VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO ANALÝZU OBSAHU SÍŤOVÉ KOMUNIKACE

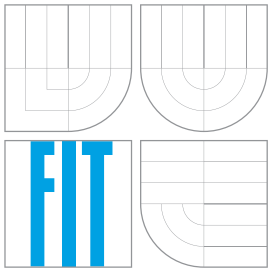
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN MAREŠ

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO ANALÝZU OBSAHU SÍŤOVÉ KOMUNIKACE

ANALYSIS TOOL FOR THE NETWORK TRAFFIC CONTENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN MAREŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR VESELÝ

BRNO 2014

Abstrakt

Tato práce se zabývá problematikou návrhu a implementace nástroje pro analýzu obsahu síťové komunikace v rámci projektu SEC6NET na FIT VUT v Brně. Jsou zde popsány požadavky na takovýto nástroj a problémy, které tyto požadavky vytvářejí. Způsob jejich řešení je diskutován v rámci návrhu vytvářeného nástroje. Jedná se zejména o zajištění persitence dat, výkonnost a celkovou architekturu nástroje. Poslední část práce popisuje nástroj Netfox Detective vytvořený na základě tohoto návrhu. Uvedeny jsou zejména možnosti a prostředky, které nástroj nabízí. Práce jako celek klade důraz na možnosti rozšiřitelnosti vytvářeného nástroje, který by měl být snadno doplnitelný jak o podporované protokoly, tak o novou funkcionalitu.

Abstract

This paper is about designing and implementing a tool for analysis of the network traffic content. Work is part of the SEC6NET project at FIT - Brno University of Technology. The paper describes specific requirement for such a tool and existing solutions. The main part is devoted to design and implementation of new tool - Netfox Detective. Several problems are handled: data persistence, performance, extensibility, etc. The last part of work describes possibilities and features of the Netfox Detective. The work emphasises extensibility and future development of a created tool.

Klíčová slova

Netfox, Detective, nástroj, analýza, síť, síťový, obsah, komunikace, rekonstrukce, reassembling, e-mail, web, IM, pcap, cap, plugin, dotaz, SEC6NET.

Keywords

Netfox, Detective, tool, analysis, network, content, communication, reconstruct, reassembling, e-mail, web, IM, pcap, cap, plugin, query, SEC6NET.

Citace

Martin Mareš: Nástroj pro analýzu obsahu síťové komunikace, diplomová práce, Brno, FIT VUT v Brně, 2014

Nástroj pro analýzu obsahu síťové komunikace

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana inženýra Vladímíra Veselého.

.....
Martin Mareš
21. května 2014

Poděkování

Rád bych poděkoval svému vedoucímu, Ing. Valdimíru Veselému, za vedení práce a poskytnuté věcné připomínky, které mi pomohly při jejím zpracování. Dále bych rád poděkoval Ing. Ondřeji Ryšavému, Ph.D. za rady a vedení v rámci projektu SEC6NET a Bc. Janu Pluskalovi za spolupráci na části týkající se Netfox frameworku a jeho kontroleru.

Vladimírovi věnuji také tento recept na táborovou svíčkovou, u které je zážitkem nejen konzumace ale zejména příprava. Potřebné suroviny : 10 kg hovězího masa, 0.5 kg slaniny, 0.3 kg soli, 1 kg tuku, 6 kg kořenové zeleniny, 1 kg cibule, 5 l smetany, 1.5 kg hladké mouky, 0.5 kg másla, 0.5 l octu, 0.5 kg cukru, pepř celý, nové koření, bobkový list, citron. Opláchnuté díly masa protkneme slaninou a osolíme. Očištěnou zeleninu a cibuli nakrájíme na plátky, mírně osmahneme na tuku, přidáme maso, krátce je v základu opečeme a podlijeme mírně vodou. K masu přidáme pepř a nové koření, bobkový list a tymián a pečlivě opláchnutý citron, nakrájený na plátky, zastříkneme octem, přikryjeme a maso dusíme (pečeme pod poklicí) do měkka. Během dušení (pečení) maso obracíme, přeléváme šťávou a doléváme vodu. Měkké maso vyjmeme, šťávu zahustíme světlou zásmažkou připravenou z másla a prosáté mouky, rozředíme vodou, rozšleháme a zvolna vaříme za občasného promíchání nejméně 20 minut. Během varu přidáme smetanu. Omáčku procedíme, dochutíme solí, octem a cukrem a znovu krátce povaříme. Doporučuji připravit si záložní mixér, první kus zpravidla odchází v nejkritičtější fázi přípravy.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá nástroji pro analýzu obsahu síťové komunikace. Cílem práce je vytvořit takovýto nástroj v rámci projektu SEC6NET (Moderní prostředky pro boj s kybernetickou kriminalitou na Internetu nové generace) ve spolupráci s Ministerstvem vnitra ČR[1].

Jedná se o nástroj, jehož oblast použití se liší oproti způsobu, kterým jsou obdobné nástroje běženě používany, například při vývoji nebo správě počítačových sítí. Jsou zde kladeny jisté specifické požadavky, které mohou být podnětnými při výzkumu v této oblasti. Může se jednat o podporované protokoly, způsob použití nebo kvalitu poskytovaných výsledků a jejich prezentaci. Cílem práce není implementovat jednoúčelový nástroj pro analýzu dané množiny protokolů předem určeným způsobem, ale získat platformu, která umožní vytvářet různé pohledy a různé způsoby práce s obsahem síťové komunikace.

Druhá kapitola práce popisuje vymezení nástroje na poli již existujících řešení - dostupné možnosti, používané postupy a koncepci vytvářeného nástroje. Další kapitola stručně popisuje Netfox Framework vyvíjený v rámci projektu SEC6NET, který nástroj používá jako funkční jádro. Čtvrtá kapitola se věnuje návrhu nástroje. Popisuje přípravu Netfox frameworku pro použití ve vytvářené aplikaci - návrh kontroleru, který umožní použít framework jako jednu komplexní knihovnu pro rekonstrukci aplikačních dat. Hlavní část kapitoly popisuje návrh vlastní aplikace pro analýzu obsahu síťové komunikace Netfox Detective. Zejména řeší celkovou architekturu, možnosti, které návrh přináší, a způsob zachování persitence dat v aplikaci s ohledem na jejich předpokládané množství a strukturu. Další kapitola stručně popisuje implemntaci nástroje - zejména použité technologie a postupy. Šestá kapitola je věnována popisu vlastního vytvořeného nástroje. Jsou zde prezentovány jednotlivé prostředky, které aplikace nabízí a možnosti práce s ní. Předposlední kapitola ukazuje řešení modelového případu z literatury pomocí vytvořeného nástroje a přednosti, které v tomto případě nabízí.

Kapitoly věnující se návrhu kontroleru a řešení persistence dat vycházejí z řešení stejnojmenného semestrálního projektu.

[1] http://www.fit.vutbr.cz/research/view_project.php?id=517

Kapitola 2

Nástroje pro analýzu obsahu síťové komunikace

Při návrhu a implementaci nového nástroje pro analýzu obsahu síťové komunikace je nejprve nutné popsat motivaci k jeho vytvoření a požadavky, které na něj budou kladeny, zejména v situaci, kdy již existuje celá řada různých typů nástrojů pro analýzu a práci s daty přenášenými pomocí počítačové sítě.

2.1 Přehled vybraných existujících nástrojů

V současné době je k dispozici mnoho kvalitních nástrojů, zaměřujících se na síťovou komunikaci z pohledu přenosových protokolů (vrstva transportní a nižší). Pohled na data v ní obsažená je zde orientován na jednotlivé PDU - rámce/pakety. Zaměřují se na protokoly síťového zásobníku, směrovací protokoly a podobně. Některé „rozumí“ také protokolům na aplikační vrstvě, ale přenášený obsah zobrazují na úrovni protokolu - příkazy, jednotlivé zprávy, servisní informace. Nástroje tohoto typu obecně nabízí vhodný pohled na data zejména pro správu sítě, vývoj (analýza protokolů po jednotlivých zprávách) nebo výzkum.[3]

Na druhé straně stojí nástroje zaměřené právě na obsah - data přenášená v aplikačních protokolech. Jedná se většinou o uzavřená řešení zaměřená na vybranou sadu protokolů. Jde především o automatizované prohledávání obsahu nebo extrakci přenášených souborů ze zachycené komunikace, jako jsou obrázky, dokumenty, zprávy nebo webové stránky. Zaměření na obsah implikuje použití zejména ve forenzních aplikacích, v kontrole dodržování firemní politiky nebo v zajištění bezpečnosti a podobně.[1][7]

Nástroje lze také rozdělit podle časového rámce, ve kterém analýza probíhá. Nástroje, které provadějí analýzu v reálném čase, jsou orientovány na bezpečnostní aplikace. Dají se sem zařadit různé typy aplikačních firewalů nebo jiné typy automatizovaného vyhledávání v přenášeném obsahu (IDS, IPS systémy). Interaktivní aplikace nabízejí práci nad množinou zachycených dat. Vzhledem k objemu síťové komunikace, který je dnes běžně přenášen, se zde ale předpokládá jistý stupeň předfiltrace, např. na zájmové komunikující strany a sadu protokolů.

2.1.1 Wireshark (TCPDump)

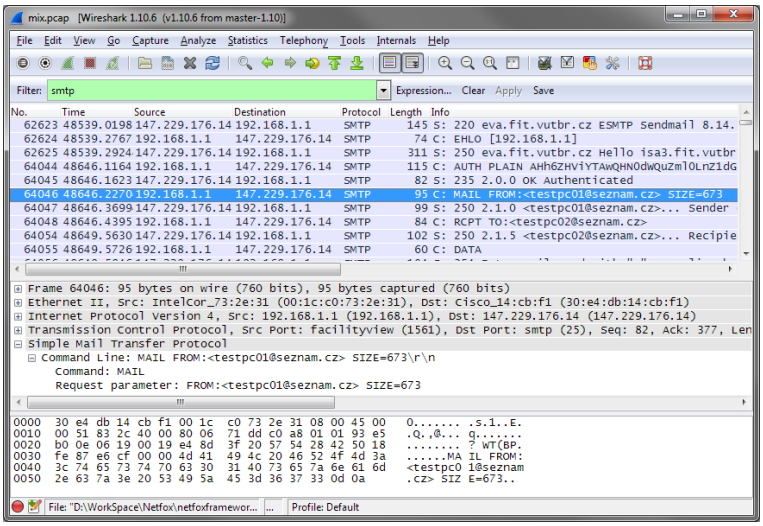
Wireshark (WS) je jedním ze základních a také nejrozšířenějších nástrojů pro analýzu síťové komunikace. Pomocí WS je možno prohlížet provoz jak on-line, tak ze souboru se zachycenou

komunikací. Pohled na data je zde zaměřen na jednotlivé přenosové jednotky - rámce. U každé přenosové jednotky lze zobrazit její naparsovaný obsah. Do toku zachycených rámců jsou zařazovány také virtuální jednotky, ve kterých je případně obsažen složený obsah pro daný aplikační protokol - obr. 2.1.

Hlavní předností tohoto nástroje je velké množství podporovaných protokolů. Mezi další výhody patří dostupnost zdrojových kódů (možnost rozšíření o vlastní parser), rychlost (díky implementaci v C) a možnost použití na různých platformách. Z hlediska analýzy aplikačního obsahu poskytuje WS pouze omezené možnosti. Jedná se zejména o procházení jednotlivých aplikačních zpráv a řadu statistik, které lze získat.

Pro export aplikačních dat jsou k dispozci pluginy pro protokoly HTTP, SMB a DICOM. Pro daný aplikační protokol jsou extrahovány všechny přenášené soubory, které lze poté uložit. Wireshark se také zaměřuje na skupinu protokolů pro VoIP. V daném souboru lze analyzovat jak signalizaci, tak vlastní hovorová data [2].

Ačkoliv se jedná nenahraditelný nástroj, jeho prostředky dané koncepcí neumožňují efektivní a komplexní analýzu obsahu směrem shora dolů.



Obrázek 2.1: Aplikace Wireshark

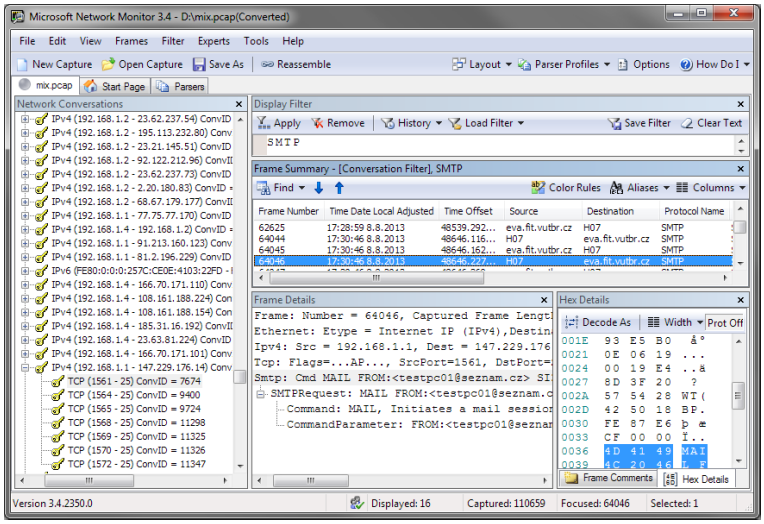
2.1.2 Microsoft Network Monitor

Microsoft Network Monitor (MNM) se svou koncepcí velice podobá Wiresharku. Je určen pouze pro stanice s operačním systémem MS Windows. I zde je pohled na data zaměřen na jednotlivé přenosové jednotky. MNM nad nimi ale vytváří hiearchickou strukturu. V první úrovni jsou jednotky tříděny podle L3 vrstvy (zdrojová a cílová adresa). V druhé úrovni podle transportní vrstvy (zrojový a cílový port) a nakonec podle *session* odpovídající aplikačnímu protokolu. V rámci dané úrovně a třídy jsou zobrazeny všechny přenosové jednotky které jí přísluší - obr. 2.2. Stejně jako v aplikaci Wireshark lze tuto množinu zúžit aplikováním filtrů.

MNM tak jako WS obsahuje velkou škálu parserů protokolů. Jednotlivé parsery jsou popsány v jazyce NPL. Do MNM lze díky tomu pomocí popisu v tomto jazyce snadno doplnit vlastní parser.

Pro analýzu aplikačních dat je v MNM k dispozici systém expertních nástrojů „Experts“[14], který umožňuje vytváření externích aplikací, přistupujících k datům ze síťového provozu

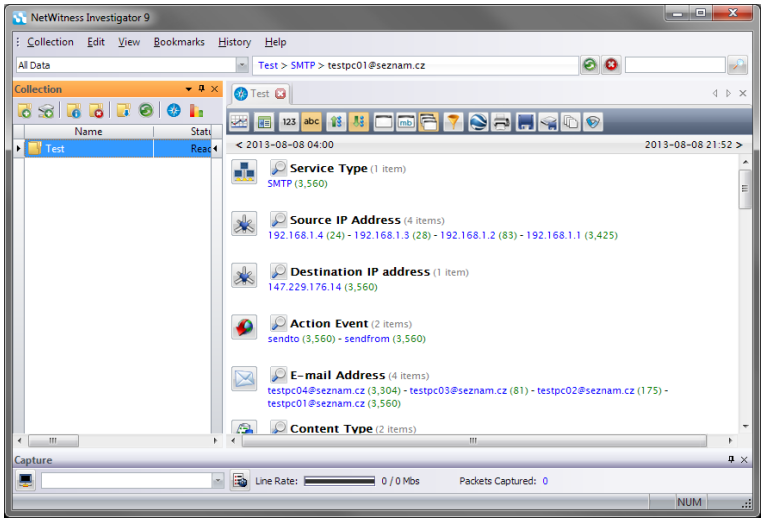
pomocí API. V základní instalaci nejsou k dispozici žádná z těchto rozšíření. Z oficiálních stránek projektu lze získat SDK pro vyváření vlastních nástrojů a několik již hotových projektů, které poskytují zejména statistiky ohledně síťového provozu.



Obrázek 2.2: Aplikace Microsoft Network Monitor

2.1.3 NetWitness Investigator

NetWitness Investigator je oproti předchozím aplikacím zaměřen na jiný způsob práce s obsahem síťové komunikace. Stejně jako MNM a WS umožňuje načíst zachycenou síťovou komunikaci, nebo sběr dat online. Zachycená data nezobrazuje v podobě jednotlivých přenosových jednotek, ale vytváří lexikon různých identifikátorů, které se v komunikaci vyskytují. Jedná se jak o informace obsažené v protokolech TCP/IP stacku (L2, L3 adresy), tak o aplikační identifikátory (emailová adresa, URL). Uživatel prochází tímto lexikonem a vytváří cestu identifikátorů, které postupně filtrují původní obsah - obr. 2.3.

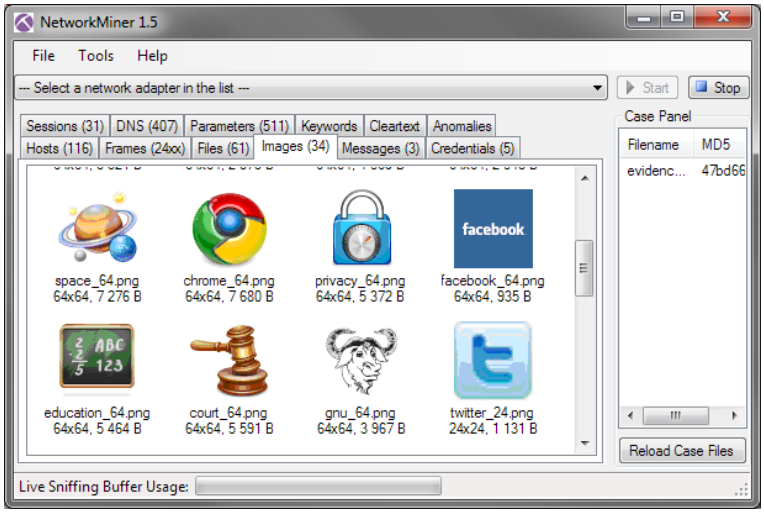


Obrázek 2.3: Aplikace NetWitness Investigator

Takto vybranou kolekci dat lze poté zpětně uložit do souboru a zpracovat dalšími nástroji. Kromě tohoto nabízí aplikace také možnost extrahovat známé typy souborů z vybrané kolekce. NetWitness Investigator se specializuje na filtraci událostí v síťové komunikaci. Na tomto poli přináší velmi účinné a intuitivní řešení. V kombinaci s aplikacemi, které umožní detailní analýzu aplikačních dat, se jedná se o velmi efektivní nástroj.

2.1.4 NetworkMiner

NetworkMiner je nástrojem ze skupiny zaměřující se převážně na extrakci obsahu. Aplikace umí obsah zachytávat, nebo ho načíst ze souboru capture. Po otevření souboru je automaticky provedena analýza jeho obsahu. Po dokončení jsou k dispozici nalezené výsledky. Ty jsou rozděleny do několika skupin (obrázky, soubory, zprávy a podobně). Každou ze skupiny výsledků lze zobrazit například ve formě tabulky, nebo galerie - obr. 2.4.



Obrázek 2.4: Aplikace NETRESEC NetworkMiner

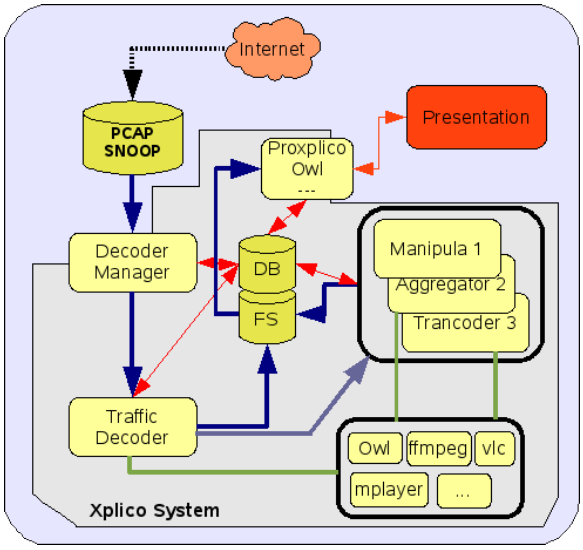
Nástroj nabízí jednoduchý způsob jak získat co největší množství obsahu ze zachycených dat. Mezi hlavní nedostatky patří slabá možnost zařazení nalezených výsledků do původního kontextu - uživatel je v převážné většině omezen na adresy a časové razítko.

2.1.5 Xplico

Xplico je nástroj pro systémy typu unix, který umožňuje analýzu aplikačních dat ze souborů se zachycenou síťovou komunikací. Oproti předchozím nástrojům se nejedná o monolitickou aplikaci, ale o zajímavý systém několika komponent - obr. 2.5. První částí je správce procesu extrakace DeMa, který řídí celý proces analýzy. O sestavení vlastních aplikačních dat se stará Xplico Traffic decoder. Takto získaná data jsou zpracována sérií manipulátorů. Extrahované dokumenty jsou uloženy do souborového systému a metainformace do databáze. Data jsou vizualizována pomocí vizualizačního systému[6].

Výhodou systému Xplico je podpora většiny běžně používaných aplikačních protokolů (HTTP, SIP, IMAP, POP, SMTP, FTP ...) [1]. Další výhodou je možnost kongruentní práce více uživatelů se systémem. Vizualizace dat probíhá pomocí webového rozhraní. Kombinace

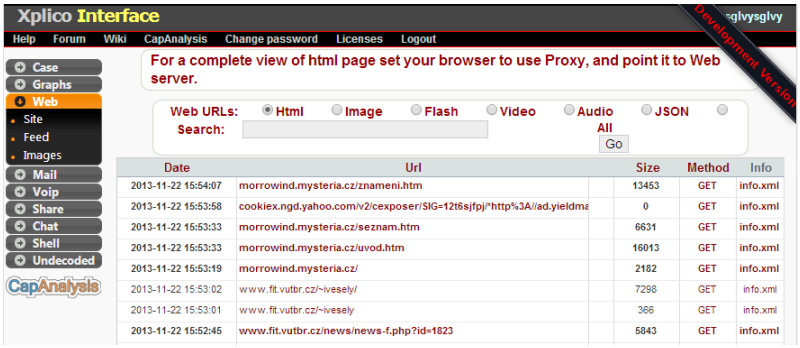
[1] http://www.xplico.org/status



Obrázek 2.5: Architektura systému Xplico

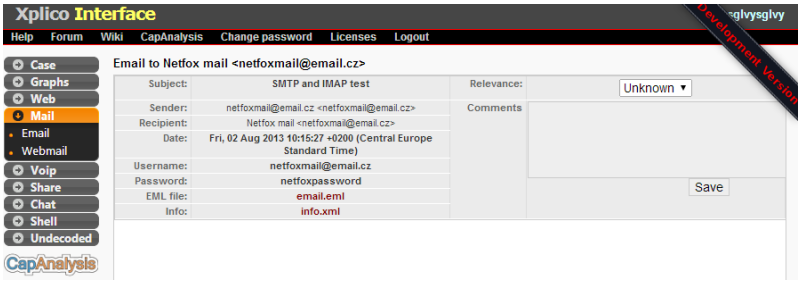
těchto přístupů vytváří možnost použití aplikace jakožto cloudového nástroje pro forenzní analýzu.

Webové rozhraní umožňuje upload soubourů s obsahem komunikace. Soubory jsou po uploadu analyzovány. K dispozici je základní statistický přehled o rozpoznaných údalostech - počet webových stránek, přijatých a odeslaných emailů, souborů přenesených přes ftp a podobně. Pro jednotlivé aplikační protokoly lze zobrazit přehled rekonstruovaných událostí - obr. 2.6, pro danou událost potom detail - obr. 2.7. Získaná data lze z rozhraní stáhnout v podobě standartního kontejneru pro daná data (eml pro e-mail, html pro webové stránky apod.).



Obrázek 2.6: Rekonstruované webové stránky v systému Xplico

Koncept systému je velmi zajímavý a vzhledem k licenci pod kterou je dostupný (GNU) se jedná o slibné řešení. Webové rozhraní systému má jisté výhody, v aktuální implementaci ale neposkytuje příliš pohodlnou práci s extrahovanými výsledky. Například filtrace v systému je možná pouze podle adresy hosta.



Obrázek 2.7: Rekonstruovaný email v systému Xplico

2.2 Koncept vytvářené aplikace

Koncept aplikace vychází z předchozího návrhu Ing. Ondřeje Ryšavého, Ph.D., který byl implementován v rámci projektu SEC6NET. Vytvářená aplikace bude vzhledem k jejímu plánovanému využití poskytovat prostředí pro offline analýzu zachycené síťové komunikace - zachycený obsah bude načítán ze souborů standardních formátů. Z hlediska pohledu na data a možností práce nabídne oba nejčastěji používané a popsané přístupy. Jejím přínosem by mělo být spojení obou do jednoho nástroje - aplikace umožní pracovat na úrovni jednotlivých přenosových jednotek a postupovat tak od nejnižších vrstev směrem k vyšším - získat a analyzovat aplikační data z vybrané množiny přenesených dat. Na druhé straně ale také přístup, kdy uživatel může na základě rekonstruovaného aplikačního obsahu (identifikátory, soubory, data) sestupovat zpět hierarchií síťových protokolů a získat tak množinu původních dat, ve kterých byla aplikační data nalezena pro další detailní analýzu. Přínosem tohoto přístupu právě směrem k forenzním aplikacím je možnost analyzovat na základě rekonstruovaných aplikačních dat podobné síťové toky, nebo naopak rekonstruovat aplikační data z vytipovaných síťových toků na základě jejich parametrů[7].

Dalším požadavkem, který je častým problémem a na který by měl být nástroj připraven, je možnost jeho rozšiřování. Toto si vyžádá kvalitní modulární konstrukci aplikace. Jedná se jednak o možnost pohodlného rozšiřování podporovaných aplikačních protokolů, společně s tímto je ale třeba umožnit snadnou implmentaci nových pohledů na data a práci s nimi.

Kromě tohoto by měla aplikace nabízet také možnost rozšíření formou externích součástí - pluginů. Ty budou moci pohodlně využít veškerá získaná data a funkcionalitu aplikace. Díky tomuto bude možno například snadno a flexibilně vytvořit specifický typ analýzy pro daný případ včetně vhodné vizualizace.

Kapitola 3

Netfox Framework

Další části práce odkazují na Netfox framework, je zde proto uveden krátký popis toho, čím framework je, co všechno obsahuje a nabízí.

Netfox (Network Forensic Extendable Analysis Tool) framework je kolekcí knihoven pro analýzu a manipulaci s obsahem síťové komunikace zachycené v souborech (soubory capture). V rámci projektu SEC6NET tvoří framework hlavní platformu pro vývoj nástrojů zaměřených na rekonstrukci aplikačního obsahu. Framework je implementován v jazyce C# za použití .NET frameworku. Jeho knihovny a funkcionalitu lze rozdělit do několika hlavních skupin[8].

Práce s capture soubory

Této části se věnuje *PmLib*. Knihovna implementuje vlastní parsery nejběžnějších typů capture souborů, jako jsou pcap/pcap-ng (tcpdump), nebo cap (Microsoft Network Monitor). *PmLib* umožňuje daný typ souboru jak načítat, tak ukládat. Lze ho proto využít pro různé typy manipulace s jednotlivými soubory a jejich obsahem - např. konverze formátů, setřídění obsahu více souborů (aplikace PCAPMerger[1]) a podobně. *PmLib* také používají téměř všechny ostatní součásti frameworku jakžto zdroj síťových dat. Pro zvýšení výkonnosti používá *PmLib* vlastní index, který vytváří a ukládá pro každý zpracovávaný soubor. Odpadá tak nutnost jeho opakovaného parsování. *PmLib* poskytuje naparsovaný obsah paketů pomocí knihovny *Packet.Net*[15].

Reassembling dat z transportní vrstvy

Tato část tvoří jádro frameworku. Implementuje postupy, pomocí kterých lze ze sekvence zachycených přenosových jednotek obsahujících transportní vrstvu TCP sestavit původní tok aplikačních dat. V rámci tohoto procesu je nejprve původní obsah rozdělen do tzv. konverzací, které odpovídají jedné TCP session, nebo flow v případě UDP. Pro tuto session jsou poté vyvořeny soubory aplikačních zpráv (tzv. L7PDUs), které lze procházet a simulovat chování aplikací pro získání původního přenášeného obsahu. Framework si umí poradit i s některými výpadky (chybějícími daty) v zachycené komunikaci. Chybějící data nahrazuje výplní, chybějící řídící zprávy aproximuje z dostupných informací. Jedná se o poměrně složitý postup, jemuž se podrobně věnuje práce Bc. Jana Pluskala - Prostředí pro zpracování dat ze zachycené síťové komunikace.[9].

[1] https://pcapmerger.codeplex.com/

Analýza a extrakce aplikačních dat

Nad původními seskládanými aplikačními daty lze provádět analýzu a extrakci přenášeného provozu. Vzhledem k velkému množství a variabilitě aplikačních protokolů je ve frameworku použit následující postup při vytváření tzv. aplikačního exportéru pro daný protokol/skupinu protokolů:

Protokol je popsán pomocí jazyka NPL (Network Parsing Language)[2]. Částečně lze využít hotové popisy dostupné společně s aplikací Microsoft Network Monitor. Takovýto vytvořený, nebo upravený popis je poté vstupem do tzv. *NplangCompileru*. Tato další část frameworku implementovaná v jazyce F# vyvoří na základě předloženého popisu implementaci parseru v jazyce C#. Ke každému parseru implementuje programátor odpovídající aplikační exportér.

Export aplikačních dat je spouštěn na vybrané konverzace. Je možné spustit vybrané exportéry na všechny konverzace, nebo vybrat exportéry pro konverzace na základě detekce aplikačního protokolu.

Každá z dvojice parser-exportér poté pracuje následujícím způsobem. Parser má k dispozici syrová rekonstruovaná aplikační data. Tato data postupně zpracovává - posouvá po nich čtecí hlavu na základě původního popisu v NPL. Během tohoto procesu ukládá do svých datových struktur (asociativní kontejnery) programátorem specifikované položky (části hlaviček, příkazy, vlastní obsah). Jakmile je dokončeno parsování daného celku (příkaz, odpověď apod.) přechází řízení do aplikačního exportéru. Zde má programátor k dispoci jednotlivé položky naparsovaného obsahu. Nejčastěji je zde implementován stavový mechanismus (může být ale také bezstavový), který simuluje chování původních aplikací. Díky tomuto je exportér schopen zrekonstruovat nejen původní obsah, ale doplnit ho také o sémantické informace získané během exportu (servisní informace, přihlašovací údaje a podobně).

Tímto způsobem je pro danou konverzaci vytvořena série tzv. aplikačních událostí. Událost většinou odpovídá přenosu hledaného obsahu - dotaz a odpověď u HTTP, stažený email u POP3, přenesená zpráva u IM a podobně. Podle daného použití frameworku je kolekce těchto výsledků společně s dalšími informacemi buďto uložena do XML dokumentu, nebo emitována v podobě události směrem k vyšším vrstvám - viz. 4.2 Kontroler pro Netfox framework. Vlastní data mohou být součástí tohoto popisu, nebo mohou být uložena do souborového systému.

[2] http://nmparsers.codeplex.com/

Kapitola 4

Návrh nástroje

Tato kapitola popisuje problémy řešené při návrhu nástroje pro analýzu obsahu síťové koumunikace *Netfox Detective*. Nástroj využívá *Netfox Framework* jakožto funkční jádro. V této kapitole je popsáno jeho doplnění o kontroler, který umožňuje jeho snadné použití nejen v této aplikaci. Data, která framework produkuje, aplikace zpracovává a ukládá. Další část návrhu se tedy věnuje persistenci dat v aplikaci. Ta je zajištěna za pomoci persistenční vrstvy a objektové databáze. Poslední část kapitoly se věnuje celkovému návrhu a použitým postupům.

4.1 Požadavky na vytvářený nástroj

Při návrhu nástroje byl kladen důraz zejména na tyto požadavky :

- **Objekový návrh - MVVM**

  Použití návrhového vzoru Model View ViewModel a důsledná separace jednotlivých vrstev aplikace tak, jak tento model předepisuje, zajistí udržovatelnost aplikace během jejího vývoje.

- **Rozšiřitelnost**

  Aplikace musí být navžena tak, aby umožňovala snadnou implementaci podpory nových aplikačních protokolů a pohledů na rekonstruovaná nebo zachycená data. Přidání podpory nového aplikačního protokolu nebo pohledu nesmí vyžadovat zásah do jádra aplikace nebo jeho detailní znalost. V nejlepším případě by měla aplikace podporovat dynamické začlenění nových kompoment bez nutnosti rekompilace - forma aplikačních knihoven nebo pluginů. Aplikace by také měla automaticky reflekovat změny v *Netfox Frameworku*.

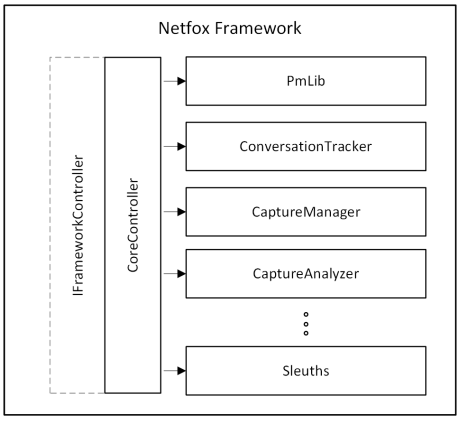
- **Škálovatelnost**

  Lze předpokládat použití aplikace pro analýzu a rekonstrukci velkých objemů zachycených síťových dat. Nelze tedy použít základní přístupy k ukládání a práci s rekonstruovanými metadaty přímo v paměti aplikace. Je třeba zvolit takový způsob, který zajistí, že v paměti aplikace bude načtena pouze ta podmnožina dat, nad kterou je prováděna analýza, a nikdy v ní nebude třeba zpracovávat celý soubor metadat.

4.2 Kontroler pro Netfox Framework

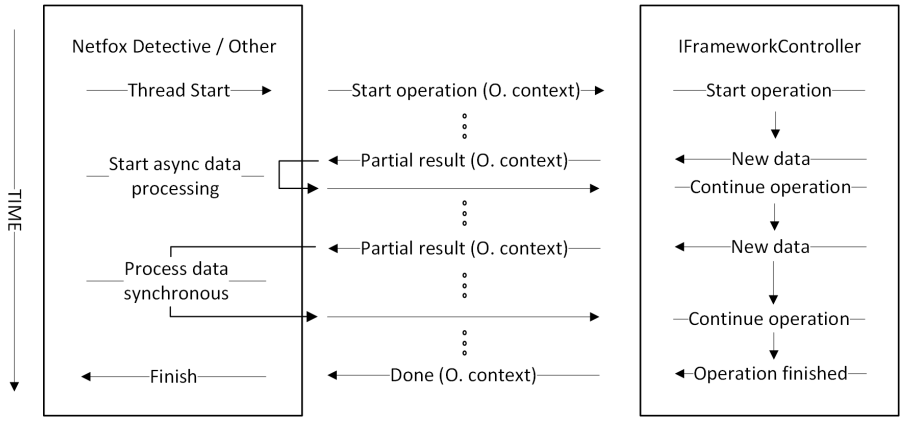
Navržený Kontroler zajišťuje veškerou interakci s Netfox frameworkem. Má za úkol zpřístupnit funkce frameworku tak, aby jej bylo možné použít pro vytvoření cílové aplikace.

Pro odstínění jednotlivých součástí frameworku a jejich implementací od aplikací, které ho mohou využívat, byla zvolena forma návrhu pomocí rozhraní - interface doplněný o sématický popis, přičemž framework musí poskytovat třídu, která toto rozhraní bude implementovat. Toto umožňuje snadné použití bez nutnosti znalosti vnitřní implementace frameworku - obr. 4.1.



Obrázek 4.1: Kontroler Netfox Frameworku

Druhým aspektem, na který byl při návrhu kladen důraz, je efektivní práce s frameworkem. To zahrnuje zejména možnost práce s více různými soubory, různou míru granularity při spouštění operací a možnost asynnchroního zpracování získaných výsledků - obr. 4.2.

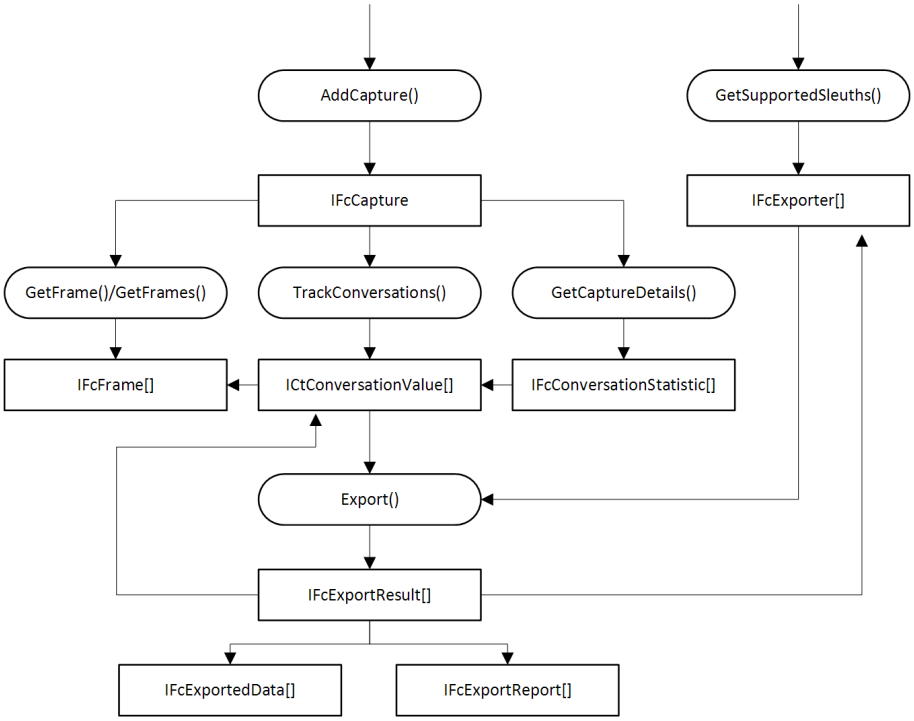


Obrázek 4.2: Řízení při operaci s kontrolerem Netfox Frameworku

Poslední důležitá požadovaná vlastnost kontroleru koresponduje s dalším předpokládaným vývojem frameworku. Tak jak se bude framework rozšiřovat o podporu nových aplikačních protokolů, musí kontroler nabízet prostředky, pomocí kterých bude dynamicky informovat nadřazené aplikace o množině prostředků, které aktuálně nabízí. Toto umožní aplikacím v čase automatickou propagaci změn až na uživatelskou úroveň bez nutnosti zásahu do jejich implementace.

4.2.1 Rozhraní kontroleru

Detailní popis rozhraní kontroleru lze naléz v programové dokumentaci Netfox frameworku. Pro přiblížení práce s kontrolerem z pohledu aplikace je zde uveden příklad jeho použití pro několik základních operací s frameworkem - obr. 4.3.



Obrázek 4.3: Workflow v kontroleru Netfox frameworku

Přidání souboru capture

Prvním krokem je přidání souboru capture do frameworku pomocí metody *AddCapture()*. Seznam již obsažených souborů lze získat pomocí *IncludedCaptures()*. Při další práci je capture soubor identifikován pomocí reference na tzv. *IFcCapture*. Pro ověření korespondence obsahu souboru, např. s předchozími výsledky práce, lze využít kontrolní součet - dostupný přes *IFcCapture*. V tuto chvíli je soubor připraven pro další zpracování ve frameworku. Knihovna *PmLib* načte jednotlivé rámce a ty jsou přístupné pomocí sady metod *GetFrame()/GetFrames()*.

Tracking konverzací

Vzhledem k tomu, že základní jednotkou, ke které se vztahují mnohé další operace frameworku, je konverzace, je pro daný soubor nutné nejprve provést tzv. tracking. Tato operace rozdělí rámce do skupin, které představují výměnu dat mezi dvěma aplikačními stranami v čase - u paketů s TCP transportní vrstvou je to chronologické pořadí packetů, tak jak byly přenášeny, u o ostatních (bezstavových), jako jsou rámce s vrstvou UDP, je to kombinace zdrojových a cílových portů a adres. Toto stačí provést pouze jednou na začátku práce s daným souborem pomocí metody *TrackConversations()*. Framework během operace postupně vrací metadata reprezentující jednotlivé koverzace.

Statistiky

Kontroler dále nabízí možnost získat z frameworku detailní statistiky pro daný soubor. Framework provede na základě reassemblingu analýzu všech zadaných konverzací a postupně pro každou vrátí základní informace, jako je celkový počet přenesených rámců/bajtů, časový rámec, který konverzace vymezuje, počet získaných bajtů na aplikační úrovni a podobně. V případě konverzace s transportní vrstvou TCP je k dispozici také počet chybějících dat.

Rekonstrukce a extrakce aplikačních dat

Nad jednotlivými konvezacemi lze provádět vlastní rekonstrukci a extrakci aplikačních dat (export). Export je zahájen pomocí volání metody *ExportData()*. Kontroler zde nabízí řadu možností identifikace zájmových konverzací - od všech v daném souboru až po seznam jednotlivých vybraných. Frameworku je třeba také specifikovat, pomocí kterých exporterů mají být konverzace zpracovány. Seznam aktuálně podporovných exportérů lze získat pomocí volání *GetSupprotedSleuths()*. Aplikace tak může uživatelům nabízet množinu aktuálně podporovaných exportérů bez nutnosti zásahu do její implementace.

Po spuštění vlastního exportu (viz kapitola 3. Netfox Framework - Analýza a extrakce aplikačních dat) jednotlivé exportéry postupně vracejí objekty reprezentující rekonstruovaná data. Zde záleží na daném aplikačním protokolu a implementaci exportéru. Nejčastější je způsob, kdy exportér uloží rekonstruovaný obsah do souborového systému a aplikaci předá objekt obsahující metainformace získané při rekonstrukci - např. cesty k souborům, kódování obsahu, časová razítka, identifikace komunikujících stran a podobně.

4.3 Persistence dat

Aplikace bude muset při svém běhu zpracovávat a ukládat velké množství dat získaných během analýzy obsahu síťové komunikace. Je třeba zajistit snadnou a efektivní možnost oragnizace dat (nejčastěji v stromových strukturách), vyhledání, filtrace, editace a další.

Vzhledem k předpokladanému množství dat toto nelze provádět nad daty načtenými v paměti, ale je třeba využít některého typu databázového uložiště, která jsou pro toto optimalizována. Pro zachování komfortu práce s daty z pohledu aplikace bylo rozhodnuto o využití tzv. noSQL objektové databáze. Tyto databáze nevyžadují schéma, a proto je jejich použití vhodné také s ohledem na plánované rozšiřování aplikace (možnost specifického formátu dat pro různé aplikační protokoly).

Během řešení bylo vytipováno několik takovýchto databází z hlediska možnosti jejich použití v rámci vytvářené aplikace. Následuje popis vlastností dvou z nich, které se ukázaly být vhodnými kandidáty.

RavenDB

Jedná se o noSQL databázi pro systémy s Windows[16]. Pro ukládání dat používá formát JSON. Výhodou je, že databáze používá .NET framework stejně jako cílová aplikace. Toto umožňuje její efektivní provázání za použití nativních datových formátů a struktur (lazy enumeration pomocí LINQ dotazů apod.). Další výhodou je možnost použít databázi jako vestavěnou součást aplikace (embeded verze) i jako klient-server architekturu. První z uvedených možností poskytuje komfort pro rychlý začátek práce s aplikací a snadnou přenositelnost aplikačních dat. Mezi hlavní nevýhody patří limit v podobě počtu dokumentů, které lze pomocí jednoho dotazu načíst. Toto je dáno zejména typem přenosu (HTTP).

Požadované dokumenty lze načíst pomocí více po sobě jdoucích přenosů, dochází tak ale ke ztrátě výkonnosti, kterou poskytuje mechanismus lazy enumeration[4].

MongoDB

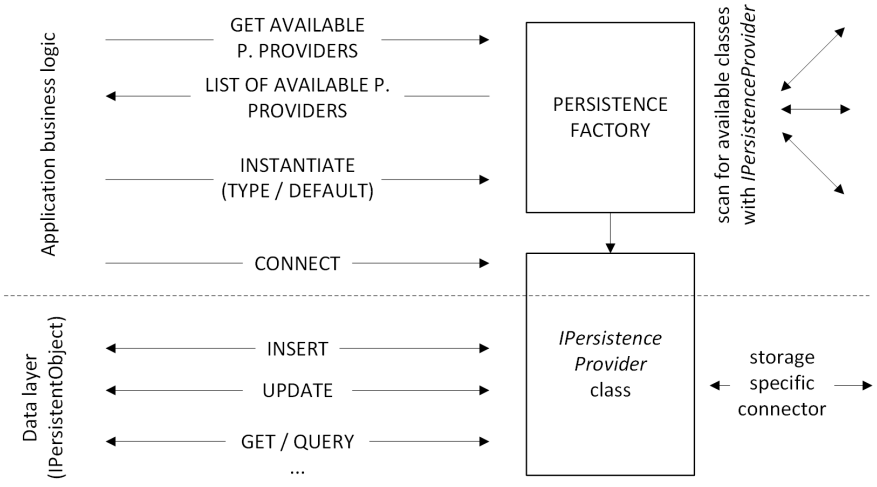
Jedná se o multiplatformní noSQL databázi[12]. Jako formát pro ukládání dat používá BSON, který je binární obdobou formátu JSON. Autoři uvádějí, že oproti JSON má nižší režii a umožňuje také rychlejší serializaci/deserializaci - používá formát datových typů jazyka C. MongoDB vždy používá architekturu klient-server. Konektor klientské aplikace se k serveru připojuje pomocí binárního protokolu nad TCP/IP *MongoDB Wire Protocol*. Oproti RavenDB v testech skutečně dosahuje vyšší výkonnosti[17], vyžaduje ale server běžící jako samostatnou nainstalovanou službu, což může být v některých případech limitující.

Použité uložiště

Vzhledem k tomu, že obě popsané databáze by mohly poskytovat zajímavé vlastnosti ve vytvářené aplikaci, bylo rozhodnuto, že aplikace bude podporovat oba typy, stejně jako možnost přidání podpory dalších typů uložišť do budoucna. Toto ovlivnilo zejména návrh persistenční vrstvy ve vyvářeném nástroji.

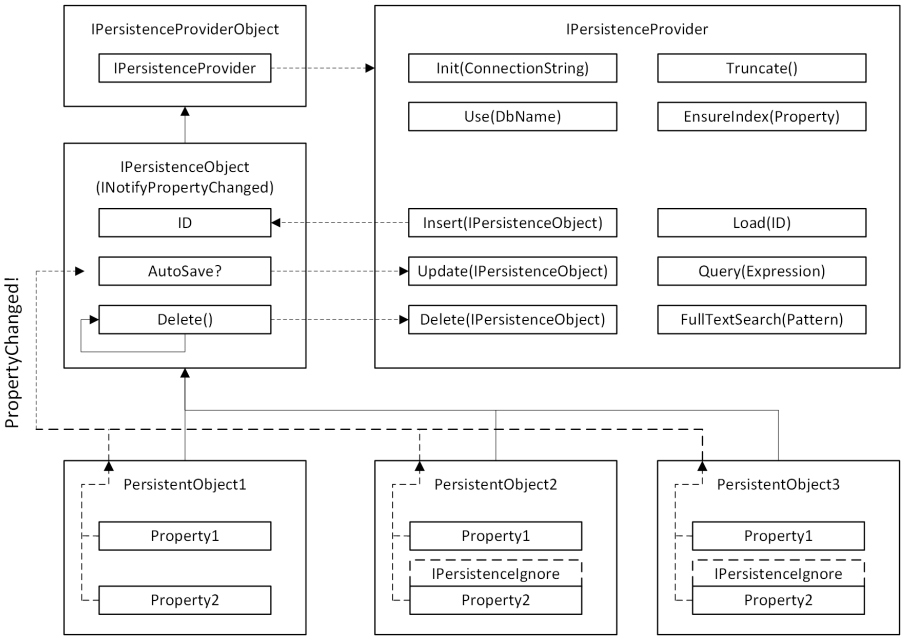
4.3.1 Návrh modelu persistence dat v aplikaci

Vzhledem k výše popsaným okolnostem jádro aplikace pouze definuje rozhraní pro zajištění peristence dat *IPersistenceProvider*. Jednotlivé typy uložišť poté toto rozhraní implementují. Jádro aplikace při spuštění detekuje všechny dostupné typy těchto základních prezistenčních vrstev a nabízí je uživateli k použití - správu instancí těchto uložišť zajišťuje třída implementující návrhový vzor *factory* - obr. 4.4.



Obrázek 4.4: Návrh zajištění peristence dat v aplikaci

Každá ze tříd implementující toto rozhraní poskytuje základní operace pro persistenci objektů a správu daného uložiště. Jedná se například o vytvoření uložiště, připojení se k němu, vložení objektu, smazání nebo jeho editaci. Rozhraní umožňuje čtení konkrétního objektu nebo získání kolekce objektů splňující určité parametry (LINQ dotaz) - obr. 4.5, 4.6.



Obrázek 4.5: Rozhraní pro zajištění peristence dat v aplikaci

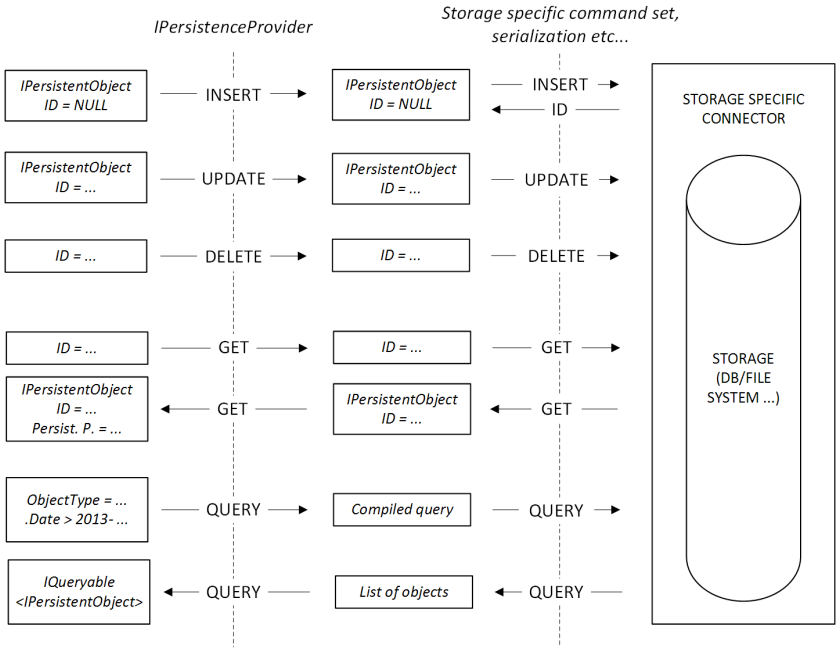
Každý objekt, u kterého má být zajištěna persistence, musí implementovat předepsané rozhraní *IPersistentObject*. To definuje zejména property ID, pomocí které je objekt v uložišti unikátně identifikován. Toto ID je při prvním vložení objektu generováno uložištěm v pro něm nativním formátu. Každému objektu implementujícímu *IPersistentObject* načtenému z uložiště je automaticky nastavena reference na rozhraní tohoto uložiště. Díky této referenci lze v aplikační logice snadno zajistit automatické ukládání objektů při změně nebo další podobné vlastnosti. Rozhraní dále definuje metody, které by měl objekt implementovat. Jedná se například o odstranění objektu, které může autonomně zajistit odstranění všech zavislých tříd z uložiště. V objektu, u kterého má být zajištěna persistence, je nutné specifikovat *properties*, které mají nebo nemají být uloženy. Toto je provedeno označením atributem s rozhraním *IPersitenceIgnore*.

4.3.2 Propagace změn v databázi směrem k vyšším vrstvám aplikace

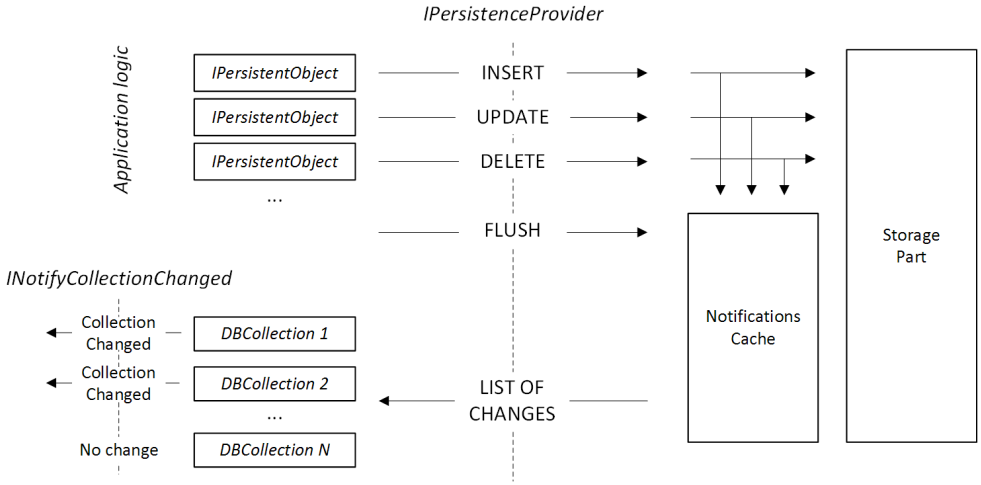
Vzhledem k tomu, že aplikace je navrhována podle struktury MVVM a předpokládá použití WPF frameworku, je vhodné zajistit automatickou notifikaci o změně obsahu objektů nebo jejich kolekcí směrem k vyšším vrstvám aplikace. K zajištění tohoto cíle byla pro aplikaci navržena speciální třída *ObservableDBCollection* reprezentující kolekci objektů v databázi, která zároveň implementuje standardní rozhraní *INotifyCollectionChanged* - viz 4.6.3 Databázové kolekce.

Notifikace o změně v databázi probíhá podle následujícího principu. Při vložení, smazání nebo editaci objektu pomocí třídy zajišťující persitenci (*IPersistenceProvider*) jsou kromě zadané operace předávány další dva volitelné parametry, a to zdali mají být o změně kolekce informány a případně jestli má být notifikace provedena okamžitě, nebo až později (odložená notifikace). Pokud má být notifikace provedena okamžitě, jsou všechny kolekce navázené na dané uložiště formou události informovány o změně (který objekt byl změněn a jakým způsobem).

Možnost odložené notifikace je zde z důvodu optimalizace výkonnosti. Velké množství



Obrázek 4.6: Princip implementace zajištění peristence dat v aplikaci



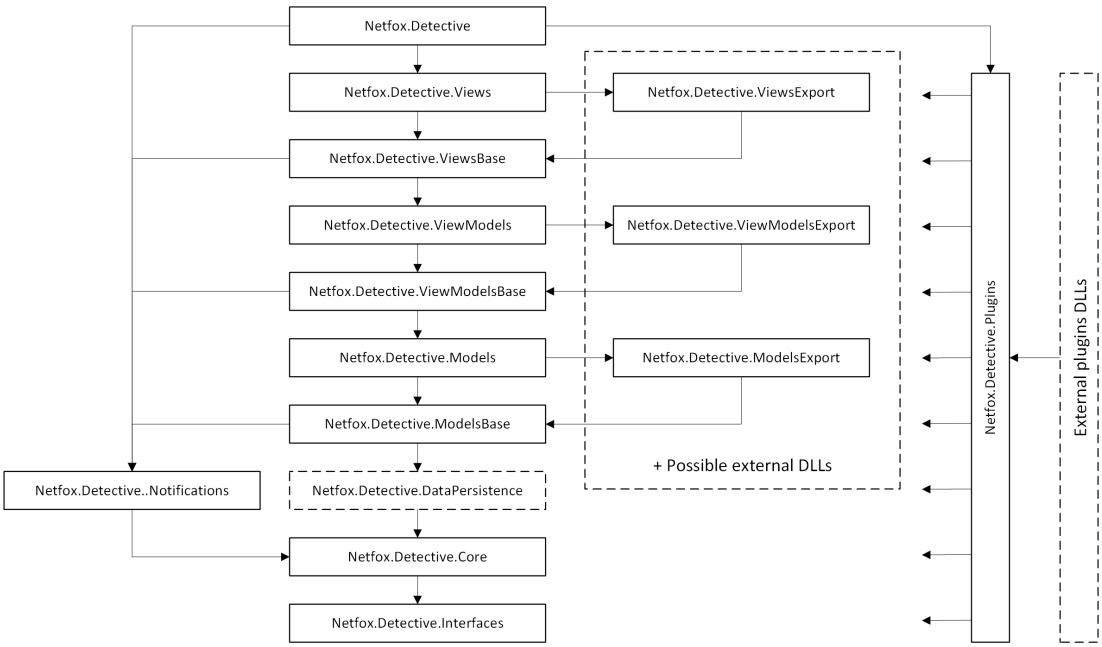
Obrázek 4.7: Princip odložené notifikace o změně databázové kolekce

po sobě jdoucích změn v databázi by generovalo velké množství událostí, které by kolekce musely opakovaně zpracovávat. Pokud jsou ale notifikace uloženy a předány kolekcím najednou až po dokončení blokové operace, dojde k výraznému snížení režie spojené s tímto mechanismem, při zachování principu automatické aktualizace na základě událostí eliminující nekonzistence mezi jednotlivými vrstvami aplikaci - obr. 4.7.

4.4 Celkový návrh aplikace

Na základě uvedených požadavků a dílčích návrhů byl vytvořen celkový návrh aplikace. Aplikace je rozdělena do jednotlivých modulů, které tvoří hierarchickou strukturu - obr.

4.8. Návrh odpovída architektuře MVVM[4].



Obrázek 4.8: Návrh struktury aplikace Netfox Detective

Nejnižší vrstvu tvoří rozhraní (*Interfaces*) viditelná všem nadřezeným součástem. Jsou zde specifikovány předpisy pro základní typy objektů, které se v aplikaci vyskytují. Kromě rozhraní týkajících se persistence dat - viz 4.3.1 - je zde například rozrhaní *ISystemComponent*, které musí implementovat všechny součásti aplikace (slouží např. k identifikaci zdroje ve výjimkovém mechanismu) a další.

Nad touto vrstvou leží jádro aplikace (*Core*). Nejdůležitější součástí této vrstvy je platforma pro univerzální komunikaci mezi komponentami aplikace pomocí messagingu[4] *DetectiveMessage* - viz. 4.6.2. Mimo jiné poskytuje tato vrstva také globální singleton zprostředkující ukládání a načítání všech aplikačních nastavení.

Mezi jádro aplikace a její datový model lze zařadit persistenční vrsvu (*DataPersistence*), tak jak byla popsána v předchozí podkapitole. Ve skutečnosti ale všechny nadřazené vrsvy závisí pouze na rozhraních, která jsou tímto modulem implementována. Implementace pro různé druhy uložišť tak může být také doplněna formou externích knihoven.

Datový model aplikace se skládá ze tří modulů. Základ tvoří modul *ModelsBase*, který obsahuje implementaci všech esenciálních a generických komponent modelu. Je zde také obsažena obecná aplikační logika (business logic), jako jsou výše uvedené databázové kolekce a podobně. Vlastní entity datového modelu a odpovídající aplikační logika jsou implementovány v modulu *Models*. Modely specifické pro dané výsledky exportu jsou implementovány v odděleném modulu *ModelsExports*, který může být postupně rozšiřován pomocí externích komponent. Detailnější popis je obsažen v následující podkapitle 4.5.

Nad vrstvou modelů se nachází korespondující struktura viewmodelů (dle architektury MVVM). Je zde opět společný základ a oddělný modul pro viewmodely specifické pro jednotlivé výsledky exportu.

Prezentační vrstva aplikace je tvořena jednotlivými pohledy *Views*. Jejich instance vytváří hlavní *ApplicationView* na základě příslušnosti k třídě dané implementací rozhraní. V aplikaci je navrženo několik typů pohledů, rozdělení ale není striktně vyžadováno. Více

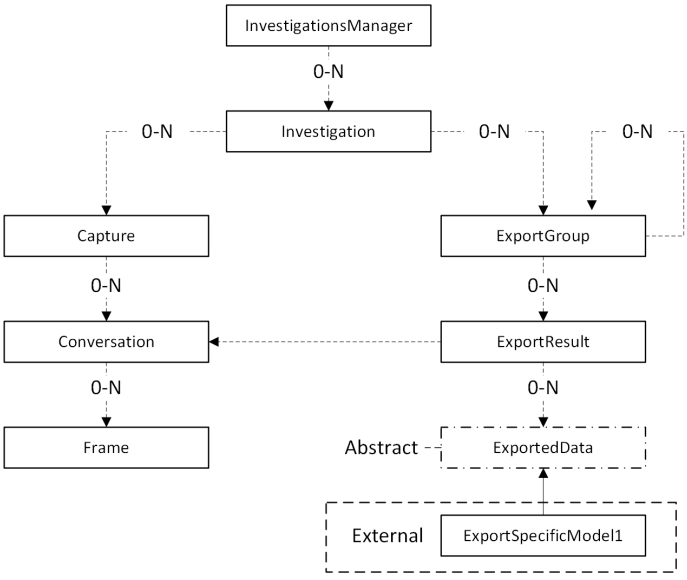
v podkapitole 4.6.6.

Za běhu aplikace vznikají události (způsobené zejména uživatelskou interakcí). Tyto události mohou vyvolávat různé a často nezavislé akce v různých částech aplikace. Může se jednat například o změnu aktuálně aktivního modelu, pohledu a podobně. Aby nebylo nutné všechny tyto části informovat formou standardních událostí nebo volání metod a udržovat tak složitou hierarchii referencí, obsahuje aplikace speciální modul, který je referencován téměř všemi ostatními (až na jádro) a který tuto funkcionalitu poskytuje - viz. také 4.6.2.

Poslední část struktury aplikace tvoří modul umožňující připojení extreních modulů (*Plugins*). Tento modul má přístup ke všem ostatním modulům. To mu a extrením modulům umožňuje implementovat funkcionalitu, která využívá jakoukoli část aplikace. Ve výjimečných případech, kdy potřebují základní moduly komunikovat s externími (např. aktivace pluginu na daný model), se tak děje pomocí rozhraní, které modul implementuje viz. také 4.6.7 Externí rozšíření aplikace - pluginy.

4.5 Datový model aplikace

Datový model aplikace je tvořen hierarchickou strukturou objektů, která ve většině případů respektuje reprezentaci objektů v Netfox Frameworku a jeho kontroleru - obr. 4.9.



Obrázek 4.9: Základní datový model aplikace

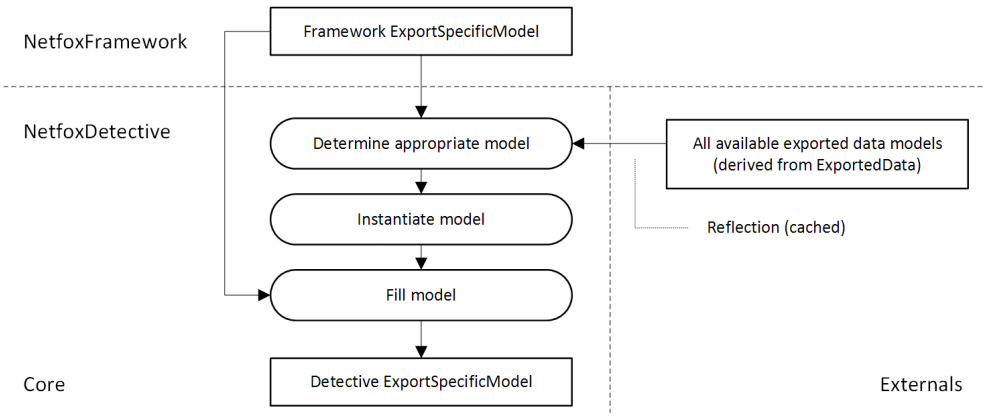
Základní pracovní jednotkou v rámci aplikace je projekt (*Investigation*). Veškeré vazby mezi objekty (soubory, konverzacemi, rámci a exportovanými aplikačními daty) existují právě v rámci tohoto projektu. Datové uložiště obsahuje 0 - N těchto projektů. Jejich správu (vytváření, rušení, aktivaci) zajišťujě *InvestigationManager*, který je právě jeden pro dané uložiště. *InvestigationManager* není persistentní. Jeho instance je vytvořena po připojení k danému uložišti. Nad tímto uložištěm provádí popsané základní operace - viz. 4.6.1. Projekt (*Investigation*) obsahuje 0 - N souborů s obsahem síťové komunikace (*Capture*) a 0 - N skupin s výsledky exportu aplikačních dat (*ExportGroup*).

Capture obsahuje 0 - N konverzací (*Conversation*), tak jak byla v něm obsažená komunikace rozdělena Netfox Frameworkem. Každá z konverzací je složená z reprezentace 0 -

N síťových rámců (*Frame*). Persistentní jsou v modelu z důvodu úspory kapacity narozdíl od ostatních částí pouze čísla rámců - v případě potřeby je obsah načten přímo z původního souboru - rámce nejsou persistentní. Konverzace je nejdůležitějším pohledem na síťová data v aplikaci. Všechny akce v aplikaci se vztahují právě ke konverzacím - např. export aplikačních dat se spouští nad vybranými konverzacemi.

Skupiny s výsledky exportu (*ExportGroup*) vytvářejí v rámci aplikace stromovou strukturu podobnou adresářům v souborových systémech. Každá ze skupin obsahuje 0 - N podskupin (opět *ExportGroup*) a 0 - N výsledků exportu z aplikačních dat (*ExportResult*). Struktura skupin může být vytvářena jak automaticky, tak na uživatelské úrovni. Výhodou použití této struktury je možnost hierarchického pohledu na data. Směrem od kořenové skupiny, kdy jsou k dispozici agregovaná data ze všech podskupin, až po konkrétní skupinu obashující například pouze výsledky týkající se daného uživatele a aplikačního protokolu.

Vzhledem k variabilitě exportovaných aplikačních dat, základní datový model aplikace definuje pouze abstraktní třídu *ExportedData*, která implementuje vlastnosti nezbytné k začlenění do něj. Vlastní model specifický pro daný výsledek poté může být implementován v externí knihovně. Správný typ objektu je za běhu rozpoznán pomocí informací dostupných z jeho rozhraní. Naplnění daty z původního výsledku je pak již plně v jeho režii - obr. 4.10.



Obrázek 4.10: Transformace specifických modelů z frameworku do aplikace

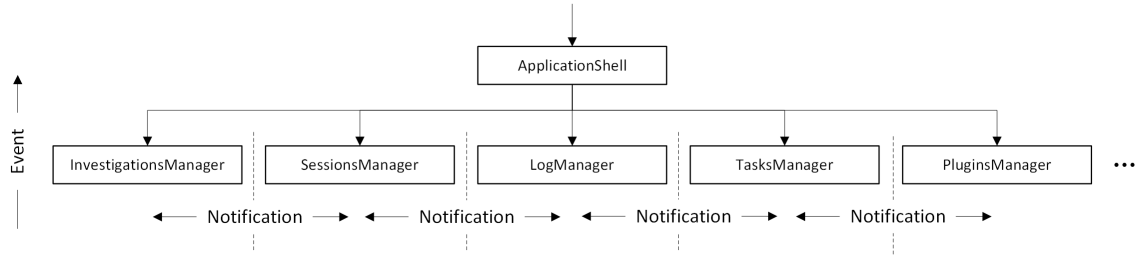
Při návrhu byla také zvažována možnost použití originálních typů objektů z frameworku. Toto řešení se ale nejevilo jako příliš vhodné vzhledem k tomu, že objekty ve frameworku nejsou navrhovány způsobem, který je vyžadován při jejich efektivním ukládání a načítání v objektové databázi. Tento transformační krok také do jisté míry izoluje změny, které mohou při vývoji frameworku nastat. Tyto získané vlastnosti byly nakonec upřednostněny před mírnou ztrátou výkonnosti danou tímto mechanismem.

4.6 Aplikační logika

Aplikační logika je složena zejména z operací s již popsaným datovým modelem a Netfox Frameworkem. Vzhledem k rozsahu aplikace zde budou popsány pouze základní principy a některé zajímavé detaily.

4.6.1 Základní principy aplikační logiky

Výchozím bodem v aplikaci je třída *ApplicationShell* z hlavního modulu *Netfox.Detective*. Její jediná instance je vytvořena po spuštění aplikace. Tato třída po vytvoření zajistí instanciaci všech ostatních tříd nezbytných k běhu aplikace. Jedná se sérii správců „sub-shellů“, které se starají o jednotlivé oddělené úlohy - obr. 4.11.



Obrázek 4.11: Vytváření základních instancí v aplikaci

Mezi nejdůležitější správce patří *InvestigationManager*, pomocí kterého v rámci aplikace vznikají a zanikají jednotlivé projekty a tedy veškerý její obsah - viz. 4.5 Datový model aplikace. Správce logu *LogManager* sbírá za běhu aplikace pomocí notifikací informace o všech důležitých událostech. Ty poté podle nastavení ukládá nebo nabízí k zobrazení v prezentační vrstvě aplikace. Důležitou instancí je také správce úloh *TaskManager*, který se za běhu stará o jednotlivé úlohy spuštěné na pozadí aplikace. Umožňuje jejich vytvoření, rušení a udržuje informace o jejich stavu, které jsou k dispozici uživateli -viz. 4.6.4. Speciálním typem správce je správce externích modulů - pluginů - *PluginsManager*. Ten tvoří vlastní typ shellu pro všechny externí moduly. Umožňuje jejich dynamické zavádění a odstraňování z aplikace. Udržuje informace o jejich vlastnostech a skrze standardní rozhraní nabízí možnost jejich aktivace dalším komponentám aplikace -viz. 4.6.7.

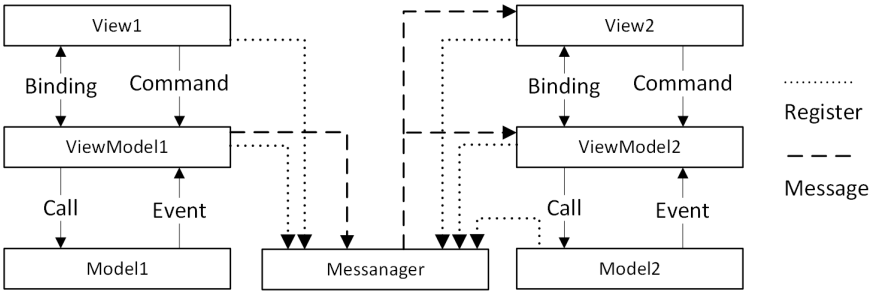
Práce s aplikací je založená na výběru objektu a následné práci s ním. *Investigation* tak např. obsahuje referenci na aktuální *Capture*. Aktuální *Capture* poté na akutální *Conversation* a podobně. Pokud dojde ke změne aktivního objektu daného typu - např. změna akutálního *Capture* - informuje třída, která tuto informaci spravuje (v tomto případě *Investigation*), všechny ostatní třídy, pro které je tato informace relevantní, pomocí zaslání zprávy. Takovouto třídou je například *view* zobrazující obsah daného *Capture*. To si díky této zprávě může upravit svůj *datacontext* na aktuálně vybraný viewmodel. Stejně tak se může dané view např. skrýt, pokud není akutuálně vybrán žadný relevantní viewmodel. Tento přístup poskytuje jistou míru autonomního chování jednotlivým částem aplikace a usnadňuje implementaci nových pohledů bez nutnosti jakéhokoliv zásahu do jejího jádra.

K zajištění dostatečné interaktivity aplikace by měly být všechny časově náročné, nebo potencionálně časově náročné operace v aplikaci spouštěny asynchronně k běhu hlavního vlákna aplikace. Tyto případy lze rozdělit do dvou skupin. Jsou to jednak spravované úlohy - jedná se o výpočetně náročné úlohy, které mohou v závislosti na velikosti vstupních dat běžet velmi dlouho (v převážné většině operace s frameworkem). Tyto úlohy by měly být

spouštěny pomocí správce úloh - viz. 4.6.4 Správa úloh v aplikaci. Je u nich možno sledovat průběh operace a operaci případně zrušit. Druhým případem jsou potencionálně časově náročné operace, které vznikají během běžné interakce uživatele s aplikací. Pro tyto operace lze použít nějaký jednodušší mechanismus (s nižší režií) např. Task v .NET frameworku. Jejich běh by měl být pro uživatele transparentní.

4.6.2 Komunikace v aplikaci

V aplikaci probíhá komunikace jak horizontálně (v dané vrstvě), tak vertikáně (mezi vrstvami). Mezi třídami na odpovídající si vrstvě neexistují reference. Jak již bylo zmíněno, komunikace zde probíhá pomocí tzv. *messagingu*. Třídy, které mají zájem o odběr daných typů zpráv, se při vytváření zaregistrují. Odeslanou zprávu pak dostanou všechny zaregistrované třídy. Jednotlivé zprávy jsou rozlišeny pomocí typů a tagů, ale nesou také přímo užitečná data, jako je například reference na objekt, kterého se zpráva týká, nebo popis události. Směrem dolů třídy komunikují pomocí volání metod, směrem nahoru pomocí událostí. Zasílání zpráv lze ale využít i v některých případech komunikace mezi vrstvami - odpadá pak složité a chybové udržování referencí - obr. 4.12.



Obrázek 4.12: Návrh komunikace v aplikaci

Zasílání zpráv by mělo probíhat vždy v hlavním vlákně aplikace. Toto zajistí minimalizaci konfiktů při přistupu k objektům u jednotlivých příjemců. Všechny časově náročné operace vyvolané příjmem zprávy by ale následně měly být spouštěny opět asynchronně.

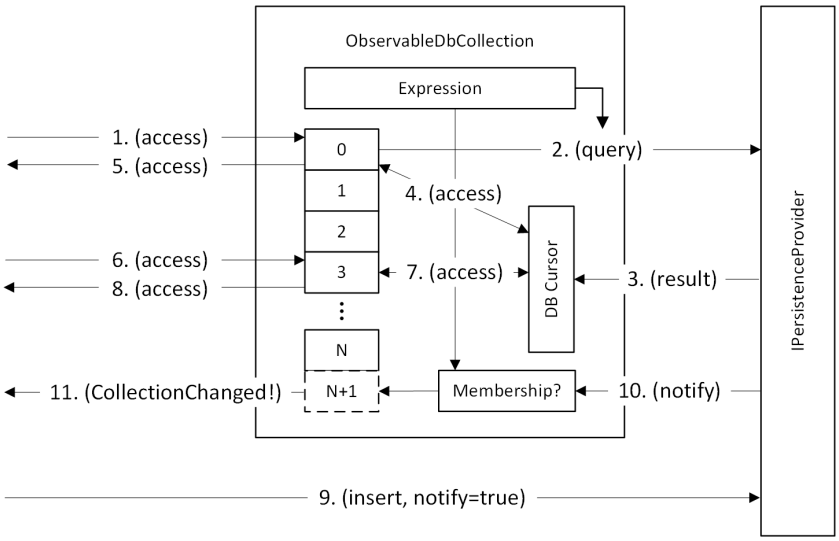
4.6.3 Databázové kolekce

Požadavky na škálovatelnost při použití aplikace kladou nároky zejména na reprezentaci jejího datového modelu. Pokud by měly být před začátkem práce načítány všechny potřebné objekty ve stromu datového modelu do nejnižší úrovně, jednalo by se o nepřijatelně dlouhé odezvy a paměťové nároky. Při návrhu bylo tedy rozhodnuto o použití tzv. *Lazy loading*[4] mechanismu.

Jednotlivé modely, které z pohledu návrhu datového modelu obsahují kolekce jiných modelů, používají tento princip. Pro danou kolekci obsahují instanci speciální navržené třídy *ObservableDbCollection*. Obsah této třídy je definován pomocí typu objektu a podmínek, které musí objekty splňovat - *Expression* (dotaz). Dále lze specifikovat další parametry jako je např. řazení. Díky tomuto po vytvoření kolekce obsahuje pouze zmíněné metainformace. Vlastní obsah nemusí být načten do paměti až do doby, kdy bude skutečně potřeba.

Přístup k položkám kolekce je možný pomocí enumerátoru, nebo asociativně pomocí indexu. Při přístupu k první položce kolekce je proveden vlastní dotaz na databázi. Databáze

vrací počet položek a kurzor pro přístup k položkám - viz. *IPersistenceProvider*. Výhodou je, že u většiny objektových databází v tomto okamžiku ještě nedochází k deserializaci jednotlivých objektů. Na funkčnost mechanismu ale nemá vliv ani případ kdy jsou objekty deserializovány přímo. Tak jak vyšší vrstvy aplikace přistupují k jednotlivým položkám, plní se v kolekci vnitřní tabulka referencemi na deserializované objekty. V případě příštího přístupu je tedy vrácena instance z paměti aplikace - obr. 4.13.



Obrázek 4.13: Životní cyklus databázové kolekce

Kolekce také dynamicky reagují na změny v databázi - viz. 4.3.2. Při změně v databázi tedy není nutné celou kolekci načítat znovu. Pokud je kolekce aktivní (je používána), automaticky se uplatní následující mechanismus. Při příchodu události informující o změně kolekce zkontroluje, jestli jí změněný objekt náleží - má do ní být zařazen nebo z ní odstraněn a podobně. Pokud ano, pak je objekt zařazen nebo odstraněn z/na příslušnou pozici v tabulce referencí. Změna je dále propagována výše pomocí standardních událostí rozhraní *INotifyCollectionChanged* - obr 4.13, obr. 4.7.

4.6.4 Správa úloh v aplikaci

Jak již bylo zmíněno v kapitole 4.6.1 Základní principy aplikační logiky, do jádra aplikace byl navržen mechanismus pro práci s tzv. spravovanými úlohami. Motivací k jeho návrhu byly některé vlastnosti, které například nativní systém úloh v .NET frameworku sám o sobě neposkytuje, zejména možnost snadného reportování průběhu operací a jejich rušení.

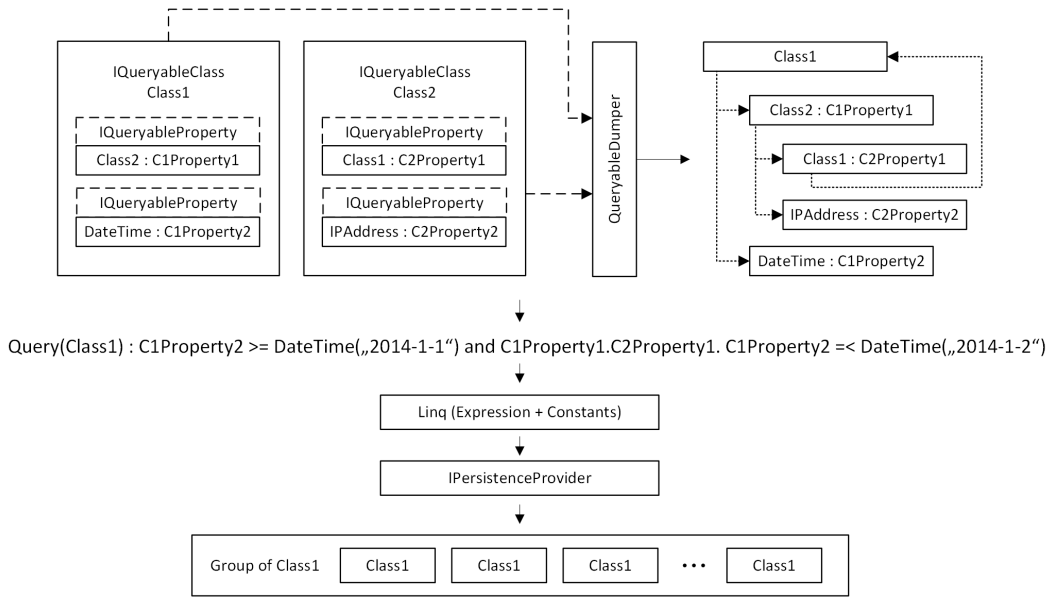
Pokud je třeba v určitém okamžiku běhu aplikace spusit operaci jako spravovanou úlohu, je z globální instance správce úloh (*BgTaskManager*) získána instance třídy reprezentující novou úlohu. Tato třída v sobě nese informace o úloze - její stav, průběh a uživatelské argumenty použitelné při spuštění. Úloha je poté spuštěna jako samostané vlákno. Vstupním bodem je libovolná metoda s argumentem, kterým je třída reprezentující úlohu. Parametry úlohy lze během provádění operace měnit (aktualní stav, cílová hodnota). Díky tomu, že všechny úlohy jsou registrovány ve správci, je vyšším vrstvám aplikace k dispozici jejich přehled. Správce také automaticky kontroluje stav úlohy. Pokud dojde při jejím provádění k chybě, ze které se nezotaví, je toto reflektováno v jejím stavu a uživatel může na tuto skutečnost reagovat.

4.6.5 Dotazování

Aplikace předpokládá analytický způsob práce. Kromě manuálního procházení a případného filtrování dat, ať už na úrovni souborů, konverzací, nebo exportovaných výsledků, byl pro aplikaci navržen systém umožňující dotazování na jednotlivé typy objektů způsobem obdobným dotazování na databázi. Efektivní provedení toho mechanismu zajišťuje právě použití databáze jakožto uložiště.

Při návrhu systému bylo nutné zohlednit zejména dynamičnost datového modelu. Uživateli bude umožněno dotazovat se na obsah objektů skrze jejich jednotlivé vlastnosti (*properties*). Vzhledem k tomu, že nové typy objektů budou do aplikace začleňovány postupně, byl navržen následující mechanismus, který umožní automaticky nabídnout uživateli pro daný typ objektu všechny jeho dostupné vlastnosti.

Třídy datového modelu, které jsou uživatelům k dispozici pro dotazování, jsou označeny rozhraním *IQueryableClass*. Jednotlivé persistentní vlastnosti jsou označeny atributem *IQueryableProperty*. Průchodem přes jednotlivé položky takovéto třídy lze získat strom těch, na které se lze dotazovat - obr. 4.14. U tříd je nutné také zajistit možnost rekurzivního vnořování jednotlivých typů.



Obrázek 4.14: Podpora pro dotazování a dotazování v aplikaci

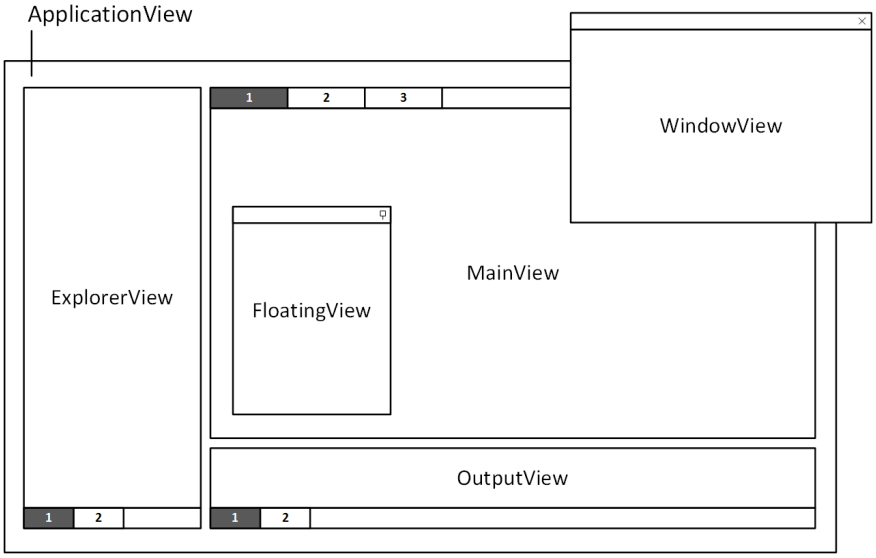
Uživatel zapíše dotaz ve formě řetězce. Řetězec je poté převeden na výraz (expression) a zadané konstanty podle datového typu na odpovídající typy objektů. Toto a typ objektu je poté vstupem do metody *Query()* z *IPersistenceProvider*. Výstup dotazu tvoří podle volby uživatele nová nebo stavající skupina (konverzací, výsledků a podobně).

Hlavní výhodou dotazování oproti filtrování, které je implicitně přítomno, je možnost agregace. Toto umožní získat zájmová data například napříč různými soubory nebo výsledky exportu.

4.6.6 Prezentační vrstva aplikace

Prezentační vrstva aplikace je navržena jako množina jednotlivých pohledů (*Views*). Jedná se o několik typů, které jsou částečně přizpůsobeny jejich předpokládanému použití. Roz-

dělení ale není striktní a v opodstatněných případech může sloužit jakýkoliv typ pohledu jako jiný - obr 4.15.



Obrázek 4.15: Základní typy pohledů v aplikaci

Jedná se o prohlížeče obsahu (*ExplorerView*), ty by měly formou stromové struktury nabízet uživateli k výběru základní prvky zkoumaného modelu. Vlastní obsah by měl být zobrazován ve formě hlavních pohledů (*MainView*). Kromě těchto dvou základních typů lze využít výstupní pohledy (*OutputView*), které vytvářejí obdobu terminálového výstupu. Tento typ pohledu je určen pro různé výpisy logů a jiných hlašení během spuštěných operací. Pro zobrazení detailních vlastností nebo doplňujících informací jsou k dispozici plovoucí pohledy (*FloatingView*), které jsou odlehčenou obdobou oken. K dipozici jsou také pohledy v podobě samostatných oken (*WindowView*), jejich použití může ale vést na nepřehledné prostředí, a proto byly navrženy pouze jako doplňkové.

Základní typy pohledů jsou podle uživatelského nastavení vytvořeny po startu aplikace. Jejich zobrazování, skrývání či případná aktivace je následně plně autonomní. Pohledy „naslouchají“ jednotlivým notifikacím a v případě potřeby na ně reagují. Pro pohledy na exportovaná data existuje korespondující množina typů pohledů. Tyto pohledy jsou do aplikace začleňovány automaticky, na základě jejich přítomnosti v aplikačních knihovnách. Každý z typů pohledů lze snadno použít i při vytváření pluginů. Pro tyto účely je k dispozci zjednodušené rozhraní které toto umožňuje - viz 4.6.7.

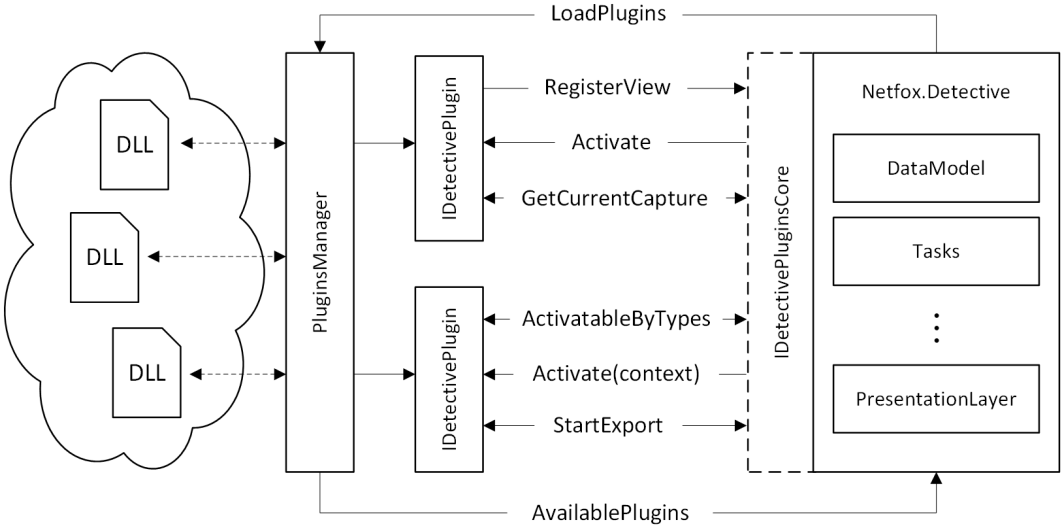
4.6.7 Externí rozšíření aplikace - pluginy

Možnosti řozšiřování aplikace lze rozdělit do dvou skupin. První z nich tvoří již popsaná dynamičnost datového modelu, odpovídajících pohledů a podobně. Toto umožňuje například snadno rozšiřovat množinu podporovaných protokolů a jiné. Základní kostra a princip fungování je zde pevně daný. To na jednu stranu posktyuje zjednodušení a zrychlení, na druhé straně ale ve specifických případech nemusí poskytnout dostatečnou variabilitu. Z tohoto důvodu byla do návrhu aplikace přidána také možnost implementovat jakékoli jiné rozšíření jakožto externí plugin.

O zavádění extreních knihoven obsahující pluginy se stará správce pluginů (*PluginsManager*). Po zavedení zadaných knihoven do aplikační domény provede jejich prohledání na třídy implementující předepsané rozhraní pluginu *IDetectivePlugin*. Třídy pluginů jsou instanciovány, a poté inicializovány pomocí příslušné metody z rozhaní. Takto také plugin získá reference na třídu implementující *IDetectivePluginsCore* - viz. níže. V tomto stavu by plugin neměl provádět žádné akce do doby, než bude tzv. aktivován.

Plugin je možné aktivovat dvojím způsobem - globálně *Activate()*, nebo kontextově na daný objekt *Activate(object context)*. O tom, jestli je možné plugin aktivovat globálně, připadně na které typy objektů (modelů) informuje skrze své rozhraní (*IsGlobalActivatable*, *ActivatableByTypes[]*). Jádro aplikace také informuje plugin v případě, že jsou skryty všechny jeho pohledy pomocí metody *Deactivate()*. Plugin tak může například přestat provádět průběžné výpočty na pozadí a šetřit tak výpočetní výkon.

Pluginy jsou organizovány ve stromové struktuře. Každý plugin hlasí cestu názvů skupin, do které má být zařazen. Na základě těchto cest je poté vytvořen jednak globální strom těchto skupin, který je v aplikaci k dispozci pro globální aktivaci pluginů, a také v případě potřeby další stromy pro jednotlivé modely, na které lze pluginy aktivovat.



Obrázek 4.16: Podpora pluginů v aplikaci

Interakci pluginů s aplikací zajišťuje třída implementující rozhraní *IDetectivePluginsCore*. Teoreticky by stačilo, kdyby poskytovala reference na základní správce (sub-shelly) - viz. 4.6.1. Pro zjednodušení ale nabízí množinu událostí a metod, které umožní interakci s aplikací bez nutnosti znalosti její vnitřní struktury. Pomocí tohoto rozhraní lze také registrovat jednotlivé pohledy, které plugin implementuje. Lze zvolit, o jaký typ pohledu se má jednat - viz. 4.6.6. Zobrazování, skrývání a aktivace je pak již plně v režii pluginu. Za standardních okolností by ale měla korespondovat s aktivacemi pluginu a dalšími uživatelskými interakcemi - obr. 4.16.

Kapitola 5

Implementace nástroje

Na základě popsaných návrhů a řešení problémů byl implementován nástroj pro analýzu obsahu síťové komunikace Netfox Detective. V této kapitole jsou popsány základní použité technologie a některé zajímavé implementační detaily.

5.1 Použité technologie

Nástroj je implementován v jazyce C# za použití .NET frameworku jako tzv. Windows Presentation Foundation (WPF) aplikace. Pro implementaci messagingu je použit *MVVM Light Toolkit*[13]. Prezentační vrstva používá zejména WPF komponenty Telerik[18] a Infragistics[11]. Pro prohlížení webových dokumentů je použito jádro Chromium (framework CefSharp). Pro implementaci dvou uvedených typů uložišť jsou použity konektory RavenDB 2.5 (Client a Embedded) a MongoDB 1.7 CSharp Driver.

5.2 Implementace návrhu

Při implementaci byl kladen důraz na dodržení MVVM architektury. Jednotlivé části aplikace tak, jak jsou popsány v kapitole 4.4 a na obr. 4.8, tvoří knihovní *assemblies*. Aplikační *assembly* poté obsahuje pouze vytvoření *ApplicationView* a *ApplicationShell* - viz. 4.6.1, vše ostatní poté probíhá v režii tohoto aplikačního shellu.

Grafické rozhraní používá přímou vazbu na obsah objektů - tzv. *WPF binding*. Toto zajišťuje neustálou konzistenci mezi datovým modelem, jeho viewmodelem a daty zobrazenými uživateli. Veškerá interakce uživatelů z grafického rozhraní s aplikační logikou probíhá pomocí příkazů - *ICommand*. Při jejich použití je možné využít unit testů pro rychlejší a bezpečnější vývoj. Zároveň také tvoří bariéru mezi prezetační vrstvou a aplikační logikou a brání tak jejímu nechtěnému přenosu pravě do prezentační vrstvy aplikace.

Výjimky, které vznikají za běhu aplikace v nižších vrstvách (včetně Netfox frameworku), jsou zpracovávány modely objektů implmentující aplikační logiku. Výjimka je zachycena a zpráva o jejím výskytu je rozeslána pomocí *messagingu* jako speciální typ zprávy tzv. *system message* ostatním komponentám aplikace. Tyto zprávy mohou být generovány také během korektního běhu aplikce. Obsahují identifikaci odesílající komponenty - např. správa projektů, typ zprávy (info, varování, chyba a pod.) a obsah - např. popis výjimky. Podle typu a nastavení uživatele jsou pak různými komponentami zprávy zpracovány, například všechny uloženy aplikačním shellem do logu a varování nebo chyby zobrazeny aplikačním *view* v dialogovém okně.

Návrh datového modelu a persistenční vrstvy používá koncept *lazy evaluation*. Ostatní procesy v aplikaci toto reflektují. Objekty jsou vytvářeny a výpočty probíhají až ve chvíli, kdy jsou jejich výsledky skutečně potřeba. Tento přístup přináší rychlejší začátek práce, ale na druhou stranu může také snižovat interaktivitu (prodlužovat odezvy během práce). Pro dosažení kompromisu mezi oběma přístupy byl v aplikaci implementovám mechanismus, který na pozadí (mimo hlavní vlákno) postupně spouští akce, jejichž výpočet vyžaduje relativně delší dobu (např. seřazení listu konverzací). Uživatel tedy např. po otevření projektu nemusí čekat, než budou všechny tyto výpočty provedeny, během nečinnosti jsou ale tato data postupně předpočítána. Poté záleží, kdy uživatel začne práci např. s daným seznamem - buď již byl výpočet proveden na pozadí, nebo je proveden právě ve chvíli, kdy je potřeba.

Tam, kde je to možné, je použito asynchroní (*Task*) a paralelní zpracování tak, aby co nejméně výpočtů probíhalo v hlavním (GUI) vlákně. Z tohoto důvodu jsou nahrazeny standardní *ObservableCollection* jejich reentrantními obdobami pro použití s WPF (používají *Dispatcher* pro vyvolání událostí). Obsah těchto kolekcí tak může být aktualizován výpočtem na pozadí a hlavní vlákno je zatíženo pouze zobrazením přehledu jejich obsahu[5].

5.3 Zajímavé detaily

5.3.1 Výkonnost databázových uložišť

Během implementace byly pozorovány výrazné výkyvy ve výkonnosti uložišť - jak mezi jednotlivými typy, tak při různých typech dotazů. Celkově výrazně lepších výsledků dosahovala databáze MongoDB. U některých typů dotazů ale naopak poskytovala rychlejší odezvu databáze RavenDB. Bylo zjištěno, že se jedná o vlastnost RavenDB týkající se indexů. Tato databáze nevytváří automaticky pouze index přes položky identifikátoru, ale také tzv. dynamické indexy[10] u položek, na které jsou prováděny dotazy. Návrh rozhraní pro persistenci dat (*IPersistenceProvider*) byl tedy doplněn o možnost vytvoření indexu pro daný typ objektu a jeho vlastnosti v databázi (*EnsureIndex()*). Po definování potřebných indexů pomocí této metody bylo u MongoDB dosaženo předpokládané výkonnosti.

5.3.2 Prezentační vrstva - cache objektů

Implementovaný nástroj velké množství dat nejen zpracovává, ale také zobrazuje. Zvýšené nároky jsou tedy kromě datového modelu kladeny na prezentační vrstvu a vrstvu viewmodelů. Většina komponent pro WPF umožňuje použít tzv. UI Virtualization. Toto zajistí, že v GUI aplikace je vytvářena a vykreslována pouze podmnožina entit, která je uživateli právě viditelná, a nedojde tak k explozi množství objektů v paměti. Právě v tomto případě a také v případě, kdy uživatel například přepíná tam a zpět mezi více entitami, dochází k opakovanému předzpracovávání dat ve viewmodelech pro jednotlivé pohledy (views). K opakovanému výpočtu stejných dat dokonce dochází podle zjištění vícenásobně i v rámci jedno zobrazení daného pohledu. Nabízí se možnost jednou vypočítáné výsledky pro příště uložit. V tomto případě by ale postupem času neúnosně narůstalo množství paměti spotřebované aplikací. Při implementaci byl použit princip obecně uplatnitelný v obdobných případech, kterým je cachování.

Pro tyto účely byla vytvořena generická třída *TimeLimitedCache*. Její základ tvoří asociativní kontejner, k jehož obsahu je možno přistupovat podle zvoleného typu klíče (předpokládaným typem je enum). Je možný tzv. synchronní nebo asynchronní přístup. V obou případech je buď vrácen nacachovaný objekt, nebo je invokována uživatelem zadaná funkce,

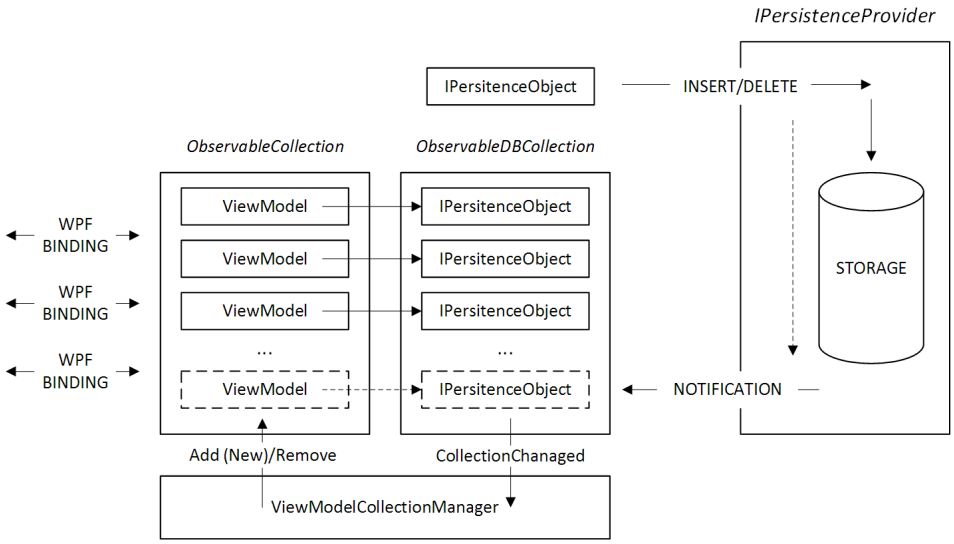
která provede jeho vytvoření (výpočet). Pokud nebyl objekt požadován po delší než nastavenou dobu, je reference na něj zahozena a v případě, že ho již nepoužívá nikdo jiný může být uvolněn z paměti. Při implementaci byl kladen důraz na co nejmenší paměťové a výpočetní prostředky (prostředky OS) a také na možnost kongruentního použití.

Použitím tohoto mechanismu společně s asynchronním výpočtem lze dosáhnout další zajímavé funkcionality. Pokud je například zobrazován pohled, u kterého jsou téměř všechna data ve vlastnostech k dispozici až na výjimku, která vyžaduje delší výpočet, je zobrazení blokováno do té doby, než je dokončen. Pokud je ale výpočet spuštěn asynchronně a je prozatím vrácena prázdná hodnota, je pohled zobrazen až na chybějící data, která jsou automaticky doplněna po dokončení asynchroního výpočtu.

5.3.3 Vytváření ViewModelů

Jak již bylo popsáno dříve, pro každý z kolekce datových modelů musí v aplikaci existovat odpovídající (obalující) viewmodel. Existuje řada způsobů, jakými se viewmodely v MVVM aplikacích vytvářejí - např. pokaždé znovu, když jsou potřeba. Vzhledem k množství a způsobu, jakým v navržené aplikaci vznikají a zanikají datové modely, byl zvolen vlastní dynamický způsob vytváření a rušení viewmodelů.

Systém pracuje nad databázovými kolekcemi *ObservableDBCollection* (popsány v kapitole o návrhu persitence dat). Využívá toho, že kolekce emituje události, pomocí kterých lze sledovat změny v jejím obsahu. Lze nad ní tedy vytvořit odpovídající kolekci viewmodelů (třída *ViewModelsObservableCollection*), která sleduje popsané změny a dynamicky vytváří a ruší odpovídající viewmodely. Implementace využívá podobné mechanismy jako *ObservableDBCollection* - lazy loading pomocí enumerátorů. Díky tomuto je zajištěno zachování řetězece tohoto mechanismu - od prezentační vrstvy (UI Virtualization), přes viewmodely a modely (*lazy loading*) až po persistenční. Dotaz je tedy nutno provést až ve chvíli, kdy mají být objekty skutečně zobrazeny v GUI - obr. 5.1.



Obrázek 5.1: Dynamické vytváření a rušení viewmodelů

Tento způsob také zajistí, že nedojde k nekonzistenci mezi kolekcemi viewmodelů, nad kterými pracují vyšší vrstvy aplikace, a zároveň sníží režii, která vzniká v případě jejich opětovného vytváření.

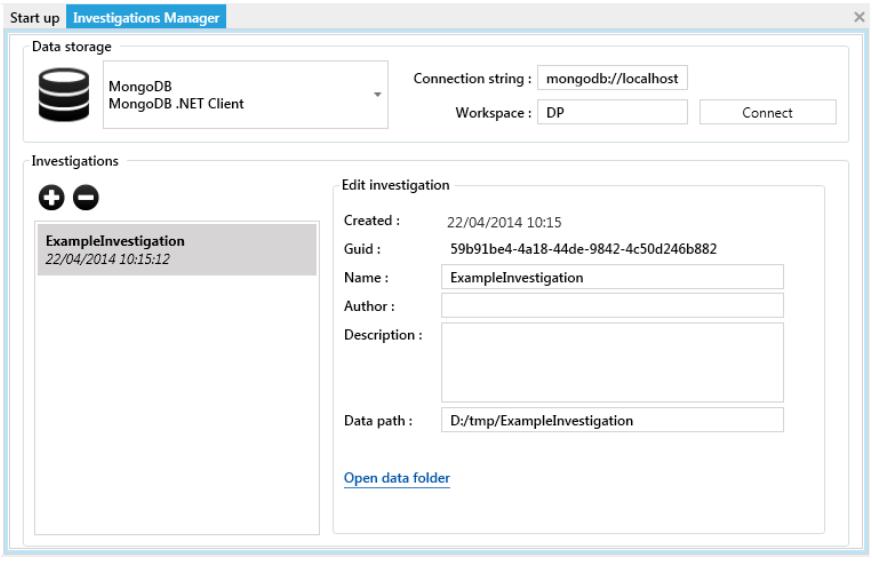
Kapitola 6

Netfox Detective

V této kapitole bude popsán nástroj pro analýzu obsahu síťové komunikace Netfox Detective vytvořený v rámci této práce. Základní možnosti, které nabízí, a způsob práce zde budou ilustrovány na příkladech použití.

6.1 Datová uložiště, správa sezení

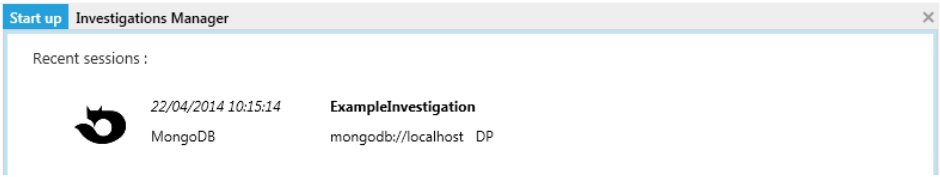
Práce s nástrojem probíhá vždy v rámci daného uložiště a v rámci daného projektu (*investigation*) - viz. 4.5 Datový model. Tato kombinace - datové uložiště a investigation tvoří sezení (*session*). K danému uložišti se lze připojit v hlavním pohledu Investigation manageru - obr. 6.1.



Obrázek 6.1: Investigations Manager

Pro připojení je nutné zadat connection string (ve většině případů adresa serveru nebo cesta v souborovém systému) a u uložišť, která to umožňují, také workspace (například název databáze). Po připojení je k dispozici seznam jednotlivých investigation, která lze vytvářet upravovat a rušit. Každé investigation má kromě názvu popisu také GUID, které ho jednoznačně identifikuje, a cestu v souborovém systému, kam budou ukládány všechny potřebné soubory (například při exportu).

Pro rychlejší začátek práce, aplikace uchovává seznam posledních aktivovaných session a použitých uložišť. Ty lze poté snadno vyvolat po startu aplikace - obr. 6.2. Kromě tohoto je k dispozici také průvodce, pomocí kterého se lze snadno připojit k uložišti a vytvořit v něm nový projekt.

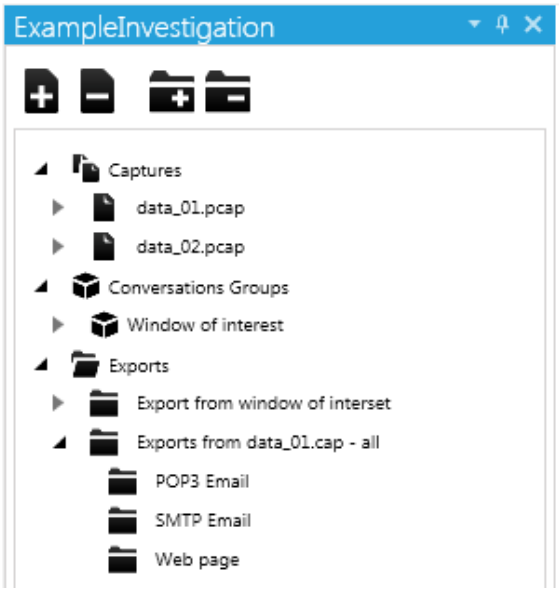


Obrázek 6.2: Seznam naposledny použitých session

Po výběru nebo vytvoření investigation jedním z výše popsaných postupů dojde k jeho aktivaci. Od této chvíle budou všechny akce provedené v nástroji (přidání souboru, export aplikačních dat, dotazování, vyhledávání) probíhat právě v rámci něj.

6.2 Investigation

Obsah investigation lze rozdělit do tří základních skupin. Jedná se o soubory se zachycenou komunikací, skupiny konverzací a výsledky exportu umístěné ve skupinách - obr. 6.3.



Obrázek 6.3: Prohlížeč investigation

Soubory capture

V rámci investigation lze pracovat s vícero soubory se zachycenou komunikací. Podpora formátů vychází z knihovny frameworku *PmLib*. Jedná se o všechny dnes běžně používané formáty : *PCAP*, *PCAP-NG* a *MNM CAP*. Soubory lze do investigation přidávat nebo odebírat například pomocí prohlížeče investigation.

Skupiny konverzací

Kromě souborů s komunikací obsahuje investigation také skupiny konverzací - *Conversation group*. Tyto skupiny vznikají na základě dotazování. Skupina obsahuje konverzace z různých souborů obsažených v projektu. Její obsah je tedy výsledkem kombinace agregace a filtrace. Se skupinou konverzací lze pracovat stejným způsobem jako se soubory.

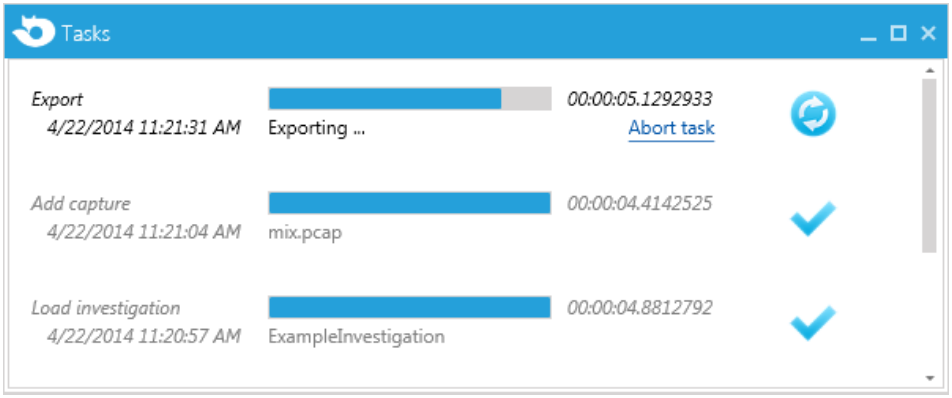
Výsledky exportu a jejich skupiny

Po spuštění exportu aplikačních dat vznikají v investigation jeho výsledky. Ty jsou organizovány do skupin, jejichž struktura odpovídá adresářové struktuře ze souborových systémů. Skupiny jsou vytvářeny jak automaticky (během operací), tak uživatelem - například pomocí prohlížeče investigation. Skupiny a výsledky mohou vznikat také na základě dotazování.

Pomocí prohlížeče investgation lze vybrat každou z výše uvedených entit. Ta se poté stavá aktivní v celé aplikaci a jsou tak například zobrazeny příslušné pohledy na vybraná data.

6.3 Správa úloh

Veškeré operace v nástroji probíhají na pozadí ve formě úloh. Přehled právě probíhajících i dokončených úloh lze získat ve správci úloh (View > Tasks) - obr. 6.4.



Obrázek 6.4: Správce úloh

Každá úloha má název a popis, je u ní také zobrazen průběh a doba trvání. Probíhající úlohu lze zrušit pomocí „Abort task“.

6.4 Práce se soubory capture a skupinami konverzací

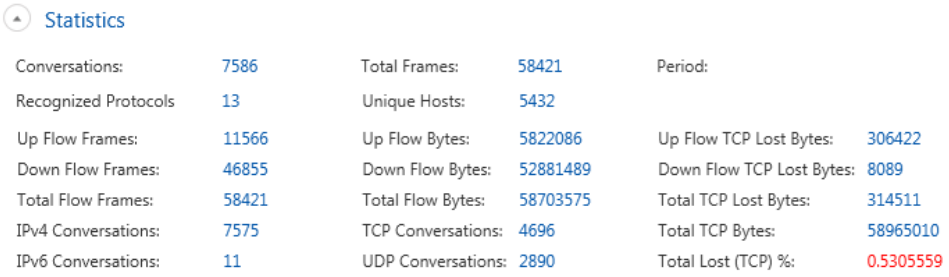
Po přidání souboru capture je jeho obsah automaticky rozdělen do konverzací. S takto rozděleným souborem, nebo po vytvoření skupiny konverzací pomocí dotazu, lze pracovat s jejich obsahem. Jsou k dispozici dva základní pohledy na obsah. První z nich je přehledový, druhý zobrazuje vlastní obsah.

Přehled o obsahu

U souboru je k dispozici kontrolní součet (SHA1) jeho obsahu. Po každém načtení investigation je zkontrolován uložený součet s aktuálním, aby bylo zajištěno, že mezi jednotlivými spuštěními nedošlo k záměně nebo poškození souboru - obr. 6.5.

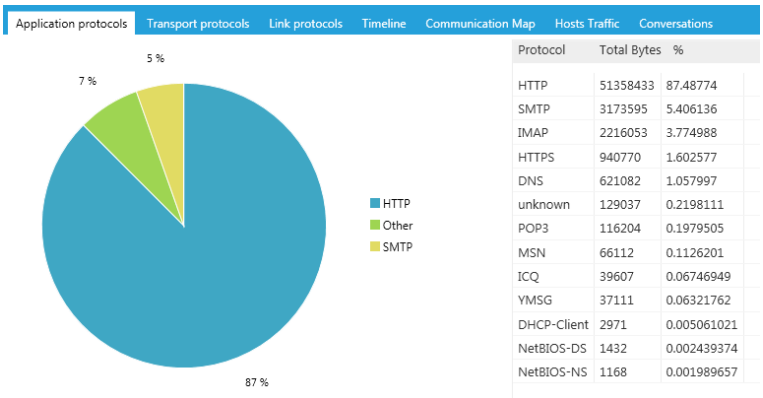
Obrázek 6.5: Kontrolení součet u souboru

Pro soubor nebo skupinu konverzací lze zobrazit základní statistické údaje, jako jsou počty přenesných rámců a bajtů, počty konverzací a podobně. Pro konverzace s transportní vrstvou TCP je k dispozici také množství přenesených aplikačních dat a množství chybějících dat - obr. 6.6.

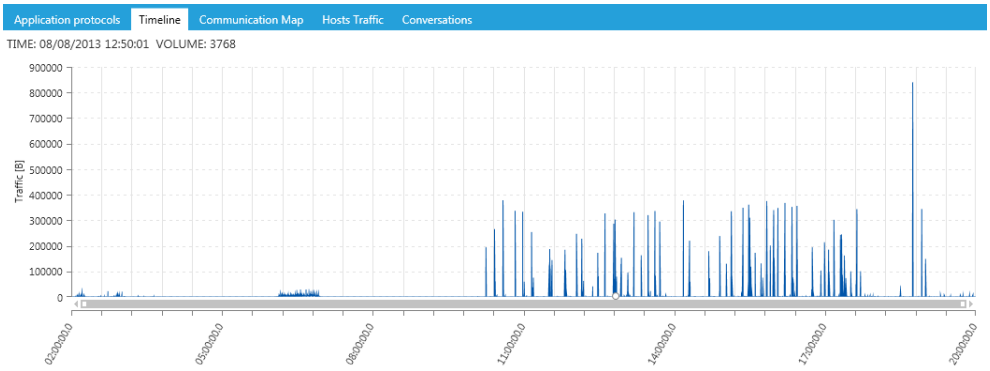


Obrázek 6.6: Statistiky konverzací

Dále jsou k dizpozici další přehledy, převážně v podobně grafů. Jedná se například o distribuci jednotlivých protokolů (na různých vrstvách) - obr. 6.7, přenesná data v čase - obr. 6.8, mapa komunikujících stanic, data přenesná jednotlivými stanicemi, rozložení konverzací vzhledem k počtu přenesených dat a délce trvání a podobně.



Obrázek 6.7: Statistiky konverzací



Obrázek 6.8: Přenesná data v souboru v čase

Obsah

Obsah je rozdělen do tří hlavních pohledů ve formě tabulek. Prvním je přehled jednotlivých konverzací. Je zde k dispozici časový rámec, transportní a aplikační protokol, koncové komunikující body a další statistiky - obr 6.9.

Conversations | Frames | Exports

Drag a column header and drop it here to group by that column

#	First Seen	Last Seen	Transport	Application	Client	Server	Frms Up	Bytes Up	Frms D
15	08/08/2013 02:04:48	08/08/2013 02:13:10	TCP	HTTP	192.168.2.101:49872	178.237.23.235:80	4	2007	4
17	08/08/2013 02:06:09	01/01/0001 00:00:00	UDP	DNS	192.168.2.101:56411	192.168.2.1:53	1	29	1
18	08/08/2013 02:06:11	01/01/0001 00:00:00	TCP	ICQ	192.168.2.101:49873	64.12.239.113:5190	1	569	1

Obrázek 6.9: Přehled konverzací

Obsah souboru lze také zobrazit v podobě jednotlivých rámců, tak jak jsou obasaženy v původním souboru. Tento pohled odpovídá základnímu pohledu výše popsaných nástrojů jako je Wireshark nebo MNM.

Conversations | Frames | Exports

Drag a column header and drop it here to group by that column

#	Time	Source	Target	Protocol	Frame Size
3	08/08/2013 02:00:01	199.47.218.150:80	192.168.2.101:49602	TCP	60
4	08/08/2013 02:00:21	FE80::6CA6:C4F5:8338:82B5:55129	FF02::C:1900	UDP	208
5	08/08/2013 02:00:33	08002736DBAF	000E2EAF1E42	ARP	42

Obrázek 6.10: Přehled rámců

Poslední tabulka zobrazuje výsledky exportu, které vznikly z obsahu souboru nebo skupiny konverzací. Výsledky exportu lze tedy procházet i podle původu dat.

Conversations | Frames | Exports

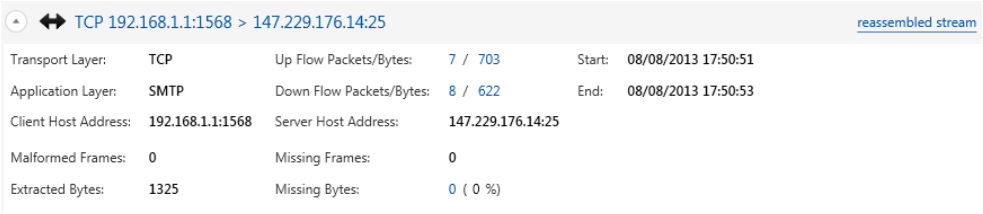
Drag a column header and drop it here to group by that column

Period	Source	Target	Exporter Type	Description
08/08/2013 02:06:43 - 08/08/2013 02:06:43	192.168.2.101:49878	178.237.25.51:80	HttpSleuth	Web page update.icq.com
08/08/2013 02:06:41 - 08/08/2013 02:06:41	192.168.2.101:49872	178.237.23.235:80	HttpSleuth	Web page update.icq.com
08/08/2013 18:44:36 - 08/08/2013 18:44:36	192.168.1.1:1569	147.229.176.14:25	SMTPSleuth	SMTP Email To:<testpc04@seznam.cz> [1]

Obrázek 6.11: Přehled výsledků exportu

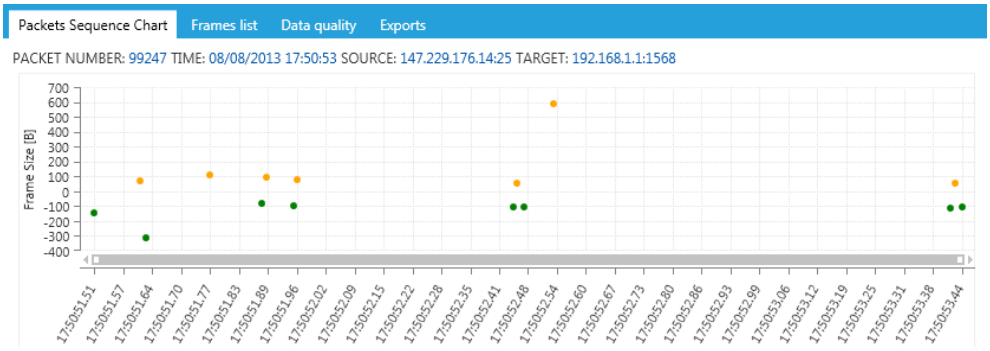
6.5 Práce s konverzacemi

Konverzaci lze zvolit například z prohlížeče projektu, z tabulky zobrazující obsah souboru, nebo jinými způsoby. Základní pohled na konverzaci nabízí statistický přehled obdobný tomu u souborů - obr. 6.12.



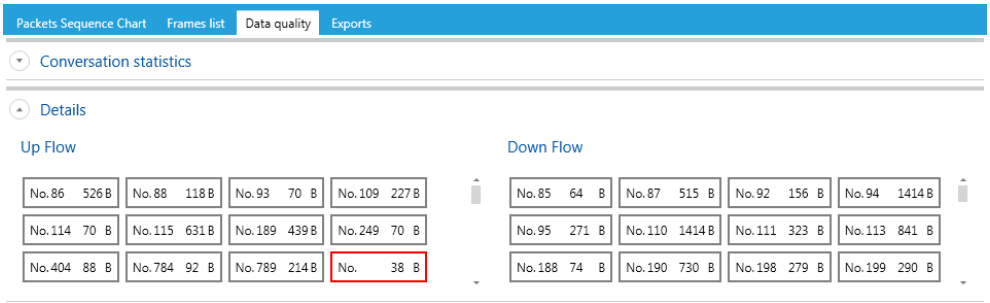
Obrázek 6.12: Základní statistiky konverzace

Kromě tohoto přehledu je k dispozici graf, zobrazující jenotlivé přenesné rámce v čase a jejich velikost (kladná velikost je u paketů příchozích, záporná u odchozích) - obr. 6.13.



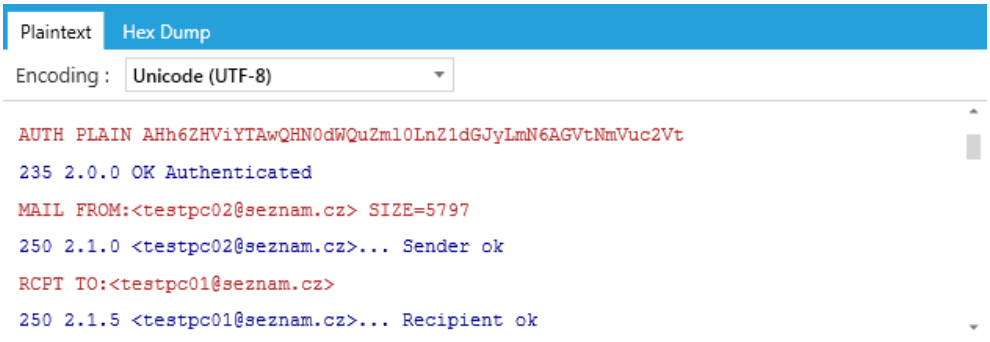
Obrázek 6.13: Jednotlivé rámce konverzace v čase

Další zajímavý pohled, který je zde k dispozici, se zaměřuje na kvalitu zachycených dat. Pro danou konverzaci zobrazuje detekovaná chybějící data, a to ve formě přehledových grafů a také na úrovni jednotlivých rámců - obr. 6.14.



Obrázek 6.14: Jednotlivé rámce konverzace v čase

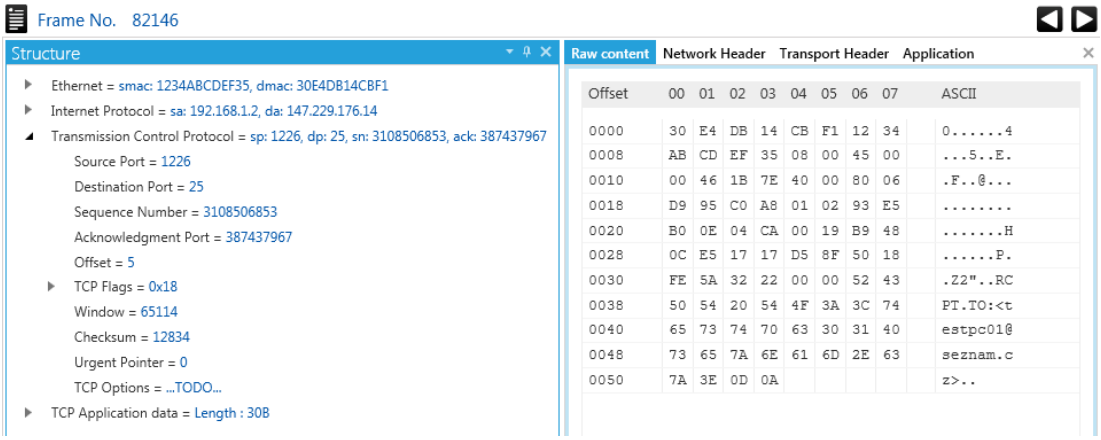
U konverzace lze také zobrazit přehled jednotlivých rámců a výsledků exportu v podobě stejných tabulek, jako u souboru nebo skupiny konverzací. Reassemblovaná aplikační data přenášená v rámci dané konverzace lze zobrazit buď jako plaintext (ve zvolém kódování) nebo jako hexadecimální výpis - obr. 6.15.



Obrázek 6.15: Rassemblovaná aplikační data

6.6 Práce s rámci

Konkrétní rámec lze vybrat z daného souboru, konverzace (z tabluky nebo grafu sekvence rámců), nebo jinými způsoby. Obsah rámce je prezentován způsoby obdobnými jako u jiných nástrojů. K dispozici je strom jednotlivých položek hlaviček až po aplikační vrstvu. Dále lze zobrazit kompletní obsah v podobě haxadecimální tabulky a jednotlivé hlavičky v podobě tabulek - obr. 6.16.



Obrázek 6.16: Zobrazení obsahu rámce

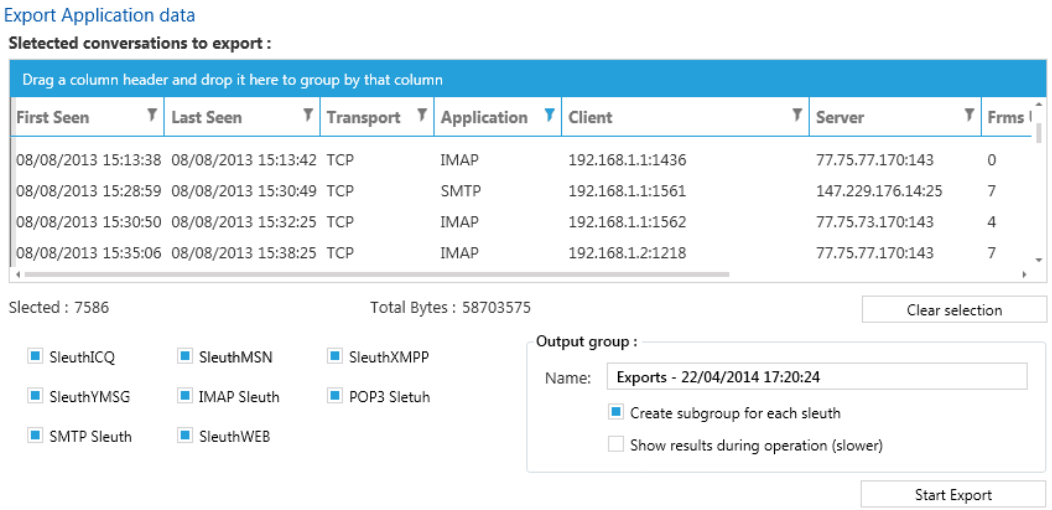
V rámci tohoto pohledu lze také přepínat mezi předchozím a následujícím rámcem pomocí navigačních šipek. Při aktivaci učitého rámce dochází automaticky také k aktivaci příslušné konverzace. Ze zajímavého rámce tak lze snadno přejít přímo k ostatním datům v rámci konverzace.

6.7 Export aplikačních dat

Nad vybranými konverzacemi lze provést analýzu a získání aplikačních dat pomocí jednotlivých exportérů. Konverzace pro export lze do seznamu vybraných konverzací přidávat různými způsoby - například jednotlivě nebo celý soubor nebo skupinu.

Export je prováděn pomocí množiny exportérů. Ty, které mají být použity, je nutno zvolit ze seznamu akutálně dostupných - viz. 4.2.1. Výsledky exportu budou uloženy do

nové skupiny výsledků. Uživatel může zvolit její název. Dále lze zvolit, zdali mají být výsledky z jednotlivých typů exportérů rozřaděny do podskupin, nebo nikoli.



Obrázek 6.17: Zobrazení obsahu rámce

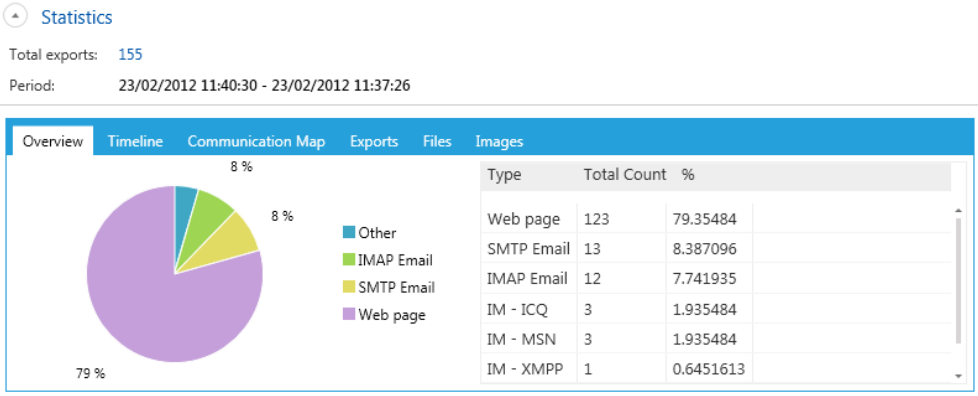
Vlastní export probíhá jako operace a její průběh lze sledovat ve správci úloh, v případě volby lze sledovat i přibývající výsledky v jednotlivých výstupních skupinách. Toto průběžné zobrazování ale snižuje výkonnost a v produkčním prostředí není doporučeno. V opačném případě jsou výsledky zobrazeny až po dokončení operace.

6.8 Práce s výsledky exportu

Práci s výsledky exportu lze rozdělit do dvou oblastí. První z nich je práce se skupinami výsledků, druhá poté práce s vlastními výsledky.

6.8.1 Skupiny výsledků

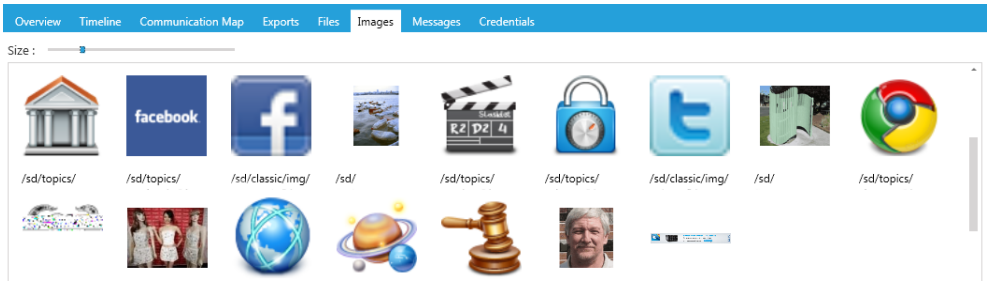
Skupiny výsledků tvoří hierachickou strukturu skupin a jednotlivých výsledků. Každá ze skupin poskytuje agregovaný pohled na veškerý obsah, který je obsažen v jejím podstromu.



Obrázek 6.18: Přehled o skupině výsledků

K dispozici je několik základních pohledů. Jedná se o rozložení nalezených výsledků podle typu, časovou přímku zobrazující nalezené výsledky v čase, mapu komunikujících uzlů a podobně - obr. 6.18. V rámci skupiny lze pracovat i se samotným obsahem. K dispozici je tabulka zobrazující jednotlivé agregované výsledky. Zobrazit lze také například přehled všech získaných dokumentů (souborů) z výsledků, galerii všech získaných obrazků, nebo všechny nalezené zprávy (IM, E-mail a další) - obr. 6.19. Od každého z uvedených typů obsahu lze snadno přejít k samotnému výsledku, ze kterého obsah pochází.

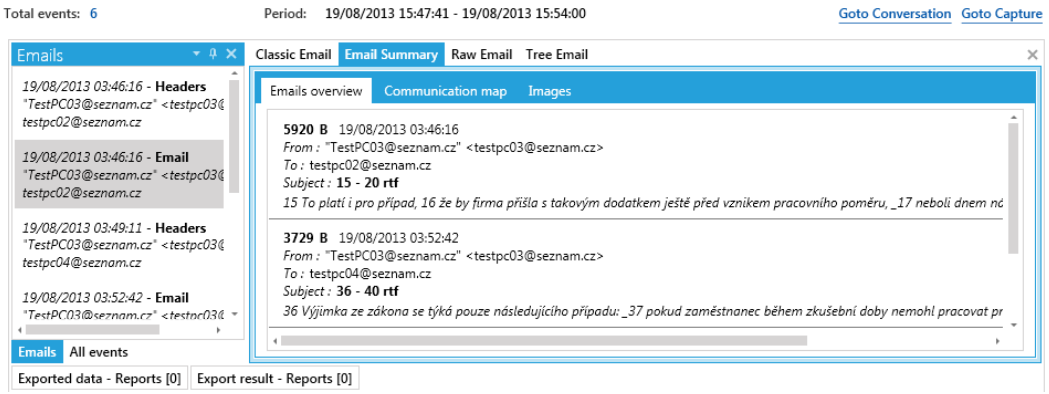
Skupiny může uživatel vytvářet, pojmenovávat a rušit. Jednotlivé skupiny a veškerý jejich obsah lze například ve správci *investigation* snadno přesouvat a vzájemně vnořovat.



Obrázek 6.19: Obrázky ze skupiny výsledků

6.8.2 Výsledky exportu

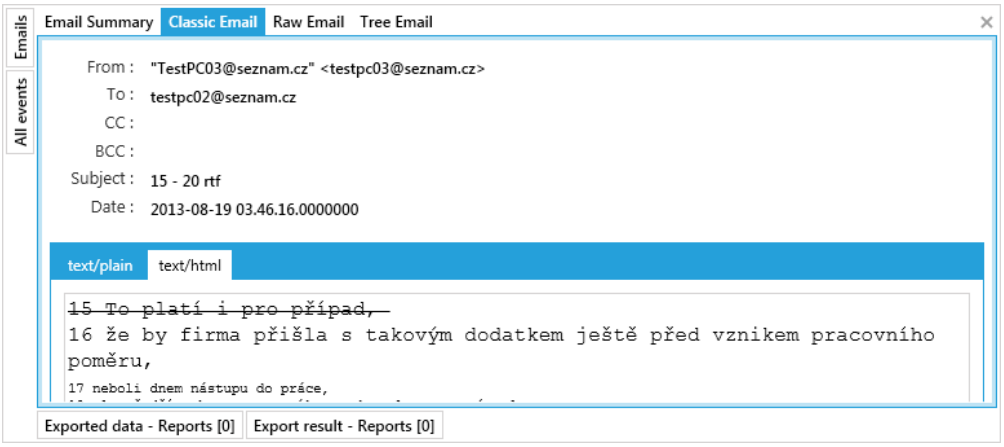
Pro práci s vlastními výsledky exportu je k dispozici prostředí, které koresponduje s globálním prostředím aplikace. K dispozici je série prohlížečů, které zobrazují události obsažené ve výsledku. Může se jednat jak o generické prohlížeče zobrazující různé typy událostí, tak o prohlížeče zaměřené pouze na určitý typ - např. e-maily - obr. 6.20.



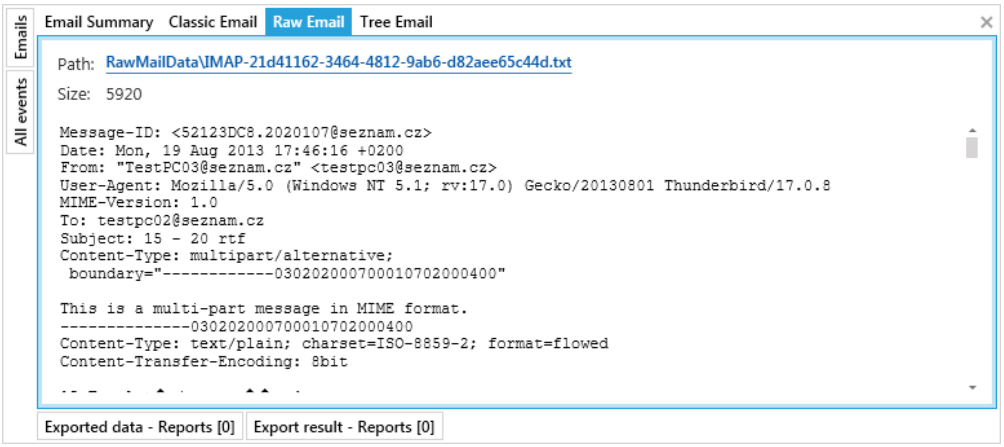
Obrázek 6.20: Výsledek exportu - e-mail

Obsah jednotlivých událostí nebo skupin událostí je zobrazován pomocí specifických pohledů. Pohledy jsou stejně jako prohlížeče implementovány v externích knihovnách. Jádro aplikace automaticky rozpozná dostupné pohledy podle standardního rozhraní. Po výběru dané události jsou zobrazeny právě všechny pohledy, které umožňují zobrazení jejího typu.

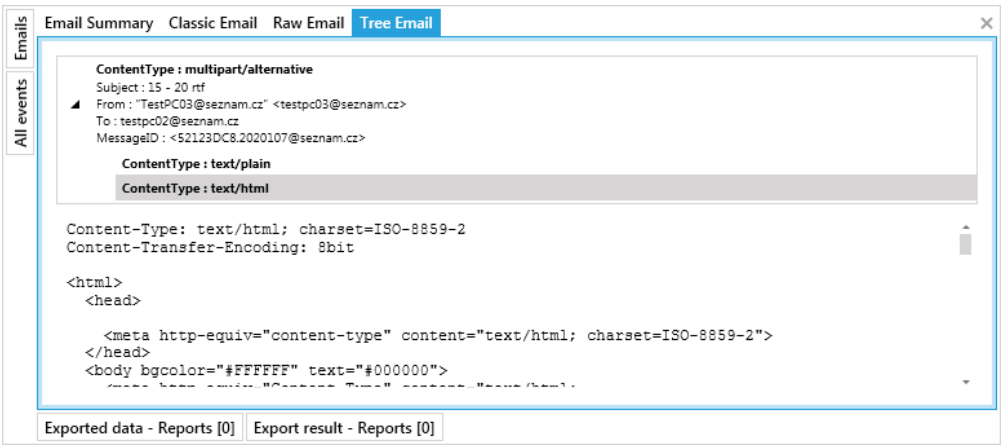
Vybraný e-mail tak může být zobrazen například v podobě obdobné standardnímu e-mailovému klientu - obr. 6.21, jako původní „plaintext“ sestavená data - obr. 6.22 nebo jako strom jednotlivých MIME částí - obr. 6.23.



Obrázek 6.21: Pohled na vybraný e-mail - e-mailový klient



Obrázek 6.22: Pohled na vybraný e-mail - plaintext data



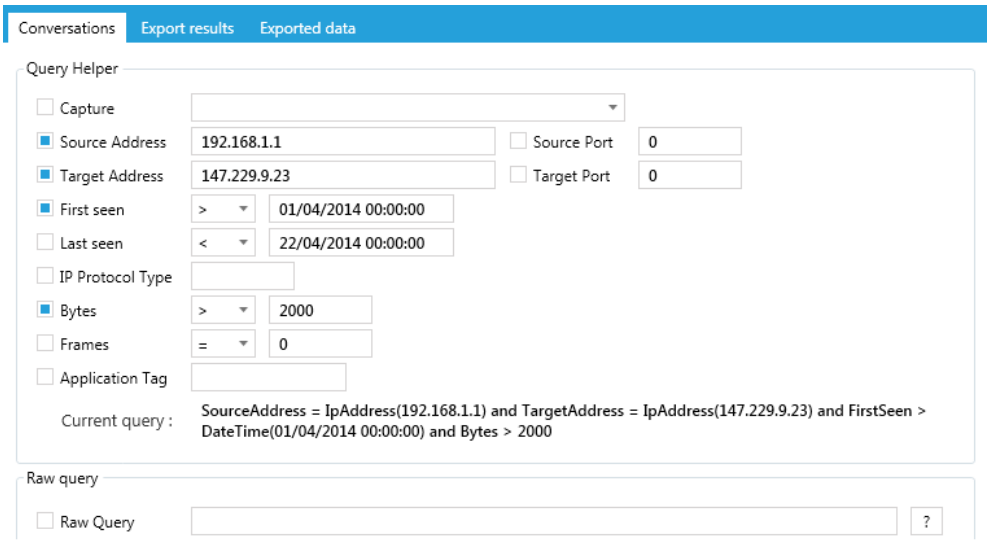
Obrázek 6.23: Pohled na vybraný e-mail - strom MIME částí

Tento přístup nabízí vysokou míru variability v práci se získaným aplikačním obsahem. Aplikace není uzavřena na jeden daný typ pohledu na data, ale lze velmi snadno rozšířit o nový pohled umožňující efektivní analýzu dat specifickým způsobem.

Od analyzovaného výsledku lze snadno (Goto Conversation / Capture - obr. 6.20) přejít ke konverzaci, případně souboru, ze kterého rekonstruovaná data pochází. Toto nabízí výhodu oproti standardnímu způsobu práce, kterou obdobné nástroje nabízejí. Uživatel má ve většině případů dostupné pouze informace o časovém rámci a komunikujících uzlech. V tomto případě má naopak k dispozici kompletní konverzaci a kontext, ve kterém probíhala.

6.9 Dotazování

Jak již bylo zmíněno v předchozích kapitolách, aplikace umožňuje analytický způsob práce pomocí dotazování. Dotazování lze rozdělit do dvou oblastí. Jedná se o dotazování na konverzace a dotazování na výsledky exportu. V obou případech se jedná o téměř identický proces, který se liší pouze vlastnostmi, na které se lze dotazovat, a ve formě výsledku (skupina konverzací / exportní skupina / výsledek exportu).

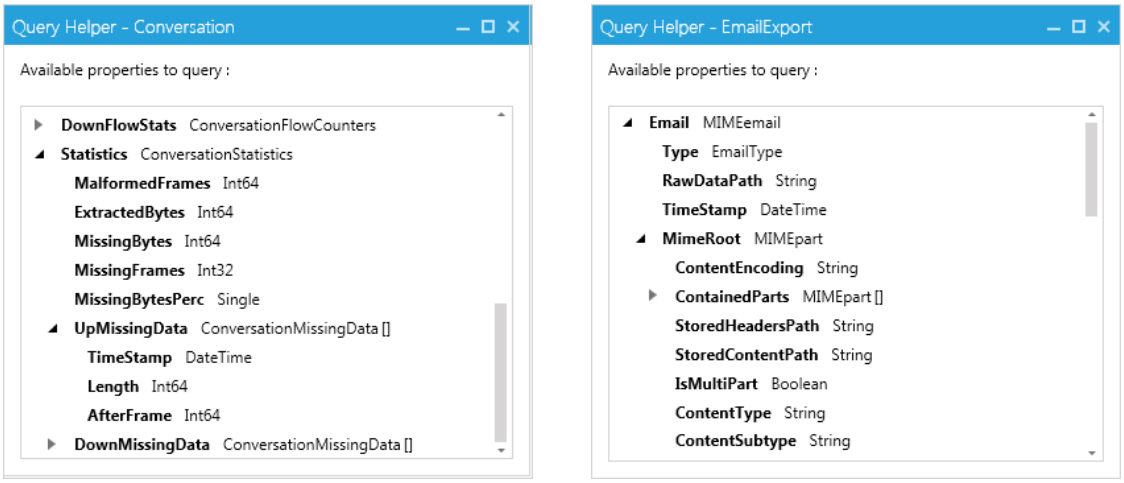


Obrázek 6.24: Dotaz na konverzace - pomocník

Vlastní dotaz lze zadat dvěma způsoby. Při využití pomocníka (*QueryHelper*) uživatel vybere vlastnosti, na které se chce dotazovat, a vyplní hodnoty a případně operátory, kterým mají odpovídat. Na základě tohoto předpisu je automaticky vygenerován dotaz ve formě řetězce, který je zobrazen uživateli - obr. 6.24. Dotaz lze také zadat přímo, ve formě řetězce (*RawQuery*). Tato varianta je obtížnější, ale přináší větší míru variability. Lze kombinovat libovolné vlastnosti s logickými operátory a konstantami - obr. 6.25.

```
(SourceAddress >= IpAddress(192.168.1.0) and SourceAddress <=
IpAddress(192.168.1.255)) and (TargetAddress = IpAddress(147.229.9.23) or
  TargetAddress = IpAddress(147.229.9.90)) and (Bytes > 2000 or Frames
   >  20) and (FirstSeen > DateTime(01/04/2014 00:00:00) and LastSeen
                  <  DateTime(10/04/2014 00:00:00))
```

Obrázek 6.25: Příklad přímo zadaného dotazu na konverzace



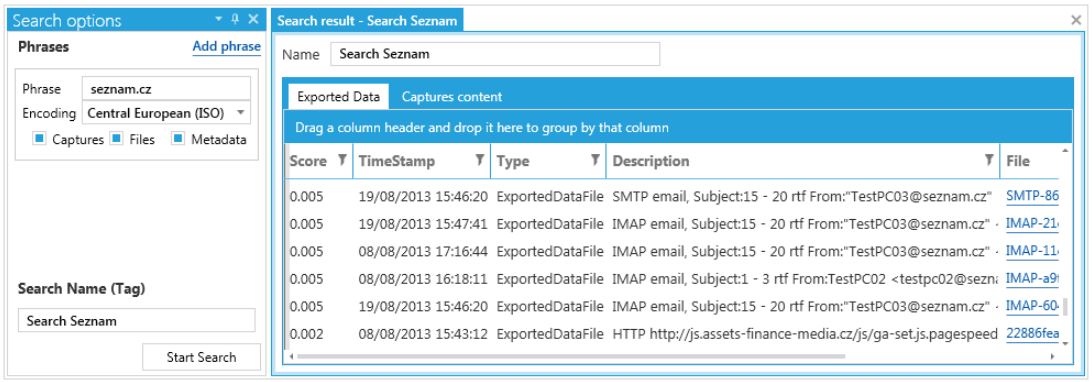
(a) Konverzace (b) Výsledek exportu - email

Obrázek 6.26: Vlastnosti dostupné k dotazování

Pro zjednodušení práce při vyváření dotazu přímo uživatelem je k dispozici další pomocník - obr. 6.26. Ten pro daný model (například konverzaci) zobrazí strom všech vlastností, na které se lze dotazovat včetně příslušného datového typu - viz. 4.6.5 Dotazování. Cestu k takovéto vlastnosti lze snadno zkopírovat a vložit do vytvářeného dotazu.

6.10 Vyhledávání

Kromě dotazování, které se zaměřuje zejména na porovnání informací obsažených v metadatech (v databázi), umožňuje aplikace také volnější fulltextové vyhledávání v obsahu - obr. 6.27.



Obrázek 6.27: Fulltextové vyhledávání v obsahu

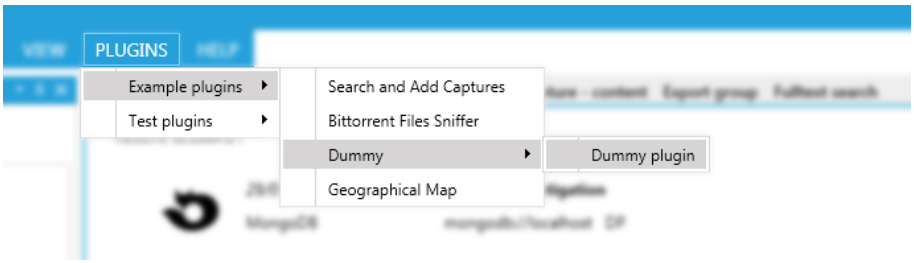
Vyhledávat lze na úrovni obsahu jednotlivých rámců (v souborech capture), v exportovaných (získaných) souborech a v metadatech (v databázi). Vyhledávat lze najednou více vzorků. Při vyhledávání v souborech je nutné specifikovat kódování hledaného řetězce.

Výsledkem vyhledání jsou dvě tabulky. První zobrazuje výsledky exportu, v jejichž metadatech nebo souborech byl nalezen hledaný vzorek, druhá zobrazuje jednotlivé rámce,

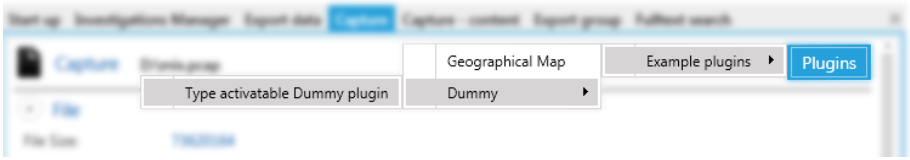
ve kterých byl vzorek nalezen. Z obou tabulek lze snadno přejít přímo do pohledu zobrazující daný výsledek nebo rámec.

6.11 Pluginy

Jak již bylo popsáno v kapitole o návrhu, aplikace umožňuje implementaci rozšíření ve formě pluginů. V rámci pluginu může být implementována téměř libovolná funkcionalita. Z pohledu uživatele se plugin včetně pohledů kromě aktivace nijak neliší od ostatních komponent nástroje. Pluginy jsou organizovány ve stromové struktuře. Toto umožňuje vytvářet jejich přehledné skupiny například podle typu funkcí, které nabízí.



Obrázek 6.28: Aktivace pluginu globálně



Obrázek 6.29: Aktivace pluginu na vybranou entitu (capture)

Plugin lze aktivovat globálně nebo na vybranou entitu nebo v obou případech - obr. 6.28, 6.29. O možných způsobech aktivace, případně včetně kompatibilních typů entit, informuje aplikaci sám plugin. Akce prováděné po aktivaci jsou již plně v režii samotného pluginu. Nejčastějším scénářem je ale zobrazení registrovaného pohledu, který umožní uživateli provést danou akci (například analýzu) nad vybranou entitou.

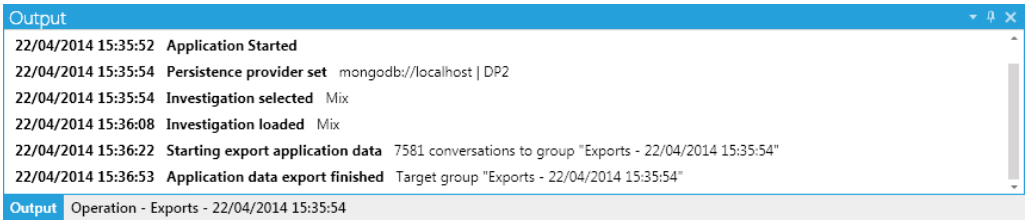
V rámci této práce bylo implementováno několik ukázkových pluginů, které demonstrují možné způsoby práce s API nástroje. Ukázky mohou v budoucnu posloužit také jako základní kostra pro snadnou implementaci nových rozšíření.

6.12 Výstup

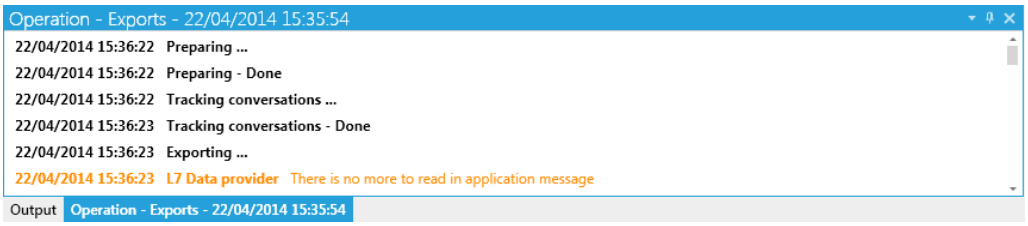
Při používání nástroje vzniká velké množství událostí. Některé jsou pouze informativního charakteru, jiné upozorňují na závažné problémy. Uživatel si může zvolit způsob, jakým o nich bude informován a jakým budou zaznamenány. K dispozici je forma dialogového okna (ve standardním nastavení pouze chybová hlášení), výstup obdobný konzolovému a log v souborovém systému. Každá z událostí je označena mírou závažnosti (typem), podle uživatelského nastavení je poté zpracována.

Kromě hlavního aplikačního výstupu - obr. 6.30 - jsou v aplikaci automaticky vytvářeny další výstupy. Jedná se zejména o výstupy zobrazující průběh a anomálie vzniklé během

operací (například během exportu) - obr. 6.31.



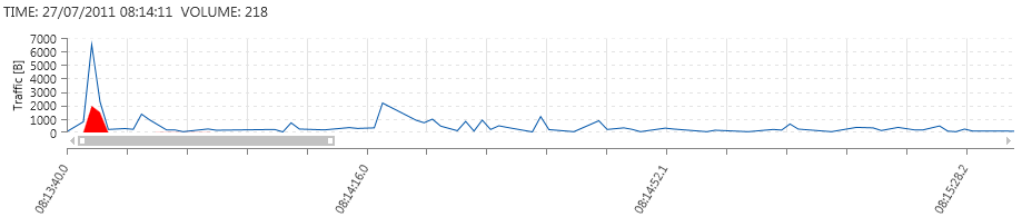
Obrázek 6.30: Hlavní výstup



Obrázek 6.31: Výstup operace - export

6.13 Kvalita získaných dat

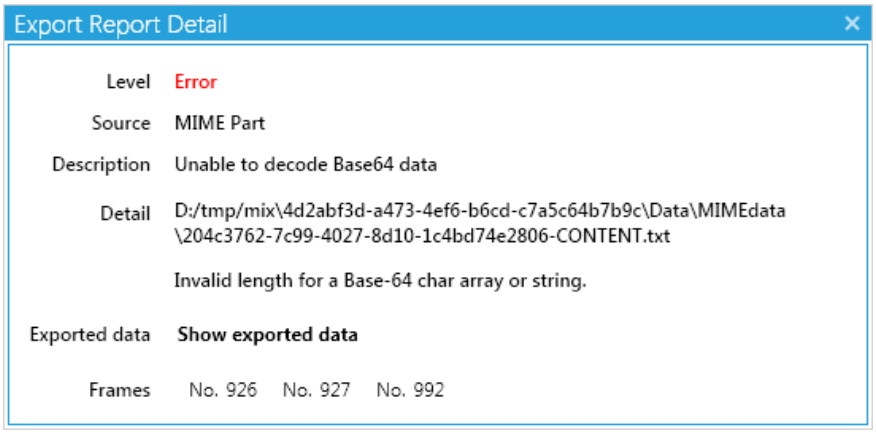
Na základě konzultací v rámci projektu SEC6NET bylo zjištěno, že při forenzní práci je kromě co největšího množství extrahovaných dat a jejich zasazení do kontextu velice důležitá také možnost porovnat jejich kvalitu. Ve vytvořeném nástroji je proto na toto kladen důraz. Problematiku lze rozdělit do dvou rovin. První z nich tvoří analýza kvality zachycených dat, druhou poté kvalita dat extrahovaných.



Obrázek 6.32: Množství chybjících dat v souboru v čase

V laboratorním přostředí lze dosáhnout velmi dobré kvality při zachytávání dat. V reálném provozu ale v rámci tohoto procesu velmi často dochází ke ztrátě rámců. Toto může být způsobeno mnoha faktory. Může se jednat o zahlcení daného zařízení, jeho výpadek, změny topologie, chyby v loadbalancingu a podobně. Díky stavovému mechanismu transportní vrstvy TCP lze ve většině případů takováto chybějící data detekovat. Lze určit množství, typ a časový úsek, kde data chybějí. Toto je důležité zejména v situaci, kdy lze kvalitu zachycených dat ovlivňovat pouze nepřímo, jako jsou například data získaná od ISP. Netfox Detective umožňuje získat statistiky v rámci souboru, nebo skupiny konverzací - viz. 6.5. Množství chybějících dat je také zobrazeno v rámci časové osy - obr. 6.32. Užitečným indikátorem u jiných transportních vrstev (UDP) a například RTP může být rozložení odeslaných a přijatých dat u daného hosta.

Kvalitu exportovaných dat lze v rámci nástroje sledovat dvěma způsoby. Prvním idikátorem je výstup generovaný během spuštěné operace - viz. 6.12. Na základě popsaných anomálií lze odhalit zásadní problémy, které se vyskytly během exportu, a například manuální analýzou dotčených dat určit možné přičiny. Kromě tohoto každý výsledek exportu i každá jednotlivá událost v něm obsažená obsahuje seznam reportů - hlášení. Tato hlášení popisují problémy týkající se rekonstrukce daných dat. Každé hlášení obsahuje úroveň závažnosti, popis (detailní) a rámce, kterých se případná chyba při rekonstrukci týká - obr. 6.33. Lze tak například ověřit, zdali se jedná o chybu v původních datech, chybu v datech zachycených, nebo chybu ve vlastním exportéru.



Obrázek 6.33: Detail hlášení o chybě týkající se výsledku exportu

6.14 Výkonnost nástroje (srovnání)

V této podkapitole je uvedeno porovnání délky trvání jednotlivých obdobných operacích v různých nástrojích pro analýzu obsahu síťové komunikace a množství získaných výsledků. Vzhledem k tomu, že jednotlivé nástroje se liší jak ve způsobu práce, množství podporovaných protokolů, tak i v datech, která získávají, nemohou zde uvedené výsledky sloužit k exaktnímu porovnání, ale pouze jako ukazatele indikující možnosti práce s různě velkými a strukturovanými vstupními daty.

Porovnání bylo provedeno na čtyřech souborech typu pcap (ten je podporván všemi) různé velikosti a struktury - tab. 6.1, tab. 6.2. Porovnávanými nástroji byly : Netfox Detective (ND), Wireshark (WS), Microsoft Network Monitor (MNM) a NetworkMiner (NM). Porovnávanými operacemi byly přidání souboru do nástroje (zahrnující tracking konverzací), načtení již jednou přídaného souboru (pokud to nástroj umožňuje), získání detailních statistik o obsahu (distribuce protokolů, data v čase, chybějící data a podobně.), export obsažených aplikačních dat (délka, počet souborů a velikost) a načtení již extrahovaných výsledků (pokud to nástroj umožňuje) - tab. 6.3. Vzhledem k tomu, že výraznou většinu aplikačního obsahu porovnávaných souborů tvoří HTTP komunikace, kterou všechny uvedené nástroje podporují, lze nástroje porovnávat i přes to, že se v podpoře ostatních protokolů liší.

Uvedená data ukazují na skutečnost, že rychlost a množství získaných dat u vybraných operací nástrojem Netfox Detective je srovnatelná nebo lepší než u porovnávaných nástrojů. Velkou výhodou je také persistence získaných výsledků - jejich načtení při příštím použití je řádově rychlejší než jejich získávání.

soubor	velikost	rámců	konverzací
1.	71 895KB	69 998	9 077
2.	770 345KB	685 041	13 244
3.	181 042KB	128 675	3 356
4.	123 483KB	90 925	2 783

Tabulka 6.1: Soubory použité při porovnání

protokol	%
HTTP	96.022
HTTPS	3.844
MSN	0.088
ICQ	0.044
IMAP	< 0.001
SMTP	< 0.001
POP3	< 0.001
YMSG	< 0.001

(a) 1.

protokol	%
HTTPS	43.012
HTTP	28.170
SSH	26.644
ICQ	0.044
XMMP	< 0.001
SMTP	< 0.001

(b) 2.

protokol	%
HTTP	73.314
HTTPS	24.685

(c) 3.

protokol	%
HTTP	71.476
HTTPS	28.202

(d) 4.

Tabulka 6.2: Zastoupení „zajímavých“ aplikačních protokolů v jednotlivých souborech

soubor	nástroj	přidání	načtení	statistiky	export	načtení (export)
1.	ND	00:08	00:02	00:05	00:31 5730 78.9MB	00:25
1.	WS	00:06	-	00:03	-	-
1.	MNM	00:55	-	-	-	-
1.	NM	-	-	-	04:07 (5 350) 70,5 MB	-
2.	ND	00:44	00:16	00:43	05:42 3928 92MB	00:21
2.	WS	00:33	-	00:26	-	-
2.	MNM	05:07	-	-	-	-
2.	NM	-	-	-	14:25 6061 102 MB	-
3.	ND	00:38	00:04	00:08	00:52 3808 153 MB	00:05
3.	WS	00:19	-	00:04	-	-
3.	MNM	01:12	-	-	-	-
3.	NM	-	-	-	04:41 3447 170 MB	-
4.	ND	00:11	00:03	00:03	04:16 2437 89 MB	00:03
4.	WS	00:25	-	00:03	-	-
4.	MNM	00:43	-	-	-	-
4.	NM	-	-	-	02:54 2167 57,4 MB	-

Uvedené časy jsou ve formátu mm:ss.
Měřeno na : MS W7, Intel Core i5-2410M, 8GB RAM, Seagate SSHD ST1000LM014

Tabulka 6.3: Porovnání jednotlivých operací

Kapitola 7

Ukázkový případ

V této kapitole bude popsáno řešení ukázkového případu z knihy *Network Forensics : Tracking Hackers through Cyberspace*[3] ze strany 135 „Ann's Randezvous" pomocí vytvořeného nástroje Netfox Detective.

7.1 Zadání

V knize je nastíněn následující scénář. Po propuštění na kauci podezřelá Ann Dercover zmizí. Vyšetřovatelé ale naštěstí monitorolovali data, která přenášela po síti. Úkolem je pokusit se zajistit informace, které by mohly přispět k určení jejího nynějšího místa pobytu.

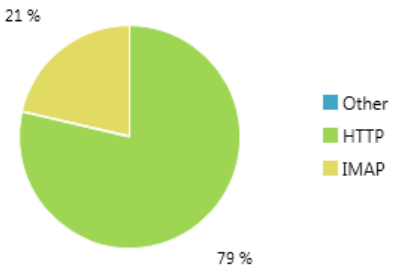
Kromě zachycených dat jsou k dispozici základní informace o síťové infrastuktuře a MAC adresa laptopu podezřelé.

7.2 Řešení - doporučený postup

Tato podkapitola ukazuje řešení případu podle postupu popsaného ve výše uvedené knize. Místo použitých nástrojů a utilit (zejména Wireshark) budou obdobné úkony prováděny v nástroji Netfox Detective.

7.2.1 Analýza obsažených protokolů

Prvním popisovaným krokem je získaní přehledu o obsažených aplikačních protokolech. Stejně jako v aplikaci Wireshark lze toto velmi snadno provést také v aplikaci Netfox Detectiv a to na základní přehledové kartě u přidaného souboru. Jsou zde například k dispozici tabulky a grafy zobrazující procentuální zastoupení jednotlivých protokolů - obr. 7.1.



Protocol	Total Bytes	%
HTTP	47245402530	78.5709
IMAP	12885193080	21.42856
(unknown)	295651	0.0004916788
DNS	20514	3.411556E-05
NETBIOS-NS	6048	1.005805E-05

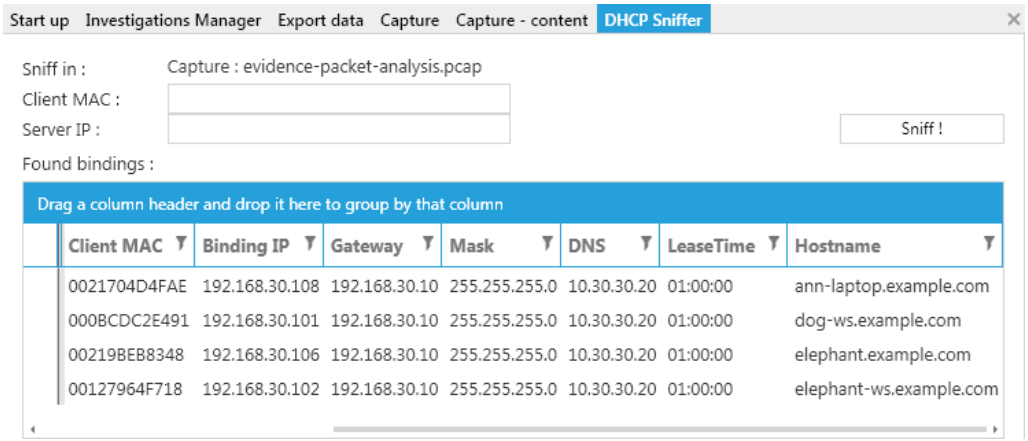
Obrázek 7.1: Zastoupení protokolů v analyzovaném souboru

Výsledkem anlýzy v tomto kroku je zjištění, že cca 1/4 zachycených dat tvoří komunikace pomocí poštovních protokolů (IMAP a SMTP), jejichž obsah může být v tomto případě zajímavý.

7.2.2 Analýza IP adres - DHCP

V druhém popsaném kroku autoři zjišťují adresu přidělenou počítači podezřelé pomocí protokolu DHCP. Činí tak na základě analýzy jednotlivých paketů. Nejprve provedou filtraci na známou MAC adresu, z obsahu paketu poté zjistí přidělenou IP adresu.

V nástroji Netfox Detective lze také provést filtraci rámců na základě MAC adresy. V rámci ukázkových pluginů je ale implmentován plugin „DHCP Sniffer“, který umožňuje snadno nalézt adresy přiřazené právě pomocí protokolu DHCP. Plugin podporuje filtraci na MAC adresu klientské stanice - obr. 7.2.



Obrázek 7.2: Výstup pluginu „DHCP Sniffer“

Na základě výstupu pluginu se podařilo určit, že stanici se zkoumanou MAC adresou (00:21:70:4D:4F:AE) byla přidělena IP adresa 192.168.30.108. Konverzace obsažené v zachycených datech lze poté pomocí této adresy snadno vyfiltrovat - obr. 7.3

#	First Seen	Last Seen	Transport	Application	Client	Server	Frms Up
85	17/05/2011 19:33:07	17/05/2011 19:33:09	TCP	IMAP	192.168.30.108:1685	205.188.58.10:143	9
86	17/05/2011 19:33:09	17/05/2011 19:33:09	TCP	IMAP	192.168.30.108:1685	205.188.58.10:143	1
87	17/05/2011 19:33:20	17/05/2011 19:33:25	TCP	IMAP	192.168.30.108:1686	205.188.58.10:143	17
105	17/05/2011 19:34:17	17/05/2011 19:34:18	TCP	IMAP	192.168.30.108:1688	205.188.58.10:143	10

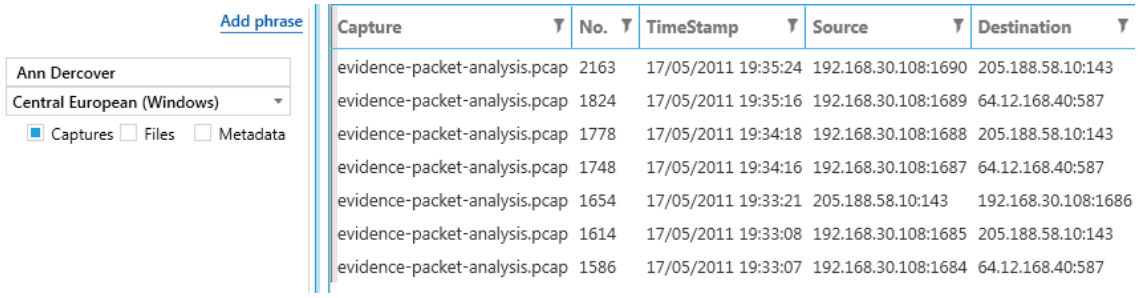
Obrázek 7.3: Konverzace s nalezenou IP adresou

7.2.3 Vyhledání klíčových slov

V dalším kroku autoři provádějí prohledání zachyceného souboru pomocí fultextového vyhledávání nástrojem ngrep. Netfox Detective umožňuje také vyhledávat v obsahu jednotlivých rámců.

Obsah souboru je prohledán na výskyt klíčového slova „Ann Dercover“. Výsledkem je stejně jako v knize sedm rámců, ve kterých se tato fráze vyskytuje - obr. 7.4. Ze zdrojových a cílových adres rámců lze určit, že klíčové slovo se vysktyuje v rámcích, které jsou

s největší pravděpodobností součástí komunikace pomocí porotokolu IMAP (TCP port 143) a SMTP (TCP port 587). Obsah nalezených rámců také odpovídá formátu používanému u e-mailových zpráv - obr. 7.5.



Obrázek 7.4: Fulltextové vyhledávání

Obrázek 7.5: Obsah rámce nalezeného fulltextovým vyhledáváním

7.2.4 Analýza poštovních protokolů SMTP, IMAP

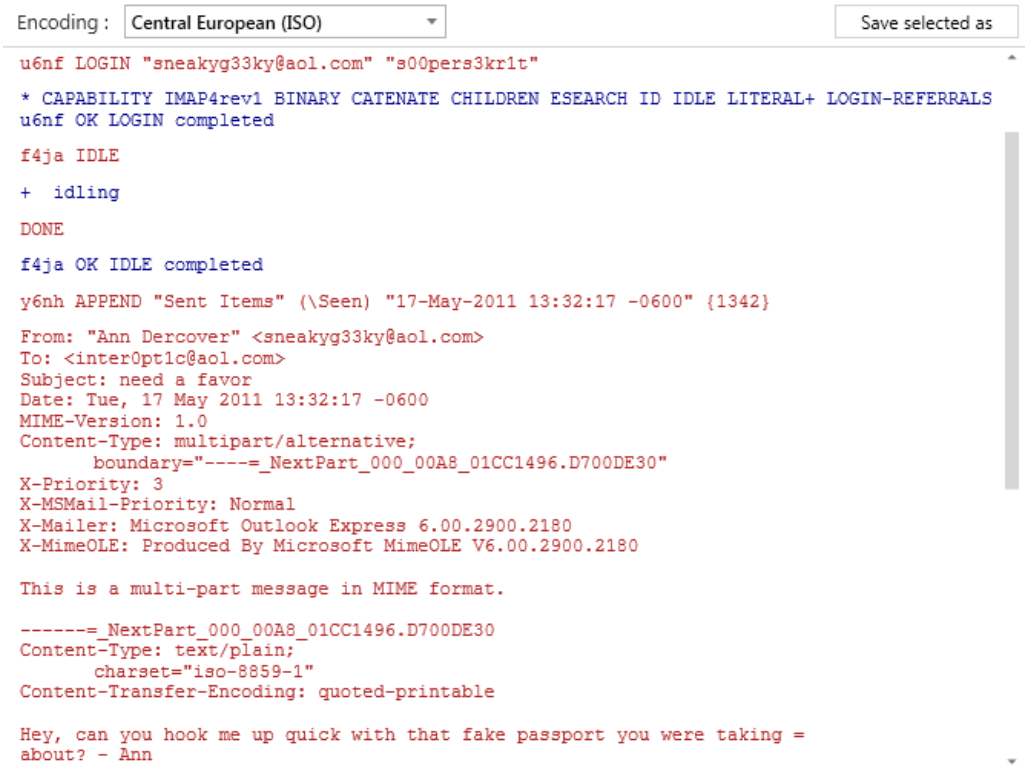
Posledním krokem v popisovaném postupu je analýza vlastního obsahu poštovních protokolů. Ta je autory prováděna pomocí zobrazení jednotlivých aplikačních zpráv vybrané konverzace. I tuto možnost Netfox Detective nabízí.

Autoři prohlíží stream sestavených aplikačních zpráv a ručně, nebo pomocí různých utilit získavají přihlašovací údaje a vlastní obsah přenášených e-mailů - obr. 7.6. Posledním krokem je uložení a dekodování přenášené přílohy - dokumentu pro aplikaci MS Word obsahující zajímavé informace.

Tento způsob práce je sice v aplikaci Netfox Detective možný, není ale vzhledem k její koncepci příliš vhodný (je zbytečně pracný). V tomto bodě by bylo vhodnější využít přímo analýzu pomocí aplikačních exportérů nad vybranou množinou konverzací - tento krok je popsán v další podkapitole „Řešení - alternativní přístup“.

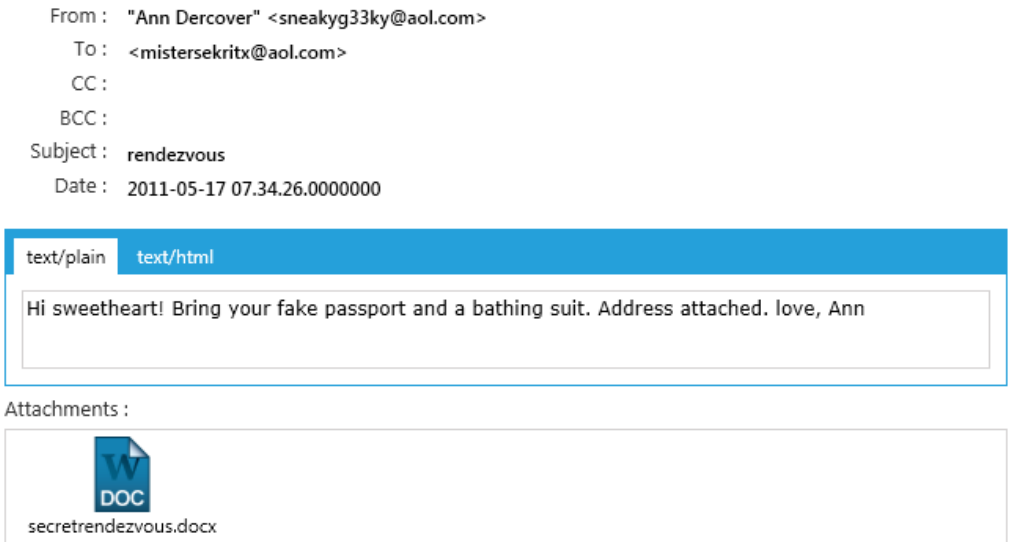
7.3 Řešení - alternativní přístup

Jako alternativu k popsanému manuálnímu postupnému způsobu řešení zadaného problému nabízí autoři možnost použít některou z aplikací, která se zaměřuje na extrakci aplikačního obsahu - v tomto případě již výše popsaný NetworkMiner.

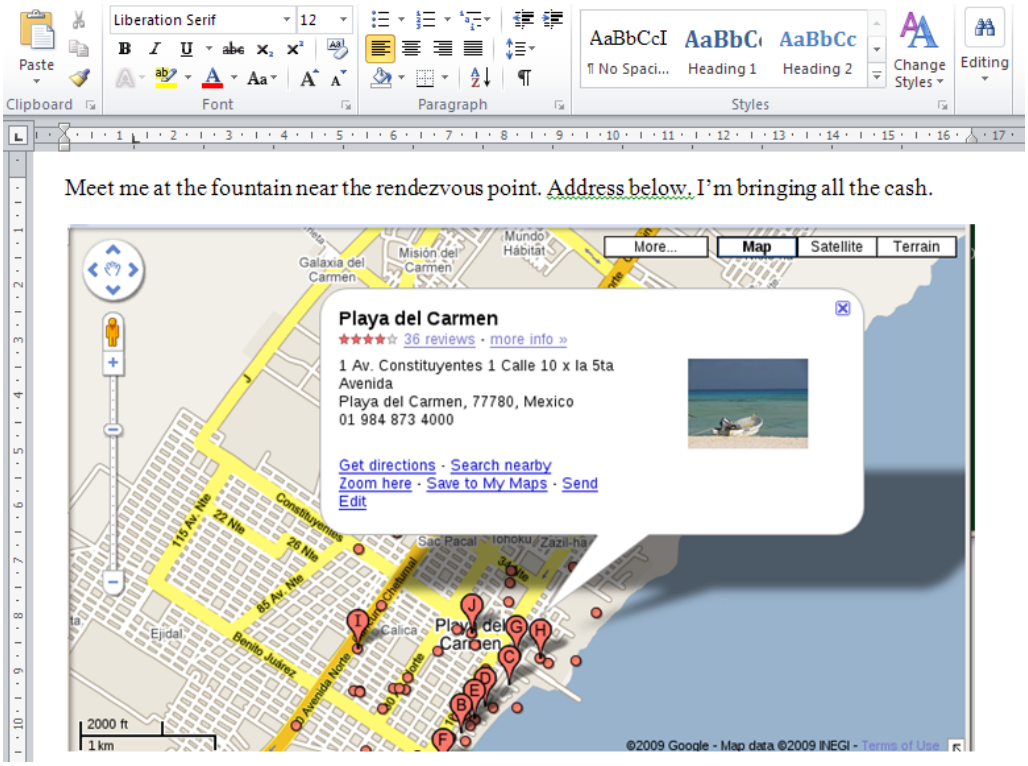


Obrázek 7.6: Stream reassemblovaných aplikačních zpráv

Toto je přístup, který je bližší návrhu nástroje Netfox Detective. Například pouze na základě zjištění, že soubor obsahuje komunikaci pomocí poštovních protokolů, lze na celou množinou v něm obsažených konverzací spustit export zaměřený pravě na ně. Výsledkem jsou potom přímo jednotlivé nalezené e-maily, včetně extrahovaných a dekodovaných příloh - obr. 7.7, obr. 7.8.

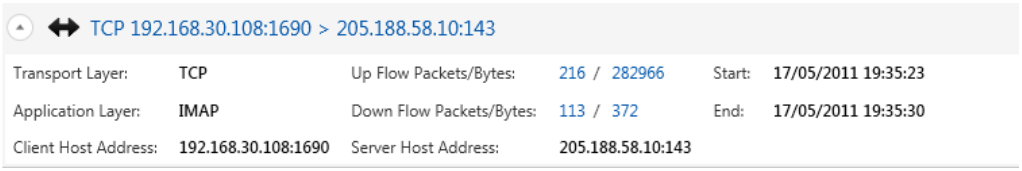


Obrázek 7.7: Rekonstruovaný e-mail



Obrázek 7.8: Obsah dekódované přílohy „secreterandezvous.docx“

Každý výsledek je spojen s konverzací, ze které byl zrekonstruován. Lze tedy snadno ověřit, že byl odeslán z laptopu podezřelé - obr. 7.9.



Obrázek 7.9: Informace o konverzaci v rámci které byl přenášen uvedený e-mail

7.4 Shrnutí

Tato kapitola ukazuje příklad řešení jednoduchého vybraného ukázkového případu pomocí vytvořeného nástroje Netfox Detective. První z popsaných způsobů řešení demonstruje vlastnosti nástroje, které umožňují pracovat směrem zdola nahoru - od jednotlivých přenosových jednotek až k hledanému obsahu. Druhý způsob naopak ukazuje možnost práce shora dolů, kdy jsou data exportována přímo a teprve na základě jejich obsahu dojde k ověření, že se jedná o data patřící k podezřelé osobě. Oba přístupy poté demonstrují možnosti nástroje Netfox Detective, který oproti většině dostupných nástrojů umožňuje libovolnou kombinaci těchto dvou nejpoužívanějších přístupů a integraci nových součástí (nástrojů) v rámci jedné komplexní platformy.

Kapitola 8

Závěr

V rámci této práce byl navržen a implementován nástroj pro analýzu obsahu síťové komunikace Netfox Detective. Koncepce nástroje byla vytvořena na základě prostudování existujících řešení, postupů a přístupů používaných pro analýzu obsahu síťové komunikace, zejména forenzní. Během návrhu a implementace bylo třeba vyřešit řadu problémů a definovat koncepce vzhledem k budoucímu vývoji aplikace.

Podařilo se vytvořit kontroler již existujícího Netfox Frameworku, který umožnil jeho použití jakožto knihovny pro analýzu a rekonstrukci aplikačních dat. Další část práce se orientovala na způsob ukládání a práce s daty ve vytvářené aplikaci. Pro tyto účely byla zvolena objektová (noSQL) databáze. Bylo vytvořeno prostředí pro použití různých typů těchto databází efektivním způsobem. Na základě dílčích návrhů byla vytvořena celková architektura aplikace reflektující všechny stanovené požadavky.

Podle návrhu se podařilo vytvořit implementaci nástroje, který byl posléze úspěšně prezentován v rámci projektu SEC6NET. Toto také přineslo užitečnou zpětnou vazbu a začlenění nových podnětů do výtvářeného nástroje. Implementace aplikace byla průběžně podrobována testům, které měly za cíl ověřit funkcionalitu aplikace a další životaschopnost použitých konceptů s ohledem na způsob použití a poskytovaný výkon. Testování také zpětně přispělo k odhalení a vylepšení slabých míst v Netfox Frameworku, ať už na úrovni kvality poskytovaných dat, tak při zvyšování výkonnosti a škálovatelnosti.

Další část práce je věnována popisu možností, které vytvořený nástroj nabízí. Postupně popisuje jednotlivé prostředky, pomocí kterých lze analýzu provádět - od jednotlivých rámců po vlastní získaný aplikační obsah. Popis se snaží ilustrovat variabilitu a rozšiřitelnost nástroje jako celku. Mezi nejzajímavější prostředky patří například systém pro analytickou práci pomocí dotazování.

Vývoj nástroje v budoucnu by se měl zaměřit na rozšíření aplikace (a Netfox Frameworku) o množinu podporovaných protokolů tak, aby se plně rozvinul potenciál vytvořeného systému. Společně s novými protokoly budou také postupně navrhovány a implementovány nové specifické pohledy na získaná data. Na základě zpětné vazby z dalšího testování a používání by bylo vhodné doplnit nástroj také o množinu pluginů (rozšíření), implementující automaticky nejčastěji prováděné úkony a analýzy. Dalším zajímavým směrem, kterým by se mohl vývoj nástroje ubírat, je jeho použití v prostředí pro distribuované výpočty a spolupráci více uživatelů. Mohlo by dojít jak ke zvýšení výkonnosti, tak k vytvoření systému pro strukturovanou a organizovanou práci.

Literatura

[1] Casey, E.: Network traffic as a source of evidence: tool strengths, weaknesses, and future needs. *Digital Investigation*, ročník 1, č. 1, 2004: s. 28 – 43, ISSN 1742-2876.

[2] Chappell, L.: *Wireshark Network Analysis: The Official Wireshark Certified Network Analyst Study Guide*. Chappell University, Protocol Analysis Institute, Chappell University, 2010, ISBN 9781893939998.

[3] Davidoff, S. and Ham, J.: *Network forensics : tracking hackers through cyberspace*. Prentice Hall, 2012, ISBN 0-13-256471-8.

[4] Garofalo, R.: *Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModel Pattern*. Microsoft Press Series, Microsoft Press, 2011, ISBN 9780735650923.

[5] Hall, G.: *Pro WPF and Silverlight MVVM: Effective Application Development with Model-View-ViewModel*. Apresspod Series, Apress, 2010, ISBN 9781430231622.

[6] Hutchens, J.: Network traffic analysis with Xplico. *eForensics Magazine*, 2013.

[7] Pilli, E. S.; Joshi, R.; Niyogi, R.: Network forensic frameworks: Survey and research challenges. *Digital Investigation*, ročník 7, č. 1?2, 2010: s. 14 – 27, ISSN 1742-2876.

[8] Pluskal, J.; Veselý, V.; Ryšavý, O.: NetFox - The network forensic extandable analysis tool. In *6th AFCEA Student conference - Future of information and communication technology*, 2014, ISBN 9786065510470.

[9] Pluskal Jan: *Prostředí pro zpracování dat ze zachycené síťové komunikace*. FIT VUT v Brně, 2014.

[10] Ritchie, B.: *RavenDB High Performance*. Community experience distilled, Packt Publishing, 2013, ISBN 9781782166993.

[11] WWW stránky: Infragistics - WPF. `http://www.infragistics.com/products/wpf/`.

[12] WWW stránky: MongoDB. `http://www.mongodb.org/`.

[13] WWW stránky: MVVM Light Toolkit. `http://www.galasoft.ch/mvvm/`.

[14] WWW stránky: Network Monitor Experts. `http://nmexperts.codeplex.com/`.

[15] WWW stránky: Packet.Net. `http://sourceforge.net/projects/packetnet/`.

[16] WWW stránky: RavenDB - 2nd generation document database. http://ravendb.net/.

[17] WWW stránky: System Properties Comparison MongoDB vs. RavenDB. http://db-engines.com/en/system/MongoDB%3BRavenDB.

[18] WWW stránky: Telerik - UI for WPF. http://www.telerik.com/products/wpf/.

Příloha A

Obsah CD

- Adresář application/src/ - zdrojové kódy aplikace
- Adresář application/bin/ - instalační balíček pro platformu MS Windows
- Adresář text/TeX - zdrojové kódy textu práce ve formátu LaTeX
- Adresář text/pdf - text práce ve formátu PDF