular segment's SEQ.

Application messages are not implicitly denoted because each application protocol has its structure and is not parsed on this level of processing. To obtain at least some level of abstraction, we can deduce boundaries of application messages from the transport layer. E.g., TCP's field *Flags* contains the *PSH* bit that is set when the last segment of a particular application message is created. In other words, when *flush()* is called on network socket which is typically done to notify the kernel that message is to be dispatch right away.

Our unique mechanism of processing network communication [15], mainly L4 segregation shown that even malformed or corrupted captures could be used as data-source and carving modules can extract otherwise lost information. We accomplish this during the last processing step, that creates *L7PDUs*, which are the approximations of application messages.

3.3 Netfox Detective Architecture

Netfox Detective was designed to be modular and modules to be inter-operable, but also to work as self-contained libraries to be used by other tools. This way, we have created a framework for network forensics and analytic application supporting the forensic investigation.

Fig. 1 describes the decomposition of the tool to small interconnected building blocks/modules. In the bottom part, the architecture of Netfox Framework processing network communication that is interconnected with Netfox Detective by *NetfoxFrameworkAPI*. This API enables easy incorporation of Netfox Framework with any additional software that may use it as a platform. Furthermore, this part is divided into two groups, the *execution* and *model* parts.

Execution part, on the left-bottom side of *NetfoxFrameworkAPI*, consists of modules that by their composition ensures polymorphic behavior and extensibility. Each new networking protocol that is to be supported requires the creation of its tracking building block and connection into the processing pipeline. The communication interface between building blocks is defined by their interfaces that buffer inputs and outputs that encapsulates data in models.

Model part consists of blocks below *DbContext*. Models serve as data carriers for parsed, extracted state information, e.g., for L3 conversation it is the *source and destination IP address* with a collection of other models representing *Frames*. Models are persisted with *DbContext* and also accessible through it to higher layers.

To ensure fast parallel processing on a single computation node with shared memory, i.e., an application running a single process, we used *Task Parallel Library* (TPL). This approach enables the creation of functional blocks that improve modularity. Each block processes immutable data; thus, all blocks might run in parallel and together create an oriented graph, a Data Flow[1]. The Netfox Framework combines buffering blocks that interconnect execution blocks to maximize the utilization of resources due to different time complexities of data processing in the functional blocks. Also, this introduces a back-pressure mechanism that is used as memory management to slow down faster blocks that might otherwise overwhelm the system and cause resource depletion and consequently, a disk swapping or an application crash.

3.4 Capture File Processing

In Netfox Framework, capture file processing is initiated by a method call of *AddCapture* in *NetfoxFrameworkAPI*. In the current implementation, the tool processes captured traffic in formats *libPCAP*, *PCAP-ng* and *MNM Cap* (Microsoft Network Monitor). Fig. 3 describes a sequence of execution calls and model passing through execution pipeline, a layer by layer to describe logical processing in an abstracted manner.

Modules are designed to ensure concurrent processing thus they do process immutable data only. Majority of modules also do run in parallel instances to increase a degree of parallelism further. This design also enables with some modifications of processing pipeline to scale up and run the data flow graph in a distributed environment. That is achieved with TPL Data Flow which also enables to change interconnection of execution block to extend the processing of capabilities to process new network encapsulations (tunneling protocols).

The rest of this section describes processing blocks and their interconnections denoted on Fig. 4.

ControllerCaptureProcessor

ControllerCaptureProcessor block is used to oversee captured traffic processing. This module interconnects particular functional and buffering block to a processing pipeline reflecting typical network layered encapsulation. A new processing data flow pipeline is created for each job. That leads to segregation of data potentially originated from multiple cases and guarantees that no data might be reconstructed into false evidence. The processing has two reading phases.

Firstly, a path to file or files with captured communication is passed to the *CaptureProcessorBlock* that takes care of parsing of particular PCAP file format and retrieving raw frames. The output of this block is *PmCapture* object collection meta information about the capture file and frames encapsulated in objects of *PmFrame*. PmFrame is obtained in the sequential streamed one-way passage of capture file and contains only information about its position in the capture file.

Secondly, additional meta information used in further processing without actual payload is filled in the second read by *IndexMetaFramesBlock*. This segregation is due to a way how frames are stored in various PCAP file formats. Some

[1] https://msdn.microsoft.com/cs-cz/library/hh228603(v=vs.110).aspx

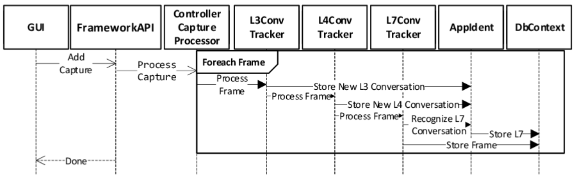


Figure 3. Abstract capture file processing scheme with a sequential passage. Data dependencies between models are omitted. New conversations are stored in relational database triggered by the processing of a first frame belonging to it.

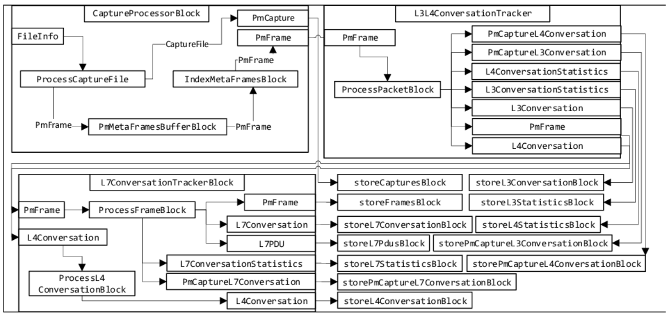


Figure 4. The figure describes the scheme of the functional and buffering block based on TPL Data Flow. This schema describes the decomposition of processing units to perform actions like reading frames from capture files, tracking conversations on L3, L4 levels and furthermore on L7 application layer with the approximation of application messages and application protocol identification.

formats (e.g., MNM) contains a frame table with this meta-information in place and spares the first PCAP read. Execution of *IndexMetaFramesBlock* block, which is a non-blocking read from PCAP file with parsing of (L2), L3, L4 layers, is done with the maximal level of parallelism. Layer 2 might be omitted in case that PCAP is captured without it.

L3L4ConversationTracker

L3L4ConversationTracker takes care of the creation of conversations on particular levels inside the *ProcessPacketBlock*. A PmFrame(s) (packets) with the same IP source and destination address compose an *L3Conversation.* This L4 conversation if furthermore a collection of smaller L4 conversations that composes PmFrame(s) (datagrams) with the same IP source and destination address and TCP or UDP source and destination ports and L4 protocol type (i.e., UDP or TCP).

In the time when conversations on layer L3 and L4 are created, meta-information in the form of PmFrames is still kept in memory. Because of that, complementary to the conversation creation, conversation statistics are generated as well. Statistics on both levels are updated by data processed from each PmFrame passing through *ProcessPacketBlock.*

Because the processing model in Netfox Framework is based on IP communication, all non-IP communication is tracked in special aggregation conversations. These conversations have invalid IP addresses as identifiers, i.e., *0.0.0.0* and *[::]* on L3 level, and invalid endpoints on L4, i.e., *0.0.0.0:0* and *[::]:0* as both source and destination. Similarly, L3 conversations containing an unknown transport protocol are aggregated into first L4 conversation with valid IP addresses but invalid transport ports, i.e., 0 port number.

L7ConversationTracker

L7ConversationTracker is a core of our reassembling engine currently supporting TCP and UDP transport protocols. Various TCP heuristics [15] are used to separated IP flow communication, i.e., L4 conversations to finer-grained units based on application session. We call them L7 conversations.

This module processes incoming datagrams in parallel respecting the following scheme. For each newly processed L4 conversation it creates a new Task and stores it into a dictionary keyed by an L4 conversation key. All consequently processed datagrams will be forwarded into this task. Tasks run in parallel on multiple cores and are scheduled by the TaskScheduler inside Common Language Runtime (CLR), which makes them much lighter than regular OS threads because they are running on existing threads stored in the ThreadPool. After a task is done or paused, the thread is returned into the ThreadPool, and a new task is immediately executed on it. This way, the overhead is minimal, and parallel processing improves performance rapidly.

Based on the transport protocol type, appropriate reassembler is selected, and the datagram is passed to it for the processing. Reassemblers incorporate heuristics [15] for advanced network traffic processing capable of accurate processing of even malformed, or missing frames.

UDP reassembler uses timeouts to separate consequential UDP sessions. Because of a lack of information from UDP protocol, application messages are created as an ordered sequence of *L7 PDUs*. Each L7 PDU contains only one datagram.

TCP reassembler is more complex and uses properties of TCP protocol like sequence numbers, flags (mainly SYN, FIN, RST, PSH) in combination with timeouts. Based on TCP properties, approximations of application messages are created in the form of the ordered sequence of *L7 PDUs*. Each L7 PDU contains one or more datagrams composing the application message.

TCP Reassembler This solves an issue with the ambiguity of L4 conversations captured in one or many simultaneously processed captures. Typically this happens when static ports are used at server and client side. In a case when a packet loss corrupts capture, it may happen that multiple TCP sessions would be merged into one because from a network point of view, communication would match the regular schema. A TCP finite state machine would process this merged communication and report missing data but would lack further information. That would result in ambiguity in determination who was communicating, whether there were one or more identities involved.

Both reassemblers (TCP and UDP) produce *L7 Conversations* that contain collections of *data* and *non-data* frames. Non-data frames are frames without payloads that serve for signaling purposes like TCP ACKs, or frames with payloads that are malformed, or retransmitted. These frames do not participate in final stream creation, but their presence is either way recorded for auxiliary forensic intents.

L7PDUs Data frames are stored inside L7 PDUs. One L7 PDU represents a data stream that is an approximation of an application message. An application message is considered to be a sequence of datagrams containing one user action, e.g., the user sends a message on online chat, or an email, or downloads a picture, etc. Although, one application message can span across multiple L7 PDUs, scarcely, one L7 PDU would contain multiple application messages. This also serves as a check-pointing mechanism in case that module extracting data from the application protocol is unable to parse the data stream due to corruption or unknown content correctly. We observed that this happens a lot when proprietary application protocols are involved because of their volatile nature and closed specification.

Storage Blocks

Storage blocks are used to assure asynchronous persistence of gathered meta-information in the form of outputs of all functional blocks, i.e., *L3, L4, L7 Conversations with statistics, L7 PDUs and Frames.* Data is stored in SQL database in bulk operations to achieve higher performance with a cost of delay introduced with buffering. Buffering and database storing operations run in separate tasks. This way, both services run in parallel and do not block one-another under ideal circumstances. Storage buffering is highly memory consumptive; therefore, in case that database is slower then processing, back-pressure mechanism protects processing pipeline against memory deprivation lowering its performance.

4 Decapsulation of Overlay Network Communication

Available network technologies provide ways to encapsulate various network protocols inside carrier traffic. This approach practically establishes an overlay network on top of an existing network infrastructure. The virtual topology of such an overlay network is usually different than the physical topology. Encapsulation methods can aim to maintain security *Confidentiality, Integrity, and Availability* (*CIA*) triad. As already explained, the goal of Netfox Detective is to offer an extensive forensic analysis of captured traffic. To fulfill this goal and provide a broader range of use-cases, our research and development further focused on the processing of encapsulated traffic. This section, therefore, outlines several encountered challenges and explains how the analysis of encapsulated satellite traffic was solved.

4.1 Generic Stream Encapsulation

Network protocol *Generic Stream Encapsulation* (*GSE*) was defined by the *Digital Video Broadcasting Project* (*DVB*), and it offers a way to transport IP traffic over generic physical layer, usually over DVB physical infrastructure [8, p. 6]. GSE, as a native IP encapsulation protocol on DVB bearers, was introduced with the second-generation satellite transmission system called DVB-S2 (Figure 5). Generic data transmission on the first generation of DVB standards was formerly possible using the *Multi-Protocol Encapsulation* (*MPE*) on MPEG-TS packets. However, MPE suffered significant overhead. GSE is also included in Satlabs System Recommendations for DVB-RCS terminals [22].

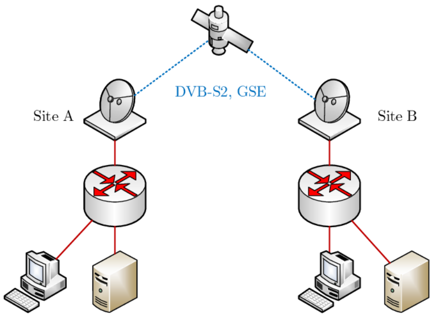


Figure 5. This example scenario is presenting a professional application of DVB-S2 and GSE. This architecture offers point-to-point or point-to-multipoint connections over a satellite link in both directions. Traffic between Site A and Site B is carried using Generic Stream Encapsulation. The figure is based on the GSE implementation guidelines [6].

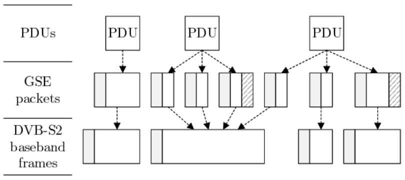


Figure 6. The figure shows the encapsulation of network layer PDUs into GSE packets and transmission of GSE packets inside physical layer baseband frames. GSE packets and baseband frames consist of a header (shown as a grey block) and a data field (shown as white space). GSE packet carrying the last fragment also contains CRC-32 (shown as a block with pattern). The figure is based on GSE protocol specification [8, p. 10].

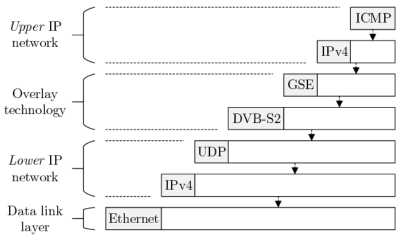


Figure 7. Example of IP traffic encapsulated in GSE layer, which is carried by another IP traffic. The resulting virtual topology can be characterized as an established overlay network.

Outline of GSE Procedures Operation of GSE allows transmission of variable size generic data encapsulated into baseband frames. GSE can encapsulate not only IPv4 traffic but a wide range of other protocols including IPv6, Ethernet, ATM, MPEG, and others. It supports addressing using 6-Byte MAC addresses, 3-Byte addresses, and even a MAC addressless mode [8, p. 6]. Encapsulation and decapsulation procedures performed by the DVB broadcast bearers are transparent to the rest of the network topology and the carried traffic. Shall a network layer PDU be transmitted over a satellite connection, GSE packets serve as a data link layer (Figure 5). This GSE layer provides encapsulation, fragmentation, and slicing. Created GSE packets are then carried in baseband frames, e.g. DVB-S2, on the physical layer (Figure 6). The receiving side performs a reassembly process, integrity check, and a final decapsulation of transmitted PDUs [4].

Moreover, it is also possible to transport GSE packets over, for example, standard IP network infrastructure. In this case, the DVB-S2 traffic can be carried like a generic payload on the application layer with the use of *User Datagram Protocol* (*UDP*) as a transport layer. Therefore, given UDP datagrams carry DVB-S2 baseband frames, which further carry GSE packets encapsulating selected protocol communication. This approach effectively establishes an overlay network infrastructure, because IP traffic can practically carry GSE packets, which can carry another layer of IP traffic. At this point, the UDP/IP layer *below* GSE can be considered the *carrier (encapsulating) traffic* while, for example, the IP layer *above* GSE can be described as the *carried (encapsulated) traffic*. This approach is presented in Figure 7.

According to specifications and recommendations published by SatLabs, implementation of a receiver with Ethernet interface can be divided into demodulation/decoding device, and a device focused on baseband processing. In such case, *L3 Mode Adaptation Receiver Header* can be prepended to received data [21, p. 10]. The receiving device would then process *DVB-S2 L3 Mode Adaptation Receiver Header*, *DVB-S2 baseband frame*, and *GSE packets* to analyze transmitted communication.

Fragmentation, Slicing, Padding and Reassembly Process As noted earlier, GSE procedures can encapsulate different protocol data units in one or more GSE packets. In general, GSE packets have variable length, and they can be sent in different baseband frames individually or in a group. Therefore, fragmentation, slicing, padding and reassembling can occur. In this context, fragmentation refers to a situation when a PDU and Extension Header is fragmented into multiple GSE packets (Figure 6). Slicing indicates a case when a GSE packet itself is divided into several contiguous baseband frames [8, p. 8]. Noted slicing, therefore, refers to physical layer fragmentation, which shall be transparent to the GSE layer [6, p. 27]. Concerning DVB-S2 applications, GSE slicing (fragmentation into baseband frames) does not occur [6, p. 31].

Shall a single PDU be fragmented into several GSE packets, each packet is assigned a *Fragmentation Identifier* (*Frag ID*) label in the GSE header [8, p. 17]. Frag ID is used to match fragments belonging to the same original PDU. This approach enables the simultaneous transmission of fragments from up to 256 different original PDUs. GSE packets carrying a complete PDU and GSE packets with PDU fragments can be distinguished using start and end flags in the GSE header. The protocol of carried PDU is indicated by Protocol Type/Extension field in the GSE header of the first fragmented packet and every not fragmented packet. The packet with the last PDU fragment further carries a CRC-32 field used to check integrity after the reassembly process (Figure 6). It is important to note that for example, DVB-S2 allows multiplexed transmission of multiple streams, each identified by its *Input Stream Identifier* (*ISI*) [6, p. 32] in baseband header [7, p. 20]. The reassembly process has to be carried out independently for each received stream [8, p. 21]. Some of the possible GSE packet formats are presented in the technical specification [8, pp. 31–32].

Concerning GSE addressing modes noted earlier, an additional fourth mode called *label re-use* can be used when multiple GSE packets are carried in a single baseband frame. Shall label re-use be indicated, current GSE packet without address belongs to the same address as the last previously processed GSE packet. More detailed analysis of GSE protocol is beyond this paper's scope. GSE packet format is defined in the protocol specification [8, p. 12]. Further information can be found in standards, recommendations, and guidelines covering GSE and DVB-S2 [8], [9], [10], [6], [11].

Implementation Outline Our main goal was to successfully decapsulate and process GSE protocol used as an overlay network technology (Figure 7). Main challenges were represented by correct decapsulation of fragmented traffic including timeout detection and also including support for recursive encapsulation. As outlined earlier, this approach represents the transmission of following protocols layered on top of each other:

- *upper* IP as an overlay network layer,
- GSE packets transmitted inside a DVB-S2 baseband frame with Mode Adaptation Header,
- *lower* IP and UDP as a network and a transport layer,
- Ethernet as a data link layer.

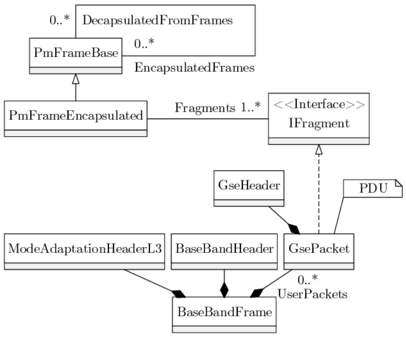


Figure 8. Extension of object model focused on the processing of GSE-encapsulated frames (simplified).

Design of the extension of the object model concerning the processing of encapsulated communication (Figure 8) is quite straightforward and reflects above-described protocol layers. Instance of *BaseBandFrame* composes of *ModeAdaptationHeaderL3*, *BaseBandHeader*, and several user packets. These user packets are, in this case, GSE packets. The instance of *GsePacket* includes *GseHeader* and carries the encapsulated PDU. Properties of these instances store values of specific protocol fields from the processed frame, e.g., address label, length, fragment ID, encapsulated protocol type, checksum, etc. All designed model classes make use of factory methods for parsing corresponding instances from network traffic. These *Parse* methods, therefore, take an instance of *PDUStreamReader*, which is responsible for providing a correct sequence of bytes belonging to the *lower* PDU, as described above.

Because GSE packets can represent fragments of the encapsulated PDU, *GsePacket* class implements *IFragment* interface utilized during reassembly procedures. With the challenge of correct reassembly and decapsulation, a new type of network traffic frame was introduced. Class *PmFrameEncapsulated* inheriting from *PmFrameBase* represents a frame encapsulated in one or more carrier datagrams. Carrier datagrams can be either baseband frames or encapsulation packets. The instance of *PmFrameEncapsulated* has references to individual fragments which form the given frame.

Processing of GSE-encapsulated communication is managed by *L7DvbS2GseDecapsulatorBlock* (Figure 9) dynamically connected to the frame processing pipeline, which was described in Figure 4. This TPL block aims to decapsulate frames from GSE packets used as an overlay network technology. Connection to the pipeline is established using *BroadcastBlock*, which is capable of forwarding *L7Conversations* from the *L7ConversationTrackerBlock* to the *StoreL7ConversationBlock* (as in the standard pipeline topology presented in Figure 4) and also to the noted *L7DvbS2GseDecapsulatorBlock* (Figure 9). Due to the possible amount of false positive detections of GSE layer, decapsulation procedures are optional. Main Netfox Detective application settings include such option to enable *Decapsulation during capture file import* for communication of *Generic Stream Encapsulation (GSE) inside DVB-S2 baseband frames with Mode Adaptation Header L3 sent as Layer 7 PDU*. Shall this option be enabled, ControllerCaptureProcessor instantiates and connects *L7DvbS2GseDecapsulatorBlock* to the pipeline.

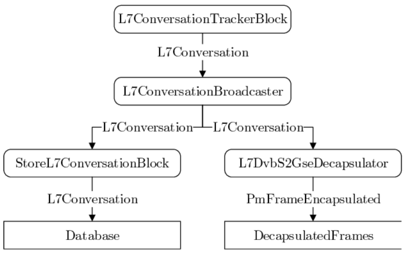


Figure 9. Scheme illustrating the connection of *L7DvbS2GseDecapsulatorBlock* to the frame processing pipeline using *BroadcastBlock* placed between *L7ConversationTrackerBlock* and *StoreL7ConversationBlock*. Standard pipeline topology is shown in Figure 4.

Because GSE packets, which can encapsulate IP traffic, can be transmitted inside another UDP/IP, recursive encapsulation can happen. In such an edge case, several GSE overlay networks could be created on top of each other. That implies that a frame decapsulated from GSE packets must be separately processed and analyzed for the presence of another GSE layer. The challenge of recursive encapsulation is handled by *ControllerCaptureProcessor*, as well. Shall the frame processing pipeline finish with some decapsulated frames, another pipeline is established, and these decapsulated frames are further processed.

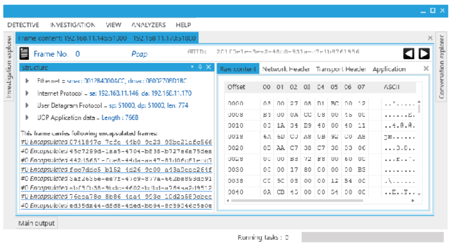
The decapsulation procedure performed by *L7DvbS2GseDecapsulatorBlock* is following. Instantiated *PDUStreamReader* handles reading bytes of the input conversation and then parsing of a GSE layer is attempted. Upon successful detection of GSE layer, DVB-S2 baseband frames are passed to the *GseReassemblingDecapsulator*. It outputs frames which have type *PmFrameEncapsulated* and are ready for further processing by consequential blocks.

The *GseReassemblingDecapsulator* manages decapsulation of frames encapsulated inside GSE packets, which are carried in baseband frames. The decapsulator is capable of reassembly procedure according to the specification [8, p. 21]. Reassembling distinguishes single input stream and multiple input streams based on *ISI* explained earlier. The reassembly procedure utilizes *GseReassemblyBuffer* for each fragment ID and for each stream identifier processed. The decapsulator, therefore, decapsulates frames from GSE packets in baseband frames. In the case of GSE fragmentation, given GSE packet (fragment) is added to the corresponding reassembly buffer. Upon successful reassembly, the carried frame is then decapsulated, too. Each *GseReassemblyBuffer* holds a counter of processed baseband frames, which is used to detect a PDU reassembly time-out error, as defined in the specification [8].

4.2 Evaluation

Every layer of decapsulated traffic is subject to further network forensic analysis performed by the Netfox Detective. The information is presented in the GUI. The view informs the user whether the current frame in encapsulated or not. It is also possible to navigate between views showing individual encapsulating frames (see Figure 10) and encapsulated frames (see Figure 11).

The implementation has been evaluated on publicly available datasets [2], and results (amount of correctly identified and extracted GSE communications) were comparable to the reference Wireshark implementation. A set of integration tests was implemented that verify the correct processing of GSE traffic in future releases and prohibit regression bugs from being introduced.



Frame content: 192.168.11.146:51000 192.168.11.170:51000

Frame No. 0 Pcap GUID: 201f0e1e-3ee2-48c0-831a-c7e1b9761956

```
▸ Ethernet = smac: 00128400ACC, dmac: 0800270BD1BC
▸ Internet Protocol – sa: 192.168.11.146, da: 192.168.11.170
▸ User Datagram Protocol – sp: 51000, dp: 51000, len: 774
▸ UDP Application data – Length : 766B

This frame carries following encapsulated frames:
#0 Encapsulated 0741847c-7cfc-44b0-9c23-93bc21afc566
#0 Encapsulated 45c7299d-1ea5-4704-b638-b727e6a75dea
#0 Encapsulated 442d5651-c0e8-44da-ae47-81d06cf1ecc7
#0 Encapsulated fcc7ddc5-b152-4d26-9c00-a43a9cac264f
#0 Encapsulated 3af3635e-ee7f-47c9-877a-662be893dc91
#0 Encapsulated eb[illegible]-9bdc-4602-b3bd-a764ea2d9512
#0 Encapsulated 76cca78c-8b86-4aa4-993c-10d2a580cccc
#0 Encapsulated ed35da44-d8d8-4ded-bb94-8c39046c5a0e
```

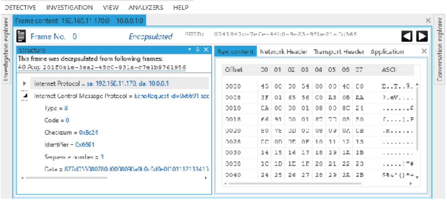
```
Offset  00 01 02 03 04 05 06 07   ASCII
0000    08 00 27 08 D1 BC 00 12   ..'.....
0008    B4 00 0A CC 08 00 45 00   ......E.
0010    03 1A 34 D9 40 00 40 11   ..4.@.@.
0018    6A 6D C0 A8 0B 92 C0 A8   jm......
0020    0B AA C7 38 C7 38 03 06   ...8.8..
0028    00 00 B8 72 E8 00 60 00   ...r..`.
0030    00 00 17 80 00 00 00 B3   ........
0038    CC 5C 08 00 00 12 B4 0C   .\......
0040    0A CB 45 00 00 54 00 00   ..E..T..
```

Running tasks : 0

Figure 10. View of the frame content of the Netfox Detective presenting a frame carrying eight other encapsulated frames. It is possible to navigate between encapsulated frames using shown links labeled with GUID of the target frame.

The main goal was to process GSE traffic used as the tunneling protocol in satellite communication networks. The current implementation of GSE processing module does not support for DVB-S2 baseband frames that can be used as the physical layer. The decapsulation procedure also does not handle GSE labels, because of the limitation of the Netfox Framework tool that does not support tracking multiple L1 conversations. Stream ID and fragment ID is correctly

[2] https://wiki.wireshark.org/DVB-S2 (last accessed 2019-04-17).



```
GUID: 0741847c-7efe-441b-9e23-951e21efc566

This frame was decapsulated from following frames:
#0 Pcap 201f0e1e-3ea2-48c0-931e-c7e1b9761956

▸ Internet Protocol – sa: 192.168.11.170, da: 10.0.0.1
◢ Internet Control Message Protocol = EchoRequest id=0x6691 sec
      Type = 8
      Code = 0
      Checksum = 0x8c24
      Identifier = 0x6691
      Sequence number = 1
      Data = 877d055080780d0008090a0b0c0d0e0f101112131415

Offset   00 01 02 03 04 05 06 07   ASCII
0000     45 00 00 54 00 00 40 00   E..T..@.
0008     3F 01 65 56 C0 A8 0B AA   ?.eV....
0010     CA 00 00 01 08 00 8C 24   .......$
0018     66 91 00 01 87 7D 05 50   f....}.P
0020     80 78 0D 00 08 09 0A 0B   .x......
0028     0C 0D 0E 0F 10 11 12 13   ........
0030     14 15 16 17 18 19 1A 1B   ........
0038     1C 1D 1E 1F 20 21 22 23   .....!"#
0040     24 25 26 27 28 29 2A 2B   $%&'()*+
```

Figure 11. Frame content view of Netfox Detective (as in Figure 10) analyzing a frame that was decapsulated from another frame of the *lower* layer.

utilized during GSE reassembling. However, the stream ID value is not used to separate L1 conversations.

5 Conclusion

Network traffic analysis is often conducted as a part of digital investigation. In most cases, Internet communication is analyzed, but sometimes the interesting communication is encapsulated in some tunneling protocol because of the network technology used. In this paper, we have presented the analysis of GSE protocol and the implementation of forensic data extraction enabling to access the encapsulated Internet traffic. The proposed implementation was evaluated against the Wireshark tool, the only available implementation of GSE analysis module in common NSM tools. The forensics tool Netfox Detective is publicly available (https://github.com/nesfit/NetfoxDetective) for all network forensic practitioners to use, including open-source source codes that can be freely modified, or integrated into other newly implemented tools.

References

[1] Nicole Beebe. 2009. Digital forensic research: The good, the bad and the unaddressed. In *IFIP International Conference on Digital Forensics*. Springer, 17–36.

[2] Eoghan Casey. 2004. Network traffic as a source of evidence: tool strengths, weaknesses, and future needs. *Digital Investigation* 1, 1 (2004), 28 – 43. https://doi.org/10.1016/j.diin.2003.12.002

[3] M. I. Cohen. 2008. PyFlag - An Advanced Network Forensic Framework. *Digital Investigation* 5 (Sept. 2008), 112–120. https://doi.org/10.1016/j.diin.2008.05.016

[4] DVB Project Office. 2015. *DVB-GSE - Generic Stream Encapsulation*. DVB Project Office. URL: https://www.dvb.org/resources/public/factsheets/dvb-gse_factsheet.pdf.

[5] Sandstorm Enterprises. 2003. NetIntercept.

[6] ETSI. 2011. *ETSI TS 102 771 V1.2.1 - Digital Video Broadcasting (DVB); Generic Stream Encapsulation (GSE) implementation guidelines*. European Telecommunications Standards Institute. URL: https://www.etsi.org/deliver/etsi_ts/102700_102799/102771/01.02.01_60/ts_102771v010201p.pdf.

[7] ETSI. 2014. *ETSI EN 302 307-1 V1.4.1 - Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 1: DVB-S2*. European Telecommunications Standards Institute. URL: http://www.etsi.org/deliver/etsi_en/302300_302399/30230701/01.04.01_60/en_30230701v010401p.pdf.

[8] ETSI. 2014. *ETSI TS 102 606-1 V1.2.1 - Digital Video Broadcasting (DVB); Generic Stream Encapsulation (GSE); Part 1: Protocol*. European Telecommunications Standards Institute. URL: https://www.etsi.org/deliver/etsi_ts/102600_102699/10260601/01.02.01_60/ts_10260601v010201p.pdf.

[9] ETSI. 2014. *ETSI TS 102 606-2 V1.1.1 - Digital Video Broadcasting (DVB); Generic Stream Encapsulation (GSE); Part 2: Logical Link Control (LLC)*. European Telecommunications Standards Institute. URL: https://www.etsi.org/deliver/etsi_ts/102600_102699/10260602/01.01.01_60/ts_10260602v010101p.pdf.

[10] ETSI. 2014. *ETSI TS 102 606-3 V1.1.1 - Digital Video Broadcasting (DVB); Generic Stream Encapsulation (GSE); Part 3: Robust Header Compression (ROHC) for IP*. European Telecommunications Standards Institute. URL: https://www.etsi.org/deliver/etsi_ts/102600_102699/10260603/01.01.01_60/ts_10260603v010101p.pdf.

[11] ETSI. 2015. *ETSI TR 102 376-1 V1.2.1 - Digital Video Broadcasting (DVB); Implementation guidelines for the second generation system for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 1: DVB-S2; Part 1: DVB-S2*. European Telecommunications Standards Institute. URL: https://www.etsi.org/deliver/etsi_tr/102300_102399/10237601/01.02.01_60/tr_10237601v010201p.pdf.

[12] Dan Farmer and Wietse Venema. 2009. *Forensic Discovery* (1st ed.). Addison-Wesley Professional.

[13] Simson L Garfinkel. 2010. Digital forensics research: The next 10 years. *Digital Investigation* 7 (2010), S64–S73.

[14] Vikram S Harichandran, Frank Breitinger, Ibrahim Baggili, and Andrew Marrington. 2016. A cyber forensics needs analysis survey: Revisiting the domain’s needs a decade later. *Computers & Security* 57 (2016), 1–13.

[15] Petr Matoušek, Jan Pluskal, Ondřej Ryšavý, Vladimír Veselý, Martin Kmeť, Filip Karpíšek, and Martin Vymlátil. 2015. Advanced Techniques for Reconstruction of Incomplete Network Data. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering* 2015, 157 (2015), 69–84. http://www.fit.vutbr.cz/research/view_pub.php?id=10864

[16] Moloch. cited April 2019. URL: https://molo.ch.

[17] NetworkMiner. cited April 2019. URL: https://www.netresec.com/?page=NetworkMiner.

[18] NfDump. cited April 2019. URL: https://github.com/phaag/nfdump.

[19] Emmanuel S. Pilli, R. C. Joshi, and Rajdeep Niyogi. 2010. Network Forensic Frameworks: Survey and Research Challenges. *Digital Investigation* 7, 1-2 (Oct. 2010), 14–27. https://doi.org/10.1016/j.diin.2010.02.003

[20] PyFlag. cited April 2019. URL: https://github.com/py4n6/pyflag.

[21] SatLabs Group. 2008. *Mode Adaptation Input and Output Interfaces for DVB-S2 equipment Version 1.3*. SatLabs Group. URL: http://satlabs.org/pdf/sl_561_Mode_Adaptation_Input_and_Output_Interfaces_for_DVB-S2_Equipment_v1.3.pdf.

[22] SatLabs Group. 2010. *SatLabs System Recommendations Version 2.1.2*. SatLabs Group. URL: http://satlabs.org/pdf/SatLabs_System_Recommendations_v2.1.2_final.pdf.

[23] Suricata. cited April 2019. URL: https://suricata-ids.org.

[24] TCPDUMP. cited April 2019. URL: https://www.tcpdump.org/.

[25] Tcpdump/Libpcap. 2018. LINK-LAYER HEADER TYPES. https://www.tcpdump.org/linktypes.html

[26] TCPFlow. cited April 2019. URL: https://github.com/simsong/tcpflow.

[27] Wireshark. cited April 2019. URL: https://www.wireshark.org/.

[28] XPlico. cited April 2019. URL: https://www.xplico.org/.

[29] Zeek. cited April 2019. URL: https://www.zeek.org.

A.4 Network Forensic Analysis for Lawful Enforcement on Steroids, Distributed and Scalable

Viliam Letavay, Jan Pluskal, and Ondřej Ryšavý. “Network Forensic Analysis for Lawful Enforcement on Steroids, Distributed and Scalable”. In: *Proceedings of the 6th Conference on the Engineering of Computer Based Systems.* ACM. 2019. ISBN: 9781450376365

Network Forensic Analysis for Lawful Enforcement on Steroids, Distributed and Scalable

Viliam Letavay
Brno University of Technology
Brno, Czech republic
iletavay@fit.vutbr.cz

Jan Pluskal
Brno University of Technology
Brno, Czech republic
ipluskal@fit.vutbr.cz

Ondřej Ryšavý
Brno University of Technology
Brno, Czech republic
rysavy@fit.vutbr.cz

Abstract

Forensic analysis of intercepted network traffic focuses on finding and extracting communication evidence, such as instant messaging, email, VoIP calls, localization information, documents, images. Due to the amount of data captured, this process is time-consuming and complicated. Most commonly used forensic network analysis tools have limited capabilities for large data processing. In this paper, we are introducing a new tool that achieves better data processing performance using available computing resources through distributed processing. Thanks to the technology used, this tool can be used on commodity hardware in a local area network, in a dedicated computing cluster or cloud environment.

CCS Concepts • **Applied computing → Network forensics**; • **Networks** → *Network monitoring*; *Network protocols*; Transport protocols; Application layer protocols; • **Social and professional topics** → Computer crime.

Keywords Network Forensics, Network Traffic Processing, Distributed Computing

ACM Reference Format:
Viliam Letavay, Jan Pluskal, and Ondřej Ryšavý. 2019. Network Forensic Analysis for Lawful Enforcement on Steroids, Distributed and Scalable. In *6th Conference on the Engineering of Computer Based Systems (ECBS '19), September 2–3, 2019, Bucharest, Romania.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3352700.3352720

1 Introduction

Network administrators, cyber-security analysts, and digital forensic investigators capture and analyze network communication to reveal the attack patterns or recover digital evidence. The traditional tools used to process captured communication have limited scalability. For instance, Wireshark is an excellent tool for troubleshooting and security analysis. However, performing analysis of captured files of several gigabytes is cumbersome. Current computing platforms offer tremendous computation power. It is mainly because of its multi-core architecture. The number of cores available per CPU constantly grows[1], contrary to the CPU frequency that is essential for single threaded applications.

However, modifying commonly used single-threaded network forensic tools, e.g., PyFlag, NetworkMiner, to utilize the full potential of modern processors is a complex task which would require extensive modification of their code base. Therefore, new tools for network forensic analysis are in high demand [9, 14].

Even more computing power can be obtained by distributing the workload among a cluster of machines. Availability of industrial strength technology for distributed data processing and scalable storage led to the emergence of distributed network security analysis systems, e.g., Moloch[2], Apache Spot[3], or Apache Metron[4]. Academic research also yields to implementations of scalable network security monitoring systems [26].

Regardless of the technology used, these systems aim to provide a high performance distributed computing environment for network security monitoring (NSM). These tools are especially useful for real-time data processing and complement other systems to defend against cyber threats such as IDS, firewalls, or SIEM. While these tools are also useful for network forensic analysis, forensic investigation favors the depth, accuracy, and reliability of processing over the fast response time. When investigating, it is necessary to reliably analyze any artifact that can be extracted, even though the source data may be corrupted and may not be complete.

1.1 Contribution

This paper discusses the design, performance, and properties of a new Network Forensic and Analysis Tool (NFAT) —

ECBS '19, September 2–3, 2019, Bucharest, Romania

ACM ISBN 978-1-4503-7636-5/19/09…$15.00
https://doi.org/10.1145/3352700.3352720

[1] Example of the state-of-the-art CPU available on the market — AMD EPYC Rome *64 cores 128 threads*, 2.35 GHz; Intel© Xeon Phi™ 7290F, *72 cores*, 1.5 GHz; Intel© Xeon© Platinum 8180M *28 cores*, *56 threads*, 2.5 GHz

[2] https://molo.ch/ (last accessed 2019-07-03).

[3] http://spot.incubator.apache.org/ (last accessed 2019-07-03).

[4] http://metron.apache.org/ (last accessed 2019-07-03).

Network Traffic Processing & Analysis Cluster (NTPAC) — that utilizes distributed computing architecture to improve the performance of network traffic analysis while being less demanding on hardware requirements than related systems. To extract the evidence from network packets, we need to thoroughly analyze them which means to perform several consecutive operations such as packet dissecting, flow identification, network stream composition, application protocol identification, and message parsing and artifact extraction (see Section 3). Contrary to the other NFAT tools (see Section 2.4), NTPAC is able to correctly process captured traffic that is malformed without yielding misleading evidence (see Section 4.2). NTPAC performs forensic network traffic analysis at high-speed networks. The system design uses a scalable approach that enables to run the tool on a single machine as well as on a computing cluster. in comparison with other NFATs tools, NTPAC is an order of magnitude faster and scales (see Section 3).

1.2 Paper Structure

Initially, background and related work are discussed presenting an overview of current network forensic and security monitoring tools. The architecture of NTPAC is introduced, and the major architectural components are outlined. The paper then provides a preliminary evaluation of the performance that focuses on demonstrating the throughput and scalability of the tool. Finally, we discuss limitations and future work.

2 Background and Related Work

This section provides a background for the paper and lists the related work. First, the actor model and network packet capture analysis are presented. Then we overview existing network forensic tools and frameworks.

2.1 Actor Model

Actor model offers to solve the problems related to parallel and distributed computing elegantly and efficiently. The actor model was first introduced as a theoretical computation model highly influenced by Lisp, Simula and packet switching in computer networks [7]. It defines a fundamental concept called *actor system* that is composed of tiny building blocks called *actors* that execute independently and massively in parallel. The actor is in the distributed world an abstraction of what is an *object* in Object-Oriented Programming; in other words, it bounds data with computation.

Actors communicate asynchronously via message passing. *Actor system* guarantees *at most one delivery*, which means that *any message* can get lost at *any time* but cannot be delivered twice or more. Actor's state changes only as a reaction on a received message. Actor's behavior determines how to process the incoming message by *creating another actor*, *sending a message* to another actor, *changing its state*.

The composition of *actors* in the *actor system* is hierarchical. Each *actor* is responsible for any other *actor* it creates, i.e., the creation of a parent-child relationship. An *Actor* is designed to be as simple as possible, typically without complex inner integrity checks, exception handling, etc. Thus, it can crash at any time. *Parent actor* is responsible for its children and knows how to deal with *children's* failures. This concept greatly simplifies the computation model and allows a programmer to focus only on the most important part that is the core application's functionality and frees him/her from the need of use of synchronization tools (such as mutexes).

2.2 Packet Capture Analysis

Network traffic analysis aims to reveal traces of network attacks and find answers to questions about the incident investigation. Packet analysis starts with dissecting network traffic which performs the following steps: i) *loading PCAP files*, parsing the PCAP file, and extracting individual packets, ii) *dissecting packets with low-level protocol parsers*, including Ethernet, IP, IPv6, TCP or UDP, iii) *collecting TCP packets into streams*, and iv) *applying higher level protocol parsers* to get the required information or extract artifacts.

However, in many cases, it is not possible to obtain plain content from communication because of encryption. Then at least some form of valuable forensic information can be identified, for instance, identities of users [1, 18], devices [17] or applications [16] based on extracted metadata.

Depending on the goal and available tools there are numerous analytic approaches to network packet analysis:

- The bottom-up approach is a prevalent method used by Network Security Monitoring (NSM) [24] oriented analysis that supports several tools, most notably *Wireshark*. All packets are parsed and presented to the investigator who uses filtering, querying and reassembling to identify and extract required artifacts.
- The top-down approach assumes that the Network Forensic and Analysis Tool (NFAT) [15], e.g., *NetworkMiner*, *Xplico*, *PyFlag*, *NetfoxDetective*, can extract information from packets into conversations or other higher level artifacts. These applications visualize this high-level information to the investigator that can then drill down into details if necessary.
- Search based approach considers network communication being just another data format in which it is possible to search for keywords or patterns [11, 20].

2.3 Network Security Monitoring Tools

Network forensic methods were implemented in various NSM tools, e.g., Wireshark, TCP dump, IDS systems (Snort, Zeek), fingerprinting tools (Nmap, p0f), and tools to identify and analyze security threats. As [15] observes, NSM tools are primarily used by network administrators and are intended for detailed bottom-up analysis that requires advanced skills.

Lukashin [12] presented a scalable internet traffic analysis system, which can process multi-terabytes libpcap dump files. It utilizes Apache Spark for data processing to analyze captured packets. The system performs basic analysis and lacks some advanced features required by network forensics. Other approaches to the big data network security analysis were presented by various researchers [2, 19, 30]. Currently, Apache Metron and Apache Spot projects are the most vital. They are frameworks for security analysis of IT threats, enabling to process also firewall and application logs, emails, intrusion-detection reports, and so on. Although they are primarily focusing on network security, they can be valuable as sources of forensic data.

Additionally, there are special appliances for network security monitoring based on custom made FPGA chips that can perform up to 100 Gbps deep packet analysis and export NetFlow with additional information extracted from application protocols [8].

2.4 Network Forensic Analysis Tools

The investigators of Law Enforcement Agencies deal with the enormous number of cases. They require specialized tools that perform top-down analysis and save valuable time [3]. The following list is a selection of notable open-source tools that were designed to support the investigators:

PyFlag is full-fledged NFAT which is intended for disk, memory, and network forensics. PyFlag's design incorporates the concept of a Virtual File System [4]. It implements a specific loader for each supported data source. PyFlag enables to reassemble the content of the communication, e.g., web pages, email conversation.

NetworkMiner is an open source tool that integrates packet sniffing and higher-layer protocol analyzing capabilities into a tool for passive network forensic analysis.

Xplico is a modular NFAT. It consists of the input module handling the loading source data, decoding module equipped with protocol dissectors for decoding the traffic and exporting the content, and the output module organizing decoded data and presenting them to the user. Xplico is a client-server application that can analyze PCAP files as large as several gigabytes.

While all these tools are very useful for investigators as they offer a variety of advanced features, their scalability is limited because they run on either a single computer or in a traditional client-server architecture.

2.5 Big Data Forensics

As distributed frameworks matured, new tools for big data security analysis and digital forensics were designed. Such tools are usually intended for the forensic investigation conducted by network administrators on corporate networks. They usually serve as a complement to Intrusion Detection Systems enabling to capture and analyze hi-speed communication at scale.

Agent-based systems for digital forensics were considered in the literature [21, 22, 29]. These models are more suitable for real-time network forensic analysis from multiple sources, such as logs and captured communication. In these systems, numerous agents perform data collection tasks. The extracted information is then sent to the forensic server and analyzed on this single node only [10], which makes this node to be the bottleneck of the whole system.

The VAST system builds upon Vallentin's previous work — The NIDS Cluster [28] which distributes the workload across multiple workers running *Zeek* to investigate online network traffic and extract *Zeek events*. The VAST system itself goes further and distributes *Zeek events* to workers running in a computing cluster which allows for on-line analysis and interactive queries. Distribution of raw packets is also supported as a 4-tuple with payload up to the speed of 3.1 Gb/s (the libpcap reading speed). According to Vallentin [27] the system does not guarantee that the storage will be able to keep up with the incoming traffic of this speed.

3 Traffic Processing

The goal of NTPAC is to capture and analyze network communication enabling to extract available information. Depending on the case, the forensic investigator may be interested in the content or metadata of application messages. NTPAC handles captured packets according to the following procedure in order to reassemble application messages:

- NTPAC organizes captured packets into separate network layer conversations based on their source and destination IP addresses, providing IP conversation.
- NTPAC then splits IP conversations into TCP/UDP conversations based on the source and destination port numbers and the transport protocol type, as shown in figure 2.
- NTPAC reassembles application conversations from packets separated into individual TCP/UDP conversations. This method utilizes heuristics [13] to recognize multiple application communication multiplexed into a stream of packets of one TCP/UDP conversation caused for example by port reuse.

Because application message extraction is a computationally challenging task, it is a good candidate to run on a computer cluster to improve overall system performance.

Extraction of the artifacts from application messages assumes that we correctly identified the application protocols. Methods based on known port numbers, characteristics patterns in the payload of packets, using statistical methods or machine learning [16] approach can be applied.

However, in many cases, application information cannot be extracted because the content is encrypted. In fact, approximately 76 % of HTTP traffic (at the time of writing this paper) is transmitted by SSL/TLS [5]. In this situation, we cannot extract application messages, but it is possible to get metadata from the SSL/TLS protocol itself, for example, cryptographic information, certificate data, etc. The only exception is two possible situations in which we can decrypt encrypted application data [5]:

1. We have access to the server's private key used in the initialization of an SSL/TLS session, we want to decrypt, and cipher-suites not supporting forward secrecy is used.
2. We can perform a Man-In-The-Middle attack with an SSL/TLS proxy [23] and store session keys.

Most agencies cannot use these techniques because of legal restrictions. For this reason, we did not consider implementing SSL/TLS encryption techniques in our tool.

4 System architecture

The architecture consists of multiple modules that form the processing pipeline (see Figure1). At the highest level, the NTPAC workflow can be divided into two main phases:

Data pre-processing reconstructs application layer conversations (L7 conversation). Each of these conversations is made up of source and destination endpoints, timestamps, and other information that is needed for subsequent processing.

Data analysis identifies application protocols in reconstructed conversations and uses an appropriate application protocol decoder to reconstruct application events from given conversations, such as visited web pages, sent emails, queried domains, etc. The output of this phase is a set of forensic artifacts.

These phases correspond to low-level analysis and high-level analysis. The separation of data pre-processing from the data analysis enables to use the actor-based computational model and offer the ability to distribute the computation. In the rest of the section, details will be given for each module of the processing pipeline.

4.1 Load balancing

The job of the *Load Balancer* nodes is to split the input packet stream, i.e., PCAP file or live traffic, into sub-streams that are then delivered to the reassembling nodes. To avoid the problem of sending packets from the same conversation to different reassemble nodes, the *Load Balancer* calculates the key used to select the destination node from the appropriate protocol fields.

The Eq. 1 calculates the routing key based on communication endpoints (*EP_A* and *EP_B*) and the transport protocol used. Value n represents the number of active *Reassembler* nodes.

$$Hash(EP_A \cdot EP_B \cdot Protocol) \bmod n \qquad (1)$$

Since all packets from the same conversation (i.e. in both directions of the conversation) should produce the same routing key, we defined an ordering relation ≤ for the endpoints[6] and ensured that $EP_A \leq EP_B$ by swapping them if necessary.

While the *Load Balancers* process each packet individually, the data is delivered to *Reassemblers* in batches. This technique helps to decrease network and processing cost of the data distribution.

Back pressure mechanism is used to control the data flow between the nodes. To increase throughput, a *Load Balancer* can submit multiple batches in parallel to the target *Reassemblers*.

IPv4 fragmentation is a challenge for *Load Balancers*. Fragmentation splits one IP packet into multiple IP packets so that the encapsulated transport layer segment header only occurs in the first IP fragment. The *Load Balancer* must, therefore, rebuild the IP fragments to identify the routing key for all fragments of a segment, before it can send them to an appropriate *Reassembler*.

4.2 Conversation Reassembling

Reassembler reconstructs conversations, i.e., two-way traffic layer flows, in batches of packets received from *Load Balancers*. The reassembly process is designed to reconstruct incomplete and corrupted data, using various heuristic techniques [13]. Reassembling is done in several steps until two corresponding flows are assembled, which is illustrated in Figure 2. The entire processing is mapped to actors performing individual steps. Individual L3 and L4 conversations are represented by corresponding actors, which form an actor hierarchy as shown in figure 3. *L3 Conversation* actors are managed by *Capture* actors, which stands for a source capture being analyzed. To enable an analysis of multiple captures at the same time, multiple *Capture* actors can be initiated. The *Captures Controller* actor manages all capture actors.

The packet blocks are first received by the *Captures Controller* actor, which passes them to the appropriate *Capture* actor. The *Capture* actor identifies affiliation of packets to L3 conversations by extracting the IP addresses of the packets and forwards them to appropriate *L3 Conversation* actors which, after identifying affiliation of packets to L4 conversation by extracting the transport protocol and port numbers, forwards the packets to appropriate *L4 Conversation* actors. At these actors, the process of reassembling depends on the transport protocol of the conversation and is performed by either *UDP Conversation Tracker* or *TCP Conversation Tracker*.

[5] https://letsencrypt.org/stats/

[6] The endpoint is a pair of IP address and port number. We consider that there is a suitable lexicographic ordering on a set of endpoints.

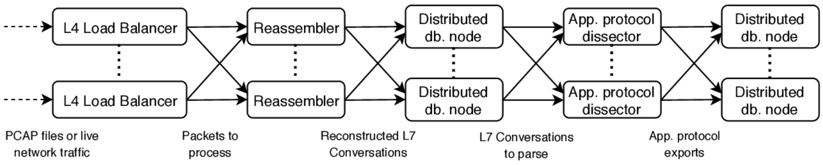


Figure 1. NTPAC's logical architecture

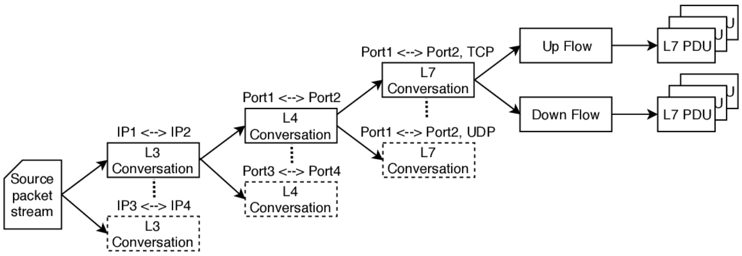


Figure 2. Separation of packets into distinct L3 conversations, L4 conversations and finally L7 conversations. L7 conversations consist of *Upflow* and *Downflow*, which contain a sequence of reconstructed *L7 PDUs.*

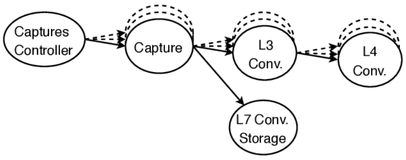


Figure 3. *Reassembler's* actor hierarchy

UDP Protocol Reassembling

UDP is transferring application data as they are, without the use of any additional control packets which implement mechanisms such as flow control or reliable data delivery. *UDP Conversation Tracker*, therefore, treats every transmitted datagram inside given L4 conversation as an individual L7 PDU (Protocol Data Unit). Another important aspect of the UDP protocol is that it is connection-less — it does not establish connections between communicating parties. To distinguish individual L7 Conversations (composed of a pair of *Upflow* and *Downflow*) inside single L4 conversation, *UDP Conversation Tracker* uses a simple heuristic based on a *time delay* between individual L7 PDUs. L7 PDUs in a given direction are considered to be part of a single *flow* if the time difference between their transmission and last recorded activity (timestamp of the last L7 PDU) of a given *flow* is less than a defined value. Experimentally we set this value to 10 minutes, but we are planning to further study UDP behavior of multiple protocols and define this threshold on application protocol bases.

TCP Protocol Reassembling

Processing of TCP protocol is different from handling UDP flows because we can use control information carried along with the data. *TCP Conversation Tracker* is capable of identifying connection initialization and its termination, handling data retransmission and reordering. In the same way as a *UDP Conversation Tracker*, *TCP Conversation Tracker* also processes segments (TCP PDUs) in separate *flows*, which are later paired to form L7 conversations. To create this *flows*, it first stores segments in the so-called *reassembling collection*, in which segments are stored and ordered by their TCP sequence number. Both directions of communication have designated their *reassembling collection*. Before a segment is stored in *reassembling collection*, its sequence number is normalized by incrementing it by *a count of detected sequence number overflows* × 2^{32} (space of TCP sequence numbers). Sequence number overflows can be caused by a natural overflow of a 32-bit integer sequence number or by establishing a new TCP connection, with ISN (Initial Sequence Number) lower as that of a previous connection. By storing segments in *reassembling collection* and ordering them by their normalized sequence numbers, we achieve that:

1. individual segments inside L7 conversation are ordered;
2. we detect data retransmissions by comparing payloads of segments of which normalized sequence numbers are overlapping;
3. individual L7 conversations inside L4 conversation are ordered by the time they were transmitted.

```
Algorithm tcp_flow_reassembling()
    forall segment in reassembling_collection do
        if SYN flag is set then
            close_flow()
            flow ← create new flow
        else if FIN flag is set then
            close_flow()
        else if flow is nil then
            flow ← create new flow
            add_segment_to_pdu()
        else
            add_segment_to_pdu()
    end
    return flows
Procedure close_flow()
    if flow is nil then
        return
    if pdu is not nil then
        add_pdu_to_flow()
    flows.insert(flow)
    flow ← nil
    return
Procedure add_segment_to_pdu()
    if segment is retransmission then
        return
    if pdu is nil then
        pdu ← create new pdu
    pdu.segments.insert(segment)
    if PSH flag is set then
        add_pdu_to_flow()
    return
Procedure add_pdu_to_flow()
    flow.pdus.insert(pdu)
    pdu ← nil
    return
```

Algorithm 1: TCP *flow* reassembling.

After all segments of L4 conversation have been stored in an appropriate *reassembling collection* (for *Up* and *Down* direction), *TCP Conversation Tracker* iterates through both of them sequentially in order to reconstruct *Upflows* and *Downflows*. Simplified *flow* reassembling algorithm is shown in Algorithm 1. For each segment containing application data, it appends it to current L7 PDU (creates it at first, if it is not already created). After it encounters packet with TCP PSH flag set, it completes current L7 PDU and adds it to the current *flow*. Segments which do not contain application data, such as packets of TCP handshake or connection termination are used to differentiate individual TCP connections by creating appropriate *flows* with assigned created L7 PDUs. Created *Upflows* and *Downflows* are paired by their ISNs (Initial Sequence Numbers) or based on their overlap on time axis in case an ISN of a particular *flow* could not be determined (missing TCP handshake).

L7 Conversation storage

L7 conversations reconstructed by *L4 Conversation* actors are passed to *L7 Conversation Storage* actor. This actor saves contents (series of reconstructed L7 PDUs), as well as metadata (timestamps, endpoints and transport protocols) of these L7 conversations in a distributed database. Our tool uses an abstract data access layer that eliminates any dependence on one database technology. Currently, our solution is primarily based on the use of the Cassandra database engine[7], which has the appropriate features — it has a distributed design, configurable *replication factor* per keyspace and *consistency factor* per query.

4.3 Application protocol parsing

In the *second stage*, a subset of reconstructed L7 conversations is retrieved from the database and further processed to identify and extract interesting application messages:

- First, *Application protocol classifier* block identifies an application protocol of the conversation. Our solution currently implements a simple application protocol classifier based on the database of known ports. However, a more advanced classifier can be used to utilize pattern recognition or statistical methods [6, 16].
- Based on the recognized application protocol, the conversation is consumed by parsing module designed to the processing of a single application protocol such as HTTP, SMTP or DNS. The parsing module processes the entire conversation by extracting individual application protocol messages and storing them back to the distributed database.

The current implementation includes only HTTP and DNS parsers. Adding support for other application protocols requires creating an application protocol parser. Implementing the parser is time-consuming and error-prone. Another option is to generate a parser using a suitable parser generator. Depending on whether the protocol is text or binary, different types of generators can be used, for example, Spicy [25], Kaitai Struct[8], etc.

[7] Note that also MSSQL and ArangoDB are supported.

[8] https://kaitai.io/

5 Performance evaluation

We focused our preliminary assessment on determining the performance parameters of the created tool. During the experiments, we considered both the data storage scenario in the distributed database and the case where data analysis uses the output from the previous step directly. The goal is to demonstrate the scalability of the proposed solution and show the available throughput in various possible configurations. We have considered two major test scenarios:

Standalone processing tests how fast is captured traffic processed on a single machine inside one process. This test-case shows total throughput of our processing algorithms (especially reassembling and application protocol parsing) on given machine type. Because the whole processing is running under one *Common Language Runtime* (CLR), it is expected to be faster than distributed processing with a low number of processing nodes. This experiment provides a baseline to which other results are compared.

Cluster Processing shows the scalability of our solution in a computing cluster. We tested it in a distributed environment with a different number of nodes. The test scenarios considered (i) processing with a single *Load Balancer* and different numbers of *Reassembler* nodes and (ii) a different number of *Load Balancer* and *Reassembler* nodes.

For our test purposes, we have chosen multiple different computing environments described in Table 1. The E.1 environment consists of 14 workstations that are all connected to the same local network. Environment E.2 is a cluster-integrated Google Cloud Platform consisting of 12 virtual machines. E.3 is a mini-cluster of four server boards in a single chassis. Finally, E.4 is a single powerful workstation.

Table 1. Testing environments used for performance evaluation.

	E.1	**E.2**	**E.3**	**E.4**
Machine Type	Desktop computers	Google Cloud Platform (*)	Mini-cluster	Workstation
Machines count	14	12	4	1
CPU Type	Intel i5-3570K	Intel Xeon E5	Intel Xeon E5520	Intel i7-5930K
Physical Cores	4	2	4	6
Logical Cores	4	4	8	12
CPU Frequency	3.40 GHz	2.60 GHz	2.26 GHz	3.50 GHz
CPU Frequency Turbo – 1 core	3.80 GHz	2.80 GHz	3.53 GHz	4.30 GHz
RAM	8 GB	7 GB	48 GB	64 GB
Sequential disk read/write	73/67 MB/s	120/118 MB/s	282/265 MB/s	490/430 MB/s
Network Card	1 Gbps	10 Gbps	1 Gbps	1 Gbps

(*) n1-highcpu-4

As the source packet capture, we used 4.7 GB file from a well known M57-Patents Scenario[9]. It captures real-world corporate network traffic over one month, consisting of 5,707,845 frames. The size of the capture file is large enough to limit the overhead to a negligible part in the initialization phase but allows us to run all test cases in a reasonable time.

To reduce the memory consumption of tracking of all processed conversations by the *Reassembler* nodes, its actors detect and remove inactive (timed out) conversations. Thus, the memory allocation corresponds to the number of active concurrent network flows within a particular time window.

Each experiment was repeated 10 times. The calculated standard deviation was in the range of $5 - 10\,\%$. Such a high value is due to the inherent non-deterministic behavior of the distributed system, including the effect of network communication, the garbage collection, and other operating system processes.

5.1 Single-node Environments

We measured the individual processing stages in the standalone test scenario in environments E.3 and E.4. Table 2 represents the performance achieved for each phase. Preliminary results show that it is possible to read and decode packets from a file at approximately 3.8 Gbps and 1.7 Gbps in test environments E.4 and E.3 respectively (second row of the table). The process of extracting conversations requires much more effort and therefore performance dropped to 972 Gbps and 380 Gbps respectively what represents about 75 % decrease compared to the previous phase. It suggests that this resource-intensive part could be most accelerated by distributed calculation. The last phase is the analysis of HTTP and DNS protocols, which resulted in a decrease in throughput of about 8 % compared to the previous phase. For comparison, Table 3 shows the results achieved by several commonly used network forensic tools (Wireshark, NetworkMiner) in the E.4 test environment.

Table 2. Processing speeds of individual network capture processing phases in *standalone* test scenario performed on test environments E.4 and E.3.

	Workstation E.4 [Mbps]	**Server** E.3 [Mbps]
PCAP file reading	5103	5719
Packet parsing	3853	1679
L7 Conversation reassembling	942	380
Application protocols parsing	880	358

Table 3. Processing speeds of commonly used network forensic tools measured on test environment Workstation E.4.

	NTPAC [Mbps]	**Netfox** [Mbps]	**Wireshark** [Mbps]	**NetworkMiner** [Mbps]
M57 Analysis	880	65.6	73.4	15.8

[9] https://digitalcorpora.org/corpora/scenarios/m57-patents-scenario (last accessed 2019-07-03).

5.2 Clustered Environments

Next, we compare the performance and scalability of our tool in a *clustered* test scenario executed in the test environment E.1. We have performed a series of experiments with the varying number of active *Load Balancer* and *Reassembler* nodes.

Additionally, we have tested configuration in which the results were persisted in a distributed database[10], as well as the configuration, where these results were discarded so we measured performance without the overhead associated with database operations.

Table 4. Performance measurements of *clustered* processing conducted in test environment E.1.

Reassemblers	**S** [Mbps]	**1** [Mbps]	**2** [Mbps]	**4** [Mbps]	**6** [Mbps]	**8** [Mbps]	**10** [Mbps]
Load Balancers Without Persistence							
1	513	380	670	768	778	797	815
2		310	574	1093	1370	1508	1542
3		290	602	1136	1713	1945	2070
4		269	660	1258	1971	2252	2580
Load Balancers With Persistence							
1	343	273	478	729	734	740	742
2		247	482	801	1009	1123	1254
3		*	501	930	1131	1326	1438
4		*	503	949	1135	1375	1710

Table 4 shows how the performance depends on the number of *Reassembler* nodes. Columns labeled 1 to 10 represent a number of participating *Reassembler* nodes. For comparison, the column labeled S represents system performance in a stand-alone mode of the processing. First set of rows (labeled Load Balancers Without persistence) denote a varying number of participating *Load Balancer* nodes without the results being stored in a database. Similarly, the second set of rows (labeled Load Balancers With Persistence) denote a varying number of participating *Load Balancer* nodes but with results being stored in a database.

In the test results, we see that performance increases to the point where one *Load Balancer* cannot provide enough data for available *Reassembler* nodes. Adding additional *Load Balancer* nodes increases the throughput of the entire system until all *Reassemblers* are fully saturated, and the processing speed reaches its limit again. Increasing a number of both *Load Balancer* and *Reassembler* nodes allows a further increase in overall throughput until the available hardware resources are exhausted. Data points marked with asterisks (*) represent incapability to complete the test run due to the overload of the *Reassembler* nodes in a given configuration (total number of active nodes).

With the knowledge of the characteristics of the distributed system obtained from experiments in the E.1 environment, we repeated the same set of experiments in E.2 (using up to 8 *Reassemblers* and up to 4 *Load Balancers*) and E.3 (using up to 3 *Reassemblers* and single *Load Balancer*). The results shown in tables 5 and 6 show a similar trend in the rate of processing per number of individual modules. Note, that we are limited by the total number of instances that we can create in environment E.2.

Table 5. Performance measurements of *clustered* processing conducted in test environment E.2.

Reassemblers	**S** [Mbps]	**1** [Mbps]	**2** [Mbps]	**4** [Mbps]	**6** [Mbps]	**8** [Mbps]
Load Balancers Without Persistence						
1	427	223	370	560	573	585
2		170	334	706	916	994
3		126	352	734	826	1016
4		104	271	580	618	920
Load Balancers With Persistence						
1	248	171	255	459	497	498
2		*	219	420	459	675
3		*	*	383	452	558
4		*	*	*	*	*

Table 6. Performance measurements of *clustered* processing conducted in test environment E.3.

Reassemblers	**S** [Mbps]	**1** [Mbps]	**2** [Mbps]	**3** [Mbps]
1 — Without Persistence	358	233	407	453
1 — With Persistence	210	158	301	388

When comparing results from different environments, it is interesting that the highest performance was achieved in the local network, although the Google Cloud Platform seems to have more powerful computing nodes and a faster network. This may be because GCP is a virtualized environment with shared hardware resources.

6 Conclusion

We have designed and implemented a system for forensic network analysis that can be used in high-speed networks for near real-time analysis. The distributed system is based on an actor model that, thanks to its good scalability, can run on a single machine as well as a computing cluster.

The proposed distributed system is comprised of different classes of cooperating nodes capable of distributing intercepted network traffic, processing identified network flows and storing reconstructed data into a distributed database. The resulting data consists of a description of network conversations and information from the extracted application communication. At this point, DNS and HTTP are supported.

The main goal of the system is to provide a scalable platform for network communication processing that is primarily designed to support a digital investigation. Experiments have demonstrated the feasibility of the proposed approach.

[10] The number of Cassandra nodes was equal to the number of active *Reassembler* nodes.

Processing throughput is scalable by adding additional processing nodes. Experiments have also shown that the proposed tool running on only one node can effectively use available resources and can offer the same or better performance than existing tools.

The NTPAC is open source, and available at https://github.com/nesfit/NTPAC under the MIT license.

References

[1] G. Alotibi, N. Clarke, Fudong Li, and S. Furnell. 2016. User profiling from network traffic via novel application-level interactions. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*. 279–285. https://doi.org/10.1109/ICITST.2016.7856712

[2] M. Aupetit, Y. Zhauniarovich, G. Vasiliadis, M. Dacier, and Y. Boshmaf. 2016. Visualization of actionable knowledge to mitigate DRDoS attacks. In *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*. 1–8. https://doi.org/10.1109/VIZSEC.2016.7739577

[3] Nicole Beebe. 2009. Digital forensic research: The good, the bad and the unaddressed. In *IFIP International Conference on Digital Forensics*. Springer, 17–36.

[4] MI Cohen. 2008. PyFlag–An advanced network forensic framework. *Digital investigation* 5 (2008), S112–S120.

[5] Sherri Davidoff and Jonathan Ham. 2012. *Network Forensics: Tracking Hackers through Cyberspace.* Prentice Hall.

[6] Alice Este, Francesco Gargiulo, Francesco Gringoli, Luca Salgarelli, and Carlo Sansone. 2008. Pattern recognition approaches for classifying ip flows. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 885–895. https://doi.org/10.1007/978-3-540-89689-0_92

[7] Carl Hewitt, Peter Bishop, and Richard Steiger. 1973. Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence. In *Advance Papers of the Conference*, Vol. 3. Stanford Research Institute, 235.

[8] L. Kekely, J. Kučera, V. Puš, J. Kořenek, and A. V. Vasilakos. 2016. Software Defined Monitoring of Application Protocols. *IEEE Trans. Comput.* 65, 2 (Feb 2016), 615–626. https://doi.org/10.1109/TC.2015.2423668

[9] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz, and Iftikhar Ahmad. 2016. Network forensics: Review, taxonomy, and open challenges. *Journal of Network and Computer Applications* 66 (2016), 214 – 235. https://doi.org/10.1016/j.jnca.2016.03.005

[10] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz, and Iftikhar Ahmad. 2016. Network forensics: Review, taxonomy, and open challenges. *Journal of Network and Computer Applications* 66 (2016), 214–235.

[11] Mark Longworth, John D Abromavage, Todd A Moore, Scott V Totman, and Vince Romano. 2006. System and method for network security. US Patent 7,016,951.

[12] Alexey Lukashin, Leonid Laboshin, Vladimir Zaborovsky, and Vladimir Mulukha. 2014. Distributed Packet Trace Processing Method for Information Security Analysis. In *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, Sergey Balandin, Sergey Andreev, and Yevgeni Koucheryavy (Eds.). Springer International Publishing, Cham, 535–543.

[13] Petr Matoušek, Jan Pluskal, Ondřej Ryšavý, Vladimír Veselý, Martin Kmet’, Filip Karpíšek, and Martin Vymlátil. 2015. Advanced techniques for reconstruction of incomplete network data. In *International Conference on Digital Forensics and Cyber Crime*. Springer, 69–84.

[14] Erik E Northrop and Heather R Lipford. 2014. Exploring the usability of open source network forensic tools. In *Proceedings of the 2014 ACM Workshop on Security Information Workers*. ACM, 1–8.

[15] Emmanuel S Pilli, Ramesh C Joshi, and Rajdeep Niyogi. 2010. Network forensic frameworks: Survey and research challenges. *digital investigation* 7, 1-2 (2010), 14–27.

[16] Jan Pluskal, Ondrej Lichtner, and Ondřej Ryšavý. 2018. Traffic Classification and Application Identification in Network Forensics. In *IFIP International Conference on Digital Forensics*. Springer, 161–181.

[17] Libor Polčák and Barbora Franková. 2015. Clock-Skew-Based Computer Identification: Traps and Pitfalls. *J. UCS* 21, 9 (2015), 1210–1233.

[18] Libor Polčák, Radek Hranický, and Tomáš Martínek. 2014. On Identities in Modern Networks. *Journal of Digital Forensics, Security and Law* 9, 2 (2014), 2.

[19] N. Promrit and A. Mingkhwan. 2015. Traffic Flow Classification and Visualization for Network Forensic Analysis. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*. 358–364. https://doi.org/10.1109/AINA.2015.207

[20] Gil Raviv. 2013. Methods for user profiling for detecting insider threats based on internet search patterns and forensics of search keywords. US Patent 8,375,452.

[21] Wei Ren. 2004. On A Reference Model of Distributed Cooperative Network, Forensics System.. In *iiWAS*.

[22] Wei Ren and Hai Jin. 2005. Distributed agent-based real time network intrusion forensics system architecture design. In *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, Vol. 1. IEEE, 177–182.

[23] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446

[24] Rommel Sira. 2003. Network forensics analysis tools: an overview of an emerging technology. *GSEC, version* 1 (2003), 1–10.

[25] Robin Sommer, Johanna Amann, and Seth Hall. 2016. Spicy: a unified deep packet inspection framework for safely dissecting all your data. *Acsac* (2016). https://doi.org/10.1145/2991079.2991100

[26] Matthias Vallentin, Dominik Charousset, Thomas C Schmidt, Vern Paxson, and Icsi U C Berkeley. 2014. Native Actors : How to Scale Network Forensics. *Sigcomm 2014* (2014). https://doi.org/10.1145/2619239.2631471

[27] Matthias Vallentin, Vern Paxson, and Robin Sommer. 2016. VAST: A Unified Platform for Interactive Network Forensics.. In *NSDI*. 345–362.

[28] Matthias Vallentin, Robin Sommer, Jason Lee, Craig Leres, Vern Paxson, and Brian Tierney. 2007. The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 107–126.

[29] Diangang Wang, Tao Li, Sunjun Liu, Jianhua Zhang, and Caiming Liu. 2007. Dynamical network forensics based on immune agent. In *Natural Computation, 2007. ICNC 2007. Third International Conference on*, Vol. 3. IEEE, 651–656.

[30] M. Wullink, G. C. M. Moura, M. MÃijller, and C. Hesselman. 2016. ENTRADA: A high-performance network traffic data streaming warehouse. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 913–918. https://doi.org/10.1109/NOMS.2016.7502925

A.5 A Scalable Architecture for Network Traffic Forensics

Viliam Letavay, Jan Pluskal, and Ondřej Ryšavý. “A Scalable Architecture for Network Traffic Forensics”. In: *The Fifteenth International Conference on Networking and Services ICNS 2019.* Athens, GR: The International Academy, Research and Industry Association, 2019, pp. 32–36. ISBN: 9781612087115

A Scalable Architecture for Network Traffic Forensics

Viliam Letavay
Faculty of Information Technology
Brno University of Technology
Brno 61266, CZ
Email: iletavay@fit.vutbr.cz

Jan Pluskal
Faculty of Information Technology
Brno University of Technology
Brno 61266, CZ
Email: ipluskal@fit.vutbr.cz

Ondřej Ryšavý
Faculty of Information Technology
Brno University of Technology
Brno 61266, CZ
Email: rysavy@fit.vutbr.cz

***Abstract*—The availability of high-speed Internet enables new opportunities for various cybercrime activities. Security administrators and Law Enforcement Agency (LEA) officers call for powerful tools capable of providing network communication analysis of an enormous amount of network traffic as well as capable of analyzing an incomplete network data. Big data technologies were considered to implement tools for capturing, processing and storing packet traces representing network communication. Often, these systems are resource intensive requiring a significant amount of memory, computing power, and disk space. The presented paper describes a novel approach to real-time network traffic processing implemented in a distributed environment. The key difference to most existing systems is that the system is based on a light-weight actor model. The whole processing pipeline is represented in terms of actor nodes that can run in parallel. Also, the actor-model offers a solution that is highly configurable and scalable. The preliminary evaluation of a prototype implementation supports these general statements.**

***Keywords*–Network forensic analysis; Network traffic processing; Actor model.**

I. Introduction

The expansion of computer networks and Internet availability opens new opportunities for cybercrime activities and increases the number of security incidents associated with network applications. The number of connected devices grows, and traffic speed increases. Security administrators and Law Enforcement Agency (LEA) officers call for powerful tools that enable them to extract useful information from network communication [1]. The network forensics that is responsible for capturing, collecting and network data analyzing is becoming more important [2].

In the forensic investigation, the network traffic is continuously captured from multiple sources. The captured network data has a form of packet traces that have to be processed and analyzed up to the application layer. The network forensic tool has to decode protocols at different network layers of the Transmission Control Protocol/Internet Protocol (TCP/IP) model and various encapsulations. For LEA officers, interesting information lies in application messages, such as instant messaging, emails, voice, localizable information, documents, pictures, etc. The form and relevance of extracted artifacts may differ from case to case. Often, communication is encrypted. In this case, meta-data can be the only piece of information available. In all cases, the network forensic processing system has to be able to extract artifacts from the network traffic reliably, even if the packet capture is corrupted, for instance, some connections are incomplete, packets are malformed, or chunks of packets were not recorded because of capturing device issues.

The amount of data that needs to be processed to extract evidence from the network communication depends on the kind of a case that is investigated but usually gets large. It is very difficult to decode, extract and store the immense mass of information for further processing. We propose a distributed network forensic framework based on the *actor model* that is computation effective and capable of linear scalability. Scalable properties of *actor model* design for network forensics are promising, as shown by the Visibility Across Space and Time (VAST) platform [3]. Similarly to VAST, our solution provides real-time data ingestion and interactive data analysis, but in addition to VAST, we consider the full artifact extraction up to the application layer. Although it requires more computation resources, we demonstrate that it can still be achieved in a more straightforward and less resource consuming environment compared to Apache Hadoop technology, which is the norm for big data processing.

In Section II, we describe tools used by network forensics practitioners. Section III addresses issues faced by investigators and our proposed solution, which architecture is broadly discussed in Section IV. Section V evaluates preliminary performance results, and Section VI concludes the paper.

II. Background & Related Work

Network forensics is a process that identifies, captures and analyzes network traffic. Network forensic techniques are used by several network forensic frameworks [4]–[9] and tools intended for intrusion detection (Zeek, VAST, Moloch) [10]–[12], network security monitoring (Microsoft Network Monitor, TShark, Wireshark, tcpdump) [13]–[16], and network forensic investigation for LEAs (Netfox Detective, PyFlag, NetworkMiner, EnCase, XPlico) [17]–[21]. Commonly available forensics tools are implemented either as a classic desktop or command line application or a traditional client-server solution.

To overcome the limitations of traditional tools, we propose to use distributed computing. The models for distributed processing [22][23] are more suitable for real-time network forensic analysis from multiple sources, such as logs and captured communication. The models are based on an agent system, where numerous agents perform the collection task. The extracted information is sent to the forensic network server and analyzed on this single node [24] only. The *forensic server* is the bottleneck that has to process all the data. To avoid this bottleneck, the Google Rapid Response (GRR) [25], a live forensic system, utilizes a cluster of servers. The system deploys agents running on users' computers that provide access to forensic information, e.g., remote raw disk and memory access. Processing of forensic data is done as flows. Each flow is maintained on the server. Server nodes run workers that process the active flows. Adding more server nodes enables to run more workers and thus it is possible to handle more clients simultaneously.

Elimination of bottlenecks in the architecture offers scalability and improved reliability. The *actor model* [26] is one of the attractive solutions that address the problem elegantly and efficiently. It comes with a separate unit called an *actor*. Actors execute independently and in parallel. They communicate with each other asynchronously via message passing, and their state is otherwise immutable. Actors are capable of spawning new actors, forming a parent-child relationship, allowing the creation of a tree-like structure of actors. Actor's current behavior determines how it processes the incoming messages. Every actor in an actor system is uniquely identified by an address which other actors use as destinations of the messages they want to send out. This address can identify actors at the local machine and also the ones at the remote machines, allowing easy means of communication between nodes of a cluster. Compared to another similar programming model, the *Communicating Sequential Processes* (CSP) [27], elementary units of computation – processes are anonymous and communicate with each other via established communication channels. The actor system is the key enabler for the VAST system [3]. In VAST, actors implement importing, archiving, indexing and exporting processed data. Actors live in nodes that map to system processes. The system scales by creating more nodes either on the single machine or a cluster of computers.

Moloch is another tool, worth to mention, that uses principles of distributed computing for massive scale network traffic monitoring, full packet capturing and indexing [12]. Moloch system consists of sensors that capture the communication and Elasticsearch database that is a distributed search and analytics engine. The system scales by adding new nodes running Elasticsearch instances.

III. Problem Statement and Solution

Our goal is to design and create a system capable of long-term, high-speed, real-time network traffic filtering and processing up to the application layer. The software solution should be scalable and hardware independent. To achieve this, we have to deal with the challenges elaborated in the rest of this section.

A. Architectural Design

How to create a system for packet filtering and analysis of communication that can identify application protocols, gets forensics artifacts and searches through them?

Network forensics is a tedious work that strictly relies on completeness and precision of all undertaken steps to gain a piece of a puzzle that fits together as a shred of evidence. Considering the current speeds of regular users' home network connection(s), a comprehensive classical analysis on a single machine would require enormous computation resources. Try to imagine, that each network packet would be analyzed by many protocol dissectors with a goal to extract, for example, an acknowledgment of email delivery. To achieve this goal, with optimal computational resources, we must revisit currently utilized methods and redesign them to work in a distributed environment which brings new challenges to architecture design, application of algorithms, data synchronization, and so on.

B. Scalability on Commodity Hardware

How can the solution be scalable and hardware independent despite the hardware limitations?

Let us consider this imaginary demonstration. The math is simple, one computer with $1\,\mathrm{Gbps}$ Network Interface Card (NIC) that has a relatively simple task to capture traffic during full line load would be required to write to a disk under the constant speed of $1000\mathrm{Mbps} \approx 125\,\mathrm{MB/s}$. Our system has to guarantee that no data loss occurs during the capture. A suspect can simultaneously download and upload data which means that the monitoring device cannot have only one $1 * 1\,\mathrm{Gbps}$ NIC, but it needs $2 * 1\,\mathrm{Gbps}$ cards, one for uplink, one for downlink. Thus, the required speed of continuous disk writing would be $2 * 125\,\mathrm{MB/s} \approx 250\,\mathrm{MB/s}$. Now, if the requirement is to store the communication for one day, the disk capacity has to be $250\,\mathrm{MB/s} * 86\,400\,\mathrm{s} \approx 21.6\,\mathrm{TB}$. This is achievable with commodity hardware, e.g., $2 * 12\,\mathrm{TB}$ drives with Redundant Array of Inexpensive Disks (RAID) 0 or $4 * 12\,\mathrm{TB}$ with RAID 1+0 — assuming higher write/read speed than $250\,\mathrm{MB/s}$. However, what if only one day is not enough? For a typical forensic case, capturing period spawns through weeks or months.

From our previous experiments, we know that a single computation node is limited and commodity hardware is hardly sufficient to perform all required operations in real-time and over long periods. Separation of frames into a conversation which requires a dissection of the network protocols up to the application layer, which speed is roughly $300\,\mathrm{Mbps}$ [28, pp. 45-51] is not sufficient. On the other hand, we are confident that the application created and optimized for this singular purpose can do the processing faster and breach the $1\,\mathrm{Gbps}$ line speed. Nevertheless, we do not believe that a single machine solution with commodity hardware is capable of doing overall analysis and extraction of information from the application layer. We have to design our solution as a distributed system across multiple machines.

C. Overall Performance

What scalability and acceleration of data processing can be achieved?

The proposed solution is based on the actor model. Each actor represents an independent processing unit. The communication between actors is managed by messaging. Actors have no shared state; thus all of them can work in parallel. If actors run on the same node, the message passing has little additional overhead compared to a function call or a loop. However, if actors scale over multiple nodes, messages need to be serialized. This process introduces latency and consumes part of the processing power. The scalability of the actor model is linear [3].

IV. Architectural Design

Incomplete data provided by unreliable traffic interception can lead to inaccurate results; some information may be lost, some fabricated by reconstruction process [29]. Keeping the above facts in mind, the processing cannot strictly follow Requests for Comments (RFCs) and behave like a *kernel* network stack implementation, but it has to incorporate several heuristics. For example, to fill missing gaps in data, and to consider these fillings during application protocol processing, or never to join multiple frames into a single conversation unless it passes more advanced heuristic-based checks. Network forensic tools that we have worked with do mostly respect RFCs and thus may produce misleading results, as shown by Matousek et al. [29].

We propose a distributed architecture composed of commodity hardware that will be capable of linear scalability, and capable of efficient resource utilization. The overall architecture is shown in Figure 1.

At the top level, we have divided the entire process into the two main stages:

- Data preprocessing — The reconstruction of conversations at the application layer (L7) of the TCP/IP model. This process consists of consecutive segregation of captured communication into the internet (L3) and transport (L4) conversations and deploying a reassembling heuristics [29] to recognize individual L7 conversations inside a parent L4 conversations and to reassemble their payloads with respect to data loss, reordering or duplication. Every L7 conversation holds information about the source and destination endpoints (IP addresses, ports), timestamps, type of transport protocol (UDP or TCP) and reassembled payloads of exchanged application messages.
- Data analysis — The analysis of each application conversation consists of the identification of the application protocol, and extraction of application events, e.g., visited web pages, exchanged emails, domain name queries, etc., with proper application protocol dissector that yields sets of forensic artifacts.

A. Data Prepossessing

The First stage is executed on a set of independent *Reassembler* nodes. These reconstruct L7 conversations from the stream of captured packets which can originate from *Packet Capture (PCAP) files* or can be captured from *the live network interface*.

In the most common use-case, we have one source stream (i.e., one PCAP file) which we want to analyze. Therefore, to utilize multiple *Reassembler* instances, we have to split packets from this stream into smaller sub-streams, which will be distributed among available *Reassembler* instances. For this split, we cannot use a naive method such as *Round Robin*, because *Reassembler* nodes operate independently of each other and to fully reconstruct L7 conversation a particular *Reassembler* has to obtain all the pieces of that particular L7 conversation. In case we would use *Round Robin*, a situation could occur when half the packets from one L7 conversation would end up in one *Reassembler* node and the second half in another; both nodes would have incomplete data and none of them would be able to reconstruct the conversation entirely.

Our proposed solution to this problem is another type of node – *L4 Load Balancer*, which will be positioned in front of the *Reassembler* nodes and which, as a name suggests, distributes packets based on their associations to L4 conversations each of which can consist of multiple L7 conversations. *L4 Load Balancer* extracts source and destination IP addresses and ports and transport protocol from each packet of the source stream and uses this information to decide to which instance from the available *Reassemblers* should it forward to. This way, all packets of a particular L7 conversation will always be forwarded to only one *Reassembler* instance.

Reassemblers build a tree-like structure of L3 and L4 conversations which are represented by the actors. Each received packet is first forwarded to an appropriate L3 conversation actor, which in turn forwards it further down to an appropriate L4 conversation actor which reassembles L7 conversations. This segregation of packets into the individual L4 conversations before actual L7 conversation reassembling is required, as implemented reassembling heuristics expect to operate on packets from a single L4 conversation at the time. The use of a hierarchical actor design allows us to perform independent portions of the processing in parallel and also to easily implement management strategies such as passing management messages to a particular L3 conversation actor and its children L4 conversation actors. The reconstructed L7 conversations are stored in a distributed database, ready to be retrieved in the second stage of the execution.

B. Data Analysis

In the *second stage*, a subset of reconstructed L7 conversations is retrieved from the distributed database and delivered to the *Application protocol dissector* nodes. For every L7 conversation, *Application protocol dissector* nodes identify the used application protocol and use a proper dissector module dedicated to the processing of a single application protocol, such as Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP) or Domain Name System (DNS), to extract application protocol messages from this L7 conversation. Obtained data are stored back into the distributed database. Processing of application messages is under normal circumstances possible only with unencrypted network communication. From Secure Sockets Layer/Transport Layer Security (SSL/TLS) communication which encapsulates application protocols, such as HTTP, we can extract only unencrypted portions of this data such as the server's cryptographic certificate. Possible ways to decrypt and subsequently, parse the SSL/TLS communication is to own a private key of a given SSL/TLS server or to deploy an SSL/TLS intercepting proxy [30].

V. Preliminary Evaluation

Our prototype implementation is based on C# actor system library *Akka.NET*. For testing and performance benchmarking, we have implemented two modes of operation:

1) *Offline* — isolated execution which combines the functionality of a single *L4 Load Balancer* and *Reassembler* node inside a single system's process. No inter-actor message serialization is therefore required.
2) *Online* — distributed execution spanning across multiple cluster nodes. The inter-actor message serialization is required as messages destined to remote

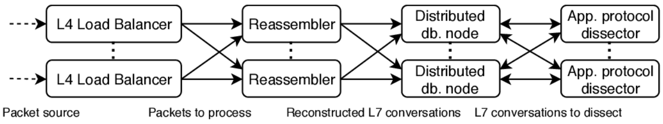


Figure 1: Architecture diagram showing the proposed system nodes with information flow between them.

actors (nodes) have to leave an originating system's process and be transmitted over a computer network in a serialized format. This introduces additional latency and performance overhead.

Additionally, for proof-of-concept benchmarking, the functionality of *Application protocol dissector* nodes was included inside *Reassembler* nodes to eliminate distributed database as a middleman between them. In the following measurements, we focus on a raw network capture's processing performance of the so-far naive implementation. Currently, our prototype implementation supports the dissection of two application protocols (DNS and HTTP).

We have measured the preliminary performance of the implementation on two different hardware configurations:

- *Workstation* — Intel i7-5930K 4.3 GHz, 12 cores, 64 GB RAM, 512 GB SSD
- *Mini-cluster* — 4x servers with Intel Xeon E5520, 2.26 GHz, 8 cores, 48 GB RAM, 1 TB SSD, 1 Gbps network

We used a public data set of M57-Patents Scenario [31], that consists of real-world data captured over a month. We merged all network traces into one PCAP file of roughly 4.8 GB and 5,707,845 frames. One large PCAP file simulates our use-case of streamed-in communication that needs to be load-balanced from a single node.

We started with measurements in an *offline* mode on a single machine, firstly with a PCAP file parsing operation and incrementally added consequent operations and measured processing speeds, as Table 1 describes. Preliminary evaluation suggests that the *raw speed* of roughly 3.8 Gbps, for PCAP file reading and packet parsing is sufficient. The process of reconstructing L7 conversations that segregates IP flows by packet source and destination IP addresses, ports and transport protocol type with additional heuristics [29], that also reassembles TCP/UDP streams, is computationally heavier, reaching ”only” 942 Mbps, and is about 4x slower than only read and parsing. With added HTTP & DNS dissection, performance slightly decreased further down to 880 Mbps.

TABLE 1. PROCESSING SPEEDS OF OUR OFFLINE TEST SCENARIO ON A SINGLE MACHINE

	Workstation [Mbps]	**Mini-cluster node** [Mbps]
PCAP file reading	5103	5719
Packet parsing	3853	1679
L7 Conversation tracking	942	380
HTTP & DNS extraction	880	358

The *CPU frequency* (performance per CPU core) plays a very important part in overall performance, that can be observed if we compare our *Workstation* with node from *Mini-cluster* — 880 Mbps vs. 358 Mbps. All other components except CPUs are otherwise roughly comparable as we can see by comparing the speed of ”PCAP file reading”.

The scalability is described in Table 2 that shows performance in *online* mode. The solution was deployed on *Mini-cluster*. The first node was reading the captured communication from a PCAP file and load-balancing it to the rest that reassembled L7 conversations and extracted HTTP and DNS artifacts. In the measurements, we can see an increase in the performance with each added *Reassembler*. When compared with the results in Table 1, the performance of a distributed processing at the *Mini-cluster* exceeded that of a single node running in an *offline* mode. Nevertheless, further optimization is required to achieve linear scalability as a single *L4 Load Balancer* fails to fully saturate available *Reassemblers* by distributing the packets fast enough. We have observed that serialization of messages containing the packets to process heavily contributes to the overall computational complexity and easily becomes a bottleneck of our solution.

TABLE 2. PROCESSING SPEEDS OF OUR ONLINE TEST SCENARIO MEASURED ON MINI-CLUSTER

Reassemblers count	**One** [Mbps]	**Two** [Mbps]	**Three** [Mbps]
HTTP & DNS extraction	233	407	453

We compare our solution, called Network Traffic Processing & Analysis Cluster (NTPAC), running in the *offline* mode at the *Workstation* with commonly used network forensic tools in Table 3. Our solution is an order of magnitude faster while delivering a comparable amount of results in terms of reconstructing L7 conversations and extracting HTTP and DNS artifacts.

TABLE 3. PROCESSING SPEEDS OF COMMONLY USED NETWORK FORENSIC TOOLS MEASURED ON WORKSTATION

NTPAC [Mbps]	**Netfox** [Mbps]	**Wireshark** [Mbps]	**NetworkMiner** [Mbps]
880	65.6	73.4	15.8

VI. Conclusion

In this research, we proposed a system for distributed real-time forensic network traffic analysis up to the application layer capable of large-scale communication processing. We intend to create a system based on the actor model that scales linearly and is hardware independent. The implementation environment of the .NET Core framework and C# language enables rapid development compared to C/C++ that is used by VAST and Moloch. Also, our solution is multiplatform and easily staged with Docker Swarm. Therefore, the deployment of the entire distributed application at the computation cluster is reduced to one command. The solution is distributed under the MIT License and hosted as an open-source project on GitHub here [32].

In the near future, we plan to measure the performance of our solution using data from real-world cases. Because of legal reasons, deployment to public cloud infrastructure is out of the question. Therefore, we need to build a private one that consists of nodes with high CPU frequencies and 10 Gbps network interfaces. Additionally, we need to profile and optimize processing and distribution mechanisms, to expand the set of protocols supported by application protocol dissectors and to add support for tunneling mechanisms.

VII. Acknowledgement

This work was supported by BUT project ”ICT Tools, Methods and Technologies for Smart Cities” (2017-2019), no. FIT-S-17-3964.

References

[1] N. Beebe, “Digital forensic research: The good, the bad and the unaddressed,” in IFIP International Conference on Digital Forensics. Springer, 2009, pp. 17–36.

[2] E. S. Pilli, R. C. Joshi, and R. Niyogi, “Network forensic frameworks: Survey and research challenges,” digital investigation, vol. 7, no. 1-2, 2010, pp. 14–27.

[3] M. Vallentin, “Scalable network forensics,” Ph.D. dissertation, UC Berkeley, 2016.

[4] S. Rekhis, J. Krichene, and N. Boudriga, “Digfornet: digital forensic in networking,” in IFIP International Information Security Conference. Springer, 2008, pp. 637–651.

[5] A. Almulhem and I. Traore, “Experience with engineering a network forensics system,” in International Conference on Information Networking. Springer, 2005, pp. 62–71.

[6] W. Wang and T. E. Daniels, “A graph based approach toward network forensics analysis,” ACM Transactions on Information and System Security (TISSEC), vol. 12, no. 1, Oct. 2008, pp. 4:1–4:33.

[7] N. L. Beebe and J. G. Clark, “A hierarchical, objectives-based framework for the digital investigations process,” Digital Investigation, vol. 2, no. 2, 2005, pp. 147–167.

[8] S. Perumal, “Digital forensic model based on malaysian investigation process,” International Journal of Computer Science and Network Security, vol. 9, no. 8, 2009, pp. 38–44.

[9] W. Halboob, R. Mahmod, M. Abulaish, H. Abbas, and K. Saleem, “Data warehousing based computer forensics investigation framework,” in 2015 12th International Conference on Information Technology-New Generations (ITNG). IEEE, 2015, pp. 163–168.

[10] Zeek, [retrieved: April, 2019]. [Online]. Available: https://www.zeek.org/

[11] Vast, [retrieved: April, 2019]. [Online]. Available: http://vast.io/

[12] Moloch, [retrieved: April, 2019]. [Online]. Available: https://molo.ch/

[13] Microsoft Network Monitor, [retrieved: April, 2019]. [Online]. Available: https://support.microsoft.com/en-us/help/933741/information-about-network-monitor-3

[14] TShark, [retrieved: April, 2019]. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[15] Wireshark, [retrieved: April, 2019]. [Online]. Available: https://www.wireshark.org/

[16] TCPDUMP, [retrieved: April, 2019]. [Online]. Available: https://www.tcpdump.org/

[17] Netfox Detective, [retrieved: April, 2019]. [Online]. Available: https://github.com/nesfit/NetfoxDetective

[18] PyFlag, [retrieved: April, 2019]. [Online]. Available: https://github.com/py4n6/pyflag

[19] NetworkMiner, [retrieved: April, 2019], https://www.netresec.com/?page=NetworkMiner.

[20] EnCase, [retrieved: April, 2019]. [Online]. Available: https://www.guidancesoftware.com/encase-forensic

[21] XPlico, [retrieved: April, 2019]. [Online]. Available: https://www.xplico.org/

[22] W. Ren and H. Jin, “Distributed agent-based real time network intrusion forensics system architecture design,” in Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on, vol. 1. IEEE, 2005, pp. 177–182.

[23] D. Wang, T. Li, S. Liu, J. Zhang, and C. Liu, “Dynamical network forensics based on immune agent,” in Natural Computation, 2007. ICNC 2007. Third International Conference on, vol. 3. IEEE, 2007, pp. 651–656.

[24] S. Khan, A. Gani, A. W. A. Wahab, M. Shiraz, and I. Ahmad, “Network forensics: Review, taxonomy, and open challenges,” Journal of Network and Computer Applications, vol. 66, 2016, pp. 214–235.

[25] M. Cohen, D. Bilby, and G. Caronni, “Distributed forensics and incident response in the enterprise,” Digital Investigation, vol. 8, 2011, pp. S101 – S110, the Proceedings of the Eleventh Annual DFRWS Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287611000363

[26] C. Hewitt, P. Bishop, and R. Steiger, “A universal modular actor formalism for artificial intelligence,” in Proceedings of the 3rd International Joint Conference on Artificial Intelligence, ser. IJCAI’73. Morgan Kaufmann Publishers Inc., 1973, pp. 235–245.

[27] C. A. R. Hoare, “Communicating sequential processes,” Commun. ACM, vol. 21, no. 8, Aug. 1978, pp. 666–677.

[28] J. Pluskal, “Framework for captured network communication processing,” Master’s thesis, FIT BUT, 2014.

[29] P. Matoušek et al., “Advanced techniques for reconstruction of incomplete network data,” in International Conference on Digital Forensics and Cyber Crime. Springer, 2015, pp. 69–84.

[30] S. Davidoff and J. Ham, Network Forensics: Tracking Hackers through Cyberspace. Prentice Hall, 2012.

[31] M57-Patents Scenario, [retrieved: April, 2019]. [Online]. Available: https://digitalcorpora.org/corpora/scenarios/m57-patents-scenario

[32] NTPAC, [retrieved: April, 2019]. [Online]. Available: https://github.com/nesfit/NTPAC/

A.6 Automated Man-in-the-MiddleAttack Against Wi-Fi Networks

Martin Vondráček, Jan Pluskal, and Ondřej Ryšavý. “Automated Man-in-the-Middle Attack Against Wi-Fi Networks”. In: *The Journal of Digital Forensics, Security and Law: JDFSL* 13.1 (2018), pp. 59–80. ISSN: 1558-7215

THE JOURNAL OF
DIGITAL FORENSICS,
SECURITY AND LAW

Journal of Digital Forensics,
Security and Law

Volume 13 | Number 1 Article 9

3-31-2018

Automated Man-in-the-Middle Attack Against Wi-Fi Networks

Martin Vondráček
Brno University of Technology, Brno, Czech Republic, xvondr20@stud.fit.vutbr.cz

Jan Pluskal
Brno University of Technology, Brno, Czech Republic, ipluskal@fit.vutbr.cz

Ondřej Ryšavý
Brno University of Technology, Brno, Czech Republic, rysavy@fit.vutbr.cz

Recommended Citation
Vondráček, Martin; Pluskal, Jan; and Ryšavý, Ondřej (2018) "Automated Man-in-the-Middle Attack Against Wi-Fi Networks," *Journal of Digital Forensics, Security and Law*: Vol. 13 : No. 1 , Article 9.
DOI: https://doi.org/10.15394/jdfsl.2018.1495
Available at: https://commons.erau.edu/jdfsl/vol13/iss1/9

Automated Man-in-the-Middle Attack Against Wi-Fi Networks

Cover Page Footnote

This paper is an extended version of the original paper that has been presented at the 9th EAI International Conference on Digital Forensics and Cyber Crime (Vondráček, Pluskal, & Ryšavý, 2018). This work was supported by Ministry of Interior of the Czech Republic project "Integrated platform for analysis of digital data from security incidents" VI20172020062; Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science" LQ1602; and by BUT internal project "ICT tools, methods and technologies for smart cities" FIT-S-17-3964.

AUTOMATED MAN-IN-THE-MIDDLE ATTACK AGAINST WI-FI NETWORKS

Martin Vondráček Jan Pluskal Ondřej Ryšavý
Brno University of Technology
Faculty of Information Technology
Božetěchova 2, Brno, Czech Republic
{xvondr20}@stud.fit.vutbr.cz, {ipluskal,rysavy}@fit.vutbr.cz

ABSTRACT

Currently used wireless communication technologies suffer security weaknesses that can be exploited allowing to eavesdrop or to spoof network communication. In this paper, we present a practical tool that can automate the attack on wireless security. The developed package called *wifimitm* provides functionality for the automation of *MitM* attacks in the wireless environment. The package combines several existing tools and attack strategies to bypass the wireless security mechanisms, such as WEP, WPA, and WPS. The presented tool can be integrated into a solution for automated penetration testing. Also, a popularization of the fact that such attacks can be easily automated should raise public awareness about the state of wireless security.

Keywords: Man-in-the-Middle attack, accessing secured wireless networks, password cracking, dictionary personalization, tampering network topology, impersonation, phishing

1. INTRODUCTION

Recent enhancements to wireless technology strengthen the benefits of wireless communication. It is convenient to access the network from any location within the network coverage area. For most of the portable devices, this is the only way to connect to the network. Installation and network setup are easy, and the network is further expandable. The main benefit of Wi-Fi, its accessibility, makes this technology a suitable target of attacks. A potential attacker needs to be in the physical proximity of a Wi-Fi network. The proposed wireless security standards aim at prevention of such unauthorized access. Unfortunately, the first standard called WEP is so weak that it is possible to crack the password in a few seconds using a conventional laptop computer. The answer was the introduction of stronger standard WPA and later even stronger WPA2. In 2017, Mathy Vanhoef announced that he discovered a vulnerability in security mechanisms that use the four-way handshake (WPA and WPA2) and demonstrated how easily this vulnerability can be exploited.

The main focus of this paper is security of wireless networks. It provides a study of widely used network technologies and mechanisms of wireless security. Analyzed tech-

[1] This paper is an extended version of the original paper that has been presented at the 9th EAI International Conference on Digital Forensics and Cyber Crime (Vondráček, Pluskal, & Ryšavý, 2018).

nologies and security algorithms suffer weaknesses that can be exploited to perform Man-in-the-Middle attacks. A successful realization of this kind of attack allows not only to eavesdrop on all the victim's network traffic but also to spoof his communication (Prowell, Kraus, & Borkin, 2010, pp. 101–120; Callegati, Cerroni, & Ramilli, 2009).

In an example scenario (Figure 1), the victim is a suspect conducting illegal activity on a target network. The attacker is a law-enforcement agency investigator with appropriate legal authorization to intercept the suspect's communication and to perform a direct attack on the network. In some cases, the suspect may be aware that his communication can be intercepted by the Internet Service Provider and harden his network. For example, he could use an overlay network technology, e.g., *VPN* (implemented by *L2TP*, *IPsec* (Kent & Seo, 2005, pp. 09–10), *PPTP*) or anonymization networks (Tor, I2P, etc.) to create an encrypted tunnel configured on his gateway, for all his external communication. This concept is easy to implement and does not require any additional configuration on endpoint devices. Generally, this would not be considered a properly secured network (Godber & Dasgupta, 2003, pp. 425-431), but this scheme, or similar, is often used by large vendors like Cisco (Deal & Cisco Systems, 2006) or Microsoft (Thomas, 2017) for branch office deployment and can also be seen in home routers[1]. In such cases, intercepting traffic on the ISP level would not yield meaningful results, because all the communication is encrypted by the hardening. On the other hand, direct attack on the suspect's LAN will intercept plain communication. But, even when an investigator is legally permitted to carry out such an attack to acquire evidence, it is scarcely used, because it requires expert domain knowledge. Thus, this process of evidence collection is very expensive and human resource demanding.

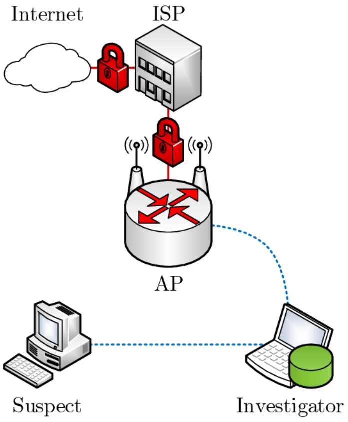


Figure 1. Example forensics scenario where the suspect has hardened his network and uses an encrypted tunnel from the gateway (*AP*).

The aim of this research is to design, implement and test a tool able to automate the process of accessing a secured *WLAN* and to perform data interception. Furthermore, this tool should be able to tamper with the network to collect more evidence by redirecting traffic to place itself in the middle of the communication and tamper with it, to access otherwise encrypted data in plain form. Using the automated tool should not require any expert knowledge from the investigator.

We designed a generic framework, see Figure 3, capable of accessing and acquiring evidence from a wireless network regardless of used security mechanisms. This framework can be split into several steps. First,

[1] Asus RT-AC5300 – Merlin WRT has an option to tunnel all traffic thought Tor.

it is necessary for an investigator to obtain access to the *WLAN* used by the suspect. Therefore, this research focuses on exploitable weaknesses of particular security mechanisms, see Section 2 for more details. Upon successful connection to the network, the investigator needs to tamper with the network topology. For this purpose, weaknesses of several network technologies can be exploited. From this point on, the investigator can start to capture and break the encryption on the suspect's communication.

Specialized tools focused on exploiting individual weaknesses in security mechanisms currently used by *WLAN*s are already available. There are also specialized tools focused on individual steps of *MitM* attacks. Tools that were analyzed and used in implementation of the *wifimitm* package are outlined in Section 2.

Based on the acquired knowledge, referenced studies and practical experience from manual experiments, authors were able to create an attack strategy which is composed of a suitable set of available tools. The strategy is then able to select and manage individual steps for a successful *MitM* attack tailored to a specific *WLAN* configuration. This strategy also includes options for impersonation and phishing for situations, when the network is properly secured, and the weakest part of the overall security is the suspect.

The created software can perform a fully automated attack and requires zero knowledge. We tested the implementation on carefully devised experiments, with available equipment. The tool is open source and can be easily incorporated into other software. The main use cases of this tool are found in automated penetration testing, forensic investigation, and education.

2. SECURITY WEAKNESSES IN WLAN TECHNOLOGIES

Following network technologies (Sections 2.1, 2.2), which find a significant utilization, unfortunately, suffer from security weaknesses in their protocols. These flaws can be used in the process of the *MitM* attack.

2.1 Wireless Security

Wired Equivalent Privacy (*WEP*) is a security algorithm introduced as a part of the IEEE 802.11 standard (Halsall, 2005, p. 665; IEEE-SA, 2012, pp. 1167–1169). Today, *WEP* is deprecated and superseded by subsequent algorithms, but is still sometimes used, as can be seen from Table 1 available from *Wifileaks.cz*[2]. Fluhrer, Mantin, and Shamir (2001) presented that *WEP* is broken. There are tools that provide access to wireless networks secured by *WEP* available (Tews, Weinmann, & Pyshkin, 2007). Regarding *WEP* secured *WLAN*s, authentication can be either *Open System Authentication* (*OSA*) or *Shared Key Authentication* (*SKA*) (IEEE-SA, 2012, pp. 1170–1174). In the case of *WEP OSA*, any *station* (*STA*) can successfully authenticate to the *Access Point* (*AP*) (Robyns, 2014, pp. 4–10). *WEP SKA* provides authentication and security of transferred communication using a shared key. Confidentiality of transferred data is ensured by encryption using the *RC4* stream cipher. Methods used for cracking access to *WEP* secured networks are based on analysis of transferred data with corresponding *Initialization Vectors* (*IV*s).

Wi-Fi Protected Access® (*WPA*^{TM}, a subset of 802.11i) was developed by the Wi-Fi Alliance® as a reaction to increasing number of security flaws in *WEP*. The *WPA* is de-

[2] `http://www.wifileaks.cz/statistika/`

signed to be backward hardware compatible with devices that used *WEP*, and vendors were expected to provide a firmware update to remedy the catastrophic situation with *WEP*. Therefore, for data confidentiality and integrity was chosen *Temporal Key Integrity Protocol* (*TKIP*). The main flaw of *WPA* security algorithm is associated with the *TKIP* and a four-way handshake. It can be identified at the beginning of client device's communication, where an unsecured exchange of confidential information is performed during the handshake. An investigator can obtain this unsecured communication and use it for consecutive cracking of the *Pairwise Master Key* (*PMK*) that is derived from *Pre-Shared Key* (*PSK*) or negotiated using an 802.1x authentication stage in case of enterprise authentication.

Wi-Fi Protected Access® *2* (*WPA2*^{TM}, full implementation of 802.11i) is a successor of *WPA*, but security flaws of the *WPA* algorithm remain significant also for the *WPA2*. Besides *TKIP*, *WPA2* has mandatory support of *Counter Mode CBC-MAC Protocol* (*CCMP*). Both *TKIP* and *CCMP* ensure data confidentiality, authentication, and access control. IEEE 802.11ad adds and 802.11ac extends a new confidentiality protocol *Galois/Counter Mode Protocol* (*GCMP*). Information exposed during the handshake can be once again used for the dictionary attack, which can be further improved by precomputing the *PMKs* (Kumkar, Tiwari, Tiwari, Gupta, & Shrawne, 2012, pp. 37–38; Liu, Jin, & Wang, 2010, p. 3). Precomputed lookup tables are already available online[3].

A critical security flaw in wireless networks secured by *WPA* or *WPA2* is the functionality called *Wi-Fi Protected Setup*^{TM} (*WPS*). This technology provides a comfort-

Table 1. Following table summarizes *WLAN* statistics provided by *Wifileaks.cz*. Users of this service voluntarily scan and publish details about *WLAN*s in the Czech Republic. Information in the table show that a significant number of *WLAN*s still use deprecated security algorithms. The statistics consisting of 97 192 922 measurements of 2 548 054 *WLAN*s were published on May 26, 2017.

Security	Count	Ratio
WPA2	1 429 518	56 %
WEP	393 579	15 %
WPA	375 984	15 %
open	67 388	3 %
other	281 585	11 %

Table 2. Results of wardriving in Bratislava and Brno focused on UPC vulnerabilities concerning default *WPA2 PSK* passwords (Klinec & Svítok, 2016b). Detailed article about these security flaws is available online (Klinec & Svítok, 2016a).

Bratislava, Slovakia, 2016-10-01	Count	Ratio
Total networks	22 172	
UPC networks	3 092	13.95 %
Vulnerable UPC networks	1 327	42.92 % UPC
Brno, Czech Republic, 2016-02-10	Count	Ratio
Total networks	17 516	
UPC networks	2 868	16.37 %
Vulnerable UPC networks	1 835	63.98 % UPC

[3] `https://www.renderlab.net/projects/WPA-tables/`

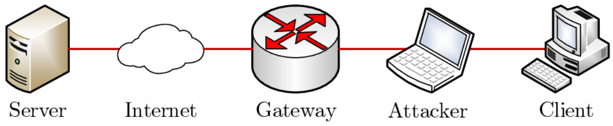


Figure 2. In an example network topology suitable for realization of *MitM* attack, the attacker's device acts towards the victim as a default gateway. All the communication routed outside the local network from the victim is sent to the default gateway, in this case to the attacker's device. From the attacker's device, the communication can be further routed to the real default gateway (Callegati et al., 2009). For the successful execution of this scenario, the attacker needs to be connected to the targeted local network.

able and supposedly secure way of connecting to the network. For a connection to the *WLAN* with *WPS* enabled, it is possible to use an individual *PIN*. However, the process of connecting to the properly secured network by providing *PIN* is very prone to brute-force attacks (Heffner, 2011). Because *WPS* is a usual feature in today's access points and that *WPS* is usually turned on by default, *WPS* can be a very common security flaw even in networks secured by *WPA2* with a strong password. Currently, there are already available automated tools for exploiting *WPS* weaknesses, e.g., *Reaver Open Source*[4].

Recently, a critical vulnerability, *Key Reinstallation Attacks* (*KRACKs*), was discovered by Vanhoef & Piessens, 2017 revealing a flaw in 801.11i and related specifications, more precisely, in the description of the four-way handshake. A security of *CCMP* and *GCMP* encryption methods expects that no *Initialization Vector* (*IV*) repeats under the same key. Authors showed that abusing this vulnerability, they can reinstall a *Pairwise Transient Key* (*PTK*) used for generation of *Key Confirmation Key* (*KCK*), *Key Encryption Key* (*KEK*), and *Temporal Key* (*TK*). *KCK* and *KEK* are used for handshake protection and *TK* for data encryption. The reinstallation resets the *incremental transmit packet number* (nonce) and *receiver packet number* (replay counter) to the initial value. Therefore, the reinstallation violates the expectation of non reusable *IV*, which consequently breaks *TKIP*, *CCMP* or *GCMP* protocols. As Vanhoef & Piessens, 2017 show, this also occasionally happens in regular conditions, without an adversary.

Newly purchased access points usually use *WPA2* security by default. Currently, many access points can be found using default passwords not only for wireless network access, but even for *AP*'s web administration. With access to the *AP*'s administration, the investigator could focus on changing the network topology by tampering the network configuration. Access to the network management further allows the investigator to lower security levels, disable attack detections, reconfigure *DHCP* together with *DNS* and also clear *AP*'s logs. There are already implemented tools, which exploit relations between *SSID*s and default network passwords, e.g., *upc_keys*[5] by Peter Geissler.[6]

[4] https://code.google.com/archive/p/reaver-wps/

[5] https://haxx.in/upc-wifi/

[6] UPC company is a major ISP in the Czech Republic, URL: https://www.upc.cz

These tools could be used in an attack on the network with default *SSID* to improve dictionary attack using possible passwords. High severity of these security flaws is also proven by the fact that a significant amount of *WLAN*s was found using unchanged passwords, as it is shown in Table 2.

2.2 Network Technologies vulnerable to MitM

Man-in-the-middle attacks are possible because of the very nature of existing network protocols. No designed for providing security per see, the common network protocols lack strong authentication capabilities that would prevent their misuse by an attacker. Man-in-the-middle attacks assume that the attacker can divert legitimate communication. Switched Ethernet and secured wireless transmission separates the communication between two endpoints thus no other device should be able to see the conversation. Fortunately for the attacker, the insecurity of existing widely deployed protocols can be used. At the minimum, the following protocols can be considered as suitable targets:

1. *DHCP* automates network device configuration without a user's intervention (Droms, 1997).
2. *ARP* translates an *IPv4* address to a destination *MAC* address of the next-hop device in the local area network (Plummer, 1982).
3. *IPv6* networks utilize *ICMPv6 Neighbor Discovery* functionality to achieve similar functionality to *ARP* in *IPv4* networks.

Because of the lack of authentication and integrity checking, these protocols are vulnerable to spoofing attacks:

1. *DHCP Spoofing* generates fake *DHCP* communication. This attack can also be referred to as *Rogue DHCP*. An investigator can perform this kind of attack to provide devices in the network with malicious configuration, most often a fake default gateway address or *DNS* address.
2. *ARP Spoofing* provides the network devices with fake *ARP* messages. This persuades the suspect's device to believe that the attacking device's *MAC* address is the default gateway's *MAC* address.
3. *IPv6 Neighbor Spoofing* is a similar concept to *ARP Spoofing*.

From the available spoofing attacks, the *ARP Spoofing* technique was implemented in our tool. This method proved itself with reasonable performance during experiments and it is simple to implement.

Of course, there are counter-measures to spoofing attacks. The defense against spoofing lies in implementing some extra functionality to network devices:

1. *DHCP Snooping* is a countermeasure against *DHCP Spoofing*. This technique focuses on detection of forged *DHCP* communication. Network device acting as a *DHCP* snooper accepts only *DHCP* messages which are coming from connections to the genuine *DHCP* server, others are discarded. This way, individual connections are classified as either trusted or untrusted. If the network contains an unknown *DHCP* server behind an untrusted connection, it is referred to as *Spurious DHCP Server* (Cisco Systems, Inc., 2013, p. 54-2).
2. *Dynamic ARP Inspection* (*DAI*) is based on analysis of *ARP* messages

transmitted over the network with aim to detect *ARP Spoofing*. Similarly, device performing *DAI* can have its connections classified as trusted or untrusted. *ARP* messages from trusted connections are not checked. Analyzing device maintains a trusted database of mapping of *IP* and *MAC* addresses in the corresponding *LAN*. *ARP* messages from untrusted connections can be verified against this trusted mapping database (Cisco Systems, Inc., 2013, p. 56-2).

3. *Neighbor Discovery Inspection* (*NDI*) uses similar approach as abovementioned *DAI*, but to detect *IPv6 Neighbor Spoofing*. Analyzing device verifies information transferred in Neighbor Discovery messages against its database of *IP* and *MAC* address mappings.

Although the mitigation techniques are known, they are applied mostly in the enterprise environments. In SOHO networks the devices either lack this feature or the protection is not enabled by the administrator.

2.3 Man-in-the-Middle Attack

The *MitM* refers to the situation, where the attacker's device is located in the network topology between two participants of the communication (Figure 2). The attacker then acts as an intermediary and the network traffic is routed through the attacking device. This state of unauthorized and intentionally changed network topology enables the attacker to eavesdrop on passed communication. The attacker is also able to focus on decryption of data and on changing the content of passed communication. That means that the attacker can inject harmful content. The attacker's prioritized intention is not only to take control over the traffic but also to perform this attack without anyone noticing it. This way, *Man-in-the-Middle Attack* is endangers maintaining confidentiality and integrity, key parts of the *CIA* triad.

HTTPS uses asymmetric cryptography with private and public keys to provide secure *HTTP* communication. If the victim is communicating using *HTTPS*, successful realization of *MitM* attack is more difficult. During communication of web browser on client's device with a web server, these two parties exchange a certificate containing a public key for providing a secure data transfer. *MitM* attack, in this case, captures transferred certificate and replaces it with a forged one (Callegati et al., 2009). The forged certificate is at this point a self-signed certificate. Upon reception of the self-signed certificate, victim's web browser can show some warning concerning possible risk. If the victim is not aware of the possible consequences, the victim can accept the certificate. In the case of success, both communicating devices are convinced of secured *HTTPS* communication, but the attacking device has the ongoing communication available.

DNS Spoofing focuses on possibilities of forging *DNS* communication used for resolution of domain names and *IP* addresses. For the successful realization of this attack, the attacker needs to detect and intercept *DNS* messages in the network. The aim of this attack is to direct the victim to a different device by providing a fake mapping of inquired domain name to a special *IP* address. The attacker is able to imitate the inquired service by running a similar rogue service on the provided spoofed *IP* address. If the victim is convinced that the inquired service is genuine, the attacker can then focus on capturing confidential information and credentials. The attacker can also use *DNS Spoofing* for providing the real service, but with

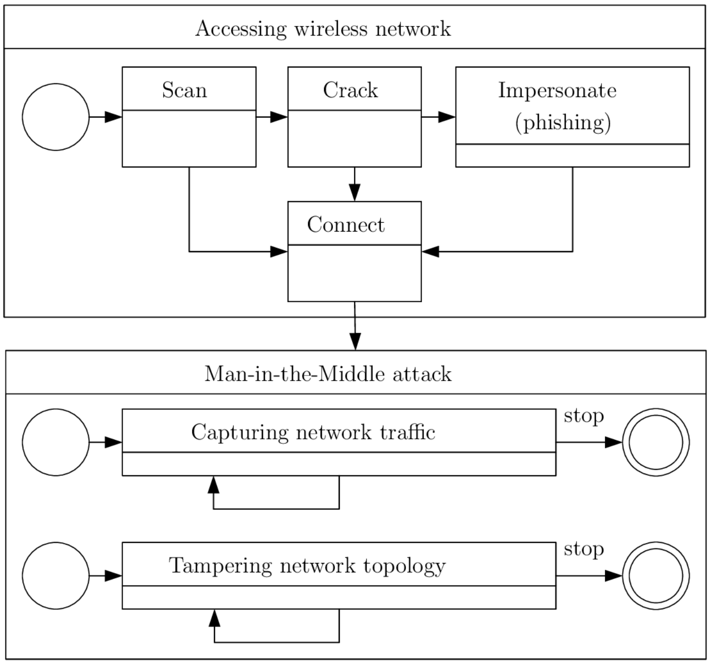

Accessing wireless network
Scan
Crack
Impersonate (phishing)
Connect
Man-in-the-Middle attack
Capturing network traffic
stop
Tampering network topology
stop

Figure 3. During the first phase – *Accessing wireless network*, the tool is capable of an attack on *WEP OSA*, *WEP SKA*, *WPA PSK* and *WPA2 PSK* secured *WLAN*s. In a case of the dictionary attack on the device deployed by the UPC company, used dictionaries are personalized by the implicit passwords. In the case of properly secured *WLAN*, impersonation (phishing) can be employed. Using this method, an investigator impersonates the legitimate network to obtain the *WLAN* credentials from the user. During the second phase – *Tampering network topology*, the tool needs to continuously work on keeping the network *stations* (*STA*s) persuaded that the spoofed topology is the correct one. An investigator is now able to capture or modify the traffic. The successful *MitM* attack is established.

enclosed harmful content. *DNS Spoofing* can be effectively applied for spoofing fake websites (Prowell et al., 2010, p. 112). If the attacker detects a *DNS* message, he intercepts it and forges a reply for the victim. The victim receives forged mapping of domain name to *IP* address and starts communication with the fake device without noticing the attack. The attacker then acts as the inquired service and therefore performs a *MitM* attack.

2.4 Available Tools for Specific Phases of the MitM Attack on Wireless Networks

From perspective of the intended functionality of the implemented tool, the whole process of *MitM* attack on wireless networks can be divided into three main phases: *Accessing wireless network*, *Tampering network topology* and *Capturing network traffic*, as explained in Figure 3.

To access secured wireless networks, *Aircrack-ng suite*[7] is considered a reliable software solution. Considering the phase *Accessing wireless network* (Figure 3), following tools were utilized. *Airmon-ng* can manage modes of a wireless interface. *Airodump-ng* can be used to scan and detect attacked *AP*. *Aircrack-ng* together with *aireplay-ng*, *airodump-ng* and *upc_keys* can be utilized for cracking *WEP OSA*, *WEP SKA*, *WPA PSK* and *WPA2 PSK*. The tool *wifiphisher*[8] can be used to perform impersonation and phishing. Connection to the wireless network can be established by *netctl*[9].

MITMf[10] with its *Spoof* plugin can be used during the *Tampering network topology* phase. For the realization of *DNS Spoofing*, it is possible to use tool *dnsspoof*, which is a part of *dsniff* collection (Song, 2001). This collection of network auditing and penetration testing tools contains several advanced programs, which could be used for tampering network topology.

Capturing traffic can be done by the tool *dumpcap*[11], which is part of the *Wireshark*[12] distribution. Behaviour, usage and success rate of individual tools, as well as possibilities of controlling them by the implemented tool, were analyzed. The software selected for individual tasks of the automated *MitM* attack were chosen from the researched variety of available tools based on performed manual experiments, further described in the thesis (Vondráček, 2016).

3. ATTACK AUTOMATION USING WIFIMITM PACKAGE AND WIFIMITMCLI TOOL

The implemented tool is currently intended to run on *Arch Linux*[13], but it could be used on other platforms which would satisfy specified dependencies. This distribution was selected because it is very flexible and lightweight. Python 3.5 was selected as a primary implementation language for the automated tool and Bash was chosen for supporting tasks, e.g., installation of dependencies on *Arch Linux* and software wrappers.

The functionality implemented in the *wifimitm* package could be directly incorporated into other software products based on Python language. This way

[7] `http://www.aircrack-ng.org/`

[8] `https://github.com/sophron/wifiphisher`

[9] `https://www.archlinux.org/packages/core/any/netctl/`

[10] `https://github.com/byt3bl33d3r/MITMf`

[11] `https://www.wireshark.org/docs/man-pages/dumpcap.html`

[12] `https://www.wireshark.org/`

[13] `https://www.archlinux.org/`

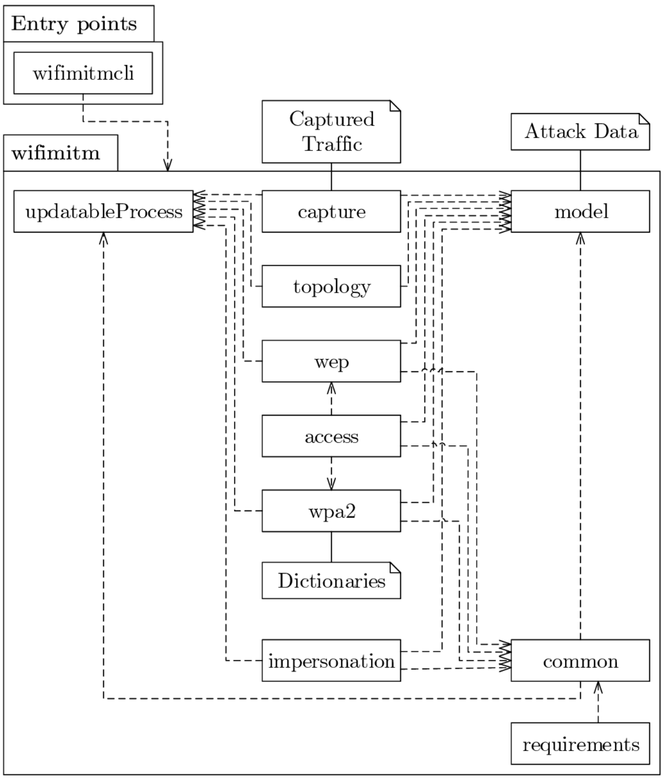


Figure 4. This figure shows the basic structure of the developed application. The tool *wifimitmcli* uses a functionality offered by the package *wifimitm*. The package is also able to manipulate attack data useful for repeated attacks and capture files with intercepted traffic. Detailed structure of the package is described in section 3.

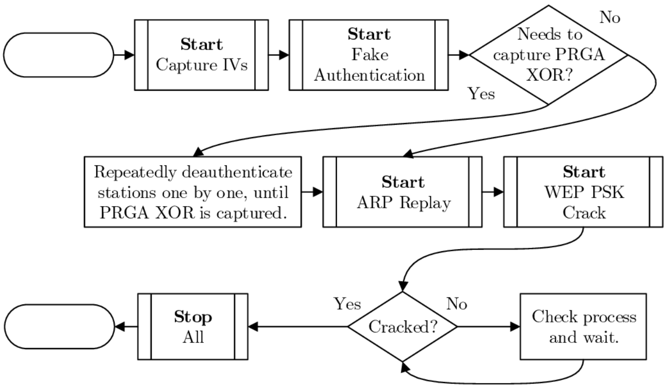


Figure 5. The figure shows a simplified flowchart of cracking *WEP OSA* or *WEP SKA* secured wireless network. Cracking procedure is a part of the first phase *Accessing wireless network* as described in Figure 3. If the given *WLAN* has already been successfully attacked, *Attack Data* (Section 3.1) contains the correct key. In such cases, repetitive cracking is unnecessary and is therefore skipped.

the package would work as a software library. Schema of the *wifimitm* package is in Figure 4.

The *wifimitm* package consists of following modules. The `access` module offers an automated process of cracking selected *WLAN*. It uses modules `wep` and `wpa2`, which implement attacks and cracking based on the used security algorithm. The `wep` module is capable of fake authentication with the *AP*, *ARP replay* attack (to speed up gathering of *IV*s) and cracking the key based on *IV*s. In the case of *WPA2* secured network, the `wpa2` module can perform a dictionary attack, personalize used dictionary and verify a password obtained by phishing (Figure 4). Verification of the password and dictionary attacks are done with a previously captured handshake. The `common` module contains functionality which could be used in various parts of the process for scanning and capturing wireless communication in monitor mode. The `common` module also offers a way to deauthenticate *STA*s from selected *AP*.

If a dictionary attack against a correctly secured network fails, a phishing attack can be managed by the `impersonation`[14] module. The `topology` module can be used to change network topology. It provides functionality for *ARP Spoofing*. The `capture` module focuses on capturing network traffic (Figure 4). It is intended to be used after the tool is successfully connected to the attacked network and network topology was successfully changed into the one suitable for *MitM* attack.

[14] For details concerning individual phishing scenarios, please see *wifiphisher*'s website. `https://github.com/sophron/wifiphisher`

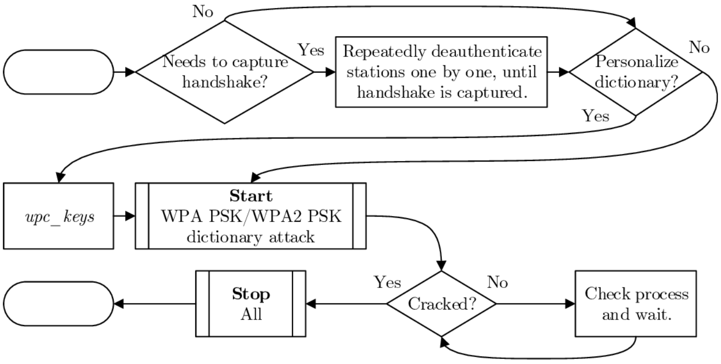

No
Needs to capture handshake?
Yes
Repeatedly deauthenticate stations one by one, until handshake is captured.
Personalize dictionary?
No
Yes
upc_keys
Start
WPA PSK/WPA2 PSK dictionary attack
Yes
No
Cracked?
Check process and wait.
Stop
All

Figure 6. This simplified flowchart illustrates cracking *WPA PSK* or *WPA2 PSK* secured network. Similarly as cracking in Figure 5, this procedure is also a part of the first phase *Accessing wireless network* (Figure 3). Cracking can also be skipped if the key is already known. As already described, impersonation (phishing) can be used in a case of unsuccessful cracking.

3.1 Attack Data

Various attacks executed against the selected *AP* require some information to be captured first. ARP request replay attack on *WEP* secured networks requires an ARP request to be obtained in order to start an attacking procedure. Fake authentication in *WEP SKA* secured network needs *PRGA XOR*[15] obtained from a detected authentication. Dictionary attack against *WPA PSK* and *WPA2 PSK* secured networks requires a captured handshake. Finally, for the successful connection to a network, a correct key is required. When the required information is obtained, it can be saved for a later usage to speed up following or repetitive attacks. Data from successful attacks could be even shared between users of the implemented tool.

3.2 Dictionary Personalization

Weaknesses in default network passwords could be exploited to improve dictionary attacks against *WPA PSK* and *WPA2 PSK* security algorithms. The implemented tool incorporates *upc_keys* for generation of possible default passwords if the selected network matches the criteria. The *upc_keys* tool generates passwords, which are transferred to the cracking tool using pipes. With this approach, the implemented tool could be further improved for example to support localized dictionaries.

3.3 Requirements

The implemented automated tool depends on several other tools, which are being controlled. The Python package can be automatically installed by its setup including Python dependencies. Non-Python dependencies can be satisfied by installation

[15] Stream of *Pseudo Random Generation Algorithm* generated bits.

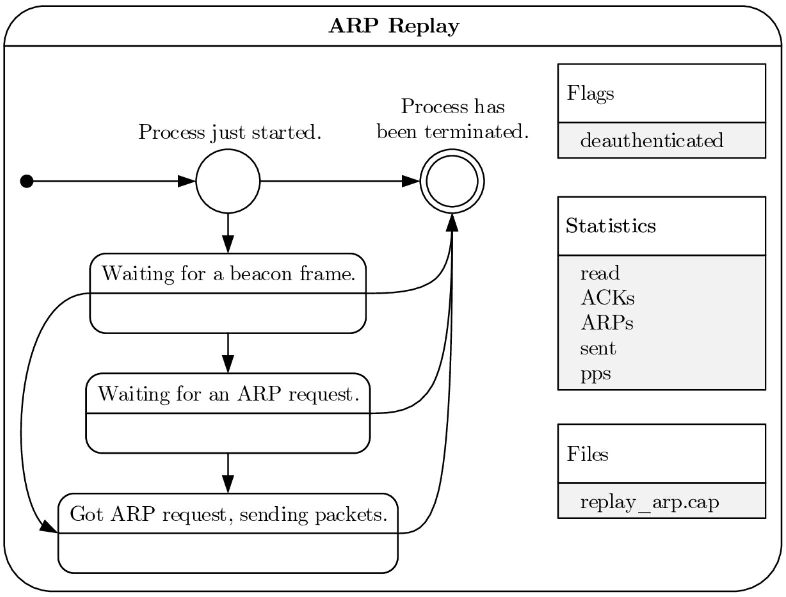

ARP Replay
Process just started.
Process has been terminated.
Waiting for a beacon frame.
Waiting for an ARP request.
Got ARP request, sending packets.
Flags
deauthenticated
Statistics
read
ACKs
ARPs
sent
pps
Files
replay_arp.cap

Figure 7. This figure presents the information model of a process controlled by *wifimitm*. In this example, the incorporated tool is *aireplay-ng* from *Aircrack-ng suite* executing *ARP replay* attack to speed up gathering of *IV*s. State of the process is modeled using a *FSM* consisting of 5 states. In a case that the attacking device receives at least one deauthentication packet, the deauthenticated flag is set. Statistics contain overall information about processed packets. Useful file created by *aireplay-ng* during this procedure is a capture file containing *ARP* request. This file is part of the *Attack Data*, as outlined in Section 3.1.

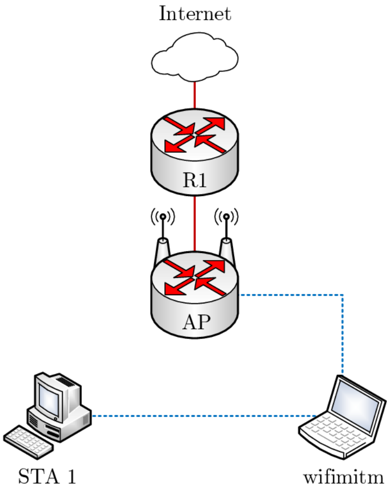


Figure 8. This figure presents the network topology used for the first performance testing (Section 4.1) and success rate measurements (Section 4.2). Results of this performance testing are in Figure 10.

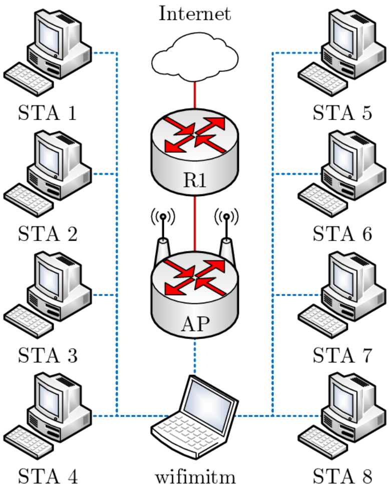


Figure 9. This figure shows the network topology consisting of 8 *STA*s and 1 *AP* which was used for the second performance testing (Section 4.1). Results of this performance testing are in Figure 11.

scripts and wrappers, which are currently developed for *Arch Linux*.

MITMf has a number of dependencies. Therefore, the installation script also creates a virtual environment dedicated to *MITMf*. After installation, *MITMf* can be easily run encapsulated in its environment. *Wifiphisher* is also installed in a virtualized environment and run using a wrapper. Tool *upc_keys* is compiled during installation. Some changes in *wifiphisher*'s source code were implemented, the installation script therefore applies a software patch. Other software dependencies are installed using a package manager.

Due to the nature of concrete steps of the attack, a special hardware equipment is required. During the scanning and capturing of network traffic without being connected to the network, an attacking device needs a wireless network interface in *monitor mode*. For sending forged packets, the wireless network interface also needs to be capable of *packet injection*. To be able to perform a phishing attack, a second wireless interface capable of master (*AP*) mode has to be available. The user can check whether his hardware is capable of packet injection using the *aireplay-ng* tool. Managing monitor mode of interface is possible with the *airmon-ng* tool.

3.4 Incorporation of tools

The implemented tool needs to interact with other software tools in order to automate attack procedures. Incorporated tools communicate using *Standard output stream* (*stdout*), *Standard error stream* (*stderr*) and optionally using generated files. *Wifimitm* needs to continuously analyze all these outputs to be aware of current state of the controlled tool. Information contained in the output can be a summary of current progress, a notification that some event occurred or a result of an intended action.

To meet requirements for efficient incorporation of other tools so that the *wifimitm* package could interact with them, the *updatableProcess* module was developed. This module contains an abstract base class *UpdatableProcess*. Individual incorporated tools have dedicated classes inherited from the *UpdatableProcess* which are used for managing these tools from *wifimitm*. When a process is spawned, using an instance of class inherited from *UpdatableProcess*, it is assigned a temporary directory for its outputs. The running process is continuously writing to *stdout* and *stderr*. The outputs are periodically analyzed. Classes inherited from *UpdatableProcess* can implement a signalization of process' state using a *Finite State Machine* (*FSM*). Process' output can include notifications of events. Upon detection of such event, appropriate flags can be set. Some processes also output summary information, which can be used to update statistics. Continuously updated information about the process can therefore consist of state, flags, statistics and created files as presented in Figure 7.

4. EVALUATION

The capabilities of the implemented tool were evaluated. Because the tool deploys man-in-the-middle type of attack, the tool necessary modifies the target environment. Thus we evaluated the footprint of the tool and the possibility to detect the running attack by the victim. The next set of experiments were conducted to show how easy it is to gain the network communication for different wireless configurations.

4.1 Attack's Performance Impact

The first experiment examines wheathe the attack is observable from end-user perspective or disrupts regular communication on the network. A scheme of the networks used for this experiment is shown in Figures 8 and 9, modeling SOHO[16] environment. The *STAs* were correctly connected to the *AP*, and they were successfully communicating with the Internet. The implemented *wifimitmcli* tool was then started and automatically attacked the network, as described in Section 3 and Figure 3.

The performance impact of the *wifimitm* was compared using typologies presented in Figures 8 and 9. As the observed metric was selected a *Round-Trip Time* (*RTT*) value describing a delay that end-user might experience when the load on the *R1* is increased.

For the first case, only one client is connected at the time. The Figure 10 plots *RTT* values measured between *STA1* and its Internet gateway *R1*. The x axe denotes each measurement, and on the y axe is shown corresponding delay in ms in a logarithmic scale.

The second case shows eight *STAs* connected to *R1* simultaneously, in Figure 9. Figure 11 shows an increase of *RTT* measured between each of *STAx* and *R1*.

Both cases were evaluated on the fact, whether the attack being performed was revealed or whether the users had any suspicion about the malicious transformation of their *WLAN*. By results comparison of both test cases, presented in Figures 10 and 11, can be concluded that regular user has no way of knowing whether the increase of la-

[16] small office/home office

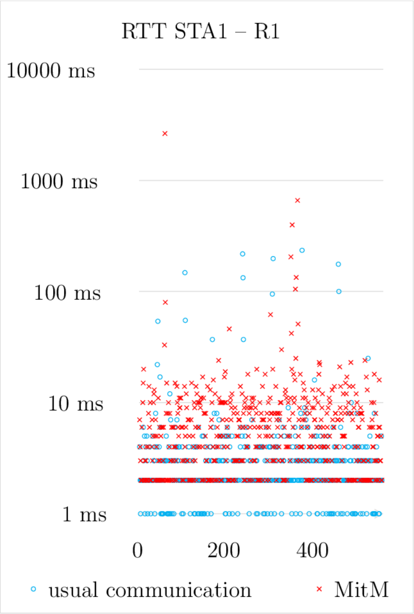


Figure 10. The first *WLAN* for performance testing was the same as for the success rate measurements described in Section 4.2. Figure shows comparison of the measured *RTT* between *STA1* and *R1* during usual communication and during successful *MitM* attack. The results show the performance impact is not critical. Discussion with the users of the attacked network proved this attack unrecognizable.

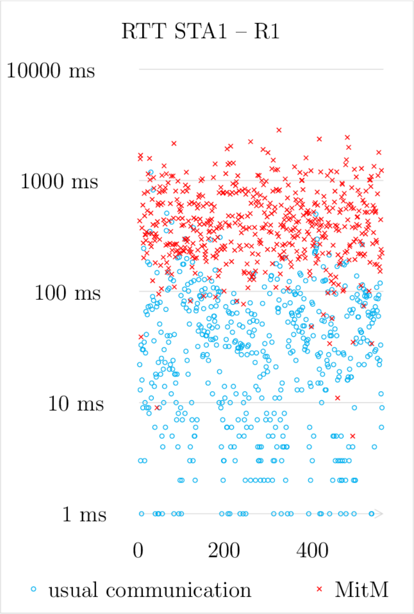


Figure 11. The second performance testing consisted of 8 *STAs* and 1 *AP* connected to the Internet – streaming videos, downloading large files, etc. The figure compares the *RTT* between *STA1* and *R1* similarly. The performance impact is more severe than in Figure 10. Despite the performance impact, the users had no suspicion that they were under *MitM* attack. Instead, they blamed the amount of devices for network congestion.

tency is caused by an attack, or by a new device connecting to the network, massive data transfer, or any other interference from the physical world.

On the other hand, there is apparent linear segregation between measurements with and without attack in Figure 11. This observation submits a future challenge, whether this condition might be used as a feature for a wireless network diagnostic without direct access to *R1* or any of *STAs*.

4.2 Experiments Concerning Various Network Devices and Configurations

The second experiment observes applicability of the *wifimitmcli* tool in different SOHO environments based on multiple AP devices with a variety of commonly used security settings in combination with numerous end-user devices. The experiment was considered successful if the *wifimitmcli* was able to perform all phases of MitM attack, Figure 3, and place itself in the middle of communication to capture network traffic according to the concept of *MitM*, Section 2.3 and Figure 2. For the test case to be correct, no help from the investigator was allowed during the attack performed by *wifimitmcli*.

The first use-case was to test all combinations of available AP devices with all available client ones. Figure 8 shows network topology used in this controlled laboratory experiment. Results of the success rate measurements are shown in Tables 3 and 4.

The second use-case was to test success rate of the *wifimitmcli* tool in a non-laboratory environment beyond our control on the end-user part. Figure 8 shows once again testing topology with*Linksys WRP400* device as an AP. Table 4 shows measurements and success rate of observations of this use-case. The experiment was conducted during the author's presentation at the Brno University of Technology, Faculty of Information Technology where visitors were invited to let their devices be attacked.

Results of experiments present in Tables 3, 4 and the thesis (Vondráček, 2016, pp. 42–43) reveal the following conclusions:

- Open – networks can be very easily attacked.
- *WEP OSA* and *WEP SKA* – secured networks can be successfully attacked even if they use a random password.
- *WPA PSK* and *WPA2 PSK* – secured networks suffer from weak passwords (dictionary attack), default passwords and mistakes of users (impersonation and phishing).

Consequently, results reveal feasibility and ease of *MitM* attack using the *wifimitm*, and its success rate in the target SOHO environments.

5. CONCLUSIONS

The goal of this research was to implement a tool that would be able to automate all the necessary steps to perform *MitM* attacks on *WLAN*s. The authors searched for and analyzed a range of software and methods focused on penetration testing, communication sniffing and spoofing, password cracking and hacking in general. To be able to design, implement and test the tool capable of such attacks, knowledge of different widespread security approaches was essential. The authors further focused on possibilities of *MitM* attacks even in cases where the target *WLAN* is secured correctly. Therefore, methods and tools for impersonation and phishing were also analyzed.

The authors' work and research resulted in creation of the *wifimitm* Python package.

Table 3. This table presents results of the success rate measurements. A successful attack is marked using a *checkmark* symbol (✓) and unsuccessful attack is marked using a *times* symbol (×). In the case when the attack was not fully successful, the question mark (?) is used. Such partially successful test (? symbol) can for example happen in situation where the suspect is sending only a portion of his traffic through the investigator. Some of the used *STA*s lack *WEP SKA* settings (□ symbol). Testing *WPA PSK* and *WPA2 PSK* networks were configured with password "`12345678`" and *WEP* secured networks used password "`A_b#1`".

		Lenovo G580, Windows 10	Lenovo G505s, Windows 8.1	Dell Latitude E6500, Ubuntu 17.04	HTC Desire 500, Android 4.1.2	Apple iPhone 4, iOS 7.1.2
Linksys WRT610N	*open*	✓	✓	✓	✓	✓
	WEP OSA	✓	✓	✓	✓	✓
	WEP SKA	□	□	✓	✓	✓
	WPA PSK	✓	✓	✓	✓	✓
	WPA2 PSK	✓	✓	✓	✓	✓
Linksys WRT54G	*open*	✓	✓	✓	✓	✓
	WEP OSA	✓	✓	✓	✓	✓
	WEP SKA	□	□	✓	✓	✓
	WPA PSK	✓	✓	✓	✓	✓
	WPA2 PSK	✓	✓	✓	✓	✓
Linksys WRP400	*open*	✓	✓	✓	✓	✓
	WEP OSA	✓	✓	✓	✓	✓
	WEP SKA	□	□	✓	✓	✓
	WPA PSK	✓	✓	✓	✓	✓
	WPA2 PSK	✓	✓	✓	✓	✓
TP-LINK TL-WR841N	*open*	?	×	✓	✓	✓
	WEP OSA	?	×	✓	✓	×
	WEP SKA	□	□	✓	✓	×
	WPA PSK	?	×	✓	✓	×
	WPA2 PSK	?	×	✓	✓	×
D-Link `DVA-G3671B`	*open*	✓	✓	✓	✓	✓
	WEP OSA	✓	✓	✓	✓	✓
	WEP SKA	□	□	✓	✓	✓
	WPA PSK	✓	✓	✓	✓	✓
	WPA2 PSK	✓	✓	✓	✓	✓

Table 4. The following table shows the results of public experiments. Testing network utilized *Linksys WRP400* as an AP and end-user devices of random people that agreed to participate in the experiment. A successful attack is marked using a *checkmark* symbol (✓).

Model	OS	Attack
HTC Desire 500	Android 4.1.2	✓
HTC Desire 820	Android 6.0.1	✓
Apple iPhone 6	iOS 10.3.1	✓
Apple iPhone 5s	iOS 10.2.1	✓
Apple iPhone 5	iOS 10.3.1	✓
Apple iPhone 5c	iOS 9.2.1	✓
Apple iPhone 4	iOS 7.1.2	✓

This package serves as a library which provides functionality for automation of *MitM* attacks on target *WLAN*s. The developed package can also be easily incorporated into other tools. Another product of this research is the *wifimitmcli* tool which incorporates the functionality of the *wifimitm* package. This tool automates the individual steps of a *MitM* attack and can be used from a *CLI*. The implemented software comes with a range of additions for convenient usage, e.g., a script that checks and installs dependencies on *Arch Linux*, a Python setuptools setup script and of course a manual page.

The *wifimitmcli* tool, and therefore *wifimitm* as well, was tested during experiments with an available set of equipment. As the results show, the implemented software product is able to perform an automated *MitM* attack on *WLAN*s successfully.

Upon successful deployment and execution of the implemented tool, an investigator can eavesdrop or spoof the passing communication. The goal of the tool was to automate *MitM* attacks on *PSK* secured *WLANs*. It does not focus on dissecting further traffic protections. This means that it does not interfere with *SSL/TLS*, *VPN*, or other encapsulations. Thanks to the tool's design, it can be easily used together with other software specialized on interception of encapsulated traffic. Traffic encapsulation is a sufficient protection against this tool. From the *WLAN* administrators point of view, available defense mechanisms are outlined in Section 2.2.

As explained earlier, all the suspect's network traffic is passing through the attacking device during a successful *MitM* attack. Unfortunately, there could be users on the network other than the ones that are subject to a court order. Making sure that only appropriate traffic is being captured may be important depending on the nature of the court order or the legislation. This challenge may be solved by setting corresponding filter rules for traffic capture software.

This research and its products can be utilized in combination with other security research carried out at the Brno University of Technology, Faculty of Information Technology. It can serve in investigations done by forensic researchers (Pluskal et al., 2015). It can also be used in automated penetration testing of *WLANs*.

In the future iterations of the development, the product could focus on exploiting the weaknesses of the widely used *WPS* technology, incorporating techniques to perform *KRACKs*, or focus on detection of attacks themselves. Concerning the current state of the product, it does not focus on enterprise *WLAN*s, which also suffer from their weaknesses.

ACKNOWLEDGEMENTS

This work was supported by Ministry of Interior of the Czech Republic project "Integrated platform for analysis of digital data from security incidents" VI20172020062; Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science" LQ1602; and by BUT internal project "ICT tools, methods and technologies for smart cities" FIT-S-17-3964.

AUTHOR BIOGRAPHIES

Martin Vondráček is currently pursuing Master's degree in Intelligent Systems at the Brno University of Technology. He has completed the Bachelor's degree with honours at the same university in 2016 and received dean's award and rector's award later that year. His interest in Computer Science increased during his thesis work at the University of Malta in 2016 and exchange study of Computer Forensics at the University of South Wales in 2017. He presented outcomes of his research at conferences ICDF2C and Excel@FIT. He is dedicated to research, information security, computer networks and software development.

Jan Pluskal is a Ph.D. student at FIT BUT, freelance lecturer and programmer. He is mostly interested in computer network forensics, machine learning, distributed computing and C# programming. The main author of Netfox Detective – a tool for network forensic analysis. In his free time, he plays around with home automation IoT technologies to ease up daily routines of his family.

Ondřej Ryšavý received the Ph.D. degree in computer science from the Brno University of Technology, Brno, Czech Republic, in 2005. He is an Associate Professor with the Department of Information Systems, Brno University of Technology, Brno. His research interests include computer networking and, in particular, network monitoring, network security and forensics, and network architectures. His work is focused on improving network security through data analysis by application of data mining, statistics, and distributed computing.

REFERENCES

Callegati, F., Cerroni, W., & Ramilli, M. (2009, Jan). Man-in-the-middle attack to the HTTPS protocol. *Security Privacy, IEEE*, 78–81. doi: 10.1109/MSP.2009.12

Cisco Systems, Inc. (2013). *Catalyst 6500 release 12.2sx software configuration guide.* Retrieved on January 29, 2018, from `http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book.html`

Deal, R., & Cisco Systems, I. (2006). *The complete cisco vpn configuration guide.* Cisco Press. Retrieved on January 30, 2018, from `https://books.google.cz/books?id=ms-8AAAACAAJ`

Droms, R. (1997, March). *Dynamic Host Configuration Protocol* (RFC No. 2131). Internet Engineering Task Force. RFC 2131 (DRAFT STANDARD). Retrieved on January 30, 2018, from `http://www.ietf.org/rfc/rfc2131.txt`

Fluhrer, S., Mantin, I., & Shamir, A. (2001). Weaknesses in the key scheduling algorithm of RC4. In S. Vaudenay & A. Youssef (Eds.), *Selected areas in cryptography* (pp. 1–24). Springer Berlin Heidelberg. Retrieved on January 30, 2018, from `http://dx.doi.org/10.1007/3-540-45537-X_1` doi: 10.1007/3-540-45537-X_1

Godber, A., & Dasgupta, P. (2003). Countering rogues in wireless networks. In *Proceedings of the international conference on parallel processing workshops* (Vol. 2003-January, pp. 425–431). Institute of Electrical and Electronics Engineers Inc. doi: 10.1109/ICPPW.2003.1240398

Halsall, F. (2005). *Computer networking and the internet.* Addison-Wesley. Retrieved on January 22, 2016, from `https://books.google.cz/books?id=QadX5XErZ9IC`

Heffner, C. (2011). *Cracking WPA in 10 hours or less – /dev/ttys0.* Retrieved on April 4, 2016, from `http://www.devttys0.com/2011/12/cracking-wpa-in-10-hours-or-less/`

IEEE-SA. (2012, March). IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, 1–2793. doi: 10.1109/IEEESTD.2012.6178212

Kent, S., & Seo, K. (2005, December). *Security Architecture for the Internet Protocol* (RFC No. 4301). Internet Engineering Task Force. RFC 4301 (PROPOSED STANDARD). Retrieved on January 30, 2018, from `https://www.ietf.org/rfc/rfc4301.txt`

Klinec, D., & Svítok, M. (2016a). *UPC UBEE EVW3226 WPA2 password reverse engineering, rev 3.* Retrieved on January 30, 2018, from `https://deadcode.me/blog/2016/07/01/UPC-UBEE-EVW3226-WPA2-Reversing.html`

Klinec, D., & Svítok, M. (2016b). *Wardriving Bratislava 10/2016.* Retrieved on January 30, 2018, from `https://deadcode.me/blog/2016/11/05/Wardriving-Bratislava-10-2016.html`

Kumkar, V., Tiwari, A., Tiwari, P., Gupta, A., & Shrawne, S. (2012). Vulnerabilities of wireless security protocols (WEP and WPA2). *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, *1*(2), 34–38. Retrieved on January 30, 2018, from `http://ijarcet.org/wp-content/uploads/IJARCET-VOL-1-ISSUE-2-34-38.pdf`

Liu, Y., Jin, Z., & Wang, Y. (2010, Sept). Survey on security scheme and attacking methods of WPA/WPA2. In *2010 6th international conference on wireless communications networking and mobile computing (wicom)* (pp. 1–4). doi: 10.1109/WICOM.2010.5601275

Plummer, D. (1982, November). *Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware* (RFC No. 826). Internet Engineering Task Force. RFC 826 (INTERNET STANDARD). Retrieved on January 30, 2018, from `http://www.ietf.org/rfc/rfc826.txt`

Pluskal, J., Matoušek, P., Ryšavý, O., Kmeť, M., Veselý, V., Karpíšek, F., & Vymlátil, M. (2015). Netfox detective: A tool for advanced network forensics analysis. In *Proceedings of security and protection of information (spi) 2015* (pp. 147–163). Brno University of Defence.

Retrieved on January 30, 2018, from http://www.fit.vutbr.cz/research/view_pub.php?id=10863

Prowell, S., Kraus, R., & Borkin, M. (2010). Chapter 6 - man-in-the-middle. In S. Prowell, R. Kraus, & M. Borkin (Eds.), *Seven deadliest network attacks* (pp. 101–120). Boston: Syngress. Retrieved on January 30, 2018, from http://www.sciencedirect.com/science/article/pii/B9781597495493000067 doi: http://dx.doi.org/10.1016/B978-1-59749-549-3.00006-7

Robyns, P. (2014). *Wireless network privacy* (Master's thesis, Hasselt University, Hasselt). Retrieved on January 30, 2018, from http://hdl.handle.net/1942/17516

Song, D. (2001, Dec). *dsniff*. Retrieved on January 27, 2018, from http://www.monkey.org/~dugsong/dsniff

Tews, E., Weinmann, R.-P., & Pyshkin, A. (2007). Breaking 104 bit WEP in less than 60 seconds. In S. Kim, M. Yung, & H.-W. Lee (Eds.), *Information security applications* (pp. 188–202). Springer Berlin Heidelberg. Retrieved on January 30, 2018, from http://dx.doi.org/10.1007/978-3-540-77535-5_14 doi: 10.1007/978-3-540-77535-5_14

Thomas, O. (2017). *Windows server 2016 inside out*. Pearson Education. Retrieved on January 30, 2018, from https://books.google.cz/books?id=rLfDDgAAQBAJ

Vanhoef, M., & Piessens, F. (2017). Key reinstallation attacks: Forcing nonce reuse in WPA2. In *Proceedings of the 24th acm conference on computer and communications security (ccs)*. ACM.

Vondráček, M. (2016). *Automation of MitM attack on WiFi networks* (Bachelor's thesis, Brno University of Technology, Faculty of Information Technology). Retrieved on January 30, 2018, from http://www.fit.vutbr.cz/study/DP/BP.php?id=18596

Vondráček, M., Pluskal, J., & Ryšavý, O. (2018). Automation of MitM attack on Wi-Fi networks. In P. Matoušek & M. Schmiedecker (Eds.), *Digital forensics and cyber crime* (pp. 207–220). Cham: Springer International Publishing.

A.7 Automation of MitM Attack on Wi-Fi Networks

Martin Vondráček, Jan Pluskal, and Ondřej Ryšavý. “Automation of MitM Attack on Wi-Fi Networks”. In: *9th International Conference on Digital Forensics & Cyber Crime*. Vol. 2018. 216. Springer International Publishing, 2017, pp. 207–220. ISBN: 9783319736969

Automation of MitM Attack on Wi-Fi Networks

Martin Vondráček( ), Jan Pluskal, and Ondřej Ryšavý

Brno University of Technology, Božetěchova 2, Brno, Czech Republic
xvondr20@stud.fit.vutbr.cz, {ipluskal,rysavy}@fit.vutbr.cz
http://www.fit.vutbr.cz/
https://mvondracek.github.io/wifimitm/

Abstract. Security mechanisms of wireless technologies often suffer weaknesses that can be exploited to perform Man-in-the-Middle attacks, allowing to eavesdrop or to spoof network communication. This paper focuses on possibilities of automation of these types of attacks using already available tools for specific tasks. Outputs of this research are the *wifimitm* Python package and the *wifimitmcli CLI* tool, both implemented in Python. The package provides functionality for automation of *MitM* attacks and can be used by other software. The *wifimitmcli* tool is an example of such software that can automatically perform multiple *MitM* attack scenarios without any intervention from an investigator.

The results of this research are intended to be used for automated penetration testing and to help with forensic investigation. Finally, a popularization of the fact that such severe attacks can be easily automated can be used to raise public awareness about information security.

Keywords: Man-in-the-Middle attack
Accessing secured wireless networks · Password cracking
Dictionary personalization · Tampering network topology
Impersonation · Phishing

1 Introduction

The main focus of this paper is security of wireless networks. It provides a study of widely used network technologies and mechanisms of wireless security. Analyzed technologies and security algorithms suffer weaknesses that can be exploited to perform Man-in-the-Middle attacks. A successful realization of this kind of attack allows not only to eavesdrop on all the victim's network traffic but also to spoof his communication [1], [16, pp. 101–120].

In an example scenario, the victim is a suspect conducting illegal activity on a target network. The attacker is a law-enforcement agency investigator with appropriate legal authorization to intercept the suspect's communication and to perform a direct attack on the network. In some cases, the suspect may be aware that his communication can be intercepted by the ISP[1] and harden his network.

[1] Internet Service Provider

P. Matoušek and M. Schmiedecker (Eds.): ICDF2C 2017, LNICST 216, pp. 207–220, 2018.
https://doi.org/10.1007/978-3-319-73697-6_16

For example, he could use an overlay network technology, e.g., *VPN* (implemented by *L2TP*, *IPsec* [9, pp. 09–10], *PPTP*) or anonymization networks (Tor, I2P, etc.) to create an encrypted tunnel configured on his gateway, for all his external communication. This concept is easy to implement and does not require any additional configuration on endpoint devices. Generally, this would not be considered a properly secured network [5, pp. 425–431], but this scheme, or similar, is often used by large vendors like Cisco [2] or Microsoft [19] for branch office deployment and can also be seen in home routers[2]. In such cases, intercepting traffic on the ISP level would not yield meaningful results, because all the communication is encrypted by the hardening. On the other hand, direct attack on the suspect's LAN will intercept plain communication. But, even when an investigator is legally permitted to carry out such an attack to acquire evidence, it is scarcely used, because it requires expert domain knowledge. Thus, this process of evidence collection is very expensive and human resource demanding.

The aim of this research is to design, implement and test a tool able to automate the process of accessing a secured *WLAN* and to perform data interception. Furthermore, this tool should be able to tamper with the network to collect more evidence by redirecting traffic to place itself in the middle of the communication and tamper with it, to access otherwise encrypted data in plain form. Using the automated tool should not require any expert knowledge from the investigator.

We designed a generic framework, see Fig. 1, capable of accessing and acquiring evidence from a wireless network regardless of used security mechanisms. This framework can be split into several steps. First, it is necessary for an investigator to obtain access to the *WLAN* used by the suspect. Therefore, this research focuses on exploitable weaknesses of particular security mechanisms. Upon successful connection to the network, the investigator needs to tamper with the network topology. For this purpose, weaknesses of several network technologies can be exploited. From this point on, the investigator can start to capture and break the encryption on the suspect's communication.

Specialized tools focused on exploiting individual weaknesses in security mechanisms currently used by *WLAN*s are already available. There are also specialized tools focused on individual steps of *MitM* attacks. Tools that were analyzed and used in implementation of the *wifimitm* package are outlined in Sect. 2.

Based on the acquired knowledge, referenced studies and practical experience from manual experiments, authors were able to create an attack strategy which is composed of a suitable set of available tools. The strategy is then able to select and manage individual steps for a successful *MitM* attack tailored to a specific *WLAN*. This strategy also includes options for impersonation and phishing for situations, when the network is properly secured, and the weakest part of the overall security is the suspect.

The created software can perform a fully automated attack and requires zero knowledge. We tested the final implementation on carefully devised experiments,

[2] Asus RT-AC5300 – Merlin WRT has an option to tunnel all traffic thought Tor.

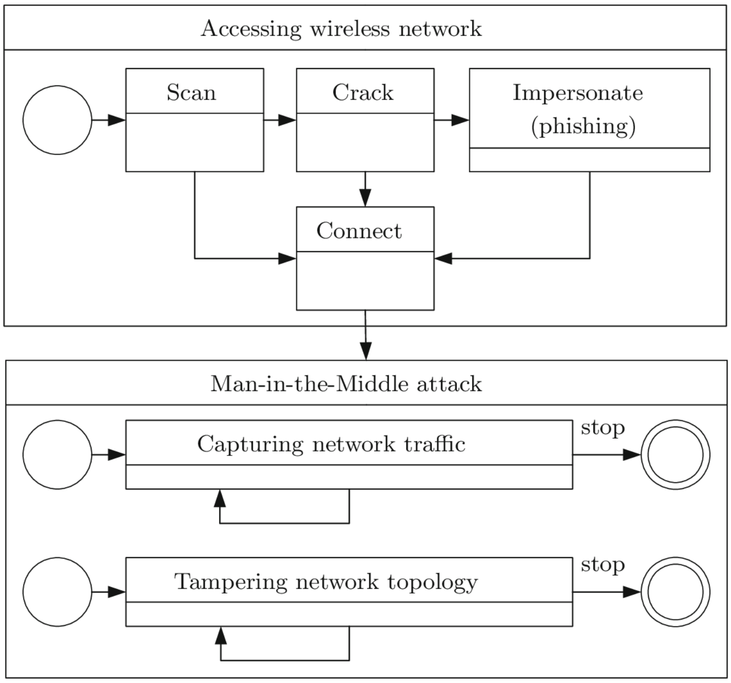

Accessing wireless network
Scan
Crack
Impersonate (phishing)
Connect
Man-in-the-Middle attack
Capturing network traffic
stop
Tampering network topology
stop

Fig. 1. During the first phase – *Accessing wireless network*, the tool is capable of an attack on *WEP OSA*, *WEP SKA*, *WPA PSK* and *WPA2 PSK* secured *WLAN*s. In a case of the dictionary attack on the device deployed by the UPC company, used dictionaries are personalized by the implicit passwords. In the case of properly secured *WLAN*, impersonation (phishing) can be employed. Using this method, an investigator impersonates the legitimate network to obtain the *WLAN* credentials from the user. During the second phase – *Tampering network topology*, the tool needs to continuously work on keeping the network *stations* (*STA*s) persuaded that the spoofed topology is the correct one. An investigator is now able to capture or modify the traffic. The successful *MitM* attack is established.

with available equipment. The tool is open source and can be easily incorporated into other software. The main use cases of this tool are found in automated penetration testing, forensic investigation, and education.

2 Security Weaknesses in WLAN Technologies

Following network technologies (Sects. 2.1 and 2.2), which find a significant utilization, unfortunately, suffer from security weaknesses in their protocols. These flaws can be used in the process of the *MitM* attack.

2.1 Wireless Security

Wired Equivalent Privacy (*WEP*) is a security algorithm introduced as a part of the IEEE 802.11 standard [6, p. 665], [8, pp. 1167–1169]. At this point, *WEP* is deprecated and superseded by subsequent algorithms, but is still sometimes used, as can be seen from Table 1 available from *Wifileaks.cz*[3]. *WEP* suffers from weaknesses and, therefore, it has been broken [4]. There are already implemented tools to provide access to wireless networks secured by *WEP* available [18]. Regarding *WEP* secured *WLAN*s, authentication can be either *Open System Authentication* (*OSA*) or *Shared Key Authentication* (*SKA*) [8, pp. 1170–1174]. In the case of *WEP OSA*, any *station* (*STA*) can successfully authenticate to the *Access Point* (*AP*) [17, pp. 4–10]. *WEP SKA* provides authentication and security of transferred communication using a shared key. Confidentiality of transferred data is ensured by encryption using the *RC4* stream cipher. Methods used for cracking access to *WEP* secured networks are based on analysis of transferred data with corresponding *Initialization Vectors* (*IV*s).

Table 1. Following table summarizes *WLAN* statistics provided by *Wifileaks.cz*. Users of this service voluntarily scan and publish details about *WLAN*s in the Czech Republic. Information in the table show that a significant number of *WLAN*s still use deprecated security algorithms. The statistics consisting of 97 192 922 measurements of 2 548 054 *WLAN*s were published on May 26, 2017.

Security	Count	Ratio
WPA2	1 429 518	56%
WEP	393 579	15%
WPA	375 984	15%
open	67 388	3%
other	281 585	11%

Wi-Fi Protected Access® (*WPA*) was developed by the Wi-Fi Alliance® as a reaction to increasing number of security flaws in *WEP*. The main flaw of *WPA* security algorithm can be identified at the beginning of client device's communication, where an unsecured exchange of confidential information is performed during the four-way handshake. An investigator can obtain this unsecured communication and use it for consecutive cracking of the *Pre-Shared Key* (*PSK*).

Wi-Fi Protected Access® *2* (*WPA2*^{TM}) is a successor of *WPA*, but security flaws of the *WPA PSK* algorithm remain significant also for the *WPA2 PSK*. Information exposed during the handshake can be used for the dictionary attack, which can be further improved by precomputing the *Pairwise Master Keys* (*PMKs*) [12, pp. 37–38], [13, p. 3]. Precomputed lookup tables are already available online[4].

A critical security flaw in wireless networks secured by *WPA* or *WPA2* is the functionality called *Wi-Fi Protected Setup*^{TM} (*WPS*). This technology was introduced with an aim to provide a comfortable and secure way of connecting

[3] http://www.wifileaks.cz/statistika/

[4] https://www.renderlab.net/projects/WPA-tables/

to the network. For a connection to the *WLAN* with *WPS* enabled, it is possible to use an individual *PIN*. However, the process of connecting to the properly secured network by providing *PIN* is very prone to brute-force attacks [7]. Because *WPS* is a usual feature in today's access points and that *WPS* is usually turned on by default, *WPS* can be a very common security flaw even in networks secured by *WPA2* with a strong password. Currently, there are already available automated tools for exploiting *WPS* weaknesses, e.g., *Reaver Open Source*[5].

Newly purchased access points usually use *WPA2* security by default. Currently, many access points can be found using default passwords not only for wireless network access, but even for *AP*'s web administration. In a case of possible access to the *AP*'s administration, the investigator could focus on changing the network topology by tampering the network configuration. Access to the network management further allows the investigator to lower security levels, disable attack detections, reconfigure *DHCP* together with *DNS* and also clear *AP*'s logs. There are already implemented tools, which exploit relations between *SSID*s and default network passwords, e.g., *upc_keys*[6] by Peter Geissler.[7] These tools could be used in an attack on the network with default *SSID* to improve dictionary attack using possible passwords. High severity of these security flaws is also proven by the fact that a significant amount of *WLAN*s was found using unchanged passwords, as it is shown in Table 2.

Table 2. Results of wardriving in Bratislava and Brno focused on UPC vulnerabilities concerning default *WPA2 PSK* passwords [11]. Detailed article about these security flaws is available online [10].

Bratislava (capital of Slovakia) 2016-10-01	Count	Ratio
Total networks	22 172	
UPC networks	3 092	13.95%
UPC networks, vulnerable	1 327	42.92% UPC
Brno (city in the Czech Republic) 2016-02-10	**Count**	**Ratio**
Total networks	17 516	
UPC networks	2 868	16.37%
UPC networks, vulnerable	1 835	63.98% UPC

2.2 Network Technologies Used in WLANs

In the context of a MitM attack on a *WLAN*, we are targeting some common network protocols:

- *DHCP* automates network device configuration without a user's intervention [3].

[5] https://code.google.com/archive/p/reaver-wps/
[6] https://haxx.in/upc-wifi/
[7] UPC company is a major ISP in the Czech Republic, URL: https://www.upc.cz

- *ARP* translates an *IPv4* address to a destination *MAC* address of the next-hop device in the local area network [14].
- *IPv6* networks utilize *ICMPv6 Neighbor Discovery* functionality to achieve similar functionality to *ARP* in *IPv4* networks.

These network protocols are vulnerable and a *MitM* attack is a coordinated attack on each of these protocols, effectively changing the network topology.

- *DHCP Spoofing* generates fake *DHCP* communication. This attack can also be referred to as *Rogue DHCP*. An investigator can perform this kind of attack to provide devices in the network with malicious configuration, most often a fake default gateway address or *DNS* address
- *ARP Spoofing* provides the network devices with fake *ARP* messages. This persuades the suspect's device to believe that the attacking device's *MAC* address is the default gateway's *MAC* address.
- *IPv6 Neighbor Spoofing* is a similar concept to *ARP Spoofing*.

ARP Spoofing technique was selected from the researched methods. This method proved itself with reasonable performance during experiments. Possible countermeasures to these attacks are further described in the thesis [20].

2.3 Available Tools for Specific Phases of the MitM Attack on Wireless Networks

From perspective of the intended functionality of the implemented tool, the whole process of *MitM* attack on wireless networks can be divided into three main phases: *Accessing wireless network*, *Tampering network topology* and *Capturing network traffic*, as explained in Fig. 1.

To access secured wireless networks, *Aircrack-ng suite*[8] is considered a reliable software solution. Considering the phase *Accessing wireless network* (Fig. 1), following tools were utilized. *Airmon-ng* can manage modes of a wireless interface. *Airodump-ng* can be used to scan and detect attacked *AP*. *Aircrack-ng* together with *aireplay-ng*, *airodump-ng* and *upc_keys* can be utilized for cracking *WEP OSA*, *WEP SKA*, *WPA PSK* and *WPA2 PSK*. The tool *wifiphisher*[9] can be used to perform impersonation and phishing. Connection to the wireless network can be established by *netctl*[10]. *MITMf*[11] with its *Spoof* plugin can be used during the *Tampering network topology* phase. *Capturing traffic* can be done by the tool *dumpcap*[12], which is part of the *Wireshark*[13] distribution. Behaviour, usage and success rate of individual tools, as well as possibilities of controlling them by the implemented tool, were analyzed. The software selected for individual tasks of the automated *MitM* attack were chosen from the researched variety

[8] http://www.aircrack-ng.org/

[9] https://github.com/sophron/wifiphisher

[10] https://www.archlinux.org/packages/core/any/netctl/

[11] https://github.com/byt3bl33d3r/MITMf

[12] https://www.wireshark.org/docs/man-pages/dumpcap.html

[13] https://www.wireshark.org/

of available tools based on performed manual experiments, further described in the thesis [20].

3 Attack Automation Using Developed wifimitm Package and wifimitmcli Tool

The implemented tool is currently intended to run on *Arch Linux*[14], but it could be used on other platforms which would satisfy specified dependencies. This distribution was selected because it is very flexible and lightweight. Python 3.5 was selected as a primary implementation language for the automated tool and Bash was chosen for supporting tasks, e.g., installation of dependencies on *Arch Linux* and software wrappers.

The functionality implemented in the *wifimitm* package could be directly incorporated into other software products based on Python language. This way the package would work as a software library. Schema of the *wifimitm* package is in Fig. 2.

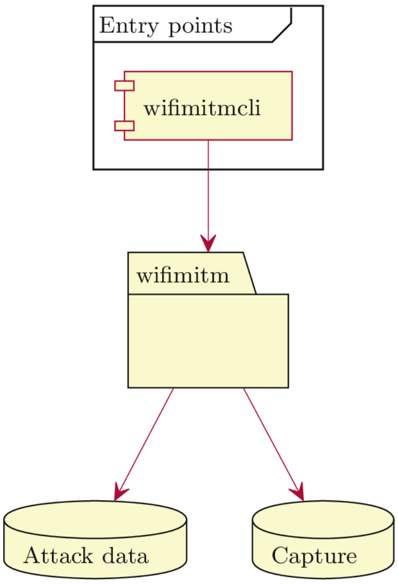


Fig. 2. This figure shows the basic structure of the developed application. The tool *wifimitmcli* uses a functionality offered by the package *wifimitm*. The package is also able to manipulate attack data useful for repeated attacks and capture files with intercepted traffic. Detailed structure of the package is described in Sect. 3.

The *wifimitm* package consists of following modules. The `access` module offers an automated process of cracking selected *WLAN*. It uses modules `wep`

[14] https://www.archlinux.org/

and wpa2, which implement attacks and cracking based on the used security algorithm. The wep module is capable of fake authentication with the *AP*, *ARP replay* attack (to speed up gathering of *IV*s) and cracking the key based on *IV*s. In the case of *WPA2* secured network, the wpa2 module can perform a dictionary attack, personalize used dictionary and verify a password obtained by phishing. Verification of the password and dictionary attacks are done with a previously captured handshake. The common module contains functionality which could be used in various parts of the process for scanning and capturing wireless communication in monitor mode. The common module also offers a way to deauthenticate *STA*s from selected *AP*.

If a dictionary attack against a correctly secured network fails, a phishing attack can be managed by the impersonation[15] module. The topology module can be used to change network topology. It provides functionality for *ARP Spoofing*. The capture module focuses on capturing network traffic. It is intended to be used after the tool is successfully connected to the attacked network and network topology was successfully changed into the one suitable for *MitM* attack.

3.1 Attack Data

Various attacks executed against the selected *AP* require some information to be captured first. ARP request replay attack on *WEP* secured networks requires an ARP request to be obtained in order to start an attacking procedure. Fake authentication in *WEP SKA* secured network needs *PRGA XOR*[16] obtained from a detected authentication. Dictionary attack against *WPA PSK* and *WPA2 PSK* secured networks requires a captured handshake. Finally, for the successful connection to a network, a correct key is required. When the required information is obtained, it can be saved for a later usage to speed up following or repetitive attacks. Data from successful attacks could be even shared between users of the implemented tool.

3.2 Dictionary Personalization

Weaknesses in default network passwords could be exploited to improve dictionary attacks against *WPA PSK* and *WPA2 PSK* security algorithms. The implemented tool incorporates *upc_keys* for generation of possible default passwords if the selected network matches the criteria. The *upc_keys* tool generates passwords, which are transferred to the cracking tool using pipes. With this approach, the implemented tool could be further improved for example to support localized dictionaries.

[15] For details concerning individual phishing scenarios, please see *wifiphisher*'s website. https://github.com/sophron/wifiphisher

[16] Stream of *Pseudo Random Generation Algorithm* generated bits.

3.3 Requirements

The implemented automated tool depends on several other tools, which are being controlled. The Python package can be automatically installed by its setup including Python dependencies. Non-Python dependencies can be satisfied by installation scripts and wrappers, which are currently developed for *Arch Linux*.

MITMf has a number of dependencies. Therefore, the installation script also creates a virtual environment dedicated to *MITMf*. After installation, *MITMf* can be easily run encapsulated in its environment. *Wifiphisher* is also installed in a virtualized environment and run using a wrapper. Tool *upc_keys* is compiled during installation. Some changes in *wifiphisher*'s source code were implemented, the installation script therefore applies a software patch. Other software dependencies are installed using a package manager.

Due to the nature of concrete steps of the attack, a special hardware equipment is required. During the scanning and capturing of network traffic without being connected to the network, an attacking device needs a wireless network interface in monitor mode. For sending forged packets, the wireless network interface also needs to be capable of packet injection. To be able to perform a phishing attack, a second wireless interface capable of master (*AP*) mode has to be available. The user can check whether his hardware is capable of packet injection

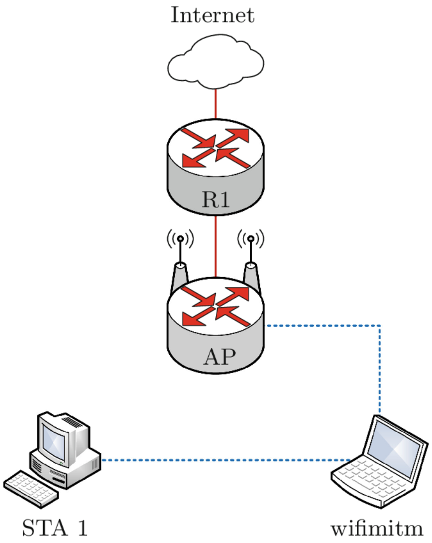


Fig. 3. This figure shows the network topology used for the first performance testing (Sect. 4) and success rate measurements (Sect. 5). Results of this performance testing are in Fig. 5.

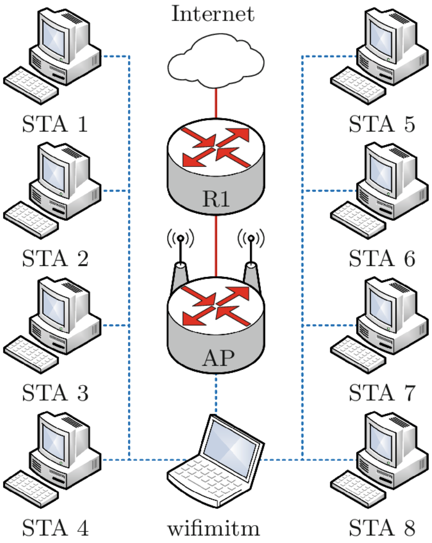


Fig. 4. This figure shows the network topology consisting of 8 *STA*s and 1 *AP* which was used for the second performance testing (Sect. 4). Results of this performance testing are in Fig. 6.

using the *aireplay-ng* tool. Managing monitor mode of interface is possible with the *airmon-ng* tool.

4 Attack's Performance Impact

A scheme of the networks used for the experiments is shown in Figs. 3 and 4. The *STAs* were correctly connected to the *AP* and they were successfully communicating with the Internet. The implemented *wifimitmcli* tool was then started and automatically attacked the network.

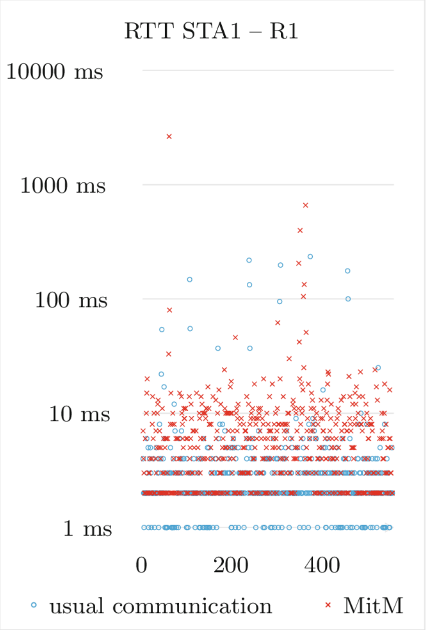


Fig. 5. The first *WLAN* for performance testing was the same as for the success rate measurements described in Sect. 5. Figure shows comparison of the measured *RTT* between *STA1* and *R1* during usual communication and during successful *MitM* attack. The results show the performance impact is not critical. Discussion with the users of the attacked network proved this attack unrecognizable.

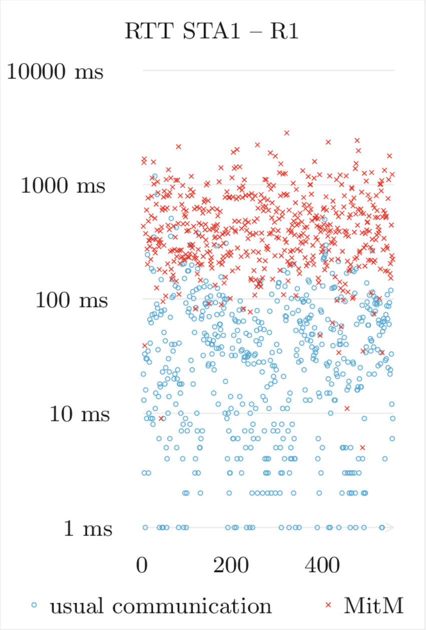


Fig. 6. The second performance testing consisted of 8 *STAs* and 1 *AP* connected to the Internet – streaming videos, downloading large files, etc. The figure compares the *RTT* between *STA1* and *R1* similarly. The performance impact is more severe than in Fig. 5. Despite the performance impact, the users had no suspicion that they were under *MitM* attack. Instead, they blamed the amount of devices for network congestion.

The performance impact of the *wifimitm* was compared using setups based on SOHO[17] environment. Both experiments were also evaluated based on the fact, whether the attack being performed was revealed or whether the users had any suspicion about the malicious transformation of their *WLAN*. Results of the testing are presented in Figs. 5 and 6.

Table 3. This table presents results of the success rate measurements. A successful attack is marked using a *checkmark* symbol (✓) and unsuccessful attack is marked using a *times* symbol (×). In the case when the attack was not fully successful, the question mark (?) is used. Such partially successful test (? symbol) can for example happen in situation where the suspect is sending only a portion of his traffic through the investigator. Some of the used *STA*s lack *WEP SKA* settings (□ symbol). Testing *WPA PSK* and *WPA2 PSK* networks were configured with password "`12345678`" and *WEP* secured networks used password "`A_b#1`".

		Lenovo G580, Windows 10	Lenovo G505s, Windows 8.1	Dell Latitude E6500, Ubuntu 17.04	HTC Desire 500, Android 4.1.2	Apple iPhone 4, iOS 7.1.2
Linksys WRT610N	*open*	✓	✓	✓	✓	✓
	WEP OSA	✓	✓	✓	✓	✓
	WEP SKA	□	□	✓	✓	✓
	WPA PSK	✓	✓	✓	✓	✓
	WPA2 PSK	✓	✓	✓	✓	✓
Linksys WRT54G	*open*	✓	✓	✓	✓	✓
	WEP OSA	✓	✓	✓	✓	✓
	WEP SKA	□	□	✓	✓	✓
	WPA PSK	✓	✓	✓	✓	✓
	WPA2 PSK	✓	✓	✓	✓	✓
Linksys WRP400	*open*	✓	✓	✓	✓	✓
	WEP OSA	✓	✓	✓	✓	✓
	WEP SKA	□	□	✓	✓	✓
	WPA PSK	✓	✓	✓	✓	✓
	WPA2 PSK	✓	✓	✓	✓	✓
TP-LINK TL-WR841N	*open*	?	×	✓	✓	✓
	WEP OSA	?	×	✓	✓	×
	WEP SKA	□	□	✓	✓	×
	WPA PSK	?	×	✓	✓	×
	WPA2 PSK	?	×	✓	✓	×
D-Link DVA-G3671B	*open*	✓	✓	✓	✓	✓
	WEP OSA	✓	✓	✓	✓	✓
	WEP SKA	□	□	✓	✓	✓
	WPA PSK	✓	✓	✓	✓	✓
	WPA2 PSK	✓	✓	✓	✓	✓

[17] Small office/home office.

5 Experiments Concerning Various Network Configurations and Devices

The test was considered successful if the *wifimitmcli* was able to capture network traffic according to the concept of *MitM*. For the test to be correct, no intervention (help) from the investigator was allowed during the attack performed by *wifimitmcli*. Results of the success rate measurements are shown in Tables 3 and 4.

Table 4. The following table shows the results of public experiments. Visitors of the Brno University of Technology, Faculty of Information Technology were invited to let their devices be attacked. Testing network utilized *Linksys WRP400* device as an AP. A successful attack is marked using a *checkmark* symbol (✓).

Model	OS	Attack
HTC Desire 500	Android 4.1.2	✓
HTC Desire 820	Android 6.0.1	✓
Apple iPhone 6	iOS 10.3.1	✓
Apple iPhone 5s	iOS 10.2.1	✓
Apple iPhone 5	iOS 10.3.1	✓
Apple iPhone 5c	iOS 9.2.1	✓
Apple iPhone 4	iOS 7.1.2	✓

Results of experiments (Tables 3 and 4 and the thesis [20, pp. 42–43]) show, that open networks can be very easily attacked. *WEP OSA* and *WEP SKA* secured networks can be successfully attacked even if they use a random password. *WPA PSK* and *WPA2 PSK* secured networks suffer from weak passwords (dictionary attack), default passwords and mistakes of users (impersonation and phishing). As Figs. 5, 6 and Tables 3, 4 show, *MitM* attack using the *wifimitm* is successfully feasible in the target environments.

6 Conclusions

The goal of this research was to implement a tool that would be able to automate all the necessary steps to perform *MitM* attacks on *WLAN*s. The authors searched for and analyzed a range of software and methods focused on penetration testing, communication sniffing and spoofing, password cracking and hacking in general. To be able to design, implement and test the tool capable of such attacks, knowledge of different widespread security approaches was essential. The authors further focused on possibilities of *MitM* attacks even in cases where the target *WLAN* is secured correctly. Therefore, methods and tools for impersonation and phishing were also analyzed.

The authors' work and research resulted in creation of the *wifimitm* Python package. This package serves as a library which provides functionality for automation of *MitM* attacks on target *WLAN*s. The developed package can also be easily incorporated into other tools. Another product of this research is the *wifimitmcli* tool which incorporates the functionality of the *wifimitm* package. This tool automates the individual steps of a *MitM* attack and can be used from a *CLI*. The implemented software comes with a range of additions for convenient usage, e.g., a script that checks and installs dependencies on *Arch Linux*, a Python setuptools setup script and of course a manual page.

The *wifimitmcli* tool, and therefore *wifimitm* as well, was tested during experiments with an available set of equipment. As the results show, the implemented software product is able to perform an automated *MitM* attack on *WLAN*s successfully.

Upon successful deployment and execution of the implemented tool, an investigator can eavesdrop or spoof the passing communication. The goal of the tool was to automate *MitM* attacks on *WLANs*. It does not focus on dissecting further traffic protections. This means that it does not interfere with *SSL/TLS*, *VPN*, or other encapsulations. Thanks to the tool's design, it can be easily used together with other software specialized on interception of encapsulated traffic. Traffic encapsulation is a sufficient protection against this tool. From the *WLAN* administrators point of view, available defense mechanisms are outlined in Sect. 2.2.

As explained earlier, all the suspect's network traffic is passing through the attacking device during a successful *MitM* attack. Unfortunately, there could be users on the network other than the ones that are subject to a court order. Making sure that only appropriate traffic is being captured may be important depending on the nature of the court order or the legislation. This challenge may be solved by setting corresponding filter rules for traffic capture software.

This research and its products can be utilized in combination with other security research carried out at the Brno University of Technology, Faculty of Information Technology. It can serve in investigations done by forensic researchers [15]. It can also be used in automated penetration testing of *WLANs*.

In the future iterations of the development, the product could focus on exploiting the weaknesses of the widely used *WPS* technology. Concerning the current state of the product, it does not focus on enterprise *WLAN*s, which also suffer from their own weaknesses.

The authors disclaim any use of this research for any unlawful activities.

References

1. Callegati, F., Cerroni, W., Ramilli, M.: Man-in-the-middle attack to the HTTPS protocol. IEEE Security Privacy **7**, 78–81 (2009)
2. Deal, R., Cisco Systems Inc.: The Complete Cisco VPN Configuration Guide. Cisco Press Networking Technology Series. Cisco Press, Indianapolis (2006)
3. Droms, R.: Dynamic host configuration protocol. RFC 2131, IETF, March 1997

4. Fluhrer, S., Mantin, I., Shamir, A.: Weaknesses in the key scheduling algorithm of RC4. In: Vaudenay, S., Youssef, A. (eds.) Selected Areas in Cryptography. LNCS, pp. 1–24. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45537-X_1
5. Godber, A., Dasgupta, P.: Countering rogues in wireless networks, vol. 2003-January, pp. 425–431. Institute of Electrical and Electronics Engineers Inc. (2003)
6. Halsall, F.: Computer Networking and the Internet. Addison-Wesley, Boston (2005)
7. Heffner, C.: Cracking WPA in 10 hours or less –/dev/ttys0 (2011). http://www.devttys0.com/2011/12/cracking-wpa-in-10-hours-or-less/
8. IEEE-SA. IEEE standard for information technology-telecommunications and information exchange between systems local and metropolitan area networks-specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007), pp. 1–2793, March 2012
9. Kent, S., Seo, K.: Security Architecture for the Internet Protocol. RFC 4301, IETF, December 2005
10. Klinec, D., Svítok, M.: UPC UBEE EVW3226 WPA2 password reverse engineering, rev 3. https://deadcode.me/blog/2016/07/01/UPC-UBEE-EVW3226-WPA2-Reversing.html. Accessed 5 Nov 2016
11. Klinec, D., Svítok, M.: Wardriving Bratislava 10/2016, 5 November 2016. https://deadcode.me/blog/2016/11/05/Wardriving-Bratislava-10-2016.html
12. Kumkar, V., Tiwari, A., Tiwari, P., Gupta, A., Shrawne, S.: Vulnerabilities of wireless security protocols (WEP and WPA2). Int. J. Adv. Res. Comput. Eng. Technol. (IJARCET) **1**(2), 34–38 (2012)
13. Liu, Y., Jin, Z., Wang, Y.: Survey on security scheme and attacking methods of WPA/WPA2. In: 2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM), pp. 1–4, September 2010
14. Plummer, D.: Ethernet address resolution protocol: or converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware. RFC 826, IETF, November 1982
15. Pluskal, J., Matoušek, P., Ryšavý, O., Kmeť, M., Veselý, V., Karpíšek, F., Vymlátil, M.: Netfox detective: a tool for advanced network forensics analysis. In: Proceedings of Security and Protection of Information (SPI) 2015, pp. 147–163. Brno University of Defence (2015)
16. Prowell, S., Kraus, R., Borkin, M.: Man-in-the-middle. In: Prowell, S., Kraus, R., Borkin, M. (eds.) Seven Deadliest Network Attacks, pp. 101–120. Syngress, Boston (2010)
17. Robyns, P.: Wireless network privacy. Master's thesis. Hasselt University, Hasselt (2014)
18. Tews, E., Weinmann, R.-P., Pyshkin, A.: Breaking 104 bit WEP in less than 60 seconds. In: Kim, S., Yung, M., Lee, H.-W. (eds.) Information Security Applications. LNCS, pp. 188–202. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77535-5_14
19. Thomas, O.: Windows Server 2016 Inside Out. Inside Out. Pearson Education, London (2017)
20. Vondráček, M.: Automation of MitM attack on WiFi networks. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology (2016)

A.8 Traffic Classification and Application Identification in Network Forensics

Jan Pluskal, Ondrej Lichtner, and Ondřej Ryšavý. “Traffic Classification and Application Identification in Network Forensics”. In: *Fourteenth Annual IFIP WG 11.9 International Conference on Digital Forensics.* Ed. by Gilbert Peterson and Sujeet Shenoi. New Delhi, IN: Springer International Publishing, 2018, pp. 161–181. ISBN: 9783319992778

Chapter 10

TRAFFIC CLASSIFICATION AND APPLICATION IDENTIFICATION IN NETWORK FORENSICS

Jan Pluskal, Ondrej Lichtner and Ondrej Rysavy

Abstract Network traffic classification is an absolute necessity for network monitoring, security analyses and digital forensics. Without accurate traffic classification, the computational demands imposed by analyzing all the IP traffic flows are enormous. Classification can also reduce the number of flows that need to be examined and prioritized for analysis in forensic investigations.

This chapter presents an automated feature elimination method based on a feature correlation matrix. Additionally, it proposes an enhanced statistical protocol identification method, which is compared against Bayesian network and random forests classification methods that offer high accuracy and acceptable performance. Each classification method is used with a subset of features that best suit the method. The methods are evaluated based on their ability to identify the application layer protocols and the applications themselves. Experiments demonstrate that the random forests classifier yields the most promising results whereas the proposed enhanced statistical protocol identification method provides an interesting trade-off between higher performance and slightly lower accuracy.

Keywords: Protocol identification, application identification, machine learning

1. Introduction

Network traffic classification is an important technique used in network monitoring, security analyses and digital forensics. In digital forensics, file types can be identified by file extensions or by searching for magic numbers at the beginning of files; known files can be identified using databases of hash values. The identification of file types and filtering of known files reduce the amount of data that needs to be analyzed. Do-

G. Peterson and S. Shenoi (Eds.): Advances in Digital Forensics XIV, IFIP AICT 532, pp. 161–181, 2018.
https://doi.org/10.1007/978-3-319-99277-8_10

ing the same with network traffic is much more complicated because each data transfer contains specific and temporary characteristics that depend on the network state, network utilization and locations of communications endpoints. The correct classification of network traffic enables an automated analyzer to determine which application protocol parser to use to extract information carried by an IP flow (a packet sequence identified by the same source and destination IP addresses, transport protocol ports and transport protocol type). This, in turn, helps speed up a forensic investigation by reducing the number of unclassified IP flows.

Traditional traffic classification methods identify applications based on the TCP or UDP ports that are used. This provides only limited accuracy (60–80%) because many applications use random or non-standard ports [3, 24], for example, peer-to-peer applications, multimedia streaming applications, computer games and tunneled traffic. Advanced traffic classification utilizes supervised machine learning methods based on payload analysis, statistical methods and hybrid approaches [17, 19, 26, 27, 29]. Each technique has its advantages and disadvantages. For example, payload analysis of encrypted communications is unacceptably inaccurate. Statistical and hybrid approaches demonstrate that it is not necessary to rely exclusively on packet content [5, 12, 21], but that it is possible to combine structural and behavioral features to increase detection accuracy [16].

Unsupervised machine learning methods can classify unknown network traffic [9] into unlabeled clusters based on their similarity. An expert investigator, upon inspection of a few samples of a cluster, can label the entire cluster.

Several researchers have investigated machine learning approaches for traffic classification. Most of the research has focused on classifying network traffic to identify the application layer protocol in order to support intelligent network filtering and security monitoring. While traffic classification for network forensics stems from the same ideas, there are some notable differences. Network forensics analysis can be performed off-line on captured data. In this case, accuracy is more important than speed. Thus, a combination of several methods or applications that are slower, but more accurate, can be considered.

In network forensics, an investigator can compensate for incorrect results by performing additional manual inspections of results. For example, some methods return a probability vector that can be inspected to consider different results.

Additionally, in network forensics, classification must be deterministic because forensic principles require that all results be verifiable.

Also, classification methods can be tuned by an investigator and can be repeated with different parameter sets to increase sensitivity while decreasing specificity.

Machine learning algorithms for network traffic classification have been studied since the 1990s. The most common algorithms include support vector machines [12], decision tree algorithms [21] and probabilistic [5] and statistical methods [16, 19], all of which involve supervised learning. The unsupervised k-means clustering algorithm [9] groups traffic based on its significant features. If the feature set is selected properly, a machine learning method can exceed 90% accuracy [26].

Surveys of classification methods by Nguyen and Armitage [27] and Namdev et al. [26] discuss protocol identification. Classification methods for encrypted traffic are reviewed in [29]. Al Khater and Overill [2] have proposed the use of machine learning algorithms to improve traffic classification methods for digital forensic applications. Foroushani and Zincir-Heywood [10] have demonstrated the possibility of identifying high-level application behaviors from encrypted network service communications. Dai et al. [6] and Miskovic et al. [23] have described methods for fingerprinting mobile applications based on their communications. Erman et al. [8] have explored flow-based classification and have proposed a semi-supervised classification method that can accommodate known and unknown applications.

While traffic classification has been applied extensively to network monitoring and security analysis, significantly less research has focused on traffic discrimination for network forensics. This research makes some key contributions to the field of network forensics. The first is the creation of a dataset that provides a means to reliably acquire ground truth for experiments. Typical datasets use information inferred from `l7-filter` [28] or `nmap` [1] and, therefore, offer only approximations of the real information. Shang and Huang [28] have shown that the precision of these techniques is always one (no false positives), but the recall varies between 0.67 and 0.87. This means that 13–33% of the samples are not labeled and the researchers would have excluded them from the datasets because they lacked labels [1, 12]. Therefore, the remaining dataset is already classifiable via deep packet inspection and is less relevant to finding better classification methods. In other cases, researchers do not include information about the data used in their experiments, or the descriptions are vague and not reproducible [28], or they do not describe how to annotate data with labels without errors [5].

For these reasons, this research captured one week's worth of packet data in an environment with eight hosts, which translates to roughly

20 GB. The data was automatically tagged with complete information about the origin application.

This research has also developed an enhanced statistical protocol identification (ESPI) method that leverages a machine-learning-based classifier. Upon evaluating the results of related studies, two additional classifiers, a Bayesian network classifier and a random forests classifier, were selected for comparison. This chapter describes all three methods and shows that they can be used to identify application layer protocols and even the applications that used the protocols. This is important because application identification provides more information about network traffic compared with what can be gleaned from the identified application layer protocols. Consider a situation where HTTPS is used to create an encrypted tunnel. A tool capable of recognizing applications (e.g., Google Drive, iTunes and OneDrive) in network traffic instead of merely the application layer protocol (e.g., HTTPS) is useful in several domains. Notably, in forensic analysis, application identification could significantly reduce the amount of data to be analyzed compared with conventional approaches.

2. Data Collection and Preprocessing

Network traffic classification takes a network traffic capture file as input, typically in the PCAP format. The captured traffic is then split into a collection of layer 4 conversations represented by one or two IP flows for one-way or two-way communications, respectively. The experiments described in this chapter employed an annotated dataset captured by Microsoft Network Monitor, which provides application labels for almost all conversations. The dataset contains regular network traffic generated by eight user workstations running the Windows operating system. The final capture file has the following characteristics:

- **PCAP File Size:** 19.5 GB.
- **PCAP Format:** Microsoft NetMon 2.x.
- **Capture Duration:** 119 hours.
- **Number of Packets:** 27,616,138.
- **Number of Layer 7 Conversations:** 269,459.
- **Number of Application Protocols:** 58.
- **Number of Communicating Applications:** 93.

Information about the dataset is available at `pluskal.github.io/AppIdent` and the dataset itself can be downloaded from `nes.fit.vutbr.cz/AppIdent`.

Before the capture file could be used, additional post-processing steps from previous work [22] were applied to enhance data extraction. The final post-processing step used a round of experiments with the enhanced statistical protocol identification method. Based on these initial results, a second instance of the dataset was created that contained ground truth about the application protocols. The ground truth supported manual hierarchical clustering analysis of the results.

The post-processing steps improved the traffic classification accuracy by reducing the noise in the extracted features caused by the following items:

- Important TCP session control information, such as synchronization segments and finalization segments, may be missing.
- Sequence numbers may overflow in long-running TCP conversations. This can result in incorrect interpretation, causing single conversations to be split or two unrelated IP flows to be joined into a single conversation.
- The joining of capture files from multiple probes must address issues related to possible packet duplication and the proper ordering of packets belonging to the same conversation.
- Some IP packets may be missing or be duplicated (e.g., in the case of TCP retransmission).
- Finally, associated IP flows in bidirectional conversations must be paired correctly.

Matousek et al. [22] have shown that other network forensic solutions do not effectively address these issues. This implies that adopting the proposed additional steps would also be beneficial in the context of network traffic classification. To address these issues, Netfox Detective (`github.com/nesfit/NetfoxDetective`), a custom tool created for these use cases, was used to process the captured PCAP files.

2.1 Application Conversations and Messages

In addition to addressing the basic issues related to processing layer 4 conversations, Netfox Detective also enabled the dataset to be processed to track layer 7 conversations and to approximate individual application messages. This increased the classification accuracy by identifying application communications patterns. It also eliminated remnants of network packet fragmentation in the Internet layer and TCP retransmission in the transport layer. Packet fragmentation and TCP retransmission are independent of application communications patterns and, thus, can negatively impact classification.

An application message was identified in the reassembled stream based on the transport protocol. The following rules were used for identification:

- If a stream uses the UDP transport protocol, then the entire payload of each UDP datagram is considered to be a single application message.
- In the case of the TCP transport protocol, segments are separated into application messages based on packets with PSH, RST or FIN flags, or based on timeouts.

These rules are simple to implement and yield accurate approximations of application messages in most cases.

3. Classification Methods

Using machine learning algorithms to classify traffic is by no means a new concept in the field of network forensics. However, the typical use case is to identify the application protocol [27, 29]. In this research, the approach was expanded to also identify the application that created the traffic. This provides more information that can be used by a forensic investigator for easier and more precise analysis.

This section describes revisions to the commonly-used feature sets [16, 19, 25] to address the task at hand and presents a feature elimination method based on feature correlation to improve the accuracy of the created classifiers. Finally, the proposed enhanced statistical protocol identification method is described along with two other classification methods from the literature that have yielded promising traffic identification results.

3.1 Feature Set

The quality of a feature set directly influences classification accuracy [32]. Common features used for traffic classification are related to key aspects of packet communications and network architecture. These include port numbers, transport protocol type, starting sequence of payload bytes, pattern occurrence, message length and message timing. Researchers have identified a list of possible features comprising 92 items that are invariant to network line characteristics [16, 19, 25]. The list is available at `github.com/pluskal/AppIdent`.

Machine learning algorithms achieve the best performance when the selected features are orthogonal (i.e., no correlation exists between the features) [14]. Several approaches have been proposed for calculating feature correlations, including the Pearson, Spearman, Kendall correlation formulas [31] and covariance matrix [13]. This research opted for the covariance matrix method due to its ease of implementation.

The covariance matrix provides a correlation value for each pair of features. This matrix was used to design an automated two-step procedure for eliminating features. In the first step, a covariance matrix was calculated based on a chosen ratio of training data to verification data (t/v). In the second step, based on a maximum allowed correlation value, feature pairs with higher correlation values were identified and features that were, on average, more correlated with all the other features, were iteratively removed from the feature set. The resulting feature set was used by the selected classification method and could be evaluated to find the optimal set.

In the experiments, more than 80% of the feature pairs had correlation values of 0.5 or higher. Table 1 lists the features that remained after feature elimination was performed on sample data with training to verification ratios of 0.1 and 0.2, based on accepted correlation values up to 0.5. Note that the correlation column shows the maximal-allowed correlation values of features listed on the corresponding line and higher. These feature sets were used by the Bayesian network and random forests classifiers.

Most of the features describe flow characteristics instead of individual packet characteristics. This confirms the assumption that relying on a signature or some pattern in packet content gives better results for encrypted or less-structured traffic.

3.2 Enhanced Statistical Protocol Identification

Hjelmvik [16] developed the statistical protocol identification (SPID) method for use with the NetworkMiner tool. The learning phase of the method creates a database of protocol fingerprints for identifying application protocols. The features utilized by the statistical protocol identification method are called "protocol attribute meters," each conveying different information. Some items are scalar values representing payload data size, number of packets in a session or port number. Other items are composite values, such as a tuple comprising packet direction, packet ordering, packet size and byte value frequency.

The original implementation uses about 35 protocol attribute meters and extracts information from the first few packets of IP flows to achieve better speed compared with other classification methods that analyze entire IP flows. The distance between the analyzed data to a

Table 1. Features remaining after elimination based on t/v ratios of 0.1 and 0.2.

Correlation	Feature (t/v = 0.1)	Feature (t/v = 0.2)
	BytePairsReoccuringDownFlow	
	DirectionChanges	
	First3BytesEqualDownFlow	First3BytesEqualDownFlow
	FirstBitPositionUpFlow	FirstBitPositionUpFlow
	FirstPayloadSize	
	MinInterArrivalTimeDownFlow	
	MinInterArrivalTimePackets UpAndDownFlow	MinInterArrivalTimePackets UpAndDownFlow
	MinPacketLengthDownFlow	MinPacketLengthDownFlow
	NumberOfBytesDownFlow	
	NumberOfPacketsUpFlow	
	PacketLengthDistribution DownFlow	PacketLengthDistribution DownFlow
	PacketLengthDistribution UpFlow	
		ThirdQuartileInterArrival TimeUp
		ByteFrequencyUpFlow
		MaxSegmentSizeDown
		MaxSegmentSizeUp
		MinInterArrivalTimePackets UpFlow
		NumberOfBytesUpFlow
		ThirdQuartileInterArrival TimeDown
<0.25	PUSHPacketsDown	PUSHPacketsDown
	ThirdQuartileInterArrival TimeDown	
		NumberOfBytesUpFlow
<0.3		FirstPayloadSize
	ByteFrequencyUpFlow	
	MinPacketLengthUpFlow	MinPacketLengthUpFlow
	NumberOfPacketsPerTimeUp	
		DirectionChanges
		BytePairsReoccuringDownFlow
<0.4		MeanPacketLengthUpFlow
<0.5	MeanPacketLengthUpFlow	

known protocol fingerprint is computed using the Kullback-Leibler divergence and the best matching protocol fingerprint has the smallest sum of Kullback-Leibler divergences over all the attributes. Kohnen et al. [19] have developed a new version of the statistical protocol identification method by adding support for UDP and handling streaming protocols using a different set of protocol attribute meters.

The research described here has drawn on this work in creating the enhanced statistical protocol identification method. The research was motivated by the fact that a forensic investigator is more interested in the precision of identification than its speed (although quicker identification is important); therefore, completed conversations are analyzed instead of just the first few packets. Additionally, as mentioned above, the intent is to identify application protocols as well as the applications themselves; therefore, approximated application messages instead of individual packets are analyzed. The enhanced statistical protocol identification method also uses a different set of features (92 features selected as described in Section 3.1) and a different method for computing the distances between measured values and learned protocol fingerprints.

The following three functions are employed:

- Function f computes the divergence of a measured value to a fingerprint value.
- Function g returns a normalized feature value for an actual measured value.
- Function w returns the weight of a feature for a protocol fingerprint.

The divergence from a learned fingerprint is computed as the Euclidean distance [7] of the weighted divergences for individual features:

$$d_{x,c} = \sqrt{\sum_{i=0}^{n} (w_i(c) \cdot f_i(g_i(x_i), c_i))^2} \tag{1}$$

where $x_1, \ldots, x_n$ denote the current flow protocol feature values; $c_1, ..., c_n$ denote the normalized feature values in the protocol fingerprint; and $w_i(c)$ denotes the weight of the i^{th} feature in protocol fingerprint c.

Equation (1) is used to compute the difference d_{x,c^j} for each protocol fingerprint c^j. The identified protocol or application k is the one such that $d_{x,c^k} = min(d_{x,c^1} \ldots d_{x,c^m})$.

Compared with other machine learning methods, the enhanced statistical protocol identification method does not suffer from overfitting due to the use of correlated features because it assigns weights on a per-feature basis. This property renders the enhanced statistical protocol identification method readily extensible to classifying new protocols and incorporating features unique to the new protocols, which could be correlated with features of other protocols.

3.3 Bayesian Network Classifier

The Bayesian network classifier [11] relies on Bayes' theorem, which defines the probability of an event based on prior knowledge about the conditions related to the occurrence of the event. The classifier incorporates Bayesian belief networks that are constructed during the learning phase. A Bayesian network is a directed acyclic graph and a set of conditional probability tables. Nodes in the network represent feature variables and edges represent conditional dependencies. The probability tables provide probability functions for the nodes.

A Bayesian network classifier identifies the application protocol by determining the node (or set of nodes) with the highest probability for the given input feature values. The advantage of the Bayesian network classifier is that it also computes the probability that the conversation belongs to the identified protocol. This information enables a forensic investigator to decide whether or not to analyze the conversation.

3.4 Random Forests Classifier

Random forests is an ensemble method that constructs multiple C4.5 decision trees during the training phase; the trees are used for classification in the verification phase, where the mode of the partial results is selected as the resulting class [4]. This makes the random forests classifier prone to overfitting [15]. Random forests are parametrized by multiple variables such as the forest count, join, and training to verification ratio. Optimal values for the parameters are determined by cross-validation and computation of an out-of-bag error that estimates the performance of specific parameter combinations. Because the classifier computes the out-of-bag error, there is no need to employ a separate data verification phase. Therefore, the random forests classifier can be trained on the entire dataset, although this approach can be computationally expensive.

4. Experimental Procedures and Results

This section presents the experimental procedures and the results obtained using the three classification methods. The experiments were designed with three goals in mind. The first goal was to compare results yielded by machine learning and statistical methods that share the same base feature set, but involve fundamentally-different approaches to classification. The second goal was to observe how the training set size and feature elimination ratio impact the accuracy of application protocol and application classification. The third goal was to prove (or disprove) that application classifiers can identify network traffic based on the applications that generated the traffic.

The Netfox Detective tool was employed as middleware for parsing and processing the captured traffic into application conversations and messages. The feature elimination algorithm and classification methods were implemented as modules in Netfox Detective for easy integration with input data. A standalone application was used to automate the experimental procedure with different parameters. The enhanced statistical protocol identification method was implemented from scratch. The Bayesian network and random forests classifiers were implemented using the Accord.NET library of machine learning algorithms.

4.1 Experimental Procedures

As mentioned above, Netfox Detective was used to parse and process the captured traffic and to extract the full set of feature values for the resulting conversations (feature vectors). Each feature vector was annotated with a label that identified the level of classification using the ground truth from the original capture file. The following labels were used:

- **Application Protocol:** Each application protocol was labeled using a tuple with the components: (i) transport protocol type; and (ii) destination transport layer port or manually assigned label (e.g., `TCP_http`).

- **Application:** Each application was labeled using a tuple with the components: (i) transport protocol type; (ii) destination transport layer port or manually assigned label; and (iii) application process information (e.g., `tcp_http_skypeexe`).

Because this task was time-intensive, but only had to be done once, the results were saved in a separate binary file. A custom application was developed to automatically execute the same experiment, but with different configuration parameter values (classification method, training to verification ratio and accepted correlation value for feature elimination).

All the experiments involved the following five steps:

- **Step 1 (Dataset Generation):** The available data was split into two disjoint datasets based on the training to verification ratio. The first dataset was used for training and the second for verification.

- **Step 2: (Feature Elimination):** The experiments using the Bayesian network and random forests classifiers used the training dataset created in Step 1 with the feature elimination algorithm described in Section 3.1. The experiments using the enhanced statistical protocol identification method employed the accepted correlation value of one to include all the features; this is because, as explained in Section 3.2, the enhanced statistical protocol identification method does not require feature elimination.

- **Step 3: (Training):** The training dataset created in Step 1 was used to train the three classifiers:
  - **Bayesian Network Classifier:** A classifier was trained for each group of feature vectors with the same label.
  - **Random Forests:** The optimal parameters specified in Section 3.4 corresponded to the most accurate classifier.
  - **Enhanced Statistical Protocol Identification Classifier:** For each group of feature vectors with the same label, an application protocol fingerprint was computed using function g.

- **Step 4 (Verification):** A cross-validation phase was used to determine the best classifiers created in Step 3. Specifically, the classifiers were used to classify each conversation from the verification dataset. They returned either: (i) multiple labels; or (ii) single labels:
  - **Multiple Labels:** Multiple labels were returned as a set of probabilities or distances. The set was ordered and the label with the highest probability or shortest distance was selected. In the case of the Bayesian network classifier, each Bayesian classifier yielded a probability of the current conversation belonging to the class of interest (application protocol or application) represented by the classifier. In the case of the enhanced statistical protocol identification classifier, the Euclidean distance between the specific conversation to each application protocol or application fingerprint was returned.
  - **Single Label:** The random forests classifier returned a single label.

- **Step 5 (Label Comparison):** In each case, the label was compared against the annotation and the statistical properties of each classification method were computed.

Table 2. Configurations of the classification methods.

Classification Method	Experiment ID	Training to Verification Ratio	Highest Feature Correlation Used
Bayesian Network	B1	0.1	0.3
	B2	0.2	0.5
	B3	0.5	0.5
	B4	0.1	0.2
	B5	0.2	0.25
	B6	0.5	0.25
ESPI	ESPI1	0.7	1
	ESPI2	0.2	1
Random Forests	RF1	0.1	0.4
	RF2	0.2	0.4
	RF3	0.1	0.5
	RF4	0.2	0.5

4.2 Experimental Results

The automated application ran many experiments with various configurations of parameters with the goal of identifying the configurations that yielded the best results. The experiments were organized based on the classification methods. For better comparisons, the most successful experiments for each method with various training to verification ratios were employed.

Table 2 lists the configurations of the classification methods with the best results. The last column specifies the highest feature correlation values used for feature elimination. The experiments were split into two categories. Experiments B1, B2, B3, ESPI1, RF1 and RF2 used classifiers for application protocol identification, for which the complete dataset contained 58 application protocol tags. The remaining experiments B4, B5, B6, ESPI2, RF3 and RF4 used classifiers for application identification, for which the complete dataset contained 93 application tags. All the experimental results are available at `pluskal.github.io/AppIdent`. The figures and tables in this section show the truncated results of the experiments. The truncation was performed by selecting the best experiment in each category as a baseline. The 20 most accurately identified labels are shown for all the experiments in a category.

The labels returned by the classification methods were compared with the ground truth from the original captured data and separated into four categories defined by the confusion matrix in Table 3. Note that a classi-

Table 3. Confusion matrix for a single label (application protocol or application).

Classification Result **Ground Truth**	**Positive**	**Negative**	**Total**
Positive	True Positive (TP)	False Positive (FP)	P
Negative	False Negative (FN)	True Negative (TN)	N
Total	P'	N'	$P+N$

fication result is positive when the classifier returns that the conversation can be labeled with the label and negative when it cannot. The ground truth is positive when the conversation in the dataset is actually labeled with the label and negative when it is not.

The F-measure, also referred to as the balanced F-score [14], was used to compare the classification methods. This single score is computed as the harmonic mean of the precision and recall using the equation:

$$F = 2 \times \frac{precision \times recall}{precision + recall} \tag{2}$$

where the precision and recall are computed from the corresponding confusion matrix values using the equations:

$$precision = \frac{TP}{TP + FP} \tag{3}$$

$$recall = \frac{TP}{TP + FN} = \frac{TP}{P} \tag{4}$$

Figure 1 presents the visualization of the application protocol identification results. The two random forest classifiers (RF1 and RF2) were very accurate. The Bayesian network classifier (B3) also performed very well, but it required a larger training set, a training to verification ratio of 0.5 and more features (see Table 2).

Figure 2 presents the visualization of the application identification results. The two random forest classifiers again yielded the best results. However, in this case, the Bayesian network classifiers were outperformed by the enhanced statistical protocol identification classifier, which also provided the best trade-off between performance and accuracy.

Figure 3 provides the aggregate statistics for all the classes. The number in each cell corresponds to the number of labels that were classified with F-measures greater than or equal to the F-measure value. Note that the size of the shaded area in a cell is proportional to the number of labels classified in the cell.

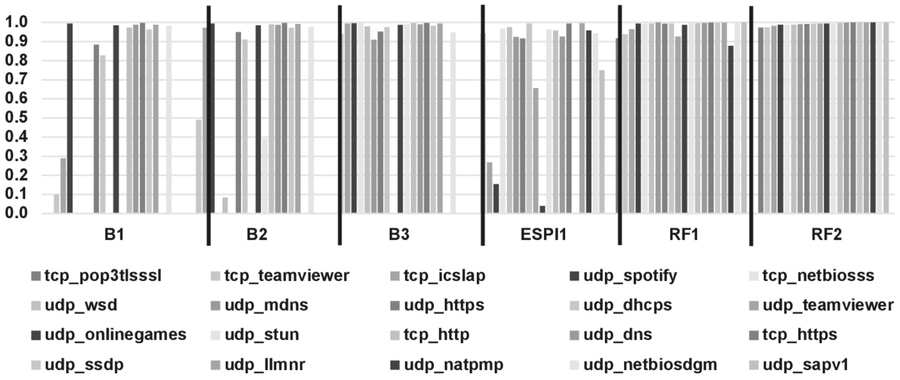


Figure 1. Performance of application protocol classifiers using the F-measure.

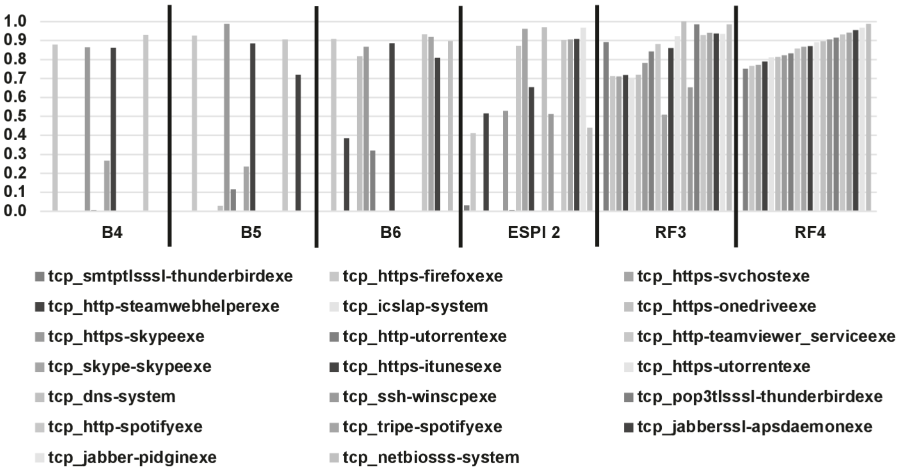


Figure 2. Performance of application classifiers using the F-measure.

Figure 4 presents the results of the performance comparison of application protocol classifiers. The first row shows the times required to complete all the steps involved in the experiments. The remaining rows show the F-measure scores of each evaluated method for the top 20 labels based on the most successful experiment in the category.

Figure 5 presents the results of the performance comparison of application classifiers. Once again, the first row shows the times required to complete all the steps involved in the experiments. The remaining rows show the F-measure scores of each evaluated method for the top 20 labels based on the most successful experiment in the category.

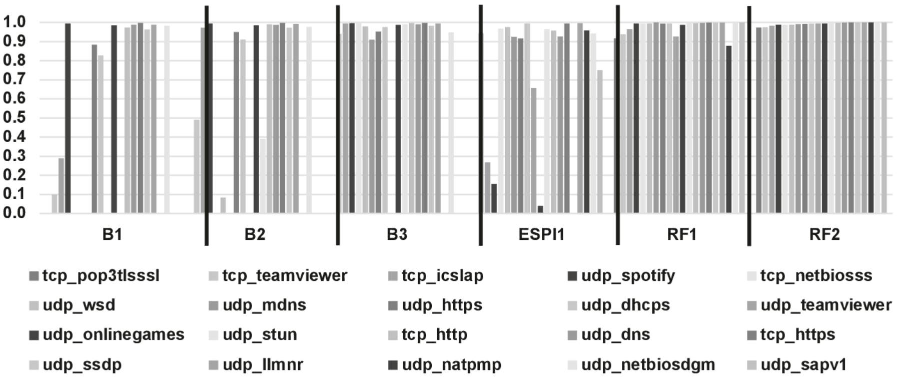


Figure 1. Performance of application protocol classifiers using the F-measure.

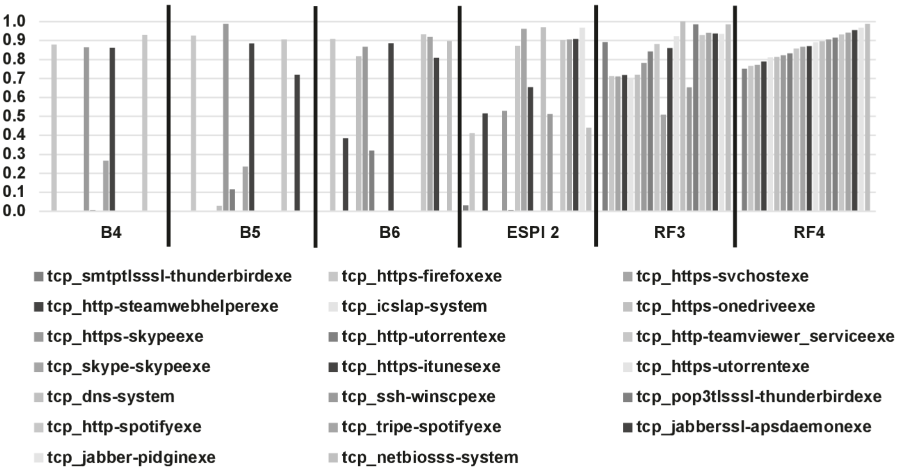
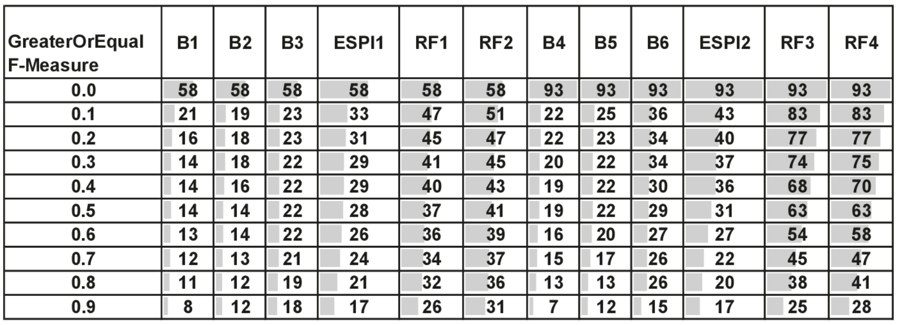


Figure 2. Performance of application classifiers using the F-measure.

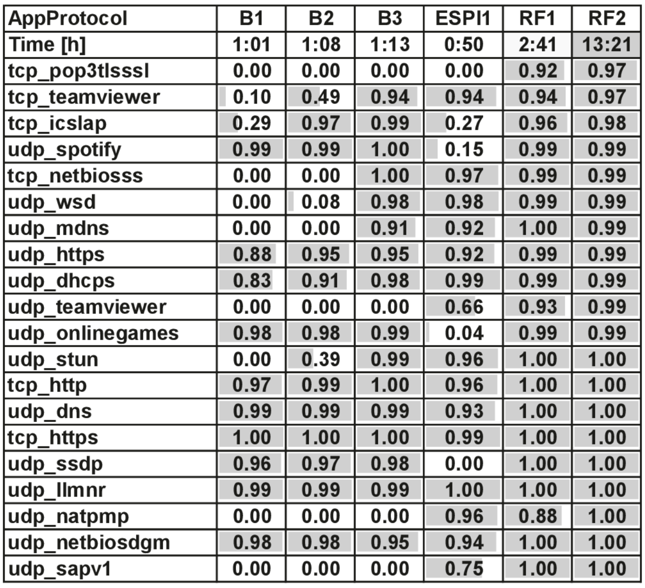
Figure 4 presents the results of the performance comparison of application protocol classifiers. The first row shows the times required to complete all the steps involved in the experiments. The remaining rows show the F-measure scores of each evaluated method for the top 20 labels based on the most successful experiment in the category.

Figure 5 presents the results of the performance comparison of application classifiers. Once again, the first row shows the times required to complete all the steps involved in the experiments. The remaining rows show the F-measure scores of each evaluated method for the top 20 labels based on the most successful experiment in the category.



GreaterOrEqual F-Measure	B1	B2	B3	ESPI1	RF1	RF2	B4	B5	B6	ESPI2	RF3	RF4
0.0	58	58	58	58	58	58	93	93	93	93	93	93
0.1	21	19	23	33	47	51	22	25	36	43	83	83
0.2	16	18	23	31	45	47	22	23	34	40	77	77
0.3	14	18	22	29	41	45	20	22	34	37	74	75
0.4	14	16	22	29	40	43	19	22	30	36	68	70
0.5	14	14	22	28	37	41	19	22	29	31	63	63
0.6	13	14	22	26	36	39	16	20	27	27	54	58
0.7	12	13	21	24	34	37	15	17	26	22	45	47
0.8	11	12	19	21	32	36	13	13	26	20	38	41
0.9	8	12	18	17	26	31	7	12	15	17	25	28

Figure 3. Summary of classification method performance.

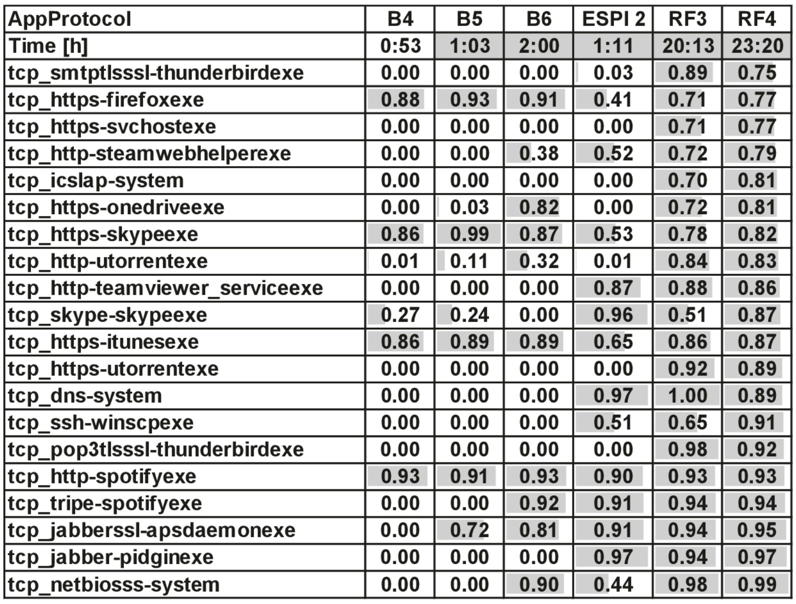


AppProtocol	B1	B2	B3	ESPI1	RF1	RF2
Time [h]	1:01	1:08	1:13	0:50	2:41	13:21
tcp_pop3tlsssl	0.00	0.00	0.00	0.00	0.92	0.97
tcp_teamviewer	0.10	0.49	0.94	0.94	0.94	0.97
tcp_icslap	0.29	0.97	0.99	0.27	0.96	0.98
udp_spotify	0.99	0.99	1.00	0.15	0.99	0.99
tcp_netbiosss	0.00	0.00	1.00	0.97	0.99	0.99
udp_wsd	0.00	0.08	0.98	0.98	0.99	0.99
udp_mdns	0.00	0.00	0.91	0.92	1.00	0.99
udp_https	0.88	0.95	0.95	0.92	0.99	0.99
udp_dhcps	0.83	0.91	0.98	0.99	0.99	0.99
udp_teamviewer	0.00	0.00	0.00	0.66	0.93	0.99
udp_onlinegames	0.98	0.98	0.99	0.04	0.99	0.99
udp_stun	0.00	0.39	0.99	0.96	1.00	1.00
tcp_http	0.97	0.99	1.00	0.96	1.00	1.00
udp_dns	0.99	0.99	0.99	0.93	1.00	1.00
tcp_https	1.00	1.00	1.00	0.99	1.00	1.00
udp_ssdp	0.96	0.97	0.98	0.00	1.00	1.00
udp_llmnr	0.99	0.99	0.99	1.00	1.00	1.00
udp_natpmp	0.00	0.00	0.00	0.96	0.88	1.00
udp_netbiosdgm	0.98	0.98	0.95	0.94	1.00	1.00
udp_sapv1	0.00	0.00	0.00	0.75	1.00	1.00

Figure 4. Performance comparison of application protocol classifiers.

5. Conclusions

This research has focused on the important network forensics problem of identifying network applications in addition to just application protocols in network traffic flows. It has studied various aspects of applying machine learning methods and the selection of features that characterize application behavior, such as message timing, content length and TCP flags instead of features related to network line characteristics. An automated feature elimination method based on the feature correlation



AppProtocol	B4	B5	B6	ESPI 2	RF3	RF4
Time [h]	0:53	1:03	2:00	1:11	20:13	23:20
tcp_smtptlsssl-thunderbirdexe	0.00	0.00	0.00	0.03	0.89	0.75
tcp_https-firefoxexe	0.88	0.93	0.91	0.41	0.71	0.77
tcp_https-svchostexe	0.00	0.00	0.00	0.00	0.71	0.77
tcp_http-steamwebhelperexe	0.00	0.00	0.38	0.52	0.72	0.79
tcp_icslap-system	0.00	0.00	0.00	0.00	0.70	0.81
tcp_https-onedriveexe	0.00	0.03	0.82	0.00	0.72	0.81
tcp_https-skypeexe	0.86	0.99	0.87	0.53	0.78	0.82
tcp_http-utorrentexe	0.01	0.11	0.32	0.01	0.84	0.83
tcp_http-teamviewer_serviceexe	0.00	0.00	0.00	0.87	0.88	0.86
tcp_skype-skypeexe	0.27	0.24	0.00	0.96	0.51	0.87
tcp_https-itunesexe	0.86	0.89	0.89	0.65	0.86	0.87
tcp_https-utorrentexe	0.00	0.00	0.00	0.00	0.92	0.89
tcp_dns-system	0.00	0.00	0.00	0.97	1.00	0.89
tcp_ssh-winscpexe	0.00	0.00	0.00	0.51	0.65	0.91
tcp_pop3tlsssl-thunderbirdexe	0.00	0.00	0.00	0.00	0.98	0.92
tcp_http-spotifyexe	0.93	0.91	0.93	0.90	0.93	0.93
tcp_tripe-spotifyexe	0.00	0.00	0.92	0.91	0.94	0.94
tcp_jabberssl-apsdaemonexe	0.00	0.72	0.81	0.91	0.94	0.95
tcp_jabber-pidginexe	0.00	0.00	0.00	0.97	0.94	0.97
tcp_netbiosss-system	0.00	0.00	0.90	0.44	0.98	0.99

Figure 5. Performance comparison of application classifiers.

matrix was employed to improve the classification results. Additionally, this research has developed the enhanced statistical protocol identification method, which was compared against the Bayesian network and random forests classification methods from the literature that offer high accuracy and acceptable performance.

The experimental results confirm that application protocols as well as the applications that generate network traffic can be classified with high confidence. For example, NetBIOS service and DNS were identified accurately and several common applications that use the HTTP(S) application protocol were identified with high accuracy. Similarly, it was possible to distinguish between communications traces of OneDrive, Skype, iTunes, Spotify, Steam and μTorrent clients, although all of them use the same application protocol (HTTPS).

The random forests classifier achieved the best results, confirming the results obtained by other researchers [20, 30] who experimented with machine learning approaches for traffic classification. The enhanced statistical protocol identification classifier yielded better results than the Bayesian network classifier and was much faster than the Bayesian network and random forests classifiers.

Classification accuracy is mainly determined by the quality of the selected features. This research has employed features based on previous observations and intuition. Future research should focus on the systematic analysis and selection of feature sets that could improve classification accuracy and robustness.

To improve the identification of applications that employ the same application protocol (e.g., removing errors when `tcp_http_skypeexe` is classified as `tcp_http_firefoxexe`, or vice-versa), future research should focus on hierarchical classification methods. An example is hierarchical clustering based on enhanced statistical protocol identification fingerprints. A forensic investigator could then infer the actual application classes by visual cluster analysis. This approach could also be extended to other levels such as application message level.

Future research should also consider combining multiple classifiers [18] to increase the confidence in the results. Research should also focus on semi-supervised classification methods [8] that enable the creation of models from partially-labeled data.

Finally, experiments should be conducted to extend the classification models and evaluate the properties of other datasets. The classification methods considered in this work require accurate models. Creating such models requires the analysis of large numbers of traffic samples. Experimenting with different datasets could provide more accurate classification models and valuable insights into the properties of individual classification methods.

A reference implementation is available under an MIT license from GitHub at `pluskal.github.io/AppIdent`. This includes the framework for parsing captured data, extracting features and eliminating features, along with the three classifiers described in this chapter and the standalone application that automated the experiments. The dataset is available at `nes.fit.vutbr.cz/AppIdent` to facilitate the reproducibility of the experiments and to serve as a benchmarking platform for testing other machine-learning-based application identification methods.

References

[1] S. Alcock and R. Nelson, Libprotoident: Traffic Classification Using Lightweight Packet Inspection, Technical Report, WAND Network Research Group, Computer Science Department, University of Waikato, Hamilton, New Zealand, 2012.

[2] N. Al Khater and R. Overill, Forensic network traffic analysis, *Proceedings of the Second International Conference on Digital Security and Forensics*, 2015.

[3] T. Auld, A. Moore and S. Gull, Bayesian neural networks for Internet traffic classification, *IEEE Transactions on Neural Networks*, vol. 18(1), pp. 223–239, 2007.

[4] L. Breiman, Random forests, *Machine Learning*, vol. 45(1), pp. 5–32, 2001.

[5] E. Bursztein, Probabilistic identification of hard to classify protocols, *Proceedings of the Second IFIP WG 11.2 International Conference on Information Security Theory and Practices: Smart Devices, Convergence and Next Generation Networks*, pp. 49–63, 2008.

[6] S. Dai, A. Tongaonkar, X. Wang, A. Nucci and D. Song, NetworkProfiler: Towards automatic fingerprinting of Android apps, *Proceedings of the IEEE International Conference on Computer Communications*, pp. 809–817, 2013.

[7] M. Deza and E. Deza, *Encyclopedia of Distances*, Springer-Verlag, Berlin Heidelberg, Germany, 2009.

[8] J. Erman, A. Mahanti, M. Arlitt, I. Cohen and C. Williamson, Offline/real-time traffic classification using semi-supervised learning, *Performance Evaluation*, vol. 64(9-12), pp. 1194–1213, 2007.

[9] A. Finamore, M. Mellia and M. Meo, Mining unclassified traffic using automatic clustering techniques, *Proceedings of the Third International Conference on Traffic Monitoring and Analysis*, pp. 150–163, 2011.

[10] V. Foroushani and A. Zincir-Heywood, Investigating application behavior in network traffic traces, *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pp. 72–79, 2013.

[11] N. Friedman, D. Geiger and M. Goldszmidt, Bayesian network classifiers, *Machine Learning*, vol. 29(2-3), pp. 131–163, 1997.

[12] G. Gomez Sena and P. Belzarena, Early traffic classification using support vector machines, *Proceedings of the Fifth International Latin American Networking Conference*, pp. 60–66, 2009.

[13] I. Guyon and A. Elisseeff, An introduction to variable and feature selection, *Journal of Machine Learning Research*, vol. 3(March), pp. 1157–1182, 2003.

[14] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, Waltham, Massachusetts, 2012.

[15] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning – Data Mining, Inference and Prediction*, Springer, New York, 2009.

[16] E. Hjelmvik, The SPID Algorithm – Statistical Protocol Identification, Gavle, Sweden (`www.iis.se/docs/The_SPID_Algorithm_-_Statistical_Protocol_IDentification.pdf`), 2008.

[17] J. Khalife, A. Hajjar and J. Diaz-Verdejo, A multilevel taxonomy and requirements for an optimal traffic-classification model, *International Journal of Network Management*, vol. 24(2), pp. 101–120, 2014.

[18] J. Kittler, Combining classifiers: A theoretical framework, *Pattern Analysis and Applications*, vol. 1(1), pp. 18–27, 1998.

[19] C. Kohnen, C. Uberall, F. Adamsky, V. Rakocevic, M. Rajarajan and R. Jager, Enhancements to statistical protocol identification (SPID) for self-organized QoS in LANs, *Proceedings of the Nineteenth International Conference on Computer Communications and Networks*, 2010.

[20] J. Li, S. Zhang, Y. Xuan and Y. Sun, Identifying Skype traffic by random forests, *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 2841–2844, 2007.

[21] Y. Luo, K. Xiang and S. Li, Acceleration of decision tree searching for IP traffic classification, *Proceedings of the Fourth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pp. 40–49, 2008.

[22] P. Matousek, J. Pluskal, O. Rysavy, V. Vesely, M. Kmet, F. Karpisek and M. Vymlatil, Advanced techniques for reconstruction of incomplete network data, *Proceedings of the Seventh International Conference on Digital Forensics and Cyber Crime*, pp. 69–84, 2015.

[23] S. Miskovic, G. Lee, Y. Liao and M. Baldi, AppPrint: Automatic fingerprinting of mobile applications in network traffic, *Proceedings of the Sixteenth International Conference on Passive and Active Measurement*, pp. 57–69, 2015.

[24] A. Moore and K. Papagiannaki, Toward the accurate identification of network applications, *Proceedings of the Sixth International Workshop on Passive and Active Network Measurement*, pp. 41–54, 2005.

[25] A. Moore, D. Zuev and M. Crogan, Discriminators for Use in Flow-Based Classification, Technical Report RR-05-13, Department of Computer Science, Queen Mary, University of London, London, United Kingdom, 2013.

[26] N. Namdev, S. Agrawal and S. Silkari, Recent advancements in machine learning based Internet traffic classification, *Procedia Computer Science*, vol. 60, pp. 784–791, 2015.

[27] T. Nguyen and G. Armitage, A survey of techniques for Internet traffic classification using machine learning, *IEEE Communications Surveys and Tutorials*, vol. 10(4), pp. 56–76, 2008.

[28] C. Shen and L. Huang, On the detection accuracy of the `l7-filter` and OpenDPI, *Proceedings of the Third International Conference on Networking and Distributed Computing*, pp. 119–123, 2012.

[29] P. Velan, M. Cermak, P. Celeda and M. Drasar, A survey of methods for encrypted traffic classification and analysis, *International Journal of Network Management*, vol. 25(5), pp. 355–374, 2015.

[30] Y. Wang and S. Yu, Machine learned real-time traffic classifiers, *Proceedings of the Second International Symposium on Intelligent Information Technology Applications*, vol. 3, pp. 449–454, 2008.

[31] I. Zezula, On multivariate Gaussian copulas, *Journal of Statistical Planning and Inference*, vol. 139(11), pp. 3942–3946, 2009.

[32] L. Zhen and L. Qiong, A new feature selection method for Internet traffic classification using ML, *Physics Procedia*, vol. 33, pp. 1338–1345, 2012.

A.9 Advanced Techniques for Reconstruction of Incomplete Network Data

Petr Matoušek, Jan Pluskal, Ondřej Ryšavý, Vladimír Veselý, Martin Kmeť, Filip Karpíšek, and Martin Vymlátil. “Advanced Techniques for Reconstruction of Incomplete Network Data”. In: *Digital Forensics and Cyber Crime.* Ed. by Joshua I. James and Frank Breitinger. Cham: Springer International Publishing, 2015, pp. 69–84. ISBN: 9783319255125

Advanced Techniques for Reconstruction of Incomplete Network Data

Petr Matoušek(✉), Jan Pluskal, Ondřej Ryšavý, Vladimír Veselý,
Martin Kmeť, Filip Karpíšek, and Martin Vymlátil

Brno University of Technology, Božetěchova 2, Brno, Czech Republic
{matousp,ipluskal,rysavy,ivesely,ikmet,ikarpisek}@fit.vutbr.cz,
xvymla01@stud.fit.vutbr.cz
http://www.fit.vutbr.cz

Abstract. Network forensics is a method of obtaining and analyzing digital evidences from network sources. Network forensics includes data acquisition, selection, processing, analysis and presentation to investigators. Due to high volumes of transmitted data the acquired information can be incomplete, corrupted, or disordered which makes further reconstruction difficult. In this paper, we address the issue of advanced parsing and reconstruction of incomplete, corrupted, or disordered data packets. We introduce a technique that recovers TCP or UDP conversations so they could be further analyzed by application parsers. Presented technique is implemented in a new network forensic tool called Netfox Detective. We also discuss current challenges in parsing web mail communication, SSL decryption and Bitcoins detection.

Keywords: Network forensic tools · TCP reassembling · Traffic reconstruction · Web mail · Bitcoin · SSL encryption

1 Introduction

Network forensics is an emerging area of digital forensics connected with the rapid network development. Many services and digital transactions are transmitted over the Internet where criminal activities and security incidents also occur. Network forensics provides post-mortem investigation of unlawful behavior using special tools that reconstruct a sequence of events occurred at the time of the attack. This reconstruction depends only on a captured network data. In some cases, these data are incomplete, corrupted, or out of order. In order to analyze the original communication using an incompletely captured data, advanced techniques of reconstruction and communication recovery are needed. Reconstruction of TCP streams is essential for any network forensic tool [1]. If the TCP reassembling fails, application data cannot be properly analyzed.

Recovery of incomplete data in network forensics is a similar task to data recovery from damaged media, e.g., hard drives, CDs, or DVDs. If some data are missing, it can be either replaced by empty data units or approximated

J.I. James and F. Breitinger (Eds.): ICDF2C 2015, LNICST 157, pp. 69–84, 2015.
DOI: 10.1007/978-3-319-25512-5_6

from known data. The goal is to provide enough data enabling reconstruction of the original content. To guarantee an admissibility of forensic results newly introduced data must be unambiguously distinguished from the original ones.

In this work, we deal with the analysis and reconstruction of incomplete or damaged network data. Our research includes the development of heuristic techniques that can detect incomplete or corrupted data on network and transport layer and restore original sessions that can be further analyzed using usual application parsers. The proposed technique was implemented in a new network forensic tool *Netfox Detective*.

1.1 Contribution

The main contribution of this paper addresses practical issues connected with network data reconstruction and proposes advanced techniques for parsing and recovery of network conversations. These techniques in combination with advanced application recognition methods increase the accuracy of content reconstruction. We also explain several issues connected with application analysis, especially with web mail services, SSL communication and Bitcoin transactions. We evaluate the implementation of proposed methods and compare them with other tools.

The paper is organized as follows: section two surveys current approaches and results in the domain of network forensic tools; section three examines issues related to network data parsing and reconstruction with focus on TCP reassembling and Layer 7 (L7, application) data reconstruction; section four deals with application detection and content analysis, which is demonstrated using examples of reconstruction of web mail, SSL traffic, and bitcoin transactions.

2 Related Work

There is a wide range of tools for network monitoring and forensics, i.e., Network Security and Monitoring tools (NSMs) and Network Forensic Analysis Tools (NFATs). NSMs include network analyzers (Wireshark, tcpdump), IDS systems (snort, Bro), fingerprinting tools (nmap, p0f), and others [2]. NFATs have similar functionality as NSMs, in addition, they also assist in a network crime investigation. They capture an entire network traffic and allow an investigator to analyze it and reconstruct the original communication. Most of the NFAT tools are proprietary, nevertheless, open source NFATs also exist, e.g., PyFlag, Network Miner, or Xplico.

In theory, parsing the network communication is straightforward. However, incompleteness and corruption of communication requires new methods involving robust parsers and complex recovery procedures. Surveys of different network forensic frameworks can be found in [2,3]. These papers discuss various approaches to network forensics, major challenges, and list available tools. In our paper, we mostly focus on techniques of network data parsing and recovery.

There are not many published works describing techniques incorporated in NFAT implementations, partly due to the protection of intellectual properties of the tools. An exception is Cohen [1] that describes several challenges connected with the stream reassembling (termination of streams, out of sequence packets, missed packets) and the combination of streams into conversations. In our work, we deeply examine issues that are essential for every network forensic tool. In addition to [1], we present an algorithm that deals with these issues, and also works with sequence number overflow, which is not discussed by other authors. A detailed description of TCP reassembling is analyzed by Paxson in [4]. However, Paxson focuses on robustness of TCP reassembling in the presence of adversaries that is out of the interest of this paper.

3 Data Parsing and Reconstruction

NFATs are designed to parse captured data, process packet headers and reconstruct high-level protocol units. Application data are regularly transmitted using TCP or UDP protocols over IP networks. By definition, IP communication does not provide reliable data exchange [5]. Application data are segmented into TCP packets and encapsulated into IP datagrams. Furthermore, IP datagrams can be fragmented into smaller IP datagrams when required by an underlying link-layer technology. The main goal of an NFAT is to extract and reconstruct original application data from possibly incomplete captured collection of IP datagrams. The method for assembling IP packet-based communications into conversations is based on the following assumptions:

- An application conversation is distinguished by a pair of IP addresses, transport ports and a protocol type. The conversation consists of a pair of flows because the most of sessions are bi-directional.
- The beginning of a TCP session is identified by a synchronization TCP segment (SYN flag). A TCP segment with FIN/PSH/RST flag closes the session.
- A TCP session consists of a collection of TCP segments each associated with a sequence number. A sequence number determines an offset of the segment content in the TCP stream [6].
- An application message can be transmitted in one or more TCP segments. Receiver must reassemble several TCP segments to obtain the original message.
- The IP fragmentation happens independently on the TCP segmentation. The IP defragmentation has to be accomplished before the application content reassembling.

3.1 Challenges in TCP Reassembling

During our research of network data analysis, following challenges connected with reassembling of TCP sessions have been identified:

- *Missing FIN packets or overlapping of TCP conversations.*
  Regularly, ephemeral source ports are dynamically assigned by OS to clients whenever a communication socket is created [7, p. 99]. It helps to distinguish several TCP sessions originating from the same node and targeting the same remote process. When the client finishes communication, these ports can be reused. Usually, the port number is not reused until the pool of ephemeral ports is exhausted. NFAT can exploit this behavior to recognize different TCP sessions safely. However, if there is a NAT translation along the communication path observable port numbers can be reused quickly. In such case, different TCP sessions can receive the same key fields within a relatively short period. While end systems and NAT can accurately track the use of port numbers, for NFAT system it may pose a problem as there is a very short interval between two TCP sessions with the same identification. NFAT can proceed as follows:
  1. FIN segment can determine closing of the first session segment while SYN segment defines a new TCP session;
  2. if these segments are missing in a captured collection, a flow needs to be detected by analyzing sequence numbers;
  3. if sequence numbers of two sessions overlap, the analysis of timestamps of expected L4 packets have to be carried out.
- *Combination of two L7 flows into a L7 conversation.*
  NFATs try to reconstruct original bi-directional communication between applications. If more TCP conversations use the same IP addresses and ports (see NAT problem above), these ports are not sufficient to unambiguously combine corresponding L7 flows into a whole L7 conversation. The proposed solution suggests considering initial TCP sequence numbers. TCP three-way handshake starts with sending three synchronization segments between a sender and a receiver. The sender sends a SYN segment with his initial, randomly chosen, sequence number. The receiver replies with an SYN+ACK segment transmitting receiver's initial sequence number and sender's next sequence number. Based on hand-shake analysis, we can match initial TCP sequence numbers of every L7 flow and its opposite L7 flow, which is necessary to create bi-directional L7 conversation based on L4 header data only. If the hand-shake is not captured, L7 flows are considered as one-directional L7 conversations.
- *TCP sequence number overflow.*
  Network data parsing and analysis is mostly based on a chronological order of packets in the flow using their sequence numbers. According to RFC 793 [6], sequence numbers occupy space up to $2^{32} - 1$ Bytes, which gives possibility to transmit maximum 4.29 GB data. This value seems large enough to avoid sequence number overflow. However, since initial sequence numbers are generated randomly, maximum data size is lower than this theoretical value. Figure 1 shows a snapshot of the distribution of maximum TCP message sizes based on randomly generated initial sequence numbers as observed on 14,000 TCP sessions. The picture does not show full distribution range. TCP sessions with possible payload greater than 500 MB are excluded, because of their irrelevance for our study. However, these data show that TCP sequence

number overflow should be taken seriously. For example, we can see that the sequence number would overflow in 0,12 % of TCP sessions with payload up to 5 MB. This situation can be solved by multi-pass processing of an L4 conversation and matching incomplete TCP sessions without SYNs when their initial sequence numbers are closed to 2^{32}.

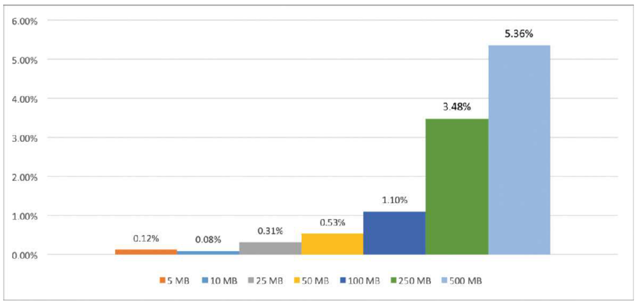


Fig. 1. Probability of TCP Seq numbers overflow related to maximal L7 payload size.

3.2 Building L7 PDUs from the PCAP File

The process of network data parsing starts with the tracking of L3 conversations based on sender's and receiver's IP addresses, see Fig. 2. Further, L4 conversations are identified using port numbers and L4 protocol type, than L7 conversations are created. In case of UDP protocol, two UDP sessions running between the same pair of ports cannot be distinguished. For example, SIP applications regularly employ the same source and destination ports, e.g., 5060, for all SIP conversations. Therefore, a L4 UDP conversation is considered to be a L7 conversation.

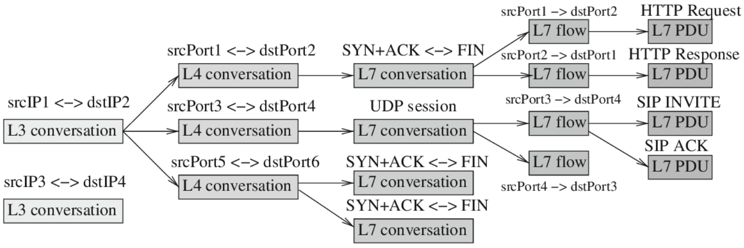


Fig. 2. Extraction of L7 PDUs from input packets.

In case of TCP protocol, the TCP reassembling is the key element in reconstruction. If all data have been properly captured, TCP reassembling is a simple task that involves port numbers, TCP sequence and acknowledgment numbers. If some packets are missing, a following procedure implementing our heuristic method can be applied to any network data. The procedure uses three heuristic parameters: *MaxLost*, which represents the maximal length of missing data that can be restored, *MaxTime*, describing the maximal permitted time delay between two consequent packets using timestamps, and *MaxPayload*, representing the maximum payload size in a TCP packet. Based on our experience, we use $MaxLost = 4kB$ and $MaxTime = 600\ sec$[1]. *MaxPayload* is computed on-the-fly as the length of the TCP packet with the maximal size of a payload in the L7 flow. Thus, application messages are built from captured data using the following steps:

1. Select L4 flows and sort packets using their sequence numbers.
2. Process each L4 flow and create L7 flows using TCP handshake. Start with the first SYN packet.
   (a) Create a new L7 PDU if does not exist or if a previous L7 PDU was closed.
   (b) Check packet sequence number Seq_{i+1}.
   (c) If $Seq_{i+1} \neq Seq_i + PS_i$ (PS stands for a payload size obtained from the packet header), i.e., the expected packet is missing, check timestamps TS and sequence numbers Seq as follows:
      i. If $TS_{i+1} - TS_i \leq MaxTime$ and $Seq_{i+1} - Seq_i \leq MaxLost$ then a virtual packet will be created to replace the missing packet.
      ii. If $TS_{i+1} - TS_i \geq MaxTime$ and $Seq_{i+1} - Seq_i \leq MaxLost$ then there is an overlapping of TCP sessions because $i+1$ packet belongs to a different L7 flow. Skip this packet and proceed with the next one.
      iii. If $Seq_{i+1} - Seq_i \geq MaxLost$ then there are too many missing data. The flow cannot be fully restored. Close it and proceed with next SYN packet.
   (d) If $Seq_{i+1} = Seq_i + PS_i$ the expected packet is present, add it into the L7 PDU.
   (e) If FIN/RST/PSH flag is found or $PS = MaxPayload$, close the L7 PDU.
   (f) GOTO 2a.
3. Process remaining packets without SYNs. Create new L7 flows using timestamps and sequence numbers only.
4. Process every L7 flow and create L7 PDUs using TCP reassembling

[1] *MaxLost* was experimentally set to 4 kB, which is more than two times greater than maximal Ethernet PDU size, i.e., 1500 Bytes. *MaxTime* is six times greater than recommended TCP connection failure timeout as defined in RFC 1122. These values say that packet loss longer than 600 secs or missing 4 kB cannot be successfully recovered.

- Add every packet of the L7 flow into the L7 PDU until FIN/RST/PSH or $PS = MaxPayload$. Then close the L7 PDU and create new one for new packets.

5. Combine opposite L7 flows into a L7 conversation using corresponding SYN and ACK numbers.

The main benefit of this approach is the reconstruction of original UDP/TCP sessions even if some important packets are missing. Based on TCP initial Seq numbers, the algorithm combines two flows into a conversation. The algorithm deals with missing SYNs, FINs, overlapping sessions, or TCP numbers overflowing. As the result, we have L7 PDU objects that can be processed on L7.

Table 1 compares our approach with a few available NSMs or NFATs. For our study, we have chosen Wireshark, Microsoft Network Monitor, NetWitness and Network Miner. In the first test we used an artificially arranged dataset with (i) one FIN packet missing, (ii) one SYN packet missing, and (iii) two SYNs missing. Original 650 kB PCAP file contained 19 conversions. Further analysis showed that in case of missing SYNs and the same port numbers, Wireshark joins two conversations into one. MS Network Monitor works well with missing SYNs, but it is not able to properly close communication if a FIN is missing. In such case, it combines two conversations into one. NetWitness also joins two conversations into one. Network Miner works similarly to Wireshark.

Table 1. Detection of network conversation when missing SYN/FIN packets.

File	NFX Det	Wireshark	MS Monitor	NetWitness	Net Miner
One FIN missing	19	19	18	17	19
One SYN missing	19	18	19	17	18
Two SYNs missing	19	17	19	17	17

The second test used 8 MB PCAP file with some packets randomly deleted. Table 2 shows results when 0 %, 1 %, 5 %, or 10 % of packets were removed. Original file contained 126 conversations. Netfox Detective shows number of L7 conversations.

Table 2. Detection of network conversations when some data are deleted.

File	NFX Det	Wireshark	MS Monitor	NetWitness	Net Miner
0 % missing	126	126	132	128	76
1 % missing	126	126	132	128	75
5 % missing	129	125	129	127	71
10 % missing	131	125	129	127	66

The table shows that Netfox Detective finds more L7 conversations than originally stored in the in-corrupted file. The reason is that when some packets are missing, a corrupted L7 conversation is divided into several L7 conversation due to the large number of missing packets or large timestamp difference, see Fig. 3. Wireshark and NetWitness also miss a conversation. However, since they consider all packets between the same src/dst ports as one conversation, missing packets usually did not reduce number of all conversations. MS Network Monitor also shows stable results. The results of Network Miner are very different but we are not able to say why.

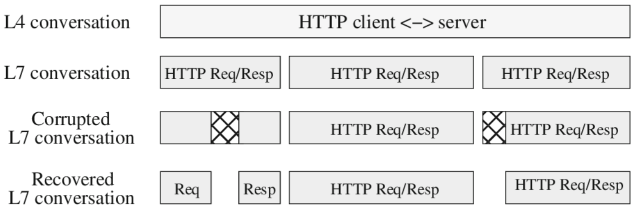


Fig. 3. Recovery of corrupted conversations.

3.3 Application Protocol Identification

The result of previously described reconstruction methods are L7 PDUs that represent L7 objects (payloads) prepared for L7 parsing. Before L7 parsing, L7 protocol should be identified in order to choose the right L7 parser. There are many methods for application protocol identification. The easiest method is based on well-known port numbers assigned by Internet Assigned Numbers Authority (IANA). Unfortunately, this method does not work well with applications using dynamic ports, peer-to-peer communication, video streaming, etc. More advanced methods use payload inspection that is suitable for protocols that can be recognized by some characteristic patterns either in a header or payload. There are also methods based on protocol fingerprinting or statistical data. In our approach, we combine several methods for application protocol identification.

1. *Identification using extended IANA database.*
   The first algorithm matches port numbers with extended IANA database of well-known ports. Our database extends IANA data by similarities, i.e., one input port number can match more applications. For example, Dropbox file hosting service can work on ports 80, 443, or 17500. Based on given application tags, L7 parser is chosen. Currently, our database can identify 1058 different application protocols.

2. *RTP Fingerprinting.*
   If there is no match on input ports, RTP fingerprinting method is applied [8]. This method uses a multi-stage classifier that observes minimal RTP header length, RTP version number, and RTP payload type number. If a packet successfully passes this filtering, per-flow checking is applied using minimal number of packets in an RTP flow to reduce false positives.
3. *Statistical Protocol Identification (SPID).*
   This method developed by Erik Hjelmvik [9] is based on supervised learning using pre-classified samples of captured network traffic where application protocols are correctly annotated. The algorithm generates protocol model database that stores application fingerprints. Currently, our database can identify 20 protocols with an ability to add new protocols.

4 Application Parsing

After building L7 PDUs and successful L7 protocol identification, application data can be processed by L7 parsers. As mentioned in Chap. 3, TCP/UDP streams are reconstructed without any knowledge of higher layers. This helps in case when an application parser is not implemented for a specific protocol. In that case application data can also be extracted from communication.

Main goal of our approach is to augment the reconstruction process when some data are missing. As mentioned earlier if only a few data is missing, lost packets can be replaced by new packets with empty payload. If more packets are lost, an original stream will be recovered as a collection of shorter streams that formed the original stream.

In this section, we will discuss how data reconstruction influences L7 processing and data presentation in case of incomplete data. For demonstration, we choose three areas that build challenges for common network parsers: web mail communication, SSL/TLS encrypted traffic, and bitcoin transactions.

4.1 Web Mail Analysis

Web mail communication is very popular today. Web mail servers employ HTTP protocol to encapsulate transactions between a user web browser and a web mail server. Mail exchange between web mail servers is mostly provided using SMTP protocol. Forensic analysis of web mail services is different from common web browsing. Many web mail servers utilize advanced web technologies like JavaScript, AJAX, JSON that dynamically create web pages. Analysis and interpretation of captured web mail data are limited due to the usage of web browser caches that store frequently used HTTP objects. These objects are not present in captured traffic, therefore, they are unavailable for forensic analysis.

The web mail analysis includes two phases: (i) the identification of web mail data between other HTTP traffic and (ii) the analysis of captured web mail data. In addition, most of web mail transmissions are SSL/TLS encrypted, so SSL/TLS decryption is required if possible (see Sect. 4.2). If encrypted, web mail traffic can be identified using a name or IP address of a particular web mail server, see Table 3. If not encrypted, a pattern matching on URLs can be applied.

Table 3. Identification of web mail services during SSL/TLS handshake.

Web mail service	Server name	Encoding
seznam.cz, email.cz	email.seznam.cz	FastRPC
Gmail	mail-attachment.googleusercontent.com	application/x-www-form-urlencoded;charset=utf-8
Yahoo	mail.yahoo.cz	application/json multipart/form-data-incl JSON
MS Live	*various*	application/x-www-form-urlencoded
Centrum/Atlas	mail.centrum.cz	application/x-www-form-urlencoded
Roundcube	*private service hostname*	application/x-www-form-urlencoded
Horde	*private service hostname*	multipart/form-data

For processing of a captured web mail data, following observations were made:

- Web mail messages transmitted over HTTP can be detected using URL patterns: _/mail/.*_ for Gmail, *o1/mail.fpp* for MS Live Mail, *appid=YahooMailNeo* for Yahoo, etc. However, these patterns usually change with a new version of the server.
- The communication from a user towards the server is transmitted via POST method of HTTP protocol [10]. GET method is employed for listing mail folders.
- Web mail messages are mostly encoded using simple *key=value* pairs in the URL. There are several types of actions that can be identified in a *key* field: *compose-message*, *send-message*, *save-draft*, *get-inbox*, *delete-message*. Each web mail service uses different names for these actions, so data analysis should be performed for every new web mail protocol.
- Some web mail objects can be transmitted as JSON objects in MIME structure, XML-RPC objects, etc.
- Because of dynamic web programming and client-based technologies (i.e., JavaScript), forensic page rendering of web mail is difficult and cannot be fully accomplished without having contents of web caches. Practically, investigator's view is limited to a simple textual form of analyzed data.

4.2 SSL/TLS Detection and Encryption

The SSL/TLS encryption is a big challenge for current NFAT tools because it completely hides the contents of the network communication. It forms a modular framework that combines various cryptography mechanisms defined by a cipher suite [11]. Clients and servers can negotiate cipher suites to meet specific security and administrative policies during initial SSL/TLS handshake. The cipher suite defines following mechanisms:

- *A key exchange algorithm.* General goal of the key exchange process is to create a pre-master secret known to the communicating parties that is used to generate the master secret. Using master secret encryption keys and MAC keys are generated. Most common key exchange algorithms are RSA, Diffie-Hellman, ECDH, etc.
- *A peer authentication.* TLS supports authentication of both peers, the server authentication with an unauthenticated client, and total anonymity. Whenever the server is authenticated, the channel is secure against man-in-the-middle attacks. Server authentication mostly requires a RSA or DSA certificate to prove an authenticity of the server side.
- *Message integrity.* Message integrity is ensured using Message Authentication Code (MAC) algorithms like MD5, SHA1, or SHA256. A cryptographic hash (often called message digest) is computed using these algorithms and added to the end of each block.
- *A bulk cipher algorithm.* This algorithm is used for a message encryption. The specification includes the cipher type (stream, block, AEAD [12]), the key size, the block size of the cipher (applied only to block ciphers), and the length of initialization vectors (or nonces). Common bulk ciphers are RC4, 3DES, AES, IDEA, or Camellia.

There are two basic approaches for SSL/TLS decryption [13]:

- *A getting server private key.* This key can be used to calculate a session key that have encrypted the conversation. The session key is generated during the key exchange.
- *A MitM attack on SSL/TLS connection.* Another method to get decrypted contents is to use man-in-the-middle (MitM) attack employing a special proxy server to track the communication between the client and server. At the same time, the communication with the user node employs different TLS keys generated by the proxy server. In this case, proxy server should offer a fake certificate in order to impersonate the original server. There are several tools implementing this proxy, e.g., SSLsplit, Fidler, etc.

Bulk cipher algorithms incorporate methods of a block cipher or stream cipher encryption that defines how a block or stream of a plain text will be encrypted and how the encryption key is generated for each data block, e.g. CBC (Cipher Block Chaining), GCM (Galois/Counter).

- The Cipher Block Chaining requires complete data for successful reconstruction because of data dependency, see Fig. 4A. If data are corrupted, successful analysis can be provided until the first error occurs in the stream. In such case, only meta information about the conversation are available, e.g. TCP completeness, probable conversation length, duration, etc.

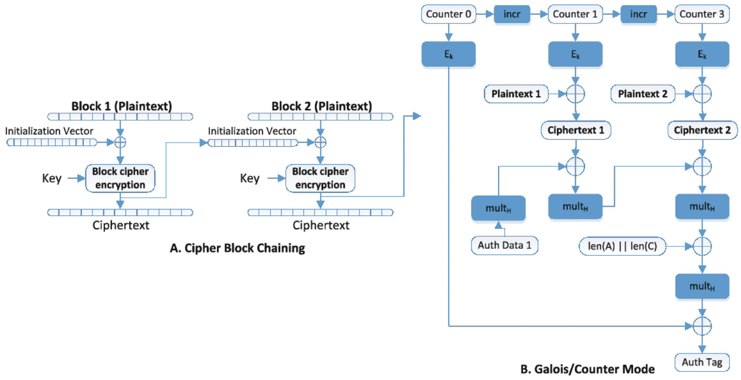


Fig. 4. CBC and GSM encryption.

– The Galois/Counter mode can be reconstructed even if some data are missing because cipher blocks are independent, see Fig. 4B.

Currently, our tool Netfox Detective supports analysis and decryption of various cipher suites, see Table 4.

Table 4. Cipher suites supported Netfox Detective.

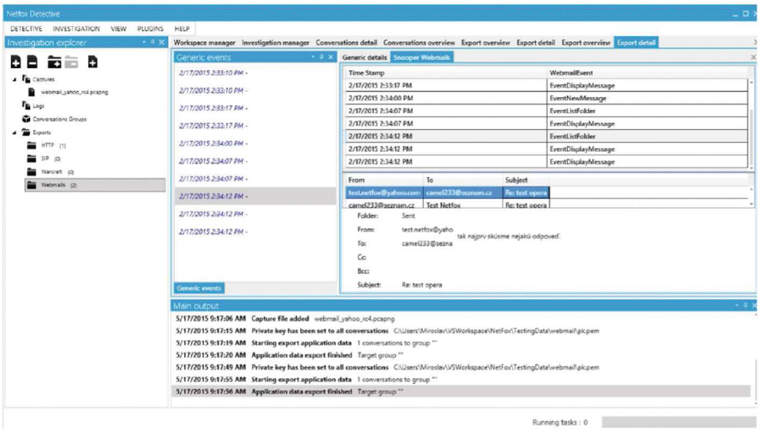
TlsRSAWithAes128CbsSha	TlsRSAWithAes256CbsSha
TlsRSAWithAes128CbsSha256	TlsRSAWithAes256CbsSha256
TlsRSAWithAes128GcmSha256	TlsRSAWithAes256GcmSha384
TlsRSAWithRc4128Md5	TlsRSAWithRc4128Sha

If a server key is available, this communication can be decrypted as presented in Fig. 5. This picture shows a successful decryption of web mail communication encrypted using TLS.

4.3 Bitcoin Detection

Bitcoins as currency (BTC) are getting more and more popular since 2008, especially because of their anonymity. Bitcoin network is secure by design against correlating transactions with individual users. However, forensic tools can at least detect bitcoin traffic within a network.

Bitcoin operates over peer-to-peer (P2P) network consisting of two node kinds: (i) clients, which send, receive, or relay BTC transactions; and (ii) miners, which verify transactions using a special proof-of-work algorithm.



2/17/2015 2:33:10 PM -
2/17/2015 2:33:10 PM -
2/17/2015 2:33:17 PM -
2/17/2015 2:33:17 PM -
2/17/2015 2:34:00 PM -
2/17/2015 2:34:07 PM -
2/17/2015 2:34:07 PM -
2/17/2015 2:34:12 PM -
2/17/2015 2:34:12 PM -
2/17/2015 2:34:12 PM -

Time Stamp	WebmailEvent
2/17/2015 2:33:17 PM	EventDisplayMessage
2/17/2015 2:34:00 PM	EventNewMessage
2/17/2015 2:34:07 PM	EventListFolder
2/17/2015 2:34:07 PM	EventDisplayMessage
2/17/2015 2:34:12 PM	EventListFolder
2/17/2015 2:34:12 PM	EventDisplayMessage
2/17/2015 2:34:12 PM	EventDisplayMessage

From	To	Subject
test.netfox@yahoo.com	camel233@seznam.cz	Re: test opera
camel233@seznam.cz	Test Netfox	Re: test opera

Folder: Sent
From: test.netfox@yaho
To: camel233@sezna
Cc:
Bcc:
Subject: Re: test opera

tak najprv skúsme nejakú odpoveď.

```
5/17/2015 9:17:06 AM  Capture file added  webmail_yahoo_rc4.pcapng
5/17/2015 9:17:15 AM  Private key has been set to all conversations  C:\Users\Miroslav\VSWorkspace\NetFox\TestingData\webmail\pk.pem
5/17/2015 9:17:19 AM  Starting export application data  1 conversations to group ""
5/17/2015 9:17:20 AM  Application data export finished  Target group ""
5/17/2015 9:17:49 AM  Private key has been set to all conversations  C:\Users\Miroslav\VSWorkspace\NetFox\TestingData\webmail\pk.pem
5/17/2015 9:17:55 AM  Starting export application data  1 conversations to group ""
5/17/2015 9:17:56 AM  Application data export finished  Target group ""
```

Fig. 5. Reconstruction of encrypted web mail data.

BTC uses three different protocols for its functionality where each protocol has a different value for the forensic investigation. These protocols are as follows:

1. Bitcoin v.1 protocol[2] is employed for P2P communication between peers (connected nodes). For forensic analysis, its detection can help to identify the end stations running Bitcoin client software. The protocol runs over TCP, port 8333. It transmits messages required for both a node discovery and Bitcoin transactions.

   Node discovery is provided twice in Bitcoin network:

– Upon software start-up, a client looks for special domain names (e.g., bitcoin.sipa.be, dnsseed.bluematt.me) in DNS in order to discover initial set of peers to get connected. Usually, the client uses a list of pre-configured stable nodes of the Bitcoin network.
– Upon successful connection to a node, the client may request a list of neighboring peers to expand its connectivity graph.

The protocol messages that helps us to detect a communication within Bitcoin P2P network area as follows: `version` and `verack` (useful for connection initiation), `address` (to detect a communication graph and provide information of known nodes), and `ping-pong` (a keep-alive mechanism). For forensic purposes, also messages `inv`, `tx`, and `block` are important since they transmit valuable information about processed transactions. The list of all Bitcoin v.1 messages is shown in Table 5.

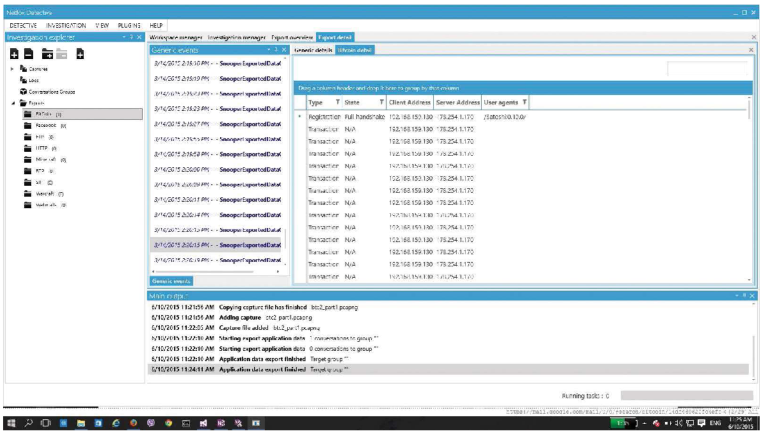
[2] See https://bitcoint.org/en/developer-documenation, June, 2015.

Table 5. Bitcoin v.1 protocol.

Messages	Description	Message	Description
version, verack	Opening messages	*tx, notfound*	Responses to *getdata*
getaddr, addr	List of known peers	*ping, pong*	Keepalive messages
inv	A new object announcement	*alert*	Broadcast notification
getdata	Request for object value	*mempool*	Retrieving a transaction
getblocks, blocks	Retrieval of a block	*filterload/add*	Bloom filter operations
getheaders, headers	Retrieval of a header	*reject*	Negative response

2. Another group of protocols (e.g., Getwork, Getworktemplate, Stratum) is used for work distribution for miners cooperating in the pool. The detection of these protocols implies an existence of bitcoin miner in the local network.
3. The last protocol group involves remote procedure call (RPC) messages that are employed for remote control of various Bitcoin related services (e.g., remote wallets controlled by a smart phone, on-line trading on Bitcoin exchanges, etc.).

Netfox Detective currently supports decoding of Bitcoin v1 protocol that helps to detect devices that run Bitcoin clients, work as Bitcoin miners, or access Bitcoin related services, see Fig. 6.



```
3/14/2015 2:19:10 PM - - SnooperExportedData(
3/14/2015 2:19:19 PM     SnooperExportedData(
3/14/2015 2:19:23 PM - - SnooperExportedData(
3/14/2015 2:19:23 PM - - SnooperExportedData(
3/14/2015 2:19:27 PM     SnooperExportedData(
3/14/2015 2:19:55 PM - - SnooperExportedData(
3/14/2015 2:19:58 PM - - SnooperExportedData(
3/14/2015 2:20:00 PM     SnooperExportedData(
3/14/2015 2:20:09 PM - - SnooperExportedData(
3/14/2015 2:20:11 PM - - SnooperExportedData(
3/14/2015 2:20:14 PM     SnooperExportedData(
3/14/2015 2:20:15 PM - - SnooperExportedData(
3/14/2015 2:20:15 PM - - SnooperExportedData(
3/14/2015 2:20:19 PM - - SnooperExportedData(
```

Type	State	Client Address	Server Address	User agents
Registration	Full handshake	192.168.159.130	178.254.1.170	/Satoshi:0.10.0/
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	
Transaction	N/A	192.168.159.130	178.254.1.170	

```
6/10/2015 11:21:56 AM  Copying capture file has finished  btc2_part1.pcapng
6/10/2015 11:21:56 AM  Adding capture  btc2_part1.pcapng
6/10/2015 11:22:05 AM  Capture file added  btc2_part1.pcapng
6/10/2015 11:22:10 AM  Starting export application data  1 conversations to group ""
6/10/2015 11:22:10 AM  Starting export application data  0 conversations to group ""
6/10/2015 11:22:10 AM  Application data export finished  Target group ""
6/10/2015 11:24:11 AM  Application data export finished  Target group ""
```

Fig. 6. Bitcoin analysis using Netfox Detective.

Based on these information, it is possible to create Bitcoin communication graphs and correlate the pool member and mining rig owner.

Captured network data can be used to provide an evidence that the seized server really conducted Bitcoin transactions, see Fig. 7.

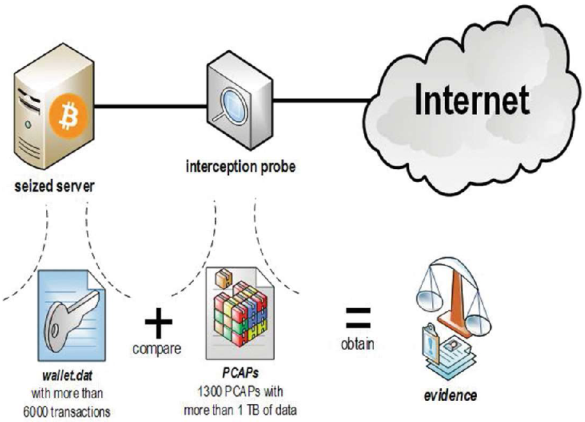


Fig. 7. Digital investigation of Bitcoin transactions.

5 Conclusion

Network forensics represent several challenges for security analysts. Network data are volatile what causes that communication traces are not captured completely. In addition, plenty of protocols are utilized in the current network communication. Many network applications also employ application-level protocol HTTP only as a data channel offering end-to-end connection. With the increased amount of traffic being encrypted, it is even complicated to recognize classes of applications in the captured communication.

In this paper, an overview of issues related to a recovery of the application content from captured traffic was presented. For identified problems, proposed methods were tested by implementing them in a novel network forensic tool. Based on the comparison to related tools, achieved results are promising for the further development of our NFAT tool.

Future work is delineated by the stated facts. Because of widely used traffic encryption, NFAT tools have to analyze meta-information associated with the traffic, e.g., recognizing events from communication, identifying end users, or approximate the meaning of information hidden in the encrypted communication. Also, the amount of communication requires NFATs to handle big data from various sources. Finally, NFATs should be extensible to deal with various classes of applications, e.g., web mail or Bitcoin traffic.

Acknowledgment. Research in this paper was supported by project "Modern Tools for Detection and Mitigation of Cyber Criminality on the New Generation Internet", no. VG20102015022 granted by Ministry of the Interior of the Czech Republic and an internal University project "Research and application of advanced methods in ICT", no. FIT-S-14-2299 granted by Brno University of Technology.

References

1. Cohen, M.I.: PyFlag - an advanced network forensic framework. Digit. Investig. **5**, 112–120 (2008)
2. Pilli, E.S., Joshi, R.C., Niyogi, R.: Network forensic frameworks: survey and research challenges. Digit. Investig. **7**, 14–27 (2010)
3. Hunt, R., Zeadally, S.: Network forensics: an analysis of techniques, tools, and trends. Computer **45**, 36–43 (2012)
4. Dharmapurikar, S., Paxson, V.: Robust TCP stream reassembly in the presence of adversaries. In: USENIX Security Symposium. (2005)
5. Postel, J.: Internet Protocol. RFC 791 (1981)
6. Postel, J.: Transmission Control Protocol. RFC 793 (1981)
7. Stevens, W., Fenner, B., Rudoff, A.M.: UNIX Network Programming: The Sockets Networking API, 3rd edn. Addison-Wesley, Reading (2004)
8. Matousek, P., Rysavy, O., Kmet, M.: Fast RTP detection and codecs classification in internet traffic. J. Digit. Forensics Secur. Law **2014**, 99–110 (2014)
9. Hjelmvik, E., John, W.: Statistical protocol identification with SPID: preliminary results. In: Swedish National Computer Networking Workshop (2009)
10. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Barners-Lee, T.: Hypertext Transfer Protocol - HTTP/1.1. IETF RFC 2616 (1999)
11. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. IETF RFC 5246 (2008)
12. McGrew, D.: An Interface and Algorithms for Authenticated Encryption. IETF RFC 5116 (2008)
13. Davidoff, S., Ham, J.: Network Forensics: Tracking Hackers through Cyberspace, 1st edn. Prentice Hall, Upper Saddle River (2012)

A.10 Netfox Detective: A Tool for Advanced Network Forensics Analysis

Jan Pluskal, Petr Matoušek, Ondřej Ryšavý, Martin Kmeť, Vladimir Veselý, Filip Karpíšek, and Martin Vymlátil. “Netfox Detective: A tool for advanced network forensics analysis”. In: *Proceedings of Security and Protection of Information (SPI) 2015.* Brno, CZ: Brno University of Defence, 2015, pp. 147–163. ISBN: 9788072319978

Netfox Detective: A Tool for Advanced Network Forensics Analysis

J. Pluskal, P. Matoušek, O. Ryšavý, M. Kmeť, V. Veselý, F. Karpíšek, M. Vymlátil

{ipluskal,matousp,rysavy,ikmet,ivesely}@fit.vutbr.cz,
{xkarpi03,xvymla01}@stud.fit.vutbr.cz

Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic

Abstract

Network forensics is a process of capturing, collecting and analysing network data for the purposes of information gathering, legal evidence, or intrusion detection. The new generation internet opens novel opportunities for cybercrime activities and security incidents using network applications. Security administrators and LEA (Law Enforcement Agency) officers are challenged to employ advanced tools and techniques in order to detect unlawful or unauthorized activities. In case of serious suspicion of crime activity, network forensics tools and techniques are used to find out legal evidences in a captured network communication that prove or disprove suspect's participation on that activity.

Today, there are various commercial or free tools for network forensics analysis available, e.g., Wireshark, Network Miner, NetWitness, Xplico, NetIntercept, or PacketScan. Many of these tools lack the ability of successful reconstruction of communication when using incomplete, duplicated or corrupted input data. Investigators also require an advanced automatic processing of application data that helps them to see real contents of conversation that include chats, VoIP talks, file transmission, email exchange etc.

Our research is focused on design and implementation of a modular framework for network forensics with advanced possibilities of application reconstruction. The proposed architecture consists of (i) input packet processing, (ii) an advanced reconstruction of L7 conversations, and (iii) application-based analysis and presentation of L7 conversations. Our approach employs various advanced reconstruction techniques and heuristics that enable to work even with corrupted or incomplete data, e.g. one-directional flows, missing synchronization, unbounded conversations, etc.

The proposed framework was implemented in a tool Netfox Detective developed by our research group. This paper shows its architecture from functional and logical point of view and its application on reconstruction of web mail traffic, VoIP and RTP transmissions.

Keywords: network forensics, forensic tools, network traffic analysis, Web mail, SIP, RTP

1 Introduction

Network forensics is a discipline that deals with obtaining and analysing digital evidences from network sources. It is an extended phase of network security where the main goal of network forensics is to track and analyse network data in order to detect security incidents and present evidences of these incidents to security administrators or investigators. Network forensics use different supporting tools and devices that (i) obtain and collect data (firewalls, IDS systems, capturing tools), and (ii) process, analyse and reconstruct captured data. Network forensic tools are mostly used by security administrators and LEA officers that try to search network data for legal evidences of unlawful behaviour. The aim of the analysis is to establish high level facts such as attribution, intent, identity, timelines and other information which may be relevant to the security incident.

Tools for network forensics can be classified into two main groups: Network Forensic Analysis Tools (NAFTs) that allow administrators to monitor network, gather all information about the traffic and assist in network crime investigation, and Network Security and Monitoring (NSM) tools that are focused more on network monitoring and management. There is a wide range of commercial and open-source NFATs and NSM tools [1]. The primary motivation behind NSM tools is network security from perspective of system administration. NSM tools are very useful in processing large amount of data in short time with limited functionality concerning application protocol dissection. NMS tools include (i) IDS/IPS systems for detection or prevention of malicious activity on network, (ii) statistical tools used for data retention to store meta-information about the traffic, (iii) packet capture and analyses tools that capture communication on local networks and analysing it. The most common NSMs focused on packet capturing and analyses are Wireshark, TCPdump, or Microsoft Network Monitor. These tools are also used for basic network forensic analysis. However, they are mostly oriented on simple analysis of internet and transport layers of TCP/IP model. Some of them even contain an application layer protocol dissector, but the provided information is a context-free parsed internal protocol structure.

In this work, we focus on NFATs. NFATs offer a wide range of research challenges in domain of analysis and reconstruction of captured traffic. Research challenges cover (i) network stream reassembling that include detection of TCP/UDP streams, dealing with out of sequence data, missing or corrupted packets, timestamps overflow, combing streams into bi-directional conversations etc. [2]; (ii) advanced identification of L7 applications using AI techniques, data mining or statistical methods [3]; (iii) processing and analysis of L7 application using application dissectors, (iv) identification and statistical processing of encrypted or tunnelled traffic, (v) efficient storage of big network data with parallel computation, (vi) correlation of different input data, etc.

This paper describes architecture and implementation of a network forensic tool Netfox Detective developed by our team in frame of security research supported by Ministry of Interior of the Czech Republic. The tool is designed for advanced reconstruction and analysis of captured network data with focus on emails (including web mails), HTTP reconstruction and intelligent detection and reconstruction of Voice over IP. Our framework combines advanced techniques and heuristics for assembling captured data, identification of L7 traffic, reconstruction of original conversations, and presentation of L7 objects to an investigator. The proposed framework uses modular programming environment with well-defined API so new modules (application dissectors, processing engines) can be added without a need to re-build the entire application. It also supports parallel processing with efficient data storage.

2 Related Work

Network forensics was formally defined in 2001 on the First Digital Forensic Research Workshop [4] where also major issues were identified: (i) time, i.e., synchronization and integrity of data and time associated with events being analysed; (ii) performance, i.e., speed and effectiveness of processing and computation; (iii) complexity, i.e., general environment with multiple operating systems, network devices, different data formats, and (iv) collection, i.e., who will collect data, when, and what to be collected?

After a decade of innovations and research, general process model for the network forensic analysis has been introduced [1]. General model was composed of blocks with separated functions and was divided into two layers: (i) *lower layer* that included preparation, detection, collection, and preservation; and (ii) *upper layer* containing examination, analysis, investigation, and presentation.

Overview of different frameworks based on distributed systems, soft computing, honeypots, graphs, formal methods, or aggregation can be found in [1]. In that paper, Pilli et al. present a survey of current network forensic frameworks. Most of discussed frameworks were designed to as research tools to prove advanced approaches and techniques in the area of network forensics. Our tool presented in this paper employs some of these ideas but its development is driven by practical usability and deployment.

On the field of free tools, there are several applications that were observed. NetWitness filters captured traffic by processing frames and creating a lexicon of identifiers found in different L3-L7 layers, e.g., IP addresses, email addresses, URIs, etc. An investigator searches this lexicon to filter interesting captured content. The result can be stored as filtered captured traffic or analysed by another NFAT. Another popular tool is NetworkMiner[1] developed by Erik Hjelmvik, an author of Statistical Protocol Identifier (SPID) algorithm for application protocol detection [3]. NetworkMiner processes captured or online communication with an instantaneous analysis of application protocol. The analysed content is grouped into categories based on its characteristics, e.g. images, messages, credentials, files, frames, hosts, sessions. The tool lacks detailed views of captured data and is not able to backtrack objects to its original representation in captured packets. Xplico[2] is an open source NFAT platform composed of functional blocks. Application data are prepared by traffic decoder and then processed by manipulators. Xplico supports various application protocols, e.g., HTTP, SIP, IMAP/POP3/SMTP, FTP, etc. with ability to provide congruent investigation for multiple investigators at once. The tool provides a user interface via a web browser which is simple to use, but it is not suitable for advance analysis, e.g., advance filtering, getting data integrity statistics, etc. Nevertheless, Xplico is the most advanced open source NFAT available.

3 Netfox Detective Architecture

By testing available NFATs we discovered that none of these tools is sufficient to accurately extract incomplete network data. In addition, advanced processing of application protocols with user-friendly presentation was mostly missing and limited large deployment of these tools for investigators. To overcome these limitations, a new network forensics framework was proposed with advanced parsing features.

[1] See http://www.netresec.com/?page=NetworkMiner.

[2] See http://www.xplico.org.

Netfox (NETwork FOrensiCS) Detective is a NFAT framework operating upon four upper layers of generic process model of NFATs as described in [1]. The tool processes input network data stored in different PCAP formats[3] using a generic algorithm that respects L2-L7 encapsulation of PDUs. As described in [2], advanced heuristics is employed to extract maximal amount of information from PDU headers.

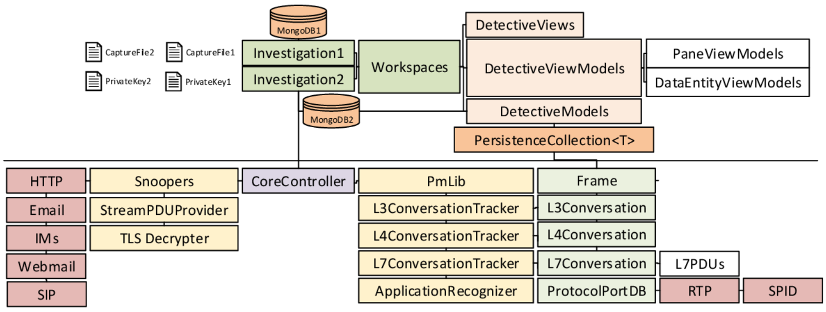


Figure 1: Functional architecture and data model of Netfox Detective.

Netfox Detective has been designed to be used on Windows 7+ platform. To ensure proper behaviour and modular architecture as shown in Fig.1, the Model-View-Viewmodel (MVVM)[4] design pattern has been chosen with asynchronous programing provided by .NET 4.5.2 and C# 6. When launching the tool, a new workspace is created or a recently used workspace is re-loaded. The workspace represents a directory structure in a file system where all data related to the workspace are stored. The workspace contains one or more investigations that can consists of one or more PCAP files, see Fig. 2. Data processing is controlled by *Core Controller* that communicates with *PmLib* module, *Conversation trackers* and *Snoopers*. *Application Recognizers* use different techniques to identify L7 applications, see below. Application analysis and presentation of the results is implemented using L7 *Snoopers* over HTTP, Emails, IMs, Web mails, or SIP.

[3] E.g., see LibPCAP and PCAP NG at https://wiki.wireshark.org/Development/LibpcapFileFormat (PcapNg), or MS Network Monitor PCAP at http://blogs.technet.com/b/netmon/p/downloads.aspx.
[4] See https://msdn.microsoft.com/en-us/library/hh848246.aspx.

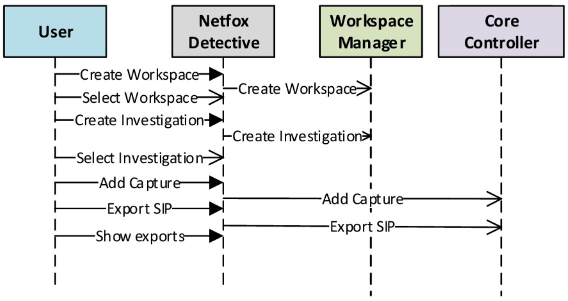


Figure 2: Logical interactions of presentation model.

NFATs usually require flexible design with extensibility that allows addition of new features and propagation of these updates throughout the modular architecture without changing internal data structures. This can be implemented using document-oriented database that processes dynamic semi-structured data types in contrast to pre-defined types in relational databases where relations between data are fixed and must be defined in advance. Netfox Detective framework employs document-oriented database system MongoDB[5]. This approach ensures persistence across the entire framework using only one implementation for each data model. Basic data models *Workspace*, *Investigation*, *Capture,* and *InvestigationInfo* are listed in Fig. 3.

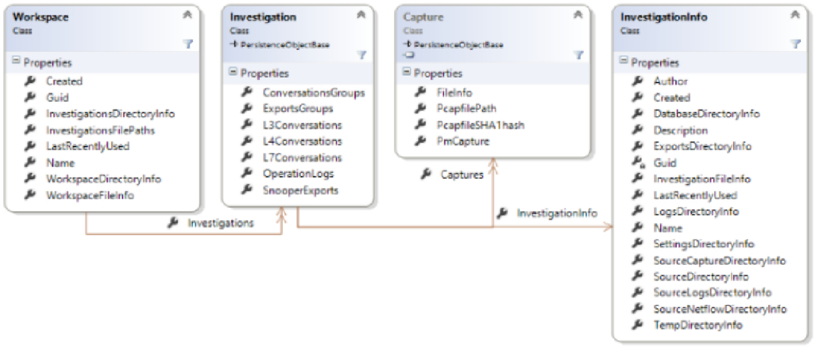


Figure 3: Database models used in Netfox Detective to ensure persistence of workspaces and investigations.

[5] See http://www.mongodb.org/about/.

Captured network data are processed using a pipeline that extracts crucial information for further analysis, see Fig. 4. At first, a PCAP file is added to an investigation and parsed in *PmLib* module that builds a frame collection. Each module *LxConversationTracker* asynchronously processes every new frame and creates an appropriate *PersistenceCollection* for *X-th* level conversation, e.g., for L3, L4, or L7 layer. The *L7ConversationTracker* builds application layer conversations over TCP or UDP sessions, and creates application protocol messages called *L7PDUs* without any syntactical knowledge of the particular application protocol. Conversation tracking and reconstruction uses port numbers and TCP sequence numbers to detect missing data or unclosed sessions. It also employs timestamps to increase accuracy of reconstruction. Detailed description of packet reassembling is described in [2].

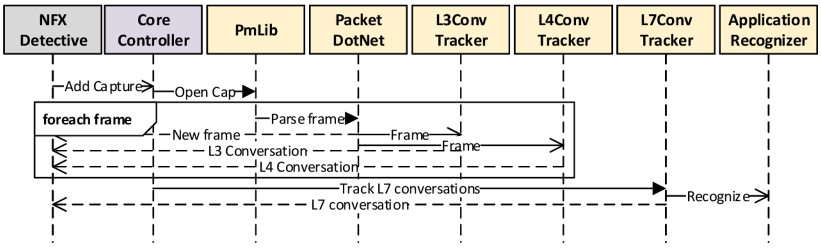


Figure 4: Asynchronous capture file processing pipeline.

The key issue for successful L7 analysis provided by application extractors (*snoopers*) is a correct identification of *L7Conversations*. Identification is provided by the *Recognizer* that assigns one or more application tags to a L*7Conversation*. The algorithm uses extended IANA database of well-known ports, RTP recognizer for dynamic RTP streams [5], or SPID algorithm [3] using statistical based identification.

Snooper modules are dynamically loaded to Netfox Detective, therefore, no recompilation is needed when a new application parser (*snooper*) is added. The *snooper* is a reconstruction engine of the application protocol. Outputs of one *snooper* can be chained into another *snooper* for further reconstruction, e.g., outputs of HTTP analysis can become inputs of web mail *snooper*. *Snoopers* export the contents of conversations with corresponding meta-data obtained during the application protocol processing into a current investigation.

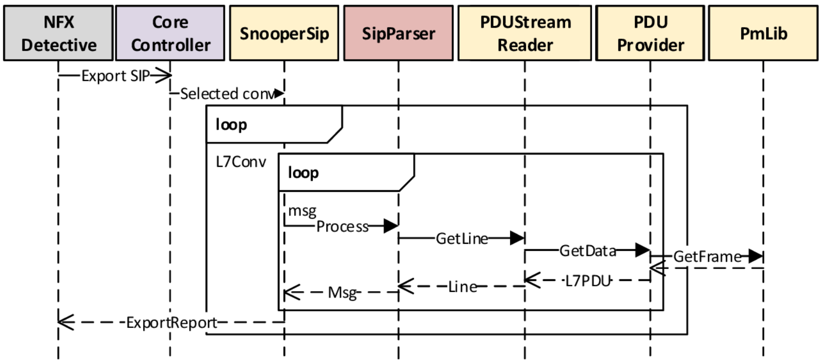


Figure 5: SIP application protocol data extraction using the SIP Snooper module.

Each *snooper* defines its own models, views and view-models to provide a detailed presentation of reconstructed data, e.g, HTTP snooper shows reconstructed web pages, an email snooper lists reconstructed emails, VoIP snooper describes VoIP session with RTP streams to be replayed, etc. Example of SIP snooper processing is at Fig. 5.

As mentioned above, s*noopers* provide a syntactical analysis of communication. Until this point, data processing has been based purely upon information obtained from layers L3 and L4. The *snooper* analyses a particular application protocol, i.e., it parses application messages. The *snooper* communicates with low level modules as *PDUStreamReader*, or *PDUProvider* that deal with missing or overlapping segments, TCP sequence number overflow, missing SYN and FIN packets, IP defragmentation, etc. The *snooper* processes logical *L7PDU*s as soon as all conversations have been successfully restored over L4. It receives data from the *PDUStreamReader* module. *PDUProvider* prepares input data for *PDUStreamReader* using one of four strategies shown on following example, see also Fig. 6:

1. *Broken Interlay* – The first application message consists at maximum of PDU1 and PDU2 transmitted in Frame 1, 2, 3. The arrival of Frame 4 on client side signals that the application message has ended. This is typical for request/response protocols. The second application message is contained only in PDU3 and the third in PDU4.
2. *Continued Interlay* – The first application message consists at maximum of PDU1, PDU2, and PDU4 without taking into account frames arriving in opposite direction.

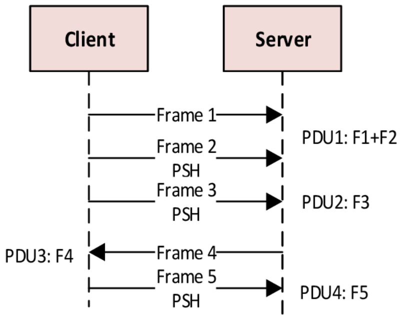


Figure 6: Processing PDUs.

3. *Mixed Interlay* – The first application message might consist of PDU1, PDU2, PDU3, and PDU4. This mode mixes PDUs from both directions into one bi-directional stream.
4. *Single Message Interlay* – Every application message consists only of one single PDU.

Generally, one application message can be composed of one or more PDUs. When some frames are missing, a virtual frame is created in order to complete proper PDU processing by a snooper. Using this approach, succeeding un-corrupted message will be properly reassembled in contrast to MS Monitor that might misinterpret succeeding messages.

4 Extracting Application Data

Application protocol data extraction is a process of analysing application layer data streams, i.e., payloads of L7 conversations. This analysis requires knowledge of application protocol syntax as well as semantics to extract significant information for forensic analysis. Following examples of application processing demonstrate how L7 parsing is implemented in NetFox Detective. They also describe advanced techniques for reconstruction of incomplete or corrupted application data.

4.1 Web mail

Communication using emails is necessity for everyone today. A majority of users uses web browser to access their mail boxes and to operate with their mail accounts. Therefore, HTTP protocol is mostly used to tunnel web mail communication. Traditional email protocols like POP3, IMAP, and SMTP have been mostly put aside from the end user perspective, even though they are still used among email providers.

In this study, we have focused on web mail traffic analysis in order to create a general model that would be able to process web mail independently on particular service used. As it is seen in Table 1, following operations similar for all analysed web mail services can be identified despite the fact that general structure of web mail is not standardised and web mail providers implement various transmission methods how to deliver web mail contents, e.g., using RPC sessions, JSON applications, etc. Table 1 shows how basic web mail operations can be identified in URL or HTTP header payload using simple pattern matching.

Operation	**Web mail patterns used in URL or HTTP header**
New Message	Keywords: from, to, subject, cc, bcc, content/body, SendMessage.
Message manipulation	URL request/HTTP header: move, delete, MoveMessageToFolder.
Email header request	URL request/HTTP header: list, search, GetInboxData.

Table 1: Common operations and methods of their detection.

Web mail services can be divided based on data privacy protection into three groups: (i) web mail service with unencrypted authentication and mail transmission, e.g. *zoznam.sk*, *tiscali.cz* (ii) web mail services with encrypted authentication and unencrypted mail transmission, e.g., *centrum.cz*, *atlas.cz* and *mujmail.cz* (iii) web mail services with encrypted authentication and encrypted mail transmission, e.g., *seznam.cz*, *gmail.com*, *email.cz*.

When web mail authentication is encrypted, web mail communication cannot be identified using standard URL analysis but other techniques can be employed. One possibility is to use client's header extension in SSL/TLS handshake where Hello message contains the server name. The server name might indicate that following SSL/TLS communication transmits web mail. Also, DNS resolution can be employed to detect web mail service, see Table 2.

Web mail	**Server name**	**Encoding**
seznam.cz, email.cz	email.seznam.cz	FastRPC

Gmail	mail-attachment.googleusercontent.com	application/x-www-form-urlencoded;charset=utf-8
Yahoo	mail.yahoo.cz	application/json multipart/form-data – incl JSON
MS Live		application/x-www-form-urlencoded
Centrum /Atlas/Mujmail	mail.centrum.cz	application/x-www-form-urlencoded
Roundcube	<private service hostname>	application/x-www-form-urlencoded
Horde	<private service hostname>	multipart/form-data;

Table 2: Identification of particular web mail service.

4.2 Voice over IP

Voice over IP (VoIP) is a technology for transmission of phone calls over IP infrastructure Main advantage of VoIP is that uses the same infrastructure for both data and voice transfers which save money but also reduce maintenance requirements. From point of view of network forensics, VoIP creates a new challenge for detection and interception of suspect's calls. Traditional call interception on telecommunication networks was subjected to strict and well-known rules. VoIP works in flexible environment of IP networks with a large variety of application protocols and codecs. The most common VoIP technologies are SIP [6] for call signalling and RTP [7] for media transmission. Following section describes how SIP and RTP protocols can analysed.

4.2.1 Signalling protocols

Session Initiation Protocol (SIP) is an application layer protocol for signalling and controlling multimedia sessions over IP networks. It is mostly used for voice/video calls and instant messaging. It defines messages that establish, modify and terminate sessions between end points. SIP is a text-based protocol with some similarities to HTTP or SMTP. It serves mainly for user registration and establishing VoIP connection. Media streams (voice or video) are transmitted using RTP protocol [7] or its secured version SRTP [8]. Description of transmitted media stream is encoded using Session Description Protocol, SDP [9].

SIP communication is independent on transport protocols and may use TCP, UDP or SCTP transport. The protocol utilizes a transaction based communication. Each transaction is represented by a request and at least one response. SIP protocol usually communicates on TCP/UDP ports 5060 or 5061 (encrypted sessions).

4.2.2 SIP analysis

The extraction algorithm iterates over *L7 conversations* identified by an application recognizer. Whenever a valid SIP message is obtained, it is processed by SIP snooper that extracts meta-data related to the call. SIP messages with the same Call-ID form a SIP event. Generally, SIP snooper uses two basic methods *INVITE* for call establishment and *REGISTER* for authentication. However, this trivial processing is not sufficient when some messages are corrupted or missing.

1	INVITE sip:10.10.10.109 SIP/2.0
2	Call-ID: D99151DA-1DD1-11B2-B23A-BC0375BD6E00@10.10.10.214
3	From: "unknown"<sip:10.10.10.214>;tag=30652209562016038532
4	To: <sip:10.10.10.109>
5	… c=IN IP4 10.10.10.214 … m=audio 49152 RTP/AVP 3 97 98 110 8 0 101 …

Table 3: Example of data transmitted in a SIP message.

Table 3 shows what kind of information can be obtained from SIP protocol:

1. *Request method or response code – this can be used to recognize a call.*
2. *Call-ID – a unique identifier used for grouping corresponding messages.*
3. *From header – identifies caller party.*
4. *To header – identifies calling party.*
5. *SDP body – identifies media stream, codecs, RTP ports, etc.*

For network forensic purposes, several SIP message are interesting to get meta-data about call exchange, e.g, *INVITE*, *BYE,* and *REFER* as requests and *100* (Trying) and *180* (Ringing) as responses. Using these requests and responses, we are able to extract SIP calls even if captured signalling is incomplete. As depicted on example in Fig. 7, even if *INVITE* message is lost, the same information can be obtained from related messages (marked by red dot).

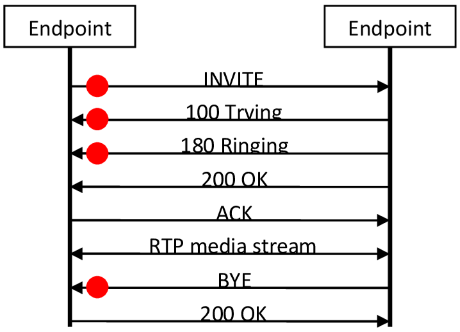


Figure 7: Typical message exchange during a SIP call.

Another issue is pairing incomplete signalling data with media streams. Network Detective implements heuristic based on RTP and TCP timestamps that result in probabilistic correlation of reconstructed VoIP calls. Utilizing these strategies, we are able to provide better reconstruction in comparison with other tools, see Table 4.

file	**NFX Detective**	**Wireshark**	**NetWitness**	**PacketScan**
Complete PCAP	2	2	2	2
PCAP without INVITE	2	0	2	0
PCAP without 200 OK	2	2	2	2

Table 4: Detection of VoIP calls when INVITE or 200 OK messages are missing.

4.2.3 Real-time Transport Protocol (RTP)

RTP [7] is a stateless application protocol used to transfer media streams over the network. The RTP also provides simple detection of lost packets and multiple streams synchronisation with minimal overhead. It is usually transferred over UDP due to minimal overhead and stateless behaviour. RTP does not retransmit lost packets because even if they had eventually arrived, they would have not been needed any longer. RTP detection is not easy due to the dynamic port assignment. As a part of RTP standard is description of RTP Control Protocol (RTCP) messages that are used to deliver additional control session data, e.g., stream source description, sent data size counter, packet loose, jitter, etc.

4.2.4 Detecting RTP without signalling protocols

Common VoIP concepts separate signalling data (SIP/SDP) from media streams (RTP). Both protocols use their own PDUs and paths through the internet. When signalling data are missing, it is generally not easy to detect RTP stream with dynamic UDP ports and identify what kind of codec is used for voice or video data transmitted. Netfox Detective uses advanced detection algorithm to identify RTP as follows. For full algorithm, see [5]:

1. *RTP header contains a fixed version 2.*
2. *Mostly all current VoIP applications use only UDP transport protocol with ports greater than 1024.*
3. *Observed packets should have a minimal packet length as required by the standard unless extension flag is set.*
4. *Typical RTP stream is collection of large number of small packets with the same SSRC identifier.*

RTP header contains *Payload type* (PT) for codec identification. This field is mostly used for statically mapped codecs like G.711, GSM, G.722, or G.729, see [10]. Dynamically assigned codecs like Speex, G.726, AMR, or Silk require identification information transmitted in signalling protocols. If signalling protocols are not present in a captured file, it is hard to identify the codec. In such case, it is possible to use an identification method based on ratio between payload size of RTP packets and timestamp differences between two successive packets. Since this ratio usually does not change, this method is sufficient for codecs identification without signalling data [5].

4.2.5 Incomplete RTP streams

In case of incomplete or corrupted RTP packets, advanced reconstruction techniques have to be applied. Following case studies present some solutions how to reconstruct such data.

The first case study (see Fig. 8) shows communication between Alice and Bob where a link towards Bob is lossy. In this case, Bob's phone will miss two RTP packets 2 and 4. When naïve approach to decode a received audio stream is applied, audio tracks would not be synchronized, see Fig. 9. This will complicate further reconstruction and forensics analysis.

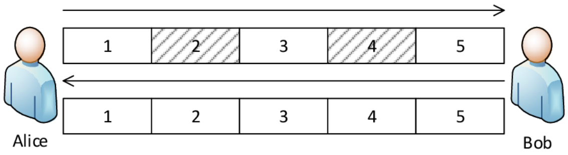


Figure 8: Incomplete RTP streams.

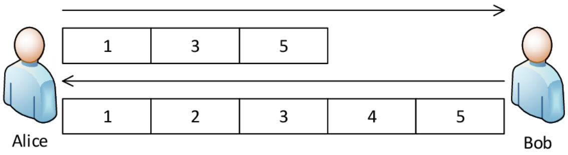


Figure 9: Naïve RTP Reconstruction.

For advanced RTP reconstruction, a following procedure is proposed:

1. *Compute the number of lost samples.*

   Using RTP timestamps a difference between the last received packet and the next one after the loss can be calculated. Then, correlation between real-time and timestamp difference indicates how many packets were lost. Although this correlation is codec dependent it can be used for reconstruction. For example, if the last received packet had timestamp 1000 and the next received packet had timestamp 9000, we may assume that 8000 audio samples were lost.

2. *Reconstruction of missing samples.*

   The knowledge of a codec used is important to encode raw audio data since the codec specifies the sampling rate that has been used. For example, codec G.711 uses sample rate 8000 Hz. In case of 8000 lost samples with sampling rate 8000 Hz one second audio is missing. Therefore, lost packet can be substitute with silence audio or white noise right after decoding to fill the specified gap and synchronize bi-directional audio steams.

Example of RTP streams after reconstruction is depicted in Fig. 10. As it is seen now, timeline of both RTP streams is properly aligned that is important for proper forensic analysis.

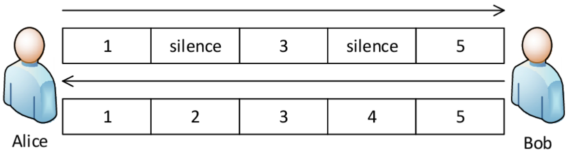


Figure 10: Reconstructed RTP streams.

5 Conclusions

This paper presented a new framework for network forensics analysis developed during security research. This framework has modular architecture with focus on two important areas: stream reassembling and application reconstruction. Stream reassembling is an important part of the tool. If not done properly, some packets can be skipped without proper analysis. On the other hand, some streams can be reconstructed incorrectly and include frames that do not belong to the reconstructed stream. The main benefit of our study is proposal of different heuristics and techniques that are able to build streams from captured packets even if some packets are missing without a need to parsing application protocol. Proposed heuristics are used to detect missing SYN and FIN packets, to identify lost packets within a stream, to detect overlapped conversations, etc., so that TCP and UDP streams are properly reconstructed for further network forensics analysis.

Following application reconstruction is provided by independent application snoopers that parse reconstructed L7 streams, extract application based meta-data, and visualize results to an investigator or security administrator. Application snoopers also implements advance techniques for proper reconstruction of incomplete application data as presented on web mails and VoIP communication. At the moment, Netfox Detective is able to work with any IP, TCP or UDP streams. It supports reconstruction of web pages, web mails, emails using SMTP, POP, or IMAP protocols, instant messaging protocols (XMPP, ICQ, Yahoo), and VoIP (SIP, RTP). The user interface allows an investigator to filter required conversations and expert interesting data for further analysis.

In this research, we concentrated more on accurate data reassembling, parsing and reconstruction. Future research will be focused on efficient analysis of big data, distributed parsing and employment of advanced detection methods using machine learning, statistical based detection, etc.

6 Acknowledgment

Presented research results were supported by project "Modern Tools for Detection and Mitigation of Cyber Criminality on the New Generation Internet", no. VG20102015022 granted by Ministry of the Interior of the Czech Republic and by project "Research and application of advanced methods in ICT", no. FIT-S-14-2299 granted by Brno University of Technology.

References

[1] S. E. Pilli, R. Joshi and R. Niyogi, "Network forensic frameworks: Survey and research challenges.," *Digital Investigation,* pp. 14-27, 2010.

[2] P. Matoušek, J. Pluskal, O. Ryšavý, V. Veselý, M. Kmeť, F. Karpíšek and M. Vymlátil, "Advanced Techniques for Reconstruction of Incompleted Network Data," in *International Conference on Digital Forensics & Cyber Crime*, Seoul, 2015.

[3] E. Hjelmvik and W. John, "Statistical protocol identification with SPID: preliminary results.," in *Sweedish National Computer Networking Workshop.*, 2009.

[4] G. Palmer, "A Road Map For Digitial Forensic Research," in *First Digital Forensic Research Workshop (DFRWS)*, Utica, New York, 2001.

[5] P. Matoušek, O. Ryšavý and M. Kmeť, "Fast RTP Detection and Codecs Classification in Internet Traffic.," *Journal of Digital Forensics, Security and Law,* pp. 99-110, 2014.

[6] H. Schulzrinne, J. Rosenberg, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley and E. Schooler, *SIP: Session Initiation Protocol,* IETF RFC 3261, 2002.

[7] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications,* IETF RFC 3550, 2003.

[8] M. Baugher, D. McGrew, M. Naslund, E. Carrara and K. Norrman, *The Secure Real-time Transport Protocol (SRTP),* IETF RFC 3711, 2004.

[9] M. Handley and V. P. C. Jacobson, *SDP: Session Description Protocol,* IETF RFC 4566, 2006.

[10] H. Schulzrinne and S. Casner, *RTP Profile for Audio and Video Conferences,* IETF RFC 3551, 2003.