

Review scientific paper/Pregledni naučni rad

TESTING THE DATA PROTECTION THROUGH IMPLEMENTATION OF SECURE COMMUNICATION

TESTIRANJE ZAŠTITE PODATAKA KROZ IMPLEMENTACIJU SIGURNE KOMUNIKACIJE

Suad Kasapović¹, Emir Skejić¹, Tarik Huremović²

Abstract: The security of using applications in cloud services and on the Internet is an important topic in the field of engineering. In this paper, two laboratory tests for data transmission protection, specifically designed for different security analysis techniques, are presented and explained. During lab tests on public Wi-Fi networks from the MIDM ("Man in the Middle") attacks, various monitoring techniques were applied, using a special lab test scenario with Kali Linux penetration tools by creating an SSH tunnel on an Android mobile device. These test benches allow easy data capturing, and the captured data is processed using available software programs. Expected outcomes, practical improvement and security performance assessment are presented in detail, and considered in terms of their value in security engineering. The aim of this paper is to detect and overcome some of the weaknesses of the application of security protocols in a Wi-Fi network environment.

Keywords: data protection, Kali Linux, security, penetration, network

Sažetak: Sigurnost upotrebe aplikacija u uslugama u oblaku i na Internetu je važna tema u području inženjerstva. U ovom radu predstavljena su i objašnjena dva laboratorijska testa za zaštitu prijenosa podataka, specijalno dizajnirana za različite tehnike analize sigurnosti. Tokom laboratorijskih testova na javnim Wi-Fi mrežama od MIDM ("čovjek u sredini") napada, primjenjivane su različite tehnike monitoringa, koristeći specijalni laboratorijski test scenario s alatima za Kali Linux penetraciju kreiranjem SSH tunela na Android mobilnom uređaju. Ove testne naprave omogućuju lako prikupljanje podataka, a prikupljeni podaci se obrađuju pomoću raspoloživih softverskih programa. Očekivani ishodi, praktično poboljšanje i procjena performansi sigurnosti su detaljno predstavljene i razmotrene u smislu njihove vrijednosti u sigurnosnom inženjerstvu. Cilj ovog rada je otkrivanje i prevladavanje nekih nedostataka primjene sigurnosnih protokola u Wi-Fi mrežnom okruženju.

Ključne riječi: zaštita podataka, Kali Linux, sigurnost, penetracija, mreža

INTRODUCTION

Organizations that implement wireless technology, often do not take into account all safety aspects of that implementation, and reasons for that are lower cost devices, simple implementation and improved productivity. Wireless networks are very often used as a tool to get free Internet access or to hide identity. It is also an opportunity for access to a network that would otherwise be difficult to attack from the Internet. Unlike a wired network, a WLAN (Wireless Local Area Network) sends data over the air and might be available outside the physical boundaries of the organization. When WLAN data is not encrypted, the packages can be viewed by anyone in the radio frequency range. For example, a person with Kali Linux (kernel 3.12) distribution on a laptop, using a program such as Driftnet, URLsnurf, SSLstrip and others, can receive,

view and store all packets in a particular WLAN. From the very beginning of computer networks, there are certain types of so-called "Man in the Middle" (MIDM) attacks. Like computer networks, these attacks have evolved with development of different systems for protection. The aim of this paper is to test the protection of data transmission in wireless networks (WLAN) via frequently used secure protocols. A security of a SSL (Secure Sockets Layer) protocol is based on the use of cryptographic methods that provide protection (confidentiality) of data moving through an insecure network [1] - [3]. Besides that, these methods can be used to verify the identity of users who participate in communication and data protection from an unauthorized modification, or to preserve their integrity.

Possible attack scenario in the WLAN and implementation of one of the solutions of adequate protection for such attacks, as well as a practical implementation of SSH (Secure Shell) protocol at the application level in Android environment will also be presented later in this paper.

The motivation to test network security protocols is the use of sensitive personal applications on Android mobile phones in WLANs, such as airports, cafes, restaurants, etc.

¹ University of Tuzla, Faculty of Electrical Engineering, Bosnia and Herzegovina

² Raiffeisen Bank dd BiH, Bosnia and Herzegovina
suad.kasapovic@untz.ba

Paper submitted: Februar 2021 Paper accepted: April 2021

A large amount of the data is being transmitted over the Internet or local network as unencrypted plain text and can be intercepted by anyone with a little bit of technical computer knowledge. SSH is a really simple, not demanding, but very powerful and robust solution for most problems of security in communications networks. In this context, the SSH protocol has been developed to replace existing insecure protocols and data encryption protection, and it is designed as a model of client/server way of communication. Authors suggest using a secure shell protocol if you want to use very sensitive personal data and applications in wireless networks. Authors also wanted to show scenarios when careless use of common security protocols can lead to data interception and open the possibility of their misuse.

This article is organized as follows: Section II includes basic mechanisms of protection that can be applied via SSH protocol and the existence of other alternatives in the market, and describes the principle of performance MIDM attack by malicious users and method of protection through SSH Android application.

Section III includes results of MIDM attack without SSH protection as well as results of the attack after applying SSH tunnel with Android application. Conclusions and results of scenarios are given in Section IV.

1. SSH PROTOCOL AND ANALYSIS OF THE SECURITY OF NETWORK ARCHITECTURE

SSH is a protocol that is applied to the application layer and hence its implementation is much simpler and cheaper than any deployment at lower layers of the network, which is the main motivation of this study to investigate the implementation of data protection by Android application [4]. Thus the future of the SSH protocol is quite certain because of a simplicity of its implementation and can only be jeopardized by a directive or an influential standard that could require a mandatory encryption of data at lower network layers, therefore becoming meaningless data encryption at the application layer [5].

SSH is a specification that defines rules that say how to enable secure communication over the Internet as shown in Figure 1.

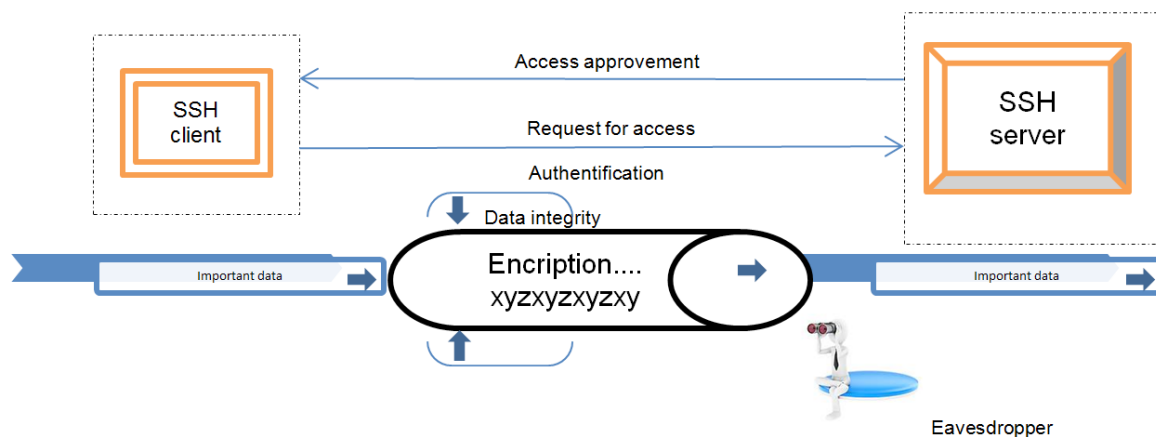


Figure 1: SSH – Authentication, Encryption and Integrity

One of the most useful features of SSH protocol is port forwarding [6]. In general, the port forwarding allows any unencrypted TCP (Transmission Control Protocol) connection to be transferred to a secure SSH connection. This also applies to SSH tunneling. It is necessary to know which port is used because each application that works on TCP has its own port number. One thing that is also important to mention is the fact that the SSH protocol is not introduced as a revolutionary new idea to protect communications in the insecure WLAN or LAN because it is based on the methods of encryption and authentication as other protocols such as SSL (Secure Socket Layer)/TLS (Transport Layer Security) and IPsec (Internet Protocol Security). All of these protocols in its operation go through, more or less, the same phase relating to the determination of the encryption algorithm, key exchange and authentication and encryption of traffic at the end of the authentication and integrity of the transmitted information [6]. What essentially distinguishes these protocols is a layer or level of the OSI reference system in which they work. SSL and TLS are protocols that are established in

the transport layer network communications, and IPsec operates at the network layer, which means that the level of protection is based on IP packets and its implementation is probably the most complex.

The easiest way to examine a network architecture security analysis is with proven penetration tools, using scenarios defined in recognized international standards for information security testing as specifications NIST (National Institute of Standards and Technology) 800-115. Kali Linux distribution is a complete platform for penetrating, and other safety tests carried out exclusively to its advanced tools to identify, uncover and present any security vulnerabilities in a computer network environment [7], [8]. Using tools of this Linux distribution, it is easy to perform standard MIDM attack [9]. These attacks are the most often carried out in three phases: collection of data on available devices on the network, the application of tools for the attack to a specific device or network component and analysis of the captured traffic using the available tools.

Table I shows two scenarios that need to be done.

Table I: Scenarios for laboratory testing

SCENARIO 1 – Collection of data traffic	SCENARIO 2 – Compromising data
OBJECTIVE:	OBJECTIVE:
Learn how to intercept unsecure data communication channel with Kali Linux penetration tools	Learn how to protect data from »men in the middle attack« described in previous steps with SSH tunneling Learn how to intercept secured (SSL) communication channel and how to collect sensitive user data.
PROCEDURE:	PROCEDURE:
-Scan WLAN network for available devices -Do "ARP (Address Resolution Protocol) spoofing" on target device -Redirect incoming and outgoing traffic between device and router to attacker device -Run tools for capturing redirected traffic -Analyse intercepted data with Kali tools	-Scan WLAN network for available devices -Do "ARP spoofing" on target device -Redirect incoming and outgoing traffic between device and router to attacker device -Run tool for compromising SSL session (SSLstrip) -Analyse intercepted data -Run and configure Android application to create secure SSH tunnel -Try to intercept data communication channel and capture data from attacked device like in previous steps

2. SSH PROTOCOL TESTING METHOD

2.1. Collection of data traffic – scenario 1

Let us consider the following network configuration presented in Figure 2, with used devices listed in Table II.

When it comes to unfamiliar WLAN network, the first action that each attacker can do is to establish a list of all available devices on the network. For this operation it is convenient to use NMAP tool, mentioned in NIST 800-115 standard.

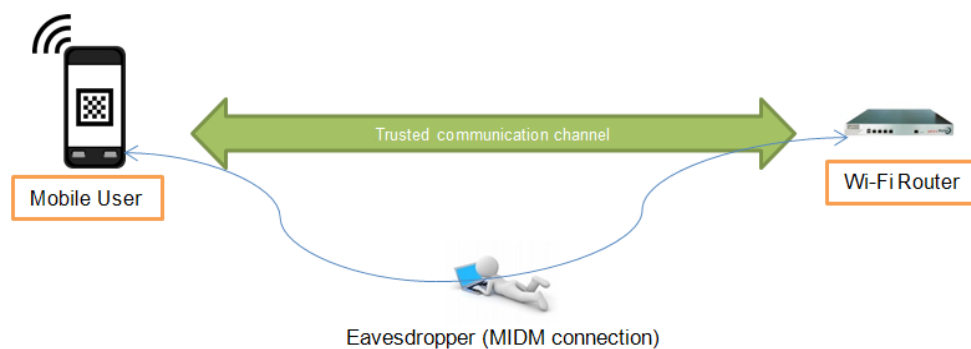


Figure 2: Schematic representation of MIDM attacks in WLAN

Table II: The list of devices used in the simulation

Network role	Device	MAC address	IP address in WLAN
Mobile user	Samsung S3 GT-I9300 Android 4.3	34:23:BA:4E:63:18	192.168.1.3
Wi-Fi Router/Modem	ZTE zxv10w300	B0:75:D5:3D:20:38	192.168.1.1
Attacker	Laptop Acer Aspire 5520 OS Kali Linux 3.12 32-bit	00:1d:d9:2f:96:29	192.168.1.6
WLAN	Interface = wlan0		

In order to scan addresses in the standard WLAN we can use the command *nmap*. Before the start of the attack, attacker usually performs a redirection of a network traffic in the way that all the traffic and information that users send to him (attacker) can be forwarded to a valid gateway that will handle the requests and responses submitted to the user. It can be done by simply setting the value of the *ip_forward* file to "1". After redirection of all the network traffic is set, we need to launch ARP Spoofing and capture traffic that goes to the router and from the router to the user so it is spoofing the user and the gateway. This way, the data collection includes the complete data traffic in communication with mobile device and the router. The traffic can be analysed with a variety of tools, and the following tools have been used in the simulation: *URLsnarf* – tools with possibility to see any URL that a user types and visits (e.g. `root@kali:~#urlsnarf -i wlan0`) and *Driftnet* – tools with possibility to pick up the graphic content on the site (e.g. `root@kali:~#driftnet -i wlan0`). After starting the tools that are in standby mode, and listening to these ports and interface wlan0 we only need to go on the device, which is the subject of analysis, launch the Internet browser and enter the address of a website. Figure 3 shows mobile device that

has the address 192.168.1.3 and entering webpage `www.arsenal.com`. The content of the browser on mobile device I9300 is on the left, and a comparative view of the content capturing tools *URLsnarf* (upper panel) and *Driftnet* (lower panel) is on the right. The Figure 3 shows that the *URLsnarf* picked up the entire HTTP (Hypertext Transfer Protocol) content, or both GET and RESPONSE traffic, and it is possible to read the complete links from browser of the mobile user and the data from the device about version of Android operating system, while *Driftnet* picked up most of the multimedia content from the site. Kali Linux penetration tool kit contains another interesting tool that can be used to conduct an analysis of the collected traffic, online or offline from the recorded log files. At first glance, analysing non-encrypted HTTP traffic may not represent serious threat for most users, and we can always rely on the fact that all major Internet services, such as electronic banking, e-mail accounts, social networks and other services that enable access to confidential private data, are secured by SSL encryption. In attacks based on the principle of MIDM, attackers often do not want the user to be aware of the attack and do not want to prevent users in regular activities and the use of these Internet services.



Figure 3: Comparison of the browser of a mobile user and display the captured traffic using *URLsnarf* and *Driftnet* tools Kali Linux (left mobile phone I9300, right Kali Linux tools)

2.2. Compromising data – scenario 2

The example in Figure 4 will show how you can not necessarily rely on SSL encryption of traffic and access to well-known Internet services via HTTPS (HTTP SSL) connections because they can be compromised. As in the previous example, this scenario was performed on a standard configuration WLAN, but this time the attack tool is *SSLstrip*, which comes with a group of tools for information gathering and analysis of SSL. *SSLstrip* is a tool that

allows unencrypted HTTP sessions to look like protected HTTPS sessions. The tool "converts" the HTTPS links to HTTP links or HTTPS links with known private key.

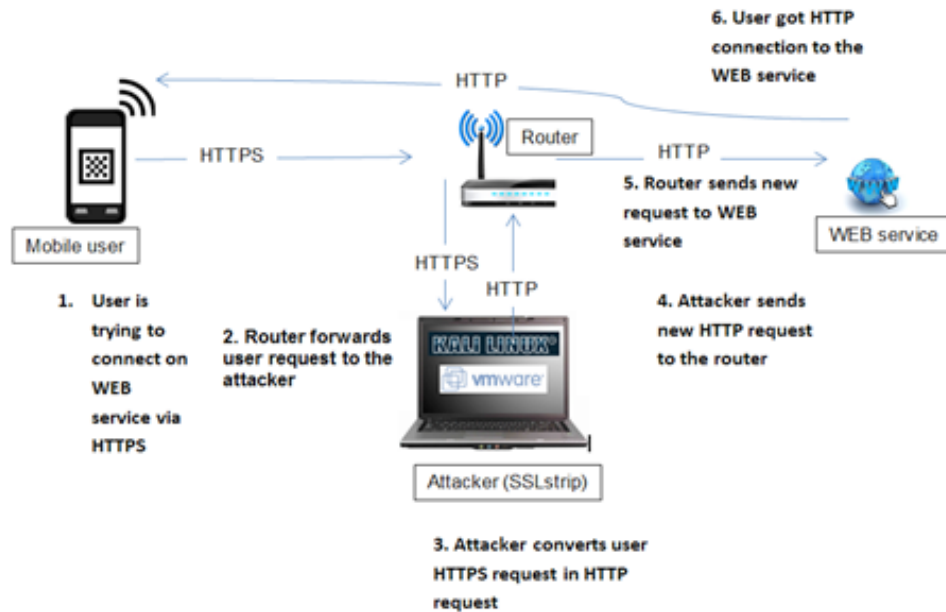


Figure 4: Schematic representation of SSLstrip MIDM attacks in WLAN

Many HTTPS pages and services enable normal access via HTTP requests, while it is quite difficult to see from the user side that it is actually HTTP traffic, not HTTPS traffic. In this case, SSLstrip actually establishes a regular connection with a valid HTTPS server, but also establishes a parallel HTTP connection to the user, which the attacker will be able to easily monitor. The working principle of SSLstrip tool may be easily explained by the following example of connecting to the Gmail service:

- The user enters a link `http://mail.google.com` and the web browser tries to connect to `port 80` using HTTP.
- Server establishes a redirection of client requests to the HTTPS version of the site using HTTP 302 code for redirection and the client connects to `https://mail.google.com` on `port 443`.
- Server generates a certificate to the client containing its digital signature, and the client takes this certificate and compares it to its list of trusted certification authority.
- If the process is completed successfully then establishes a secure connection.

The process above was considered extremely safe until recently, but security expert and ethical hacker Moxie Marlinspike presented SSLstrip tool. SSLstrip attacks exactly this "bridge redirections" between unsafe and safe communication because users rarely enter HTTPS links directly, because links are usually entered through HTTP 302 redirects or by calling a submit button. SSLstrip exploits a weakness at the point of transition session between HTTP and HTTPS in a way that SSLstrip begins normal exchange of the certificates to the server, and then converts all HTTPS links to HTTP and makes a map of all changes so he can monitor the traffic between the server and the client and to be invisible. The only difference that the client can see is the disappearance of the safety lock, and that a HTTP link is placed instead of HTTPS link in the address field of the browser until the server notices the difference between the initial user requests and SSLstrip

requests. As in the previous example, before the start of redirecting network traffic it is necessary to set up file `ip_forward` to the value "1".

The second line of code is intended to establish a routing HTTP traffic to SSLstrip. IPTABLES is a user-space application program that allows a system administrator (root) to configure the tables provided by the Linux kernel firewall and the chains and rules it stores. Simply shown, this line of code adds a new IPTABLE rule to PREROUTING chain, which tells the attacker's device that accepts incoming traffic, which means in particular that all traffic intended for `port 80` (HTTP, Web traffic) is redirected to `port 8080` where we will later setup SSLstrip to the listen port. The rule is set to PREROUTING because these packages are included in the current chain before making a decision about routing these packages. Once we have established this rule is necessary to begin listen `port 8080` with the program SSLstrip.

The last thing that we need to do is the same as in the previous example because it is necessary to modify the ARP values or initiate ARP spoofing to convince the network to begin to send all traffic to the target device:

```
root@kali:~# sslstrip -l 8080
root@kali:~# echo 1 > /proc/sys/net/ipv4/
ip_forward
root@kali:~# iptables -t nat -A PREROUT-
ING -p tcp --destination-port 80 -j REDI-
RECT --to-port 8080
root@kali:~# arpspoof -i wlan0 -t
192.168.1.3 -r 192.168.1.1
```

After the preparation of an attack, it is necessary to go to familiar web services that have SSL protection. Figure 5 shows the example when *Facebook* and *mail.google.com* services are accessed before and after the attack with SSLstrip software tools [10], [11].

Table III: Device in list used the simulation with the addition of Linux AWS SSH server

Network role	Device	MAC address	IP address
Mobile phone	Samsung S3 GT-I9300 Android 4.3	34:23:BA:4E:63:18	192.168.1.3
Wi-Fi Router/Modem	ZTE zxv10w300	B0:75:D5:3D:20:38	192.168.1.1
Attacker	Notebook Acer Aspire 5520 OS Kali Linux 3.12 32-bit	00:1d:d9:2f:96:29	192.168.1.6
WLAN	Interface = wlan0		
SSH server	AWS – Amazon Web Services EC2 t2.micro ubuntu-trusty-14.04-amd64-server		52.19.17.81

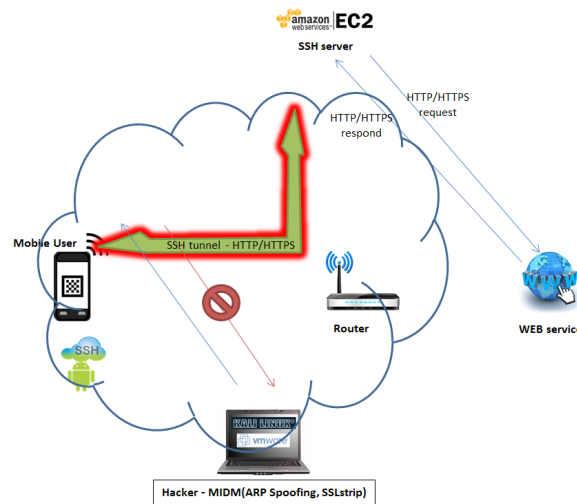


Figure 7: Schematic diagram of the network structure with the addition of SSH server (AWS EC2)

The first phase of the simulation is done as in previous simulations; launch the attacking tools ARP spoofing and SSLstrip, and then the tools to analyse the collected data (URLsnarf and Driftnet). On the running Linux terminals, the analysis tools are displaying and filtering the traffic caught from mobile device GT-I9300 which is sent and received from the Internet.

In the second phase of a simulation, when the device GT-I9300 is still connected to the same network, interface wlan0 runs the Android "SSH aplikacija" with the options checked as in Figure 8.

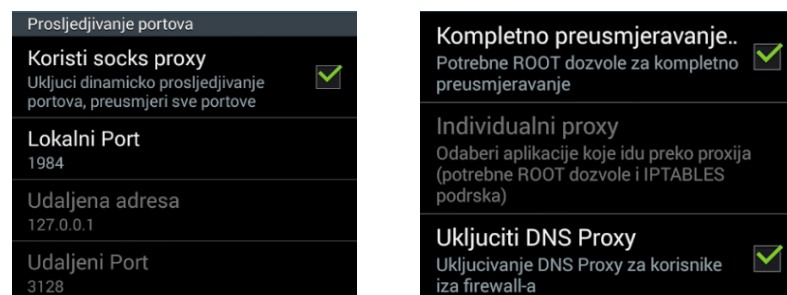


Figure 8: The options to turn SOCKS proxy and forward complete traffic and DNS requests through tunnel

Besides establishing the SSH tunnel it is necessary to "tell" what the tunnel is supposed to serve. Using local SOCKS proxy server (to 127.0.0.1) allows the establishment of a transparent HTTP redirector, which serves to secure transmission of the network traffic between the client and server. SOCKS actually define a standard mechanism for a client that can connect to the server via a proxy [12].

Port forwarding is a very good possibility. If we find ourselves in a public wireless network which is blocking our essential services through the firewall, such restrictions can simply be bypassed via port forwarding. If, for example, we cannot access the port 25 through the mail server *moj.email.ba* then just via SSH tunnel we can do the following command:

```
ssh -L 2525:moj.email.ba:25 52.19.17.81
```

This way, through the local *port 2525*, which is not forbidden by the firewall, we can perform the translation of connection via the SSH server at *52.19.17.81*, which has not banned *port 25*, and regularly access the mail service. This is a standard way of local port forwarding where we need to know the server address and both ports. This can be done even more efficiently via a SOCKS proxy using dynamic forwarding:

```
ssh -D 1984 52.19.17.81
```

With this approach, we make the connection to *52.19.17.81* and raise a proxy on the local host (mobile device) on *port 1984*. After this, we can use all the applications that are able to use a SOCKS proxy. Anyway, SOCKS protocol is defined in the session level of the OSI reference system.

For this simulation scenario an instance of Ubuntu Server is being created on Amazon EC2 Cloud solution for creating virtual servers. Initially, without additional configuration, the SSH server on this Linux AWS instances is configured and started with open *port 22* for a traffic to the server. In order to be able to connect to the SSH server, you need to enter data relating to user authentication. The authentication is performed by a pair of authentication keys (private and public). Since it is also the only possible way for the authentication to server instances run through Amazon AWS, the only extra work that was made is conversion of the original AWS key is in "PEM" extension to OpenSSH format keys and a key store to the correct location on the internal "SD card" on mobile device (to allow application to properly read "keyPath" variable). Generic user which is used to access to Ubuntu micro instances AWS server is "Ubuntu" and in the settings it can be seen that login account is without entered password, only with the pair of keys [13] - [15]. After adjusting the options above the only thing we need to do is to activate SSH tunnel via option "Prekidac za tunel" (the switch) in Figure 9.

Before starting up the SSH tunnel, the current IP address of the router has been checked: *89.146.160.178*. In order to write to a log, device is connected to the Android Studio and application is run in debug mode.

After starting the tunnel by clicking on the "Prekidac za tunel (Enable/disable SSH Tunnel)" program started to track using Logcat, a mechanism for the collection and analysis of logs in Android Studio environment. By monitoring the sequential execution of program commands

with subsequent filtering we can see the most important commands when creating an SSH tunnel, and then the establishment of a SOCKS proxy.

The procedure of establishing an SSH tunnel begins with the start of SSHTunnelService's with forwarded parameters needed to establish the connection. Once the connection starts, next important command on the side of SSH tunnel is:

```
iptables -t nat -D OUTPUT -p udp --dport 54 -j REDIRECT --to 8154
```

IPTABLES, in this case, is the program structure at the level of Android (Linux) kernel, which contains a set of rules for filtering and management of network packets (IPv4). The rules are contained within the table, and each tablet contains chains of rules. Chains of rules are in fact the list of rules that apply to certain groups of packages. There are built-in chains (PREROUTING, OUTPUT, POSTROUTING), and we also can create a user chains with command N. This command means that using NAT rules table (-t nat) we delete the rule in the OUTPUT chain (for local packages) for UDP protocol, where destination *port 54* is redirected to *port 8154*. It is logged that authentication is done using a public key (line in the log "Authenticate with public key"), which is concluded with the message "Forward Successful". After that, the SOCKS proxy is started with rules defined in shell script "proxy_socks.sh" on the local *port 1984*. After that, the program executes the command to create new chains: SSHTUNNEL and SSHTUNNELDNS with commands N (new chain) and F (flush – initially clear all rules in new chain). Since in the options of the application "DNS proxy" option is checked, the command is executed:

```
iptables -t nat -A SSHTUNNELDNS -p udp --dport 53 -j REDIRECT --to 54351
```

This command adds a new rule in the chain SSHTUNNELDNS for UDP protocol where it is said that the destination *port 53* is redirected to *port 54351*. Then on the built-in OUTPUT chain is added for the UDP protocol chain SSHTUNNELDNS which currently contains only previously created rule.

```
iptables -t nat -A OUTPUT -p udp --j SSH-TUNNELDNS
```

The following set of rules is added (Command A - append) on the chain SSHTUNNEL:

Postavke SSH tunela	
Host	52.19.17.81
Port	22
Informacije za account	
Korisnik	ubuntu
Sifra	SSH sifra

SSH aplikacija	
Kontroler servisa	
Prekidac za tunel	
Ukljuci/Iskljuci SSH Tunel	☐

Figure 9: Authentication settings on the SSH server

```
iptables -t nat -A SSHTUNNEL -p tcp !
-d 52.19.17.81--dport 80 -j REDIRECT --to
8123
iptables -t nat -A SSHTUNNEL -p tcp ! -d
52.19.17.81--dport 443 -j REDIRECT --to
8123
iptables -t nat -A SSHTUNNEL -p tcp --dport
5228 -j REDIRECT --to 8123
```

The first two rules say redirecting all TCP packets with destination *port 80* (HTTP) and *443* (HTTPS) on *port 8123* except those packages that come from address *52.19.17.81*, which is actually the address of the SSH server. As in the case of SSHTUNNELDNS chain, on built-in OUTPUT chain, the chain SSHTUNNEL is added for the TCP protocol, which currently contains the three rules added earlier.

```
iptables -t nat -A OUTPUT -p tcp --j SSH-
TUNNEL
```

After these rules, the procedure of establishing the connection ends. Finally, as evidence that the mobile device, which is still in the local WLAN, has an IP address of AWS SSH server, is the following: *52.19.17.81*. Switching the SSH tunnel on mobile device, on penetration tools previously used in the simulation (DRIFTNET, URLsnurf, SSLstrip), stops any activity in the form of interception of traffic that is sent to or coming to mobile devices, because the ARP values changed between the mobile device and router, in the WLAN at *192.168.1.1*, but the new output to the Internet is address of SSH server (*89.146.139.129*), by which we fully protect the outgoing and incoming communication. As the MIDM attacks are most often used attacks on public wireless networks, this principle of the protection on mobile devices can be quite sufficient level of security to our private, financial and other sensitive data.

4. CONCLUSION

The main purpose of the experimental part of this paper is to show how vulnerable wireless communications are in present days and how we can protect our mobile devices with a little knowledge of SSH protocol mechanisms. The focus of the work is based on SSH protocol because of its ease of deployment and very little investment in infrastructure. Android application is made on the basis of open source solutions and software packages. Based on the preliminary design proposed in this paper it can be beneficial whether you're a private user or a financial institution that wants to provide its customers, in every situation, secure access to services regardless of where they are with their mobile device. For the ordinary user, it is enough to raise the SSH application server in his home Wi-Fi network, provide access to the Internet through a router (by passing the selected port) and via the Android SSH application access whenever he finds himself in a situation where he cannot provide secure access to the Internet or whenever he is connected to a public WLAN. Corporations, especially financial ones, can offer access to a personal SSH application server to its customers and thus strengthen their confidence so that clients can feel safe when using any of the e-commerce

websites or access their accounts, credit cards, or simply enter their personal information that no third person who has access to what they do is involved in their communication.

Testing the security of the communication channel (tunnel), which was established between the SSH application and the SSH server, is done by Kali Linux penetration tool which is supported by the official PCI DSS (Payment Card Industry Data Security Standards) standards. Proof of this is written in the document Meet PCI DSS Requirements with FOSS (Free and Open Source Software), which reads about using Open Source solutions for compliance testing at the PCI-DSS standard and which is in line with the requirements of the provisions of the "Requirement 11.3.2", where it is said that Kali Linux distributions can be used as a valid and relevant penetration tool. During simulation, tools which were used for MIDM attacks have not recorded any logs or traffic activity when a tunnel, using SSH application and SSH server had been established, while the activities of capturing traffic between the mobile device and the router in the WLAN had been recorded immediately after the tunnel was lifted, which serves as a proof that communication security has been realized through SSH application tunnel.

REFERENCES

- [1] W. Stallings: Network Security Essentials: Applications and Standards, 4th Edition, Prentice Hall Press, Upper Saddle River, NJ, USA, pp. 170-188, 2010
- [2] Z. Golebiewski, M. Kutylowski, F. Zagorski: Stealing Secrets with SSL/TLS and SSH – Kleptographic Attacks. In: Pointcheval D., Mu Y., Chen K. (Eds) Cryptology and Network Security, CANS, 2006
- [3] Lecture Notes in Computer Science, vol 4301 Springer, Berlin, Heidelberg. https://doi.org/10.1007/11935070_13
- [4] M. Stahnke: Pro OpenSSH, Apress, pp. 147-164, 2006
- [5] N. Smyth: Android Studio Development Essentials, eBookFrenzy, pp. 107, 201, 2015
- [6] R. Cameron, N. R. Wyler: Juniper(r) Networks Secure Access SSL VPN Configuration Guide, April 2011, Syngress, ISBN: 9780080556635
- [7] D. J. Barrett, R. Silverman: SSH, The Secure Shell: The Definitive Guide, O'Reilly, 2nd Edition, pp. 39-49, 2005
- [8] T. Ylonen: The Secure Shell (SSH) Transport Layer Protocol, Informational RFC 4253, IETF, Network Working Group, 2006
- [9] D. W. Dieterle: Basic Security Testing with Kali Linux, eBook, ISBN-13: 978-1494861278, pp. 185-190, 2013
- [10] T. Anjali, S. Krishna, Z. Poonam: Comparison between SSL and SSH in Network and Transport Layer, International Journal of Computer Science and Information Technologies, Vol. 6 (6), pp. 5432-5434, 2015
- [11] A. Garimella, D. R. Kumar: Secure Shell – Its Significance in Networking (SSH), International Journal of Application or Innovation in Engineering & Management, Volume 4, Issue 3, pp. 187-196, March 2015
- [12] R. W. Beggs: Mastering Kali Linux for Advanced Penetration Testing, Packt Publishing Ltd., pp. 258-278, 2014

- [13] T. Ylonen, C. Lonvick: The Secure Shell (SSH) Protocol Architecture, RFC 4251, January 2006
- [14] Secure Shell Configuration Guide, Cisco IOS Release 15S, Americas Headquarters Cisco Systems, Inc., USA, 2015
- [15] M. Souppaya, K. Scarfone, L. Feldman: Securing Interactive and Automated Access Management Using Secure Shell (SSH), National Institute of Standards and Technology U.S. Department of Commerce, January 2016

BIOGRAPHY

Suad Kasapović was born on 21 February 1970 in Tuzla, Bosnia and Herzegovina. He received his PhD degree in technical science from the field of electrical engineering in 2007 at the Faculty of Electrical Engineering and Computing, University of Zagreb. Since 2003, he has been employed at the Faculty of Electrical Engineering at the University of Tuzla, where he is currently the associate professor in the field of Telecommunications and Informatics.

Emir Skejić was born on 16 January 1977 in Dobož, Bosnia and Herzegovina. He received his PhD degree in technical science from the field of electrical engineering in 2007 at the Faculty of Electrical Engineering, University of Tuzla. Since 2001, he has been employed at the Faculty of Electrical Engineering at the University of Tuzla, where he is currently the associate professor in the field of Computer and Information Science.

Tarik Huremović was born on 01 December 1979 in Tuzla, Bosnia and Herzegovina. He received his MSc in technical science from the field of Electrical Engineering in 2015 at the Faculty of Electrical Engineering, University of Tuzla. Currently, he works as the senior developer at Raiffeisen Bank dd BiH.