Macrothink Institute™

Network Protocols and Algorithms
ISSN 1943-3581
2018, Vol.10, No.1

Survey of the Protection Mechanisms to the SSL-based Session Hijacking Attacks

Md. Shohrab Hossain, Arnob Paul, Md. Hasanul Islam

Department of CSE, Bangladesh University of Engineering and Technology, Bangladesh

E-mail: arnobpl@gmail.com, hasanuli10@gmail.com, mshohrabhossain@cse.buet.ac.bd

Mohammed Atiquzzaman

School of Computer Science, University of Oklahoma, Norman, Oklahoma, USA

E-mail: atiq@ou.edu

Received: January 13, 2018 Accepted: March 28, 2018 Published: March 31, 2018

DOI:10.5296/npa.v10i1.12478 URL: http://dx.doi.org/10.5296/npa.v10i1.12478

Abstract

Web communications between the server and the client are being used extensively. However, session hijacking has become a critical problem for most of the client-server communications. Among different session hijacking attacks, SSL stripping is the most dangerous attack. There are a number of measures proposed to prevent SSL tripping-based session hijacking attacks. However, existing surveys did not summarize all the preventive measures in a comprehensive manner (without much illustration and categorization). The objective of this paper is to provide a comprehensive survey of existing measures against SSL stripping-based session hijacking attacks and compare those measures. In this paper, we have classified all the existing preventive measures for SSL stripping-based session hijacking attacks into two main categories: client-side measures and server-side measures. We have illustrated the proposed solutions comprehensively with useful diagrams for clarification. We have also compared them based on different performance criteria. This paper will help web security researchers to have a comparative analysis of all solutions for the SSL stripping based attacks, thereby improving existing solutions to better protect the users from session hijacking attacks.

Keywords: Session hijacking, SSL stripping, Man-in-the-middle attack, HTTPS.

1 Introduction

Session is an exchange of information between two or more devices. Every session must have a session ID to identify a session. Session IDs are typically granted by a server to its

clients. Session hijacking involves knowing that session ID to masquerade as authorised user. Once the session ID is known, the attacker can do anything as the authorised user can do on the server. Session ID is usually stored within a cookie. Cookie is a small chunk of private data. When a client browses any website, web server sends this data. Browser resend this private data to server to keep track of activity of client whenever client reloads this web page. Every time user loads the website, the browser sends the cookie back to the server to notify user's previous activity. If a web page uses HTTP, this cookie can easily be sniffed as plain text by a sniffing [1] tool (e.g. Wireshark [2], Ettercap [3] etc). But to protect this man-in-the-middle (MITM) attack, the information should be encrypted so that no attacker can retrieve the actual text. HTTPS (HTTP over SSL/TLS [4]) provides the authentication of visited website and the protection of privacy (secrecy) and the integrity of exchanged information. HTTPS protects against MITM attack and eavesdropping the information between server and client.

SSL (Secured Sockets Layer) or TLS (Transport Layer Security) is a security protocol that secures data on the Internet everyday while transmitting confidential information. It provides privacy and data integrity by encrypting information that are passed between web server and web client (e.g. browser). SSL-secured web addresses begin with HTTPS prefix rather than HTTP prefix. So the web address of an SSL-enabled website is *https://www.example.com* rather than *http://www.example.com* which is vulnerable to attacker. Session hijacking attack can be launched by stripping [5] this SSL. Some technique is used to strip SSL. The details about SSL stripping [6] attack technique are described in Section 2.

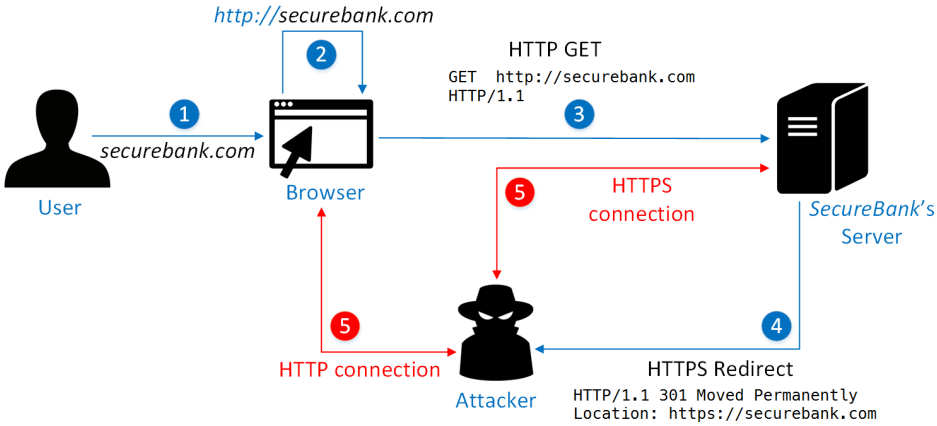


Figure 1: SSL Stripping

There exists several recent works [7–13] on threats and security analysis of mobile applications and SSL/TLS deployment in mobile applications. Other HTTPS and TLS/SSL-based attack techniques are discussed in several works [14–21] There have been several existing solutions proposed specifically for SSL stripping attack, for example, ARP-related solutions [22, 23], web script based solutions [24–27], MITM based solutions [28] and others [29–33]. However, there has been no survey that compiles all the solutions against SSL stripping-based session hijacking attacks which are crucial security threats for web users.

The main *objective* of this paper is to perform a comprehensive survey to identify and categorize all the existing solutions related to SSL stripping-based session hijacking attacks and to present them with illustrations. *No such comprehensive survey* exists in the literature.

The *contributions* of this work are (i) survey on solutions of SSL stripping attack with

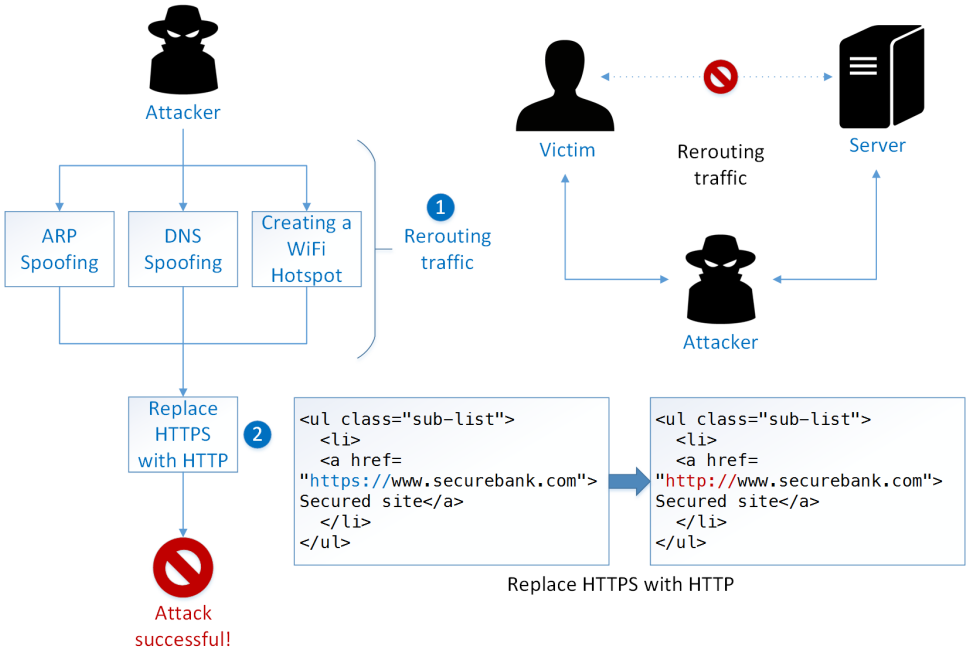


Figure 2: SSL Stripping Attack Technique

illustrations, (ii) classifying the solutions according to solution approach, (iii) comparing these solutions with the help of a table and illustrating each comparison, and (iv) proposing future research scopes on the solutions of SSL stripping attack.

This paper will help future researchers to get all the solutions together and propose new solutions against SSL stripping based session hijacking attack.

The rest of the paper is organized as follows. In Section 2, we briefly explain SSL-based session hijacking attack. In Section 3, the existing protection mechanisms are categorized. In Section 4, the client-side protection mechanisms are explained, followed by the server-side approaches in Section 5. In Section 6, we compare the all the protection mechanisms against SSL stripping-based session hijacking attacks. Future research are outlined in Section 7. Finally, Section 8 has the concluding remarks.

2 SSL Stripping

SSL stands for Secure Socket Layer. All security mechanisms on the Internet are actually based on SSL. SSL stripping is the most dangerous attack for session hijacking. Because in SSL stripping attack, web browsers generally does not give any warning to end users. In Fig. 1, a basic SSL stripping attack has been demonstrated which is explained as follows:

1. User visits a site (for example, a banking site) with its url address, e.g.,: *www.securebank.com*
2. Web browser automatically adds *http://* preceding *www.securebank.com*, thereby making it *http://www.securebank.com*
3. Now web browser requests the *SecureBank*'s server by *HTTP GET* method
4. *SecureBank*'s server receives the request and redirects it to *https://www.securebank.com*

5. Since the *HTTPS* redirection goes through a possibly unsecured network, following are the possibilities:

   - Attacker can strip *HTTPS* for the user
   - Attacker can communicate to SecureBank via *HTTPS*, and can do everything on behalf of the actual user
   - Generally no warning is shown from web browser
   - User generally has no knowledge about this attack
   - User can still connect to SecureBank's server through HTTP via the attacker even though the Bank actually has provided *HTTPS* for its clients

2.1 Severity of SSL Stripping attack

SSL Stripping-based session hijacking attack is a very dangerous attack. This is because by performing successful SSL strip attack, hackers can easily fetch victims' credentials and use them silently. Those credentials might be bank accounts, credit card details, Social Security numbers, other sensitive personal / financial information which are very crucial for any person. Thus, hackers can still huge amount of money from the victims. The nature of SSL strip attack is done in a such the way that victim will not notice the presence of an attack.

2.2 SSL Stripping Attack Technique

Fig. 2 shows SSL stripping attack technique. To launch a successful SSL stripping attack, the following two steps are performed:

1. Rerouting traffic: There are three existing techniques to reroute web traffic. They are:

   - ARP Spoofing: ARP table binds IP address with its corresponding MAC address. If an attacker broadcasts his MAC address binded with the network's gateway IP, then all traffic intended for the gateway will go through the attacker. Similarly, the attacker can also send his MAC address binded with a victim's IP address to the gateway. Thus any traffic that is intended for the victim, will pass through the attacker. The attacker requires to be connected with the LAN that the victim is connected with. But it is also possible for the attacker to get access to a LAN through a remote vulnerability or a weak password on just one machine on the network. The popular tool to perform ARP spoofing is *arpspoof*.
   - DNS Spoofing: DNS stores the human-readable web URL binded with its corresponding IP address. By DNS spoofing, the URL binds with an attacker malicious website's IP address. As same as ARP spoofing, the attacker requires to be connected with the same LAN or to use a remote vulnerability. The popular tool to perform DNS spoofing is *dnsspoof*.
   - Creating a WiFi Hotspot: An attacker needs to create a WiFi hotspot (possibly a fake one) and allows victims to connect to it. Thus all traffic related to the victims will go through the attacker's hotspot. In this method, the attacker's hotspot acts as a proxy server. The related tools for this method are *easy-creds*, *airodump-ng*, *airbase-ng*. [34]

2. Replace HTTPS with HTTP: This step is done after successfully rerouting victim's traffic to the attacker's machine. In this step, all HTTPS links and redirects are replaced with their corresponding HTTP ones. The ultimate tool for this step is *sslstrip*. [35]

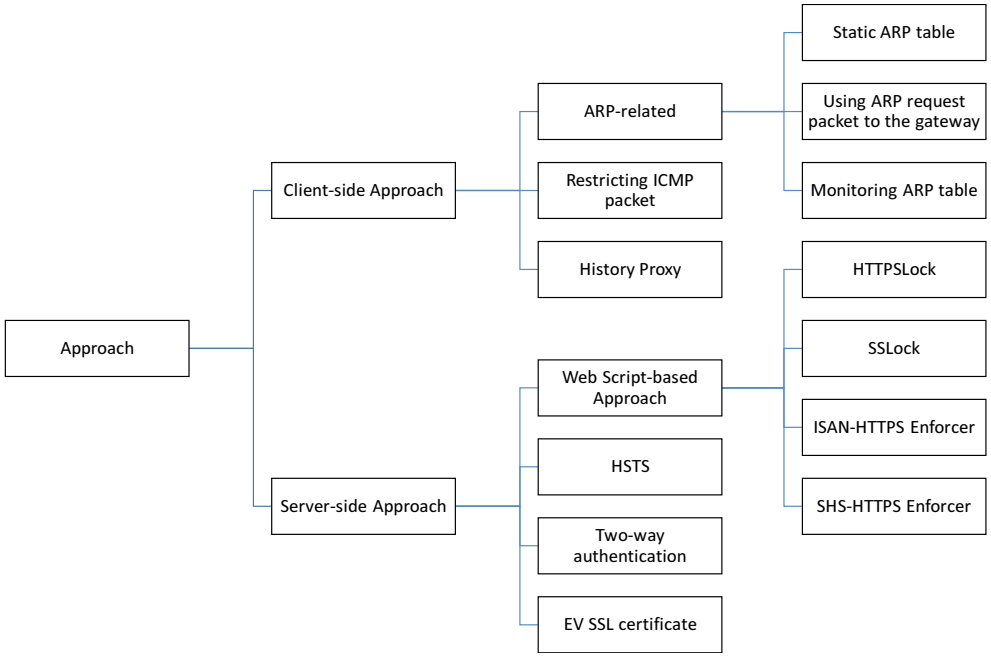


Figure 3: Categorization of existing approaches.

2.3 Weakness of Users for SSL Stripping Attack

In general, an attacker uses two weaknesses of users to carry out SSL stripping-based session hijaching attacks:

1. Not explicitly visiting HTTPS sites: Users tend to just go to the address *securebank.com* or *www.securebank.com* rather than *https://securebank.com* or *https://www.securebank.com*. If rerouting traffic is done successfully by the attacker, he/she can easily perform SSL stripping attack.

2. Users' tendency to accept fake certificates: Users generally wants to bypass any warnings of web browsers and visit a website without considering whether it is vulnerable or not. Some of the warnings are very critical such as certificate error due to a fake certificate. By accepting fake certificates, the attacker can easily perform session hijacking attack.

2.4 Existing Solutions to SSL Stripping Attack

The current solutions to SSL stripping attack are:

- History Proxy [36]
- Static ARP table [22] [23]
- EV SSL certificate [22]

- Two-way authentication [22]
- HTTPSLock [24]
- Using ARP request packet to the gateway [23]
- Monitoring ARP table [23]
- Restricting ICMP packet [23]
- SSLock [25]
- HSTS [37]
- ISAN-HTTPS Enforcer [26]
- SHS-HTTPS Enforcer [27]
- Cookie Proxy [38]

3 Categorization of Existing Approaches

SSL stripping attack is targeted towards client. So client must defend against the attack. Server may also ease its clients to protect against the attack even though client is always involved to protect itself. Thus, the existing approaches against SSL stripping attacks can be categorized as follows (shown in Fig. 3):

1. Client-side Approach: The approaches in this category does not include any server involvement. In this category, only client is responsible to protect itself against SSL stripping attacks. Different client-side approaches are:
    - ARP-related: The approaches in the category actually defend against ARP spoofing. Since ARP spoofing is one of the steps to perform SSL stripping attack, the approaches in this category can be used to protect against SSL stripping attack.
    - Restricting ICMP packets.
    - History Proxy.
2. Server-side Approach: The approaches in this category includes some sort of involvement of server. Some of the approaches in this category can be further categorized:
    - Web Script-based Approach: The approaches in this category use some sort of web-based scripts such as JavaScript, Python or Perl. Server sends web-based scripts to its clients so that browser in client's device can execute the scripts to protect against SSL stripping attack. Since the approaches in the category highly depend on client's web-based technology, a supported web browser must be used in client side.
    - HSTS
    - Two-way authentication
    - EV SSL certificate.

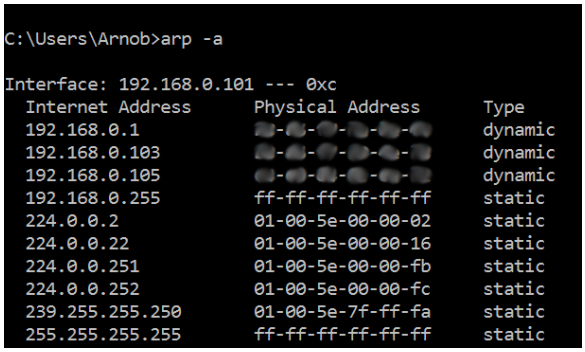


Figure 4: ARP table in a client device.

4 Client Side Countermeasures

In this section, we explain all the client side measures that have been proposed by researchers to prevent SSL-based session hijacking attacks.

4.1 ARP related countermeasures

There are three types of countermeasures that are based on ARP-related approaches. They are explained in the following:

1. Keeping Static ARP.
2. Monitoring ARP table periodically.
3. Using ARP request to gateway periodically.

4.1.1 Keeping Static ARP Table

ARP table binds IP address with corresponding MAC address. However, ARP is not secured by design. It is highly vulnerable to ARP spoofing [39]. ARP spoofing can be used to initiate man-in-the-middle attack, such as SSL stripping. Use of static ARP table can prevent ARP spoofing and SSL stripping attacks. There are two scopes found in which static ARP may be used:

- in client's device
- in the gateway

In Client's Device: Each client device contains an ARP table that is used in sending packets in the LAN. To prevent ARP table from poisoning, user may add static entries in the ARP table. In this way, the gateway router's MAC address cannot be replaced by attacker's MAC address [22, 23]. Fig. 4 shows a sample ARP table in a client machine with static and dynamic ARP entries.

Advantages

- It can be used in home network if the gateway is rarely changed.

```
#!/bin/sh
while[1]
do
        a = arp -a | awk '{print $4}' | grep e5:1b:d4:74:62:bf
        if [ $a -gt 1]
        then
                notify-send -t 0 System is ARP poisoned
        fi
        sleep 10
done
```

Figure 5: Shell script to detect ARP poisoning.

- It is very suitable for a static network such as a wired network.

Limitations

- For a mobile device, static ARP entry is very impractical. Because device has to change ARP entry all the time while connecting to a different hotspot.

In the Gateway: All activities between the client and the server (on the Internet) pass through the default gateway. Since ARP is not secure, attacker can exploit the gateway of the victim node. To prevent this attack, static ARP table may also be used in the gateway [23].

Advantages

- It is very suitable for the users who are connected to a wired network (such as LAN).
- It can be used in a wireless network (such as WLAN) if the network configuration is fixed, such as in a home network.

Limitations

- It is not suitable in a wireless network (with DHCP configuration) such as a public WiFi hotspot.

4.1.2 Monitoring ARP table periodically

A user can check ARP cache before sending any data to the gateway. A Linux shell script can control the ARP table for this purpose. Fig. 5 shows a shell script having 'awk' command to identify ARP poisoning. It checks ARP table at a fixed interval of time. It checks the MAC address of the gateway router to protect against ARP poisoning. It usually checks the number of IP addresses which has the MAC address of the default gateway. If more than one mapping exists for the default gateway router, it implies that attacker are spoofing the default router. In this case, it alerts the user to take proper action. On the other hand, user can execute 'arp -d' command to delete all the entries of ARP cache to avert this attack.

Limitations

- User must be conscious about the MAC address of the gateway.
- It cannot protect against ARP poisoning.

```
#!/bin/sh
while[1]
do
        arping -f 192.168.142.1
        sleep 5
done
```

Figure 6: Shell script to prevent ARP poisoning

4.1.3 Using ARP request to gateway periodically

In a LAN, when a source machine would like to know the MAC address of a destination machine, it broadcast ARP request in the LAN. After receiving this request, the destination machine unicasts its MAC address to the requesting source machine. The source machine then stores the IP to MAC address mapping of the destination machine in the ARP cache. Thus, use of ARP cache speeds up future communication. However, ARP request does not carry any validity. In ARP spoofing attack, the attacker sends false ARP response, claiming that it contains the MAC address of requested IP address. Thus, the ARP cache of the victim node may contain wrong IP-to-MAC address mapping in the ARP cache. This may cause future frames to be passed through the attacker's machine.

In order to solve this problem, user sends ARP request to the default gateway (router) at regular interval to collect MAC address of the gateway and store it in the ARP cache. A shell script can help us for this case. Fig. 6 is such a shell script which is run on background in advance to prevent ARP poisoning. This script broadcasts ARP request to the gateway router at regular interval. In reply, the gateway sends its MAC address. Thus, The attacker cannot modify the ARP cache of user because this process runs at regular intervals.

This script sends ARP request to gateway having IP address 192.168.142.1. "-f" is used to finish after first reply confirming that target is alive.

Limitations

- It will create unnecessary traffic to the network as there are many broadcast ARP request and unicast ARP reply.

4.2 Restricting ICMP Packet

An attacker can send ICMP ping request to the victim and the gateway. When victim gets this fake request, then ARP cache of the victim saves the IP address of the attacker. It stores this address with IP address of the gateway. Similarly, ARP cache of the gateway saves the MAC address of the attacker. It stores this address with the IP address of the victim. So the attacker can sniff all the traffic from victim to gateway or gateway to victim. If ICMP packets are blocked, this may obstruct all the ports and terminate future communication that could be occurred through these ports. In order to block ICMP packet, firewall should be activated. But blocking ICMP packet is not a better solution. Sometimes it may be harmful.

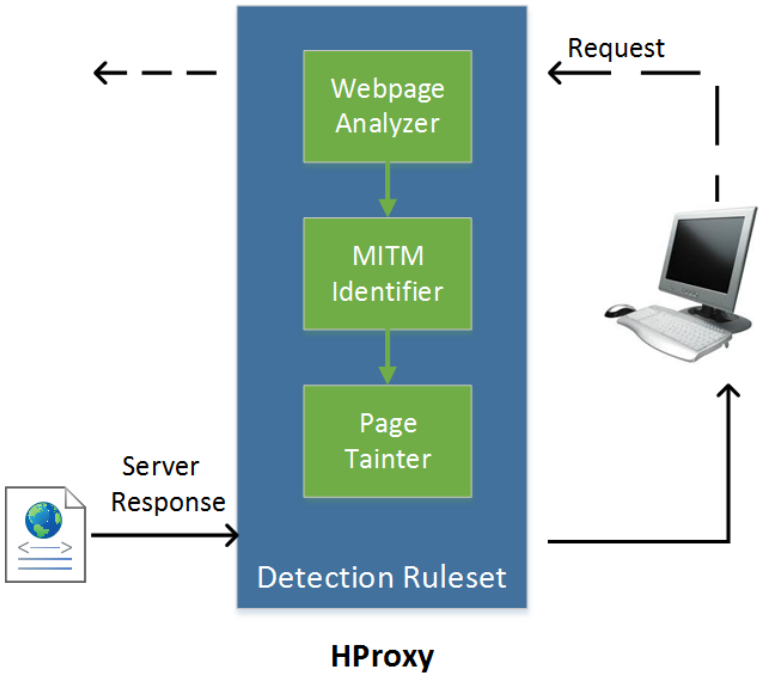


Figure 7: Architecture of HProxy

4.3 History Proxy

Nikiforakis et al. [36] present a countermeasure against SSL stripping attack that uses the browser's history to create a security record for each visited website. Each record contains information about the exact use of SSL at each website and all future connections to that site are validated against it. This system uses these records and some rules to detect whether any webpage has been attacked by any malicious MITM attacker. After detecting, this system block that connection alerting user about the existence of a malicious attacker. It has been presumed that mainly users use secured connections and use insecure connections circumstantially. Regular browsing of SSL-enabled websites can be enough to create a record. HProxy uses record of a website, current browser request and response along with some ruleset to detect whether a page is maliciously modified by an SSL stripping attack.

4.3.1 Architecture of HProxy

The core functionality of HProxy is detection ruleset and some components that use it to protect from attack. Fig. 7 shows the HProxy architecture whose components are webpage analyser, MITM identifier, page tainter and detection ruleset. The components of HProxy architecture are explained as follows:

- Webpage Analyser: It identifies critical parts of a webpage such as JavaScript blocks, HTTP forms and their targets, iframe flags, HTTP moves messages and records them in the page's current profile along with their attributes.

- MITM identifier: This identifier utilizes current record of a webpage that is performed by webpage analyser. It compares this current record with original record of this web page.

If if identifies any modification after comparing two records, than it blocks this web page from forwarding to the client. In the case of new request, the webpage is soted as new record in the database.

- Page Tainter: If MITM identifier declares malevolent webpage as secure webpage, the page tainter support HProxy to halt in time to protect privte and authentic data of user. To recognize authentic data from login page, it inserts a Javascript program. This program sends username and password of the user to HProxy when the user types in login form. These private data is stored in HProxy. It also traces all the forms those has an extra secret field. Page tainter module checks for the extra hidden field for stored password using hidden domain field. If the page is authorized, password of domain field will never be visible in the HTTP traffic because this password is communicated only over SSL.

- Detection ruleset: Each rule helps to detect malicious webpage. HProxy cannot identify JavaScript which is original and appended. Only by using whitelisting, It can identify. At first time, all JavaScript blocks are identified from HTTPS form of a webpage and these blocks of code are stored in record of that webpage. If any request from this page is found in future, this page is compared with stored record of this webpage. If any modification is found, this page is considered as malicious and is not redirected to the user. Absence of HTTP forms, new forms, modified HTTP forms, HTTP moved messages, Iframe tags are very sensitive data structures for MITM attack.

4.3.2 Advantages

- It prevents fake HTTP connection from taking place.

4.3.3 Disadvantages

- HProxy has a excessive overhead because of the proxy of client side.
- It can only discern about SSL stripping attack but cannot prevent this attack.
- It needs a modification on web browser.

5 Server Side Countermeasures

In this section, we explain all the server side measures that have been proposed by researchers to prevent SSL-based session hijacking attacks.

5.1 Web Script based Approaches

There are four types of countermeasures that are based on Web script-based approaches. They are as follows:

1. HTTPSLock
2. SSLock

3. ISAN-HTTPS Enforcer
4. SHS-HTTPS Enforcer

5.1.1 HTTPSLock

In a study [40], one demonstrated that it is very easy to obtain users' online credentials by exploiting the fact that users disregard whether HTTPS is being used. It is the basis of SSL stripping attack. Users also tend to ignore certificate error warnings. Ignoring those warnings actually leads to SSL sniffing attack. To defend against both SSL stripping and SSL sniffing attacks, HTTPSLock is proposed. It is actually a JavaScript-based HTTPS enforcement with browser cache. [24]

Main Idea Since users tend to ignore certificate error warnings and whether HTTPS is being used, a JavaScript-based mechanism is developed in which browser caching is involved. The role of browser caching is similar to the trust-on-first-use model. The goal of the mechanism to prevent users to make security-critical decisions. The mechanism itself will do it on behalf of users. So in this scenario, if SSL certificate error is detected, web browser will rather prevent users from visiting it than give them choice to make a decision. Similarly, if a website sends HTTP pages but sent HTTPS pages previously, the mechanism will show non-bypassable warning to the user. Thus HTTPSLock can defend against both SSL stripping and SSL sniffing attacks.

Deployability HTTPSLock is based on JavaScript with browser caching. If web browser supports caching for secured connection, HTTPSLock can be deployed. Browser caching for secured sites is available for all current major web browsers. HTTPSLock solution is actually the collection of several HTML and JavaScript (JS) files. They are deployed in web server to ensure HTTPSLock for the website. Full-page browser warning is also necessary to work HTTPSLock properly. All current major web browsers support full-page browser warning for some certain cases.

Advantages As previously mentioned, HTTPSLock can defend against both SSL stripping and SSL sniffing attacks. The advantages of HTTPSLock are:

- HTTPSLock prevents user from bypassing certificate error warning. It does so by using cached JavaScript files. So it works against SSL sniffing attack which is found in deceptive captive portal.
- HTTPSLock also prevents SSL stripping attack after once the website is legitimately visited for the first time.
- It becomes harder for attacker to perform SSL stripping and SSL sniffing to the websites which are regularly visited.
- It can be immediately implemented without depending on HSTS.

Limitations

- HTTPSLock is only effective after once the website is legitimately visited for the first time. If attacker manages to perform SSL stripping during first visit, then this method is still vulnerable.
- If browser cache gets deleted deliberately by user or by limited cache size, then it is vulnerable.
- HTTPSLock depends on full-page certificate warning in web browsers. Even though most web browsers use full-page certificate warning, they are not obliged to do so. They may be changed at any time. So depending on the UI in different browsers for security purposes cannot be a *good* solution.

5.1.2 SSLock

SSLock [25] is a technique to enforce SSL-protection in SSL protected domain. This domain can be used as top level domain (TLD). For example www.example.secure and example.secure. Since an new top-level domain can create time cost and difficulties, therefore 'secure' can be used as subdomain. For example, secure.www.example.com, www.secure.example.com and secure.example.com. This format will also break some existing service. To extend SSL protection across other subdomain like www, a cacheable redirection response header is used to redirect browsers from an unprotected domain to SSLock-protected one.

5.1.3 ISAN-HTTPS Enforcer

HTTPS is a SSL based security control protocol which is designed to protect web application services from malicious users and eavesdroppers. When a user wants to connect with a website, he generally types just URL of this website (such as example.com or www.example.com). URL with HTTPS header initiates a normal HTTP connection with web server. When this request is received by web server, the web server redirects this HTTP connection to HTTPS connection. After this, the web server responds messages (webpages, JavaScript files) to the user. Now the communication between web server and web browser follows HTTPS connection. If any client uses HTTPS header prior to URL, the SSL stripping attack would not be taken place. Therefore, SSL stripping attack can be prevented through HTTPS enforcement.

Main Idea During a SSL stripping attack, the attacker redirects the HTTPS connection as a HTTP connection and is sent to client. Thus, the communication between the attacker and the client is over HTTP connection. So, credential information are passed as plain text from the client to the attacker. HTTPS enforcer [26] uses a list of URLs that is kept at the client side. When a web page is requested and the communication between the web browser and web server is done over HTTP connection, the HTTP enforcer checks the list of URLs. If this web page matches any URL in the list, HTTPS enforcer will redirect the connection to HTTPS. HTTPS enforcer has been implemented using JavaScript. Fig. 9 shows a flow diagram to enforce a connection to HTTPS by HTTPSenforcer.js.

Resource Requirement of ISAN-HTTPS Enforcer: Since ISAN-HTTPS Enforcer uses simple JavaScript at client-side, it does not take too much resource. A test was performed using for ISAN-HTTPS Enforcer [26] on a web server containing Intel XEON 2.40 GHz processor with a RAM of 2GB. The Operating system in the server was CentOS 5.3, web server was Apache/2.2.0, PHP 5.2.4; The client machine used was Intel Core2Duo E7300 with 2.66 GHz processor and 2GB RAM (with Backtrack 5 as the operating system in the client)

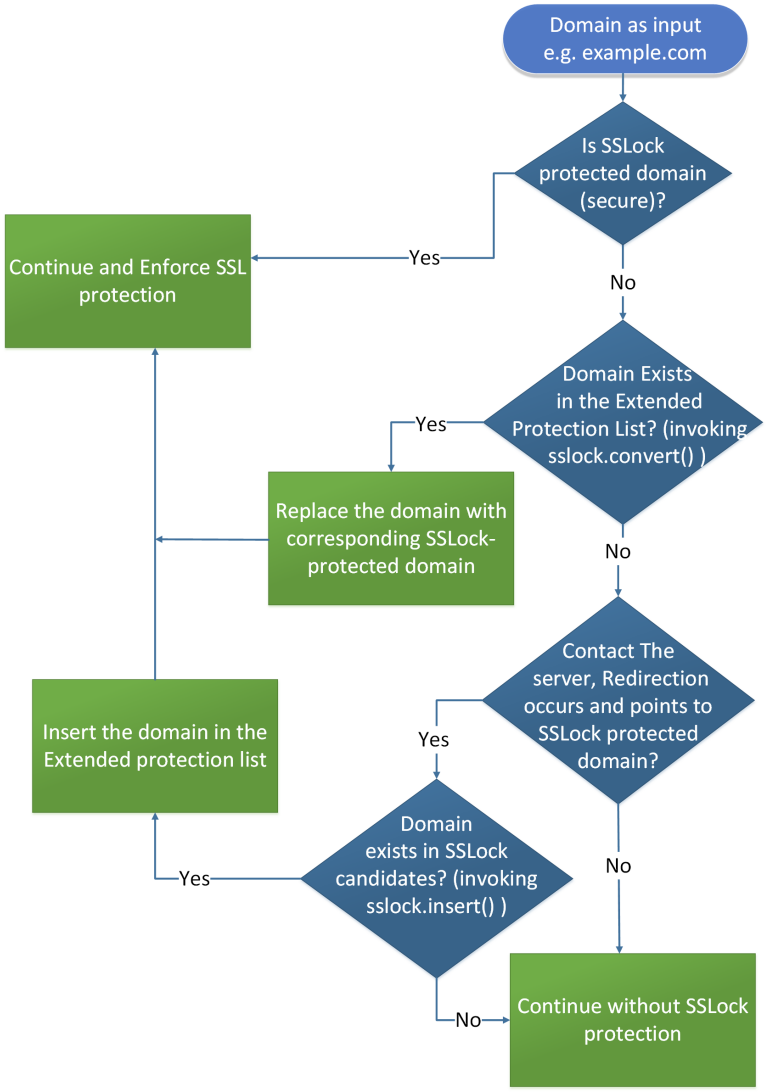


Figure 8: Flow diagram to enforce SSLock protection

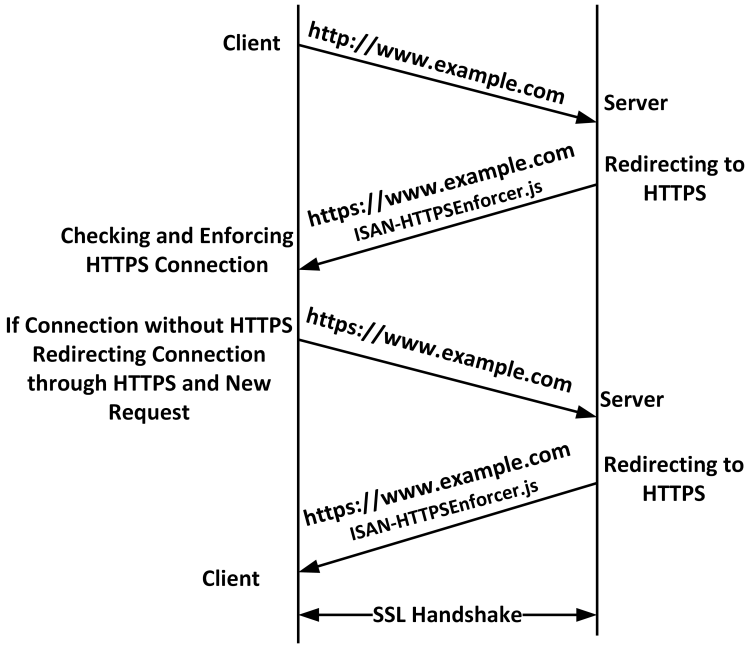


Figure 9: Flow diagram to enforce HTTPS connection

Advantages

- HTTPS enforcer can not only detect attack, but also protects from SSL stripping attack.
- It can work for all web browsers that use JavaScript in various platform.
- Web master can easily develop web pages to enforce HTTPS by including and calling JavaScript API.
- There is no need to modify or load any plug-in into the web browser.

Limitations

- Response time of HTTPS enforcer has overhead higher than other techniques (such as HSTS).

5.1.4 SHS-HTTPS Enforcer

Main Idea To reduce vulnerability of insecure initial handshaking by HSTS, SHS-HTTPS enforcer is proposed. In this solution, when a user agent sends a new request to server, this enforcer enforces https during (if needed) initial handshaking by URL redirection before request flows out. Again, this solution merges all the list of static HSTS URL lists of different browsers and enforces all the Fig. 10 shows a block diagram of SHS-HTTPS enforcer.

Rerouter Module When a user sends a website name without http or https header in front of domain name, in the most cases it is conducted through http request. Then this rerouter module shown in Fig. 11 sends all http traffic to a local lightweight server named as Squid server. Using IP table, a rule is written to do this task.

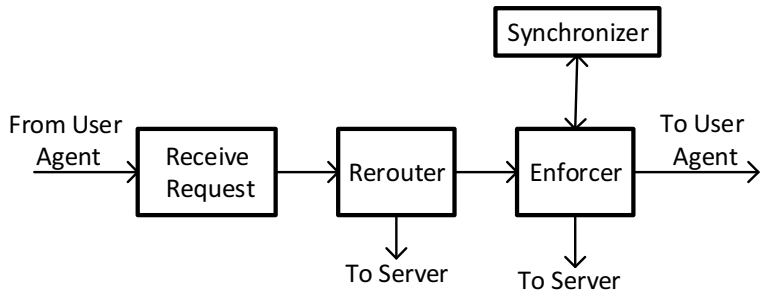


Figure 10: Block Diagram of SHS-HTTPS Enforcer

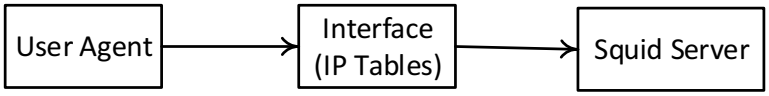


Figure 11: Rerouter module to reroute HTTP traffic

Enforcer Module After stripping URL by local server, this URL is compared against a preloaded list to verify whether this request should be converted to https as shown in Fig. 12. Algorithm for converting http to https is below: After getting URL which is requested, this request is compared with HSTS records of browsers. If there is a match between URL with any record of HSTS, this module redirects http to https. If there is no similarity between HSTS record and URL, this URL is redirected to interface of this enforcer.

Synchronization module This module maintains a list of URL which is syncronised with HSTS list of other web browsers. This module is shown in Fig. 13.

5.2 HSTS

HTTP Strict Transport Security (HSTS, or previous known as STS) is a web security protocol which is intended to prevent SSL stripping attack. HSTS is designed to work against protocol downgrade attack.

5.2.1 Main Idea

In HSTS, some special information are sent from a server to a client (web browser) through HTTP response header field. The header field is called ”Strict-Transport-Security”. When the browser receives the header, it must impose HSTS policy for the website. The header field contains the following fields:

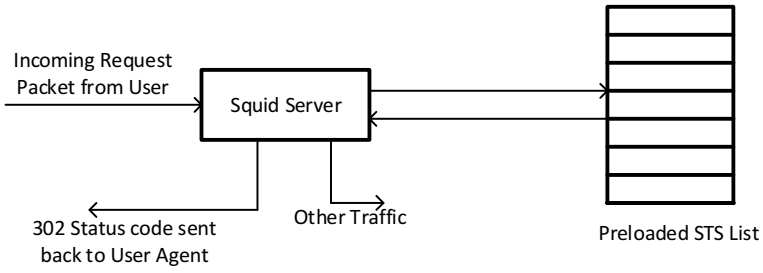


Figure 12: Enforcer module to convert HTTP to HTTPS

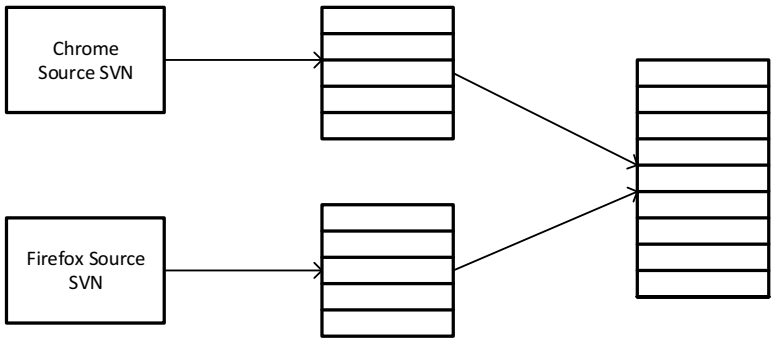


Figure 13: Synchronization module to sync STS Lists

- The max-age Directive: It specifies a period of time during which the browser should only access the website via HTTPS. It is a required field.
- The includeSubDomains Directive: It specifies whether HSTS policy is applicable in the website's subdomains. It is an optional field.

By HSTS policy, the web browser sets HTTPS for HSTS-enabled websites even if *https://* is not explicitly specified or *http://* is specified in the URL.

5.2.2 HSTS Preloaded List

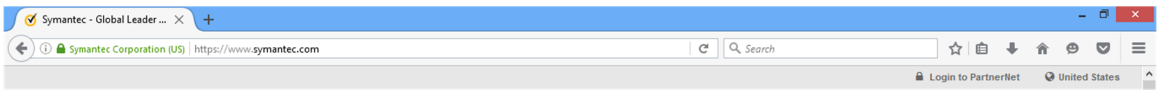
Although HSTS-enabled websites tell browsers that they should be visited through HTTPS, but it is still vulnerable to SSL stripping attack in user's first visit. This is called *Bootstrap MITM Vulnerability*. To prevent it, there is a preloaded list in browsers which describes HSTS-enabled domains. HSTS preloaded list is stored similarly as how root CA certificates are embedded in browsers by default.

5.2.3 Advantages

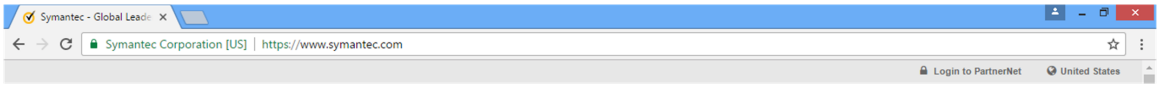
- HSTS protects against SSL stripping attack by imposing HTTPS explicitly. Even unspecified or HTTP connection will be always converted into HTTPS connection when HSTS is imposed.
- HSTS preloaded list can prevent SSL stripping attack during user's first visit to a website if the site is stored in the preloaded list.

5.2.4 Limitations

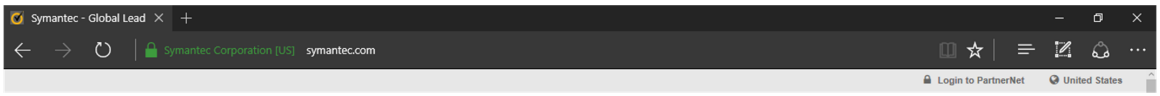
- HSTS uses trust-on-first-use model. So the HSTS header can be stripped by the attacker during user's first visit to a new website.
- Although HSTS preloaded list can prevent bootstrap MITM vulnerability, it is not scalable for all HSTS-enabled sites. If a HSTS-enabled website is not in the preloaded list, SSL stripping attack is still possible.
- HSTS preloaded list can be modified by user. But the process is not user-friendly for a normal user.



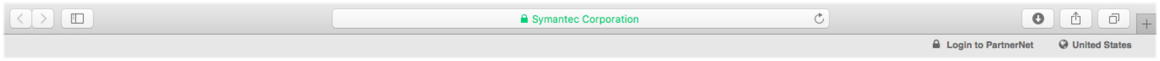
Firefox 49.0.2



Chrome 54.0.2840.71



Edge 38.14393.0.0



Safari 10.0.1 (12602.2.14.0.7)

Figure 14: User interface in different web browsers for EV SSL-enabled site

5.3 Two-way Authentication

Two-way authentication requires both client and server side certificate authentication. Without two-way authentication, only client knows server's authenticity by its provided certificate. But in two-way authentication, both client and server must have to acquire their certificates from CAs. Thus SSL connection will be successfully established after both sides are validated by each other. [22]

5.3.1 Advantages

- SSL stripping is practically impossible for an attacker. Because attacker needs to fake both server and client certificates.
- Even if user accepts fake server certificate, attacker cannot convince the server to accept fake client certificate. As a result, the connection will be still encrypted by the fake server certificate's public key (from client to server) and the legitimate client certificate's public key (from server to client). So it will be no use for the attacker.

5.3.2 Limitations

- Every client need to obtain a SSL certificate from a CA. It can be troublesome for a client.
- Asymmetric encryption-decryption such as SSL (public key cryptography) is approximately 1000 times slower than symmetric cryptography. In two-way authentication, both server and client need to validate each other with SSL. Thus it will slow down connection and increase server overhead.

5.4 EV SSL Certificate

EV SSL certificate is a new type of SSL certificate. It is also called *Extended Validation Certificate*. Normal SSL certificate is comparatively easy to obtain from a certificate authority (CA). So normal SSL certificate has encouraged some phishing attack. On the other hand, obtaining EV SSL certificate requires rigorous identity validation check by a CA. All CAs must follow the uniform criteria which requires the applicant to prove its legal identity to use the domain. [22]

5.4.1 Advantages

- All web browsers show a special and larger indicator for an EV SSL-enabled website on the user interface (shown in Fig. 14). So user can easily notice the difference in case of SSL stripping attack.
- Since obtaining EV SSL certificate requires strict identity validation, it is not possible for a phisher to obtain an EV SSL certificate to create a 'secured' phishing website.

5.4.2 Limitations

- The cost of obtaining EV SSL certificate is 1.5 to 2.5 times higher than the cost of obtaining normal SSL certificate. So many web administrators are not encouraged to obtain an EV SSL certificate to maximize their security.
- Many users lack their own safety awareness so that they cannot notice any difference in the user interface even though EV SSL-enabled website shows a large clear indicator in every web browser.
- Some users, especially in poor developing countries, tend to use outdated web browsers which do not separately recognize EV SSL certificates. In that case, no UI awareness is possible.

5.5 Cookie Proxy

In the Cookie Proxy approach, two proxies have been added between the server and the client as shown in Fig 15. Proposed new topology contains two proxies: (i) Secured LAN Guaranteed Proxy (SLGP) and (ii) Secured Server Guaranteed Proxy (SSGP) (Fig 15) The steps used in the cookie proxy are as follows. The client sends a web request to server which goes through SLGP. SLGP reconstructs the cookie and sends it to SSGP. SSGP receives the reconstructed cookie and encrypt this cookie using its private key. Then, SSGP sends this cookie to server and server receives this request and creates a set-cookie. The sever then sends this set-cookie to SSGP. SSGP remodels the set-cookie and sends the newly modified set-cookie to SLGP. After receiving this set-cookie, it verifies this using public key of SSGP. SLGP receives set-cookie and verifies the signature using public key of SSGP. In case of any SSL stripping attack, this signature will not be valid and thus this cookie is dropped. If the cookie is valid, SLGP sends the set-cookie to client. The client receives the set-cookie and caches it. Client sends new request again with a new cookie.

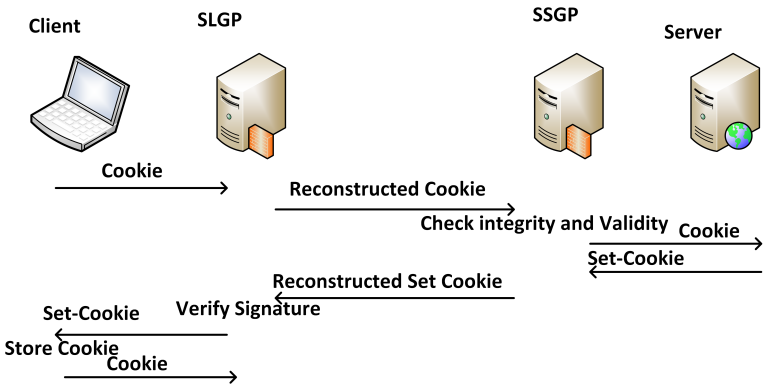


Figure 15: Flow diagram of Cookie Proxy Mechanism

5.5.1 Advantages

- It can prevent against SSL stripping attacks effectively.

5.5.2 Limitations

- It takes extra time to complete this prevention preocess.

6 Comparison among Countermeasures

The existing solutions to SSL Stripping attack are analyzed in Table 1. The criteria for the comparison are described next. Later a brief analysis is shown.

6.1 Effectiveness

It is noticeable that some of the existing solutions can only detect the presence of SSL Stripping attack. But others can both detect and protect against the attack. For example, HSTS protects against SSL stripping attack using HSTS preloaded list. But the approach like monitoring ARP table only detects suspicious behaviour in ARP table. So it cannot alone protect against this attack. Users have to explicitly abstain from using that vulnerable network.

6.2 Browser Support

Not all web browsers can work with all the existing solutions. Some web browsers still use old technologies. So the existing solutions to the attack need browser support. If an existing solution is not supported by all major web browsers, the solution is ineffective. Generally JavaScript-based solutions are supported by all major browsers. But the approach like two-way authentication is not supported by mobile browser or has very limited support. Some approaches such ARP table related solutions are not related to browsers.

6.3 User-friendliness in Server-side

If an existing solution need too much configuration from either server-side, it will not be not user-friendly. If it is expensive to deploy, it will not be user-friendly too. And if it is not user-friendly, the solution is not very effective. A solution can be said user-friendly if no special configuration is needed. For example, the ARP table related solutions do not need any server-side involvement. But SSLock needs some modifications in server-side.

Table 1: Comparison among the existing solutions

Approach	**Effectiveness**	**Browser Support**	**User-friendliness in Server-side**	**User-friendliness in Client-side**	**Resource Utilization**
History Proxy	Detect only	All Browsers	No support is needed from web server	Acts as a client side proxy	High overhead due to time
Static ARP table	Protect	Not related to browser	No configuration needed	Much configuration needed for different connections	Low
EV SSL certificate	Detect only	All major browsers	Expensive to obtain certificate	Awareness of browser UI needed	Low
Two-way authentication	Protect	All major desktop browsers	Client certificate validation needed for each session	Client certificate needed which may be costly	Overhead due to response time
HTTPSLock	Protect	All major browsers	One-time server deployment needed	No configuration needed	Adequate storage for browser cache needed
Using ARP request packet to the gateway	Protect	Not related to browser	No need to configure server	User initiates a shell script	Cost effective
Monitoring ARP table	Detect only	Not related to browser	No need to configure server	User initiates a shell script	Cost effective
Restricting ICMP packet	Protect	Not related to browser	No need to configure server	User has to activate firewall	Cost effective, but not a feasible solution
SSLock	Protect	Not all browsers	Need involvment of server as new domain is introduced	JavaScript code is placed at client side without user's involvment	Light weight
HSTS	Protect	All major browsers	No configuration needed	User-friendly if default HSTS preloaded is OK	Low
ISAN-HTTPS Enforcer	Protect	All browsers	Can easily develop to enforce https just including and calling JavaScript API	More user friendly than HSTS and typing *https://* at address bar	Overhead due to response time
SHS-HTTPS Enforcer	Protect	Irrespective of browsers	Local squid server is used which is easily configured	No user interaction is needed	Relatively low
Cookie Proxy	Protect	Browser Independent	User Friendly	User Friendly	High

6.4 User-friendliness in Client-side

If client has to do a lot of things for deploying an existing solution, it will not be not user-friendly. For example, HTTPSLock approach do not need any client-side configuration. But the ARP table related solutions need client-side involvement.

6.5 Resource Utilization

Some existing solutions need high CPU, RAM, storage or network bandwidth usage in either server or client-side. If powerful hardware is not available, the solutions would either fail or work very slowly. For example, HTTPSLock needs adequate storage for browser cache. But static ARP table does not use any special resource for its operation.

6.6 Comparative analysis among the approaches

From Table 1, some points are noted:

- HProxy only detects the attack and acts as a client-side proxy. For this reason, it has high overhead due to its detection mechanism.
- Static ARP table is protective against the attack and its overhead is very low. But it is not user-friendly in case of different connections.
- EV SSL certificate helps to detect the attack. But it highly relies on browser UI and certificates are usually more expensive than standard SSL certificates.
- Two-way authentication highly protects against the attack. But it needs both server and client certificates. It also has high overhead due to verification from both sides.
- HTTPSLock is protective against the attack but it is not effective on a website's first visit. Also it highly relies on browser UI for security purpose which is not a good idea.
- ARP request packet prevents ARP spoofing, so does the attack. But this technique unnecessarily creates request packets in the network at some interval.
- Restricting ICMP packets also protects against the attack. But ICMP packets are used for legitimate reasons too. So blocking valid usage cannot be a good solution.
- SSLock is protective against the attack but relies on a specific subdomain 'secure'. JavaScript is also provided by server for caching the subdomain redirection to HTTPS.
- HSTS also protects against SSL stripping attack but it relies on Trust-On-First-Use (TOFU) policy. It is ineffective if attacker somehow manages to attack a client at the time of a website's first visit.
- ISAN-HTTPS Enforcer uses JavaScript to protect against the attack but it has overhead due to response time.
- SHS-HTTPS Enforcer uses local squid server.

7 Future Research

The above discussed solutions have a number of demerits on protecting against SSL stripping attack. If we can remove or reduce these demerits, we can get better solutions to prevent this attack. Some improvements are mentioned below:

- Avoiding trust-on-first-use policy: HSTS trusts on the first session between server and browser. But SSL stripping attack can be made at the first session. If we can prevent this trust-on-first-use policy somehow, it will be a great work for us.
- Reducing response time: If we can reduce response time of ISAN-HTTPS enforcer, it will be great improvement on this solution.
- Protection rather than detection only: Despite high overhead, if we can configure HProxy such that it can protect from SSL stripping, it will be also very useful improvement on this solution.

The weaknesses of users must be mitigated to prevent SSL stripping attack successful. The probable approaches to mitigate the weaknesses of users are described below:

- Explicitly visiting HTTPS sites: One of the weaknesses described previously was not to visit HTTPS sites explicitly. SSLock approach has tried to remove this weakness by introducing 'secure' subdomains. But this is not user-friendly. Because in this approach there must have an extra text 'secure' for every secured web addresses. Rather doing that, it may be better to have a mapping for HTTPS-enabled websites in *a secured fashion*. The secured fashion may be done by the method the certificates are verified today, or by some other way we have not discovered yet.
- Denying fake certificates: Another weaknesses described previously was the tendency to accept fake certificates. Users generally do not realize the risk of ignoring certificate errors. This may be stopped by not giving any chance to users to ignore certificate errors. Web browsers may reject fake certificates without giving any choice to accept them. But for expert users, there may be a cryptic setting such as *about:config* in Mozilla Firefox.

By combining these approaches, there should be a much better solution against the session hijacking attack.

8 Conclusion

In this paper, we have described SSL based session hijacking attack and existing preventive measures against this attack. To facilitate research work for the web security based researchers, we have classified all these preventive measures into two main catagories: client-side measures and server-side measures. We have explained all the client-side measures and server-side measures with comprehensive illustrations. We have presented a comparative study on the proposed measures for SSL stripping based session hijacking attacks. Finally, we have also pointed out future research scopes on this topic. This paper will help web security researchers to improve existing solutions into better one that will protect the users from session hijacking attacks.

References

[1] "dsniff," accessed: June 5, 2017. [Online]. Available: http://monkey.org/~dugsong/dsniff/

[2] "Wireshark network protocol analyzer," Web page, accessed June 10, 2017. [Online]. Available: https://www.wireshark.org/

[3] "Ettercap: A ARP poisoning tool," 2010, accessed: June, 2017. [Online]. Available: http://ettercap.sourceforge.net/

[4] A. Freier, P. Karlton, and P. Kocher, "The SSL protocol," IETF RFC 6101, Aug 2011.

[5] "Moxie Marlinspike's sslstrip," 2009. [Online]. Available: http://www.thoughtcrime.org/software/sslstrip/

[6] M. Marlinspike, "New Tricks for Defeating SSL in Practice," in *BlackHat DC briefings*, DC, USA, Feb 16-19, 2009.

[7] Y. Guo, Z. Cao, W. Yang, and G. Xiong, "A measurement and security analysis of ssl/tls deployment in mobile applications," in *International Conference on Communicatins and Networking in China (ChinaCom)*, Chongqing, China, Spetember 24-26, 2016.

[8] Y. Liu, C. Zuo, Z. Zhang, S. Guo, and X. Xu, "An automatically vetting mechanism for ssl error-handling vulnerability in android hybrid web apps," *World Wide Web*, vol. 21, no. 1, pp. 127–150, 2018.

[9] M. Zaman, M. R. Amin, M. S. Hossain, and M. Atiquzzaman, "Behavioral malware detection approaches for android," in *IEEE ICC*, Kuala Lumpur, Malaysia, May 22-27, 2016.

[10] M. S. H. Shaikh Shahriar Hassan, Soumik Das Bibon and M. Atiquzzaman, "Security threats in bluetooth technology," *Journal of Computers and Security*, vol. 74, pp. 308–322, 2018.

[11] M. Zaman, T. Siddiqui, M. R. Amin, and M. S. Hossain, "Malware detection in android by network traffic analysis," in *1st International Conference on Networking Systems and Security (NSysS)*, Dhaka, Bangladesh, Jan 5-7, 2015.

[12] C. Amrutkar, P. Traynor, and P. C. van Oorschot, "An empirical evaluation of security indicators in mobile web browsers," *IEEE Transactions on Mobile Computing*, vol. 14, no. 5, pp. 889–903, 2015.

[13] rançois Gagnon, M.-A. Ferland, Marc-Antoine, F. Desloges, J. Ouellet, and C. Boileau, "Androssl: A platform to test android applications connection," *Lecture Notes in Computer Science*, vol. 9482, pp. 294–302, 2016.

[14] D. G. N. Benítez-Mejía, A. Zacatenco-Santos, L. K. Toscano-Medina, and G. Sánchez-Pérez, "Https: A phishing attack in a network," in *7th International Conference on Information Communication and Management*, Moscow, Russian Federation, August 28-30, 2017, pp. 24–27.

[15] P. Zhou and X. Gu, "Httpas: Active authentication against https man-in-the-middle attacks," *IET Communications*, vol. 10, no. 17, pp. 2308–2314, 2016.

[16] K. Bhargavan, C. Fournet, and M. Kohlweiss, "mitls: Verifying protocol implementations against real-world," *IEEE Security and Privacy*, vol. 14, no. 6, pp. 18–25, 2016.

[17] S. Kyatam, A. Alhayajneh, and T. Hayajneh, "Heartbleed attacks implementation and vulnerability," in *IEEE Long Island Systems, Applications and Technology Conference*, Farmingdale, NY, USA, August 8, 2017, pp. 1–6.

[18] S. Kiljan, K. Simoens, D. D. Cock, M. V. Eekelen, and H. Vranken, "A survey of authentication and communications security in online banking," *ACM Computing Surveys*, vol. 49, no. 4, 2016.

[19] A. E. W. Eldewahi, T. M. H. Sharfi, A. A. Mansor, N. A. F. Mohamed, and S. M. H. Alwahbani, "Ssl/tls attacks: Analysis and evaluation," in *International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*, Khartoum, Sudan, Sept 7-9, 2015.

[20] S. Fan, W. Wang, and Q. Cheng, "Attacking openssl implementation of ecdsa with a few signatures," in *ACM Conference on Computer and Communications Security*, Vienna, Austria, Oct 24-28,, 2016.

[21] Y. Jia, Y. Chen, X. Dong, R. Saxena, J. Mao, and Z. Liang, "Man-in-the-browser-cache: Persisting https attacks via browser cache poisoning," *Computers and Security*, vol. 55, pp. 62–80, 2015.

[22] K. Cheng, M. Gao, and R. Guo, "Analysis and research on HTTPS hijacking attacks," in *International Conference on Networks Security, Wireless Communications and Trusted Computing*. Wuhan, China: IEEE, April 24-26, 2010, pp. 223–226.

[23] G. N. Nayak and S. G. Samaddar, "Different flavours of Man-In-The-Middle Attack, consequences and feasible solutions," in *International Conference on Computer Science and Information Technology*. Chengdu, China: IEEE, July 9-11, 2010, pp. 491–495.

[24] A. Fung and K. Cheung, "HTTPSLock: Enforcing HTTPS in Unmodified Browsers with Cached Javascript," in *International Conference on Network and System Security (NSS)*. Melbourne, Australia: IEEE, September 1-3, 2010, pp. 269–274.

[25] A. P. Fung and K. Cheung, "SSLock: Sustaining the trust on entities brought by SSL," in *ACM Symposium on Information, Computer and Communications Security*, Beijing, China, April 13-16, 2010, pp. 204–213.

[26] S. Puangpronpitag and N. Sriwiboon, "Simple and Lightweight HTTPS Enforcement to Protect Against SSL Striping Attack," in *International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN)*. Phuket, Thailand: IEEE, July 24-26, 2012, pp. 229–234.

[27] S. B, H. P. R, and S. S, "SHS-HTTPS enforcer: enforcing HTTPS and preventing MITM attacks," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 6, pp. 1–4, November 2013.

[28] H. Xia and J. Brustoloni, "Hardening web browsers against man-in-the-middle and eavesdropping attacks," in *International conference on World Wide Web*. New York: ACM, May 10–14 2005, p. 489–498.

[29] C. Jackson and A. Barth, "ForceHTTPS: Protecting high-security web sites from network attacks," in *17th International World Wide Web Conference (WWW2008)*, Beijing, China, April 21-25 2008.

[30] J. Hodges, C. Jackson, and A. Barth, "Strict Transport Security," Web page, December 18, 2009, accessed: June 10, 2017. [Online]. Available: http://bit.ly/438ir0

[31] T.-E. Inc., " Extended validation and VeriSign brand," January 2007. [Online]. Available: http://www.verisign.com/static/040655.pdf

[32] D. W. C. Karlof, J. Tygar and U. Shankar, "Dynamic pharming attacks and locked same-origin policies for web browsers," in *Proceedings of the 14th ACM conference on computer and communications security (CCS)*. New York, USA.: ACM, October 28 2007.

[33] " DNSSEC : the DNS security extensions," accessed: June 10, 2017. [Online]. Available: http://www.dnssec.net/

[34] S. Avinash, "SSL stripping for newbies," March 8, 2015. [Online]. Available: https://www.linkedin.com/pulse/ssl-stripping-newbies-avinash-sm

[35] M. Marlinspike, "sslstrip," May 15, 2011. [Online]. Available: https://moxie.org/software/sslstrip

[36] N. Nikiforakis, Y. Younan, and W. Joosen, "HProxy: Client-side detection of SSL stripping attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, vol. 6201. Bonn, Germany: Springer, July 8-9, 2010, pp. 200–218.

[37] J. Hodges, C. Jackson, and A. Barth, "HTTP Strict Transport Security (HSTS)," *Internet Engineering Task Force (IETF)*, no. RFC 6797, pp. 1–46, November 2012. [Online]. Available: https://tools.ietf.org/pdf/rfc6797.pdf

[38] S. Zhao, D. W. andSicheng Zhao, and W. Y. andChunguang Ma, "Cookie-Proxy: A Scheme to Prevent SSLStrip Attack," in *International Conference on Information and Communications Security, ICICS*, vol. vol 7618. Berlin, Heidelberg: Springer, Oct 29 2012, pp. pp 365–372.

[39] "ARP spoofing," Web page, accessed: June 10, 2017. [Online]. Available: https://www.veracode.com/security/arp-spoofing

[40] M. Marlinspike, "New tricks for defeating SSL in practice," in *Technical Security Conference*. Las Vegas, NV: Black Hat, July 25-30, 2009, pp. 1–145.