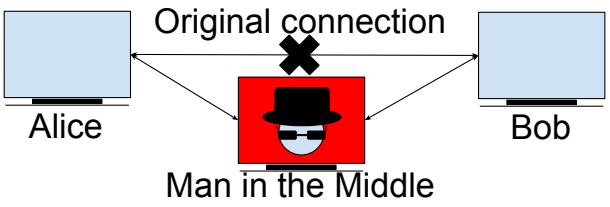
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2016
http://excel.fit.vutbr.cz

Automatization of MitM Attack for SSL/TLS Decryption

Marek Marušic*



Abstract
SSL/TLS are protocols used to encrypt network traffic. They provide secure communication between clients and servers. The communication can be intercepted with MitM attack. This paper is aimed to describe the automatization of MitM attack and demonstrate its results. The automatization is done by MitM probe and a python script, which configures the probe and starts the attack. The script is easy to use, without great effort. It takes care of configuration of the probe, then it starts the tools used for network traffic capture and at last it starts MitM tools to do the attack. During the MitM attack, users are warned by client applications about insecure connection. The client applications either provide an option to establish a connection anyway or it forbids clients to establish the connection with insecure parameters. In this paper, the users can learn what are SSL/TLS protocols and about a possibility how to intercept the network traffic encrypted by these protocols.

Keywords: MitM — attack — SSL — TLS — decryption

Supplementary Material: N/A

*xmarus05@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

SSL/TLS encrypted network traffic is nowadays used for secure transmission of various network protocols over an insecure infrastructure of Internet in order to provide secure communication between end points. However, sometimes there is a need to decipher and analyze the communication between criminals or suspects by the government, police or secret services. Existing tools for the decryption are very complicated to use, therefore it would take a great amount of time to perform the decryption by a common user. Furthermore, the user would have to use various tools in order to be able to intercept the communication in networks with various properties (ipv4/ipv6, various cipher suites, various type of connection into the network, etc.). The goal of this work is to provide easy to use solution, which supports various network properties for interception of the SSL/TLS traffic.

A great number of the web hosts on the Internet use SSL/TLS protocols (see Subsection 2.1). It would take a great amount of time to decipher the captured SSL/TLS traffic with brute force attack. This means that we need a better solution with faster and easier access to the decrypted data. There is a lack of a tool witch would support both ipv4 and ipv6, capture the packets into pcap file (most common format for network analysis) with ability to decipher them, support DH key exchange (see Subsection 2.1), support various types of connection into the network, with ability to act as transparent proxy and last but not least with an easy usage for the users.

There are many tools focused on deciphering the

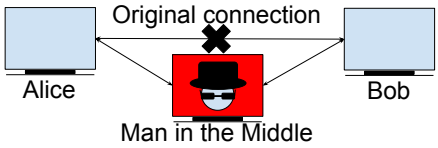


Figure 1. An illustration of the Man in the Middle attack

SSL/TLS traffic, which use various scenarios for the decryption. A great number of the tools are open source and can be used free of charge. Few of these tools will be described in a Subsection 2.4. These tools often require complicated setup of a device and of the tool itself.

During this work there was set up a MitM probe (see Section 3) which is very easy to use. The probe uses a python script which will automatically configure the probe and run the MitM attack. There is used SSL/TLS intercepting proxy (see Section 2), which terminates the connection between clients and servers and establishes its own connections with both of them. Then it forwards the data from clients to servers and vice versa. For clients, the proxy acts as visited server with its own certificate. The probe captures all network packets into its SSD disk and the captured network traffic is then deciphered by a private key of the SSL/TLS proxy or with captured *master secrets* of SSL/TLS sessions.

2. Man in the Middle Attack

The classic Man in the Middle (MitM) attack (see Figure 1) is a scenario where an attacker intrudes into server-client connections. This attack allows the attacker to eavesdrop and modify network traffic and connections while the communicating endpoints believe that they are talking with each other.

The attacker must have access to the network that endpoints are using and all the intercepted traffic has to go through the MitM device. There are few ways how to achieve this:

1. On LAN network this can be done with a *arpspoof* tool, which redirects packets from a target host (or all hosts) on the LAN intended for another host on the LAN (see Subsection 3.2).
2. The MitM device can be connected to the upstream network, e.g., ISP connection (see Subsection 3.1).

2.1 SSL/TLS Protocols

The major enemies of the MitM attack are SSL and TLS protocols. They are ”cryptographic protocols designed to provide secure communication over insecure infrastructure.” [1] ”The primary goal of these protocols is to provide privacy and data integrity between two communicating applications.” [2] This means that the intercepted network traffic packets are useless for the attacker hence the data are encrypted and only client and server can access the data.

Client		Server
Client hello	$\longrightarrow$	
	$\longleftarrow$	Server hello
	$\longleftarrow$	Server certificate*
	$\longleftarrow$	Server key exchange*
	$\longleftarrow$	Certificate request*
	$\longleftarrow$	Server hello done
Client certificate*	$\longrightarrow$	
Client key exchange	$\longrightarrow$	
Certificate verify*	$\longrightarrow$	
[Change cipher spec]	$\longrightarrow$	
Finished	$\longrightarrow$	
	$\longleftarrow$	[Change cipher spec]
	$\longleftarrow$	Finished
Application Data	$\longleftrightarrow$	Application Data

Table 1. SSL/TLS Handshake [2] (*optional message, [Change cipher spec] is an independent protocol)

”When a SSL/TLS client and a server first start communicating, they agree on cryptographic parameters of the session produced by the SSL/TLS Handshake Protocol” (illustrated in Table 1). ”They use public-key encryption techniques to generate shared secrets.” [2] There are different algorithms for the exchange of generated shared secrets over network e.g RSA [1] (page 36) , Diffie–Hellman (DH)[3], ephemeral Diffie-Hellman (DHE), Elliptic Curve Diffie-Hellman (ECDH), ephemeral Elliptic Curve Diffie-Hellman (ECDHE) etc. Modern web applications prefer the most secure ones, i.e., DHE and ECDHE key exchange. The public-key encryption techniques use asymmetric encryption. The shared secrets are then used in symmetric encryption of the network traffic. The public key is a part of a host certificate. The certificate is a digital document, which bears the public key, a digital signature of the certificate issuer (CA authority) and the information about its validity and owner.

2.2 HTTP Strict Transport Security (HSTS)

Another enemy of the MitM attacks is the HSTS protocol, which enforces usage of the SSL/TLS encryption between the client and server and forbids the client to accept an untrusted or a fake certificate as well. The client software will warn the user about possible MitM attack and block all the communication with HSTS host until there is a secure connection. The HSTS can be enabled through a *Strict-Transport-Security* HTTP header or through preloaded list of HSTS hosts in the

client's software application, e.g., web browser. [4]

2.3 SSL/TLS Decryption

There are two common scenarios how to get to decrypted data of the captured SSL/TLS network traffic. In the first case, we need to capture whole SSL/TLS Handshake and have the server's private key, which was used during the Handshake. This way we can obtain the shared secrets, which was used for the symmetric encryption of the session. However, this scenario does not work with key exchange algorithms based on a Diffie-Hellman algorithm, where the shared secrets are not transmitted through the network, but they are calculated on the endpoint devices.

The second option is to capture connections with SSL/TLS proxy. The proxy will split the connection of the intercepted client and server to downstream and upstream connections. Downstream communication between client and proxy will use proxy's certificate. The upstream connection between proxy and server will use server's certificate. This solution eliminates the problem with DH key exchange since the sessions were negotiated with the proxy, thus it has access to all the parameters of the sessions. [5]

2.4 MitM Tools for SSL/TLS Decryption

Besides the classic MitM attack, we need to decrypt intercepted SSL/TLS network traffic. For this purpose, we can use tools implementing the aforementioned decryption scenarios. This can be called sslplitting. There is another way how to get to the decrypted data of the SSL/TLS network traffic and it is called sslstripping. Sslstripping also works as SSL/TLS proxy, however only the communication between proxy and server is encrypted and the communication between client and proxy is not encrypted. In the following, I will discuss few open-source tools in this section.

SSLsplit[1] works as a SSL/TLS proxy between client and server. "*SSLsplit* supports plain TCP, plain SSL, HTTP and HTTPS connections over both IPv4 and IPv6. For SSL and HTTPS connections, *SSLsplit* generates and signs forged X509v3 certificates on-the-fly, based on the original server certificate subject DN and subjectAltName extension." [6] With this tool we can not intercept HSTS preloaded hosts since client applications disallow clients to accept untrusted certificates.

Mitmproxy[2] is "an interactive, SSL-capable man-in-the-middle proxy for HTTP with a console interface." [7] This tool captures the data from the application layer as well as *SSLsplit*. However, it uses its own certificate with the public and private key for the SSL/TLS communication, which can be used for decryption of the captured packets from tcpdump.

MITMf[3] **and Bettercap**[4] use *SSLstrip+*[5] and special DNS server *dns2proxy*[6] to implement partial HSTS bypass. The main difference is the scenario used in SSL/TLS proxy of this tool. The SSL/TLS connections are also terminated by them, however, the downstream connection between client and attacker does not use SSL/TLS encryption and remains decrypted. The partial HSTS bypass redirects the client from domain name of the visited web host to a fake domain name by sending HTTP redirection request. The client is then redirected to a domain name with extra *w* in *www* or *web.* in the domain name e.g. *web.example.com*. This way the web host is not considered as a member of HSTS preloaded hosts list and the client can access the web host without SSL/TLS. The fake domain names are then resolved to real and correct IP addresses by the special DNS server, which expects these changes in the domain names. The downside of this attack is that the client has to start the connection over HTTP due to the need of HTTP redirection.

The main disadvantage of the *SSLsplit* and *mitmproxy* is a poor support of HSTS bypass for HSTS preloaded hosts, compared to *MITMf* and *bettercap* which support HSTS bypass and allow clients to connect to HSTS preloaded hosts.

All mentioned tools are not able to save the network traffic into a pcap file format since they work with the application layer of the *ISO/OSI* network model [8]. However, we can use a tcpdump to capture the data into a pcap format and decrypt captured packets using the private key of the used fake certificate. This is not possible with cipher suites using DH key exchange algorithm. However, *mitmproxy* can save the *master secrets* of sessions into a file with NSS Key Log Format[7] and use this file for decryption of the data instead of the private key, which eliminates the problem with DH key exchange.

[1] https://www.roe.ch/SSLsplit
[2] https://mitmproxy.org/
[3] https://github.com/byt3bl33d3r/MITMf
[4] https://www.bettercap.org/
[5] https://github.com/byt3bl33d3r/sslstrip2
[6] https://github.com/LeonardoNve/dns2proxy
[7] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format

	SSLsplit	Mitmproxy	MITMf	bettercap	MitM Probe
ipv4	✔	✔	✔	✔	✔
ipv6	✔	✘	✔	✔	✔
master keys logging	✘	✔	✘	✘	✔
partial HSTS bypass	✘	✘	✔	✔	✘*
sslsplitting	✔	✔	✘	✘	✔
sslstripping	✘	✘	✔	✔	✘*
arpspoof	✘	✘	✔	✔	✔
bridge	✘	✘	✘	✘	✔
CA certificates	✘	✔	✘	✘	✔
pcap output	✘	✘	✘	✔	✔
auto traffic redirection	✘	✘	✔	✔	✔

Figure 2. Comparison of the MitM tools (*these properties should be also implemented in the future)

3. SSL/TLS Intercepting Probe

The purpose of the probe is to simplify and automatize the MitM attack on the connected network. The goal is to be able to connect input and output Ethernet cables, login to OS on the probe and start the MitM attack just by a double click on an icon. In the future, the used software should provide a GUI for greater scalability with a simple control. The setup and start of the MitM are provided by my python script, which starts and configures all the necessary software and settings.

The OS is installed on a 32GB USB 3.0 flash for an easy transportation. The probe contains SSD 120GB disk, 3.2 GHz Intel Pentium G3258 processor and 8GB RAM for fast packet capture. The greatest criterion during an HW selection process was the motherboard. It should own two GLAN interfaces with very good throughput in order to be able intercept and capture communications of the whole subnet. The selected one is Mini ITX Intel Motherboard GIGABYTE GA-Z97N-WIFI - Intel Z97 with Intel (*eno1*) and Realtek (*enp2s0*) GLAN interfaces. Moreover, it supports Wifi connection, therefore, we are able to perform the MitM attack wirelessly as well.

I have used Archlinux[8] as OS due to its powerful user repositories, which provide most of the used tools and packages, without the need to compile them from sources manually. I have subsequently installed all the necessary packages needed to realize the MitM attack. I will describe all the required packages and their usage in this document. The python script, responsible for the MitM attack initiation, encapsulates setup of the settings and tools which will be described in this Section.

With OpenSSL tool, I have created CA certificate with public and private key pair, which are required by *SSLsplit*. This CA certificate is then used to sign on-fly created certificates for the visited hosts. Users are able to use their own CA certificate.

3.1 Bridge

This setup is needed in case of access to the upstream Ethernet cable before the intercepted LAN or Host (see Figure 3). The scenario is considered as default setup of the MitM probe. Bridge is created with Linux tool *brctl*, which will create a bridge between the 2 GLAN interfaces of the probe. It is necessary to connect an Ethernet cable with the Internet connection to eno1 interface and to the second interface enp2s0 the Ethernet cable with the connection to the intercepted network. The bridge acts as a virtual network switch working transparently, this behavior can be called transparent firewall. This way the probe will forward all the traffic from the client to the Internet and vice versa. With this setup, we have access to all packets transmitted from the clients and servers without the need to use any additional tools (e.g *arpspoof*) to redirect the traffic into our interfaces. For the correct behavior it is necessary to start *dhclient* process as well, which will enable clients to automatically acquire IP addresses.

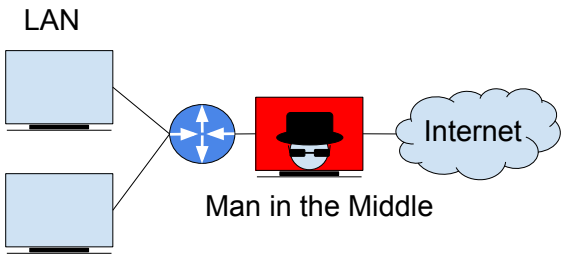


Figure 3. An illustration of the MitM on LAN

3.2 Arpspoof

Arpspoof is required to use, in case we have no access to the upstream network, instead, we have access to LAN connection (see Figure 4). The client and the MitM are connected to the default gateway. The default gateway is a network device which connects LAN with the Internet. It forwards the traffic from the local subnet to the other connected subnets. The tool is spoofing Address Resolution Protocol (ARP) messages to fake the ARP records in ARP tables of devices in the network. ARP is used for resolution of IP address to a physical Ethernet (MAC) address [9]. The IP address of the gateway is then mapped to MAC address of attacker's interface rather than the original MAC address of the gateway. It allows users to choose whether to spoof the whole LAN or a specific IP address as well. It is not recommended to attempt to spoof busy LAN networks with an insufficient (slow) network card, due to possible denial of service on the network or rapid

[8] https://www.archlinux.org/

slowdown of the network. This approach is unfortunately not as fast, reliable and transparent as the bridge solution.

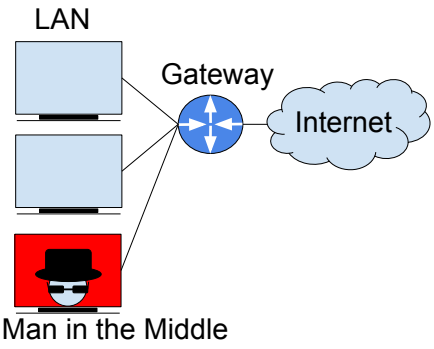


Figure 4. An illustration of the MitM on LAN

3.3 Iptables and Ip6tables

They are ”administration tools for IPv4/IPv6 packet filtering and NAT” and ”are used to set up, maintain, and inspect the tables of IPv4 and IPv6 packet filter rules in the Linux kernel.” [10]. It is able to redirect the incoming and outgoing traffic to the specified port. I have used this tool to create redirection of all HTTPS (tcp port 443) and HTTP (tcp port 80) network traffic to the address and port where the *SSLsplit* is listening and waiting for incoming traffic. Besides the use of *iptables*, users have to enable a packet forwarding on the used interface in order to enable a communication between clients and servers.

3.4 Tcpdump

This tool is used to capture all the traffic of the intercepted network. It uses pcap file format readable by a packet capture library libpcap[9], which is used in network traffic analyzer tools, e.g., Wireshark. All captured data will be stored on the SSD disk. File size is limited to 100MB per file. Users can set a limit for a number of stored files and after reaching max file count, it will start rewriting the created files from the oldest file. There can be set a filter on packets as well to eliminate unnecessary packets and capture just relevant ones, e.g., TCP and TLS packets. This is useful in case of low disk space or for a longer monitoring.

3.5 SSLsplit

For the MitM attack I have chosen the *SSLsplit* tool, which is written in C programming language, thus it is very fast. It supports all TLS and TCP traffic, compared to other mentioned tools, which provide only HTTPS and HTTP support. The traffic redirection is done by linux firewall called *iptables* and *ip6tables* for ipv6 to *SSLsplit*'s address and port. For HTTPS traffic over ipv4, it is address 0.0.0.0 and port 10443, for traffic over ipv6, it is listening on address ::1 and port 10443. The same addresses are used for HTTP traffic, however, there is a difference in used port number 10080. As an NAT engine is used *tproxy*, due to its support of ipv4 and ipv6. It is the only engine with ipv6 support on Linux OS, however on FreeBSD, OpenBSD there are few more, e.g., *pf* or *ipfw*. Aforementioned generated CA keys, i.e., ca.crt, ca.key, ca.pem, are used as root authority keys in *SSLsplit*. It is necessary to use private key with parameter -K (should be *.key or *.pem key format) in order to be able to decrypt captured packets. However, this is only possible when cipher suites without DH key exchange algorithm are used, nonetheless, the modern client applications prefer DH key exchange cipher suites. The script provides two solutions to this problem. The first solution forbids DH cipher-suties and offers only ciphersuites using RSA key exchange algorithm to the client application. Another solution is to log *master secrets* of sessions and use them for decryption of the captured packets. I had to implement this feature into *SSLsplit*, and now it can log the *master secrets* with parameter -A into the file. This is the second solution providing the ability to decrypt all the captured data regardless the used key exchange algorithm.

4. Experiments

The basic experiment is to connect MitM probe between the upstream and downstream network (see Subsection 3.1) and run the *SSLsplit* by the aforementioned python script. Clients will get warning about untrusted certificate used by the visited host in the address bar of the web browser (see Figure 5) and there will be a more explanatory warning instead of a visited web page content. This warning will provide the user with

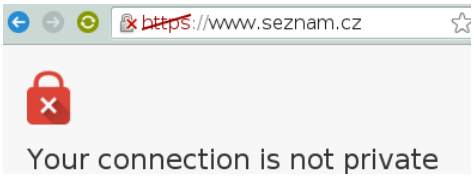


Figure 5. Insecure HTTPS connection

advanced options. There is an option during the visit of the ordinary host without HSTS to accept the untrusted certificate and proceed to the web page with the security risks (see Figure 6) example of web host without HSTS is a *www.seznam.cz*. This warning will be present every time the client visits an unvisited webpage, during the MitM attack. In case, the host is preloaded in the HSTS list of the web browser there is

[9] http://www.tcpdump.org/manpages/pcap.3pcap.html

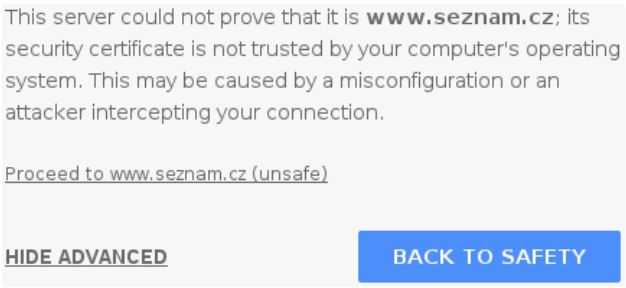


Figure 6. Warning with option to continue with insecure connection

not any option to accept and proceed to the web page. This behaviour can be observed during an attempt to visit *google.com* under MitM attack (see Figure 7). The warning about untrusted certificate disappears at

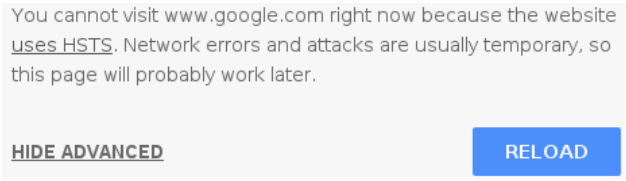


Figure 7. Warning without option to continue to HSTS web page over insecure connection

the moment the client imports the CA authority inside of his OS or client application. There is the possibility to use HSTS bypass from the aforementioned *bettercap* as well. However, the connection is just over the plain unecrypted HTTP (see Figure 8). Captured pcap

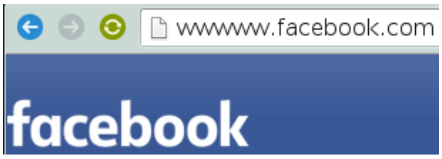


Figure 8. HSTS bypass on *wwwww.facebook.com*

files can be opened and decrypted with commonly used tool Wireshark along with the created private key or with the captured SSL *master secrets* log file. The *master secrets* log file or RSA keys can be applied in the Preferences - Protocols - SSL menu. Decrypted data of the SSL/TLS packets can be examined with the Wireshark. Users have to select the Decrypted SSL data option in the selected packet (see Figure 9). The decrypted HTTP data are very usefull in the further analysis, where we can discover what are the subjects chatting about, doing on the Internet and intercept user credentials and sessions. The application layer data can be examined with the option *Follow SSL stream* (see Figure 10), where it is possible to see the decrypted raw HTTP protocol data.

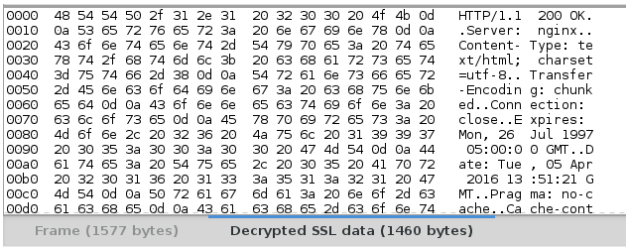


Figure 9. Decrypted SSL packet

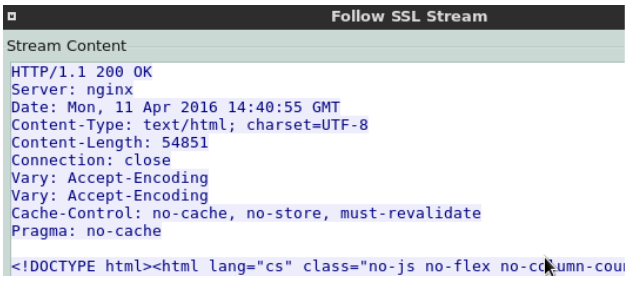


Figure 10. Decrypted HTTP response in Wireshark

5. Contributions

I Made a research about abilities of the MitM tools and created comparison in the survey (see Figure 2). Choose the best tool for the automatization which support the most of the properties. Implemented the master secrets logging into *SSLsplit*, in order to solve the problem with DH key exchange. Created and preinstalled a script into MitM probe with a support for most of the aforementioned properties. The script introduces an ability to work as a transparent proxy and ability to automatically capture the data into pcap file format, which is the most used format in network analysis. Last but not least the script and MitM probe is very easy to use.

6. Conclusions

The paper describes protocols SSL/TLS and MitM attack aimed to intercept SSL/TLS encrypted traffic. Scenarios used to decrypt the SSL/TLS traffic with private key or with the *master secrets* log file of SSL/TLS sessions. Some of MitM attacks and tools, e.g., *SSLsplit*, *mitmproxy*, and *MITMf* capable of performing the MitM attack. Hardware and software setup of MitM probe, which is used to automatize SSL/TLS interception. There was a demonstration how to connect the probe to the LAN by usage of arpspoof or to the upstream network by usage of the bridge.

There was an illustration how to get to deciphered packets of the SSL/TLS traffic. Variations of the MitM attack and their results, e.g, web browser warnings (see Figure 6) and restrictions (see Figure 7) were shown. The paper illustrates what HSTS is and how HSTS can be partially defeated by *MITMf* and *bettercap* as well.

In order to set up a MitM probe I had to research

and learn how to use all mentioned tools used in the script. I have selected HW and OS. Installed the OS with necessary tools. Created script for automatic start of the attack. The script deals with DH key exchange by forcing the RSA key exchange. I have implemented *master secrets* logging into the *SSLsplit* as well.

In the future, the python script for automatization should support HSTS bypass (see Subsection 2.2) and JavaScript keylogger as well. These functionalities should be implemented by adding corresponding modules from *MITMf* and from other open-source tools.

Acknowledgements

I would like to thank my supervisor Ing. Jan Pluskal for his help.

References

[1] Ivan Ristic. *Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications*. Feisty Duck, 2014.

[2] Tim Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[3] Eric Rescorla. Diffie-hellman key agreement method. 1999.

[4] Collin Jackson, Adam Barth, and Jeff Hodges. Http strict transport security (hsts). 2012.

[5] Sherri Davidoff and Jonathan Ham. *Network forensics: tracking hackers through cyberspace*. Prentice hall New Delhi, 2012.

[6] Daniel Roethlisberger. Sslsplit - transparent ssl/tls interception. https://www.roe.ch/SSLsplit, April 2016.

[7] Aldo Cortesi. mitmproxy. https://mitmproxy.org/.

[8] H. Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 1980.

[9] Bruce Hartpence. *Packet guide to core network protocols*. O'Reilly Media, Inc., 2011.

[10] Herve Eychenne. iptables man page, 2002.