An Intelligent System for Preventing SSL Stripping-based Session Hijacking Attacks

Mainduddin Ahmad Jonas[1], Md. Shohrab Hossain[1], Risul Islam[1], Husnu S. Narman[2], and Mohammed Atiquzzaman[3]
[1]Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Bangladesh
[2]Weisberg Division of Computer Science, Marshall University, Huntington, WV
[3]School of Computer Science, University of Oklahoma, Norman, Oklahoma, USA
Email: jonas.maj@gmail.com, mshohrabhossain@cse.buet.ac.bd, risulcse09@gmail.com, narman@marshall.edu, atiq@ou.edu

***Abstract*—An intelligent system to prevent SSL Stripping based session hijacking attacks is proposed in this paper. The system is designed to strike a delicate balance between security and user-friendliness. Common user behavior towards security warnings is taken into account and combined with well-known machine learning and statistical techniques to build a robust solution against SSL Stripping. Users are shown warning messages of various levels based on the importance of each website from a security point of view. Initially, websites are classified using a Naive Bayes classifier. User responses towards warnings messages are stored and combined at a central database server to provide a modified and continuously improving rating system for websites. The system serves to both protect and educate users without causing them an unnecessary annoyance.**

***Index Terms*—session hijacking, SSL stripping, naive bayes classifier, HTTPS, split-half correlation.**

I. Introduction

Session Hijacking is one of the most widespread security problems in modern computing [1], [2]. As more and more people rely on internet services for conducting sensitive activities like banking, online shopping, earning money online, accessing academic record, etc., session hijacking is proving to be a very damaging problem.

A *session* is a lasting connection between a user (or user agent, i.e., browser) and a server, usually involving the exchange of many requests and responses. The server typically maintains it, and on the server end, there is a data store or table to hold user state and user-specific information.

Each row of the table contains information about a particular session. The index to this table is known as the *Session ID* or *Session Key*. The key is generated on the first request or after an authentication process and is exchanged between the browser and the server on every request. The most common method for exchanging Session IDs is through HTTP Cookies.

Session hijacking refers to the exploitation of a valid computer session where an attacker takes over a session between two computers. This is done by stealing the Session ID, which can be used to get into the system and snoop the private data.

In general, unencrypted HTTP sessions are quite simple to hijack using Man-in-the-Middle (MITM) attacks [3]–[5]. Usually, HTTPS connection is recommended for preventing such Session Hijacking [6]. However, even HTTPS, which is HTTP over SSL, is vulnerable to hijacking [7].

SSL consists of three protocols: Handshake Protocol; Record Protocol; and Alert Protocol. Handshake protocol establishes a secure connection between the server and the client by agreeing upon a common set of cipher suites and parameters. Record protocol encrypts the data using the agreed-upon key established in the handshake protocol. Alert protocol alerts the other party by sending custom messages whenever an intrusion is detected.

The handshake protocol is the most vulnerable part of the SSL connection. The reason is that this protocol is done over unencrypted plain text, which makes it vulnerable to man-in-the-middle attacks. Attacks on SSL can be classified into two types primarily: SSL Sniffing attacks and SSL Stripping attacks. The SSL Sniffing attack is carried out through spoofed certificates. However, modern browsers will show a warning dialog when spoofed certificates are detected; hence, users enjoy a protection layer. On the other hand, SSL Stripping type of attacks does not result in any warning messages for users, making them more dangerous.

In SSL Stripping attack, the attacker strips out the HTTPS from server responses and presents the users with HTTP links. This can be done mainly due to two reasons. Firstly, users do not usually type in the full URL in the browser address bar including ”https://...”. They only type in the domain name. This cause the browser to send a HTTP request to the server. The server then sends a response redirecting it to a HTTPS site. Secondly, many websites use HTTP for the landing page to improve performance. It only uses HTTPS for login and other sensitive parts.

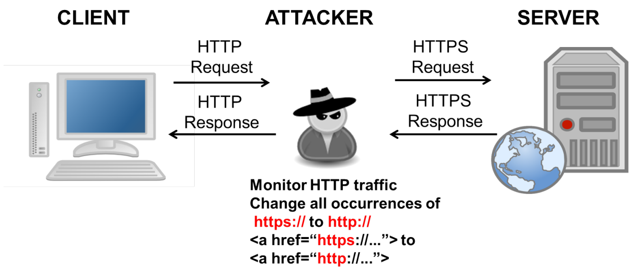

Fig. 1. SSL Stripping attack

Because of this, the MITM attacker can intercept the response from the server, and strip out the HTTPS to send the client HTTP links. Thus, the attacker establishes an HTTP connection with the client while maintaining an HTTPS connection with the server. The client does not see any warning message and sends all sensitive data over plain text, which the attacker can easily intercept.

The main *contributions* of this paper are (i) proposing a system for preventing SSL stripping attacks through the combination of machine learning and statistical techniques with common user behavior patterns (ii) real implementation of our system. We have come up with an accurate rating system for websites with a combination of web page content classification and user behavior. Then we present users with relevant and accurate warning messages of various levels according to the rating of the websites in order to both educate them and prevent them from being victims of SSL stripping-based session hijacking attacks.

The rest of the paper is organized as follows. In section II, we describe various existing work on SSL Stripping attacks. In section III, we explain our system in detail. In section IV, we discuss our experimental setup and a sample demonstration of how the system comes up with gradually evolving website ratings. Section V has on our plans to improve the system further by enhancing the rating algorithms and user-friendliness, and finally, Section VI concludes the paper.

II. Existing works

There have been few research works on SSL vulnerabilities [8]–[18], particularly focusing on SSL Stripping.

Burkholder [8] used `dsniff` and `webmitm` to implement a MITM attack through ARP spoofing on the SSL protocol. Callegati et al. [9] used ARP poisoning and DNS spoofing to intercept all traffic between the host machine and the server. Finally, `webmitm` was used to forge a fake certificate to launch the MITM attack. `ssldump` tool was used to dump the captured traffic into a text file.

Hossain et al. [10] performed a detailed survey on protection mechanisms to the SSL-based session hijacking attacks.

Cheng et al. [11] identified two potential vulnerabilities. Firstly, the SSL handshaking is done over plaintext, which leaves the certificate vulnerable to forging. Secondly, websites often initiate HTTPS connection by HTTP connections first. The authors used both types of vulnerabilities to launch two different types of attacks. The first one shows the users a fake certificate dialog, but the second one does not show any warning. The authors proposed Static ARP tables, EVSSL certificates, and Two-way authentication as potential solutions to the vulnerabilities.

Prandini et al. [12] detailed the SSL Stripping attack. They demonstrated that websites typically use HTTPS for only a selected few pages while using HTTP for the other parts of the website. This is done because HTTPS connection is about 20-50 times slower than HTTP connection. However, this left users vulnerable to the SSL stripping attacks.

Joshi et al. [13] proposed a method to prevent SSL stripping attacks with a browser plugin which hashed the password sent by the client with the server's certificate. This prevented fake certificate attacks.

Nikiforakis et al. [15] proposed a method called HProxy to prevent SSL stripping attacks by using browser history to gain a profile of which websites used HTTPS connection, and prevented a fake HTTP connection from taking place. It built a profile of safe SSL-enabled websites from the history of requests and responses.

Fung et al. [16] proposed a system called SSLock that works by enforcing special protected domains which enforce SSL connection. The same authors also proposed HTTPSLock [17] which was designed to enforce the HTTPS protection and forbid users to embrace invalid certificates.

The ISAN HTTPS Enforcer [18] enforces HTTPS connection from the client end using a custom javascript API which can be easily called from secure web servers. This overcomed the problem of user bypassing security warnings, as no security warning was shown, and also prevents SSL stripping attacks by introducing the HTTPS enforcer to handle redirections from the client side. SSL stripping attacks cannot trick the HTTPS enforcer because it maintains a list of secure URLs which are always forced to use the SSL connection.

From analyzing the reasons behind successful SSL Stripping attacks, we summarize the main features of SSL Stripping and its preventive measures as follows:

- SSL stripping is successful primarily because users are not educated about the difference between HTTP and HTTPS connections, and therefore are not aware of the importance of using encrypted connections while sending sensitive data to websites.
- Users cannot to be expected to type in HTTPS in the URL bar to ensure secure a connection.
- Users have a habit of ignoring warning dialogs even if the warning cautions against the possibility of leakage of sensitive data.
- False negative rate is very high, while the false positive rate is relatively low in user response towards security warnings.

Based on the above findings, we have proposed a solution to the SSL Stripping attack addressing the above mentioned issues.

III. Proposed system

In the section, we present a novel method of preventing SSL Stripping-based hijacking attacks based on our research on existing work, and overcoming the various limitations of them. We use a heuristic approach and apply machine learning to identify potential vulnerable websites and try to protect casual users from SSL Stripping attacks. In our system, there is a balance between security and user-friendliness based on the security rating of a particular website.

Our system can be broken down into two parts – the client side and the server side systems. The system is depicted in Fig. 2. The client downloads a global filter list from a central

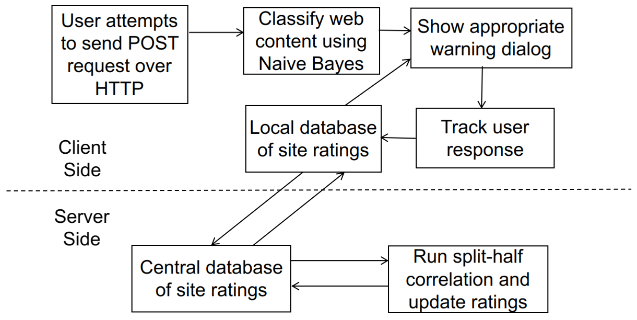
User attempts to send POST request over HTTP
Classify web content using Naive Bayes
Show appropriate warning dialog
Local database of site ratings
Track user response
Client Side
Server Side
Central database of site ratings
Run split-half correlation and update ratings

Fig. 2. Full system flowchart

web server. This list contains the security rating of all the websites in the database. This list is used to show a various level of warning messages to end users as described below. Periodically, the client sends users' response to the server so that machine learning can be applied to improve the security ratings of the websites.

A. Client-side System

The client-side system works by monitoring the HTTP POST requests sent from web browsers and intercepting the connection. If the connection is HTTPS, then the request is not intercepted at all. Furthermore, if the request type is not POST type, then again no interception takes place. The client-side flow chart is shown in Fig. 3.

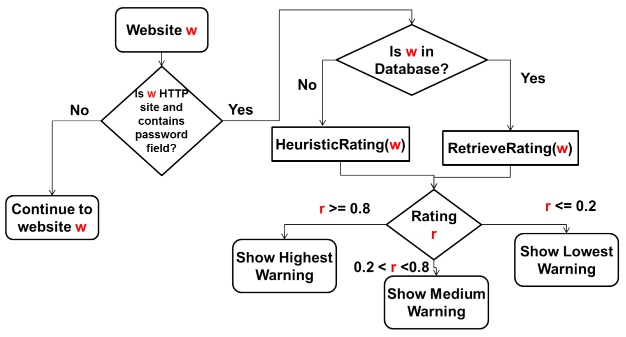

Fig. 3. Client-side system flowchart

Here, as the user sends each POST request, we check if it is an HTTPS site. If so, we do not take any action. If it is an HTTP site and includes a password or login field, then we intercept the request.

Once we have intercepted a request, we check if the site is in our database. If it does exist in our database, then we fetch the security rating from the database.

If the site is not in the database, we at first classify the web page to one of the five pre-defined classes. These classes are banking, e-commerce, education/email, social media, and miscellaneous. In our particular implementation, we used a Naive Bayes classifier on the content and URL of the web page for classification. We used a sample of 100 web pages of various classes to train our Naive Bayes classifier.

At first, we use the `dump` flag of the UNIX tool `w3m` to fetch the content of our web page, and then apply our Naive Bayes classifier on the content to classify it to one of the five pre-defined classes. Each class of website has a different security rating. Banking and e-commerce sites are assigned higher ratings, as they are more vulnerable to SSL stripping type of attacks. The ratings for various classes are shown in Table I.

TABLE I
CLASSES OF WEBSITES

Class of website	Rating	Security Level
Banking	1.0	High
E-commerce	0.9	High
Education/Email	0.7	Medium
Social Media	0.5	Medium
Miscellaneous	0.1	Low

We found that the URL of a web site can be valuable in classifying it. The reason is that it is widespread for banking sites to contain keywords such as ”bank”, educational sites to contain ”edu”, and so on. Therefore, we separate the components of the URL into words and then run a separate Naive Bayes classifier on it. We look for keywords such as ”bank”, ”mail”, ”shop”, ”edu”, ”uni” etc., and separate them from other parts. For example, xyzbank.co.uk will be separated into ”xyz”, ”bank”, ”co”, and ”uk”. The classifier will then easily be able to classify it as a banking site. Through an empirical approach, we found that giving the URL-based classifier 25% weight, and the content-based classifier 75% weight produces the best results.

Once having the rating for a website, an appropriate warning message is displayed based on the rating of the site. There are three levels of warning:

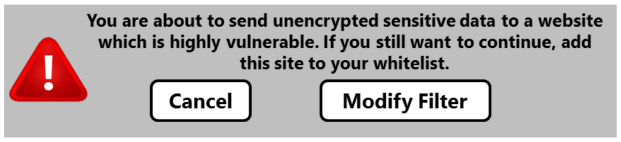
You are about to send unencrypted sensitive data to a website which is highly vulnerable. If you still want to continue, add this site to your whitelist.

Fig. 4. Highest warning level

a) Highest level: This warning level is shown if the security rating of the website is higher than or equal to 0.8. This level of warning blocks the POST request and forces the user to manually add the website to a white list before proceeding.

b) Medium level: This warning level is shown if the security rating of the website is higher than 0.2 but lower than 0.8. This level of warning also blocks the POST request, but the user can continue browsing to the site by clicking on the ”continue” button.

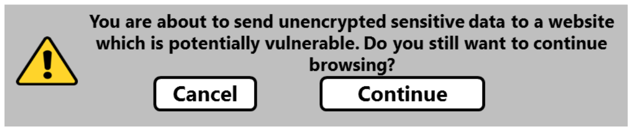
You are about to send unencrypted sensitive data to a website which is potentially vulnerable. Do you still want to continue browsing?

Fig. 5. Medium warning level.

c) Low level: This warning level is shown if the security rating of the website is lower than 0.2. This warning does not block the request, and browsing continues as usual. However, the user can optionally add the website to a blacklist.

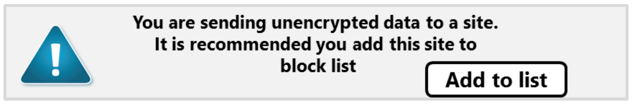
You are sending unencrypted data to a site. It is recommended you add this site to block list

Fig. 6. Low warning level

Users' actions are tracked in each level of warning and recorded it in the database. These records are periodically sent to a central server. Also, the updated database of website ratings is periodically fetched from the central server.

B. Server-side System

The server gathers the user's actions from various clients and uses that information to improve the security rating of websites, as depicted in Fig. 7. The algorithms in the server side can be broken down into mainly two parts.

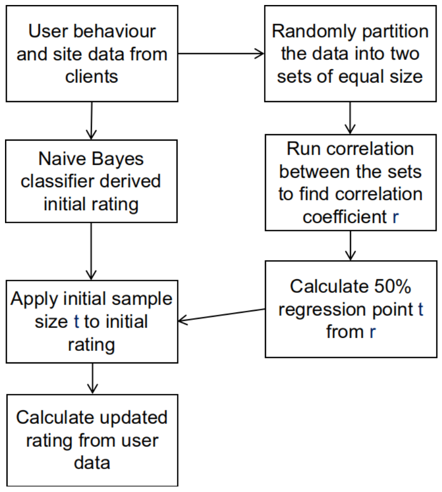
User behaviour and site data from clients
Randomly partition the data into two sets of equal size
Naive Bayes classifier derived initial rating
Run correlation between the sets to find correlation coefficient r
Apply initial sample size t to initial rating
Calculate 50% regression point t from r
Calculate updated rating from user data

Fig. 7. Server-side flow diagram

First of all, we take all the website in our existing database to calculate the 50% regression point for our dataset. We use a well known statistical approach called split-half reliability testing to find the correlation coefficient r. In this approach, we partition the user data for each website into two equal sized sets and find out the correlation between the two partitions. Once finding the correlation coefficient r, we then use this value, along with the average sample size x to find out the point of 50% regression by using Equation (1) in Algorithm 1.

$$t = \frac{1-r}{r}x \tag{1}$$

t, the point of 50% regression tells us is how much weight to give to our classifier-derived rating as opposed to the rating we learn from user behavior. If the value of t is higher than the number of user samples, then more weight is given to the classifier. If the sample is higher, then user's actions have more influence on the website's rating.

Algorithm 1 To measure the point of 50% regression

```
Input: Websites in database
Output:t point of 50% regression, t = (1-r)/r * x
1: get all websites from database
2: partition all websites into two sets of equal size
3: run a correlation between the two sets
4: find correlation coefficient $r$
5: $x$ = average sample size
6: point of 50% regression, $t = (1-r)/r * x$
7: return $t$
```

Algorithm 2 To update the rating of each website

```
Input: Websites and entries in database with $t$
    Initialization :
1: for each website w in database do
2:     sample of w = $t$
3:     sum of w = $initialRating(w) * t$
4:     score of w = sum of w / sample of w
5: end for
    Updating :
6: for each entry d in database do
7:     w = get website of d
8:     p = warning level of d for w
9:     a = user_action of d for w
10:    if a == respect then
11:        increment sample of w by 1
12:        increment sum of w by 1
13:        score of w = sum of w / sample of w
14:    else if a == bypass then
15:        if p == high then
16:            increment sample of w by 0.8
17:            score of w = sum of w / sample of w
18:        else if p == medium then
19:            increment sample of w by 0.5
20:            score of w = sum of w / sample of w
21:        else if p == low then
22:            increment sample of w by 0.2
23:            score of w = sum of w / sample of w
24:        end if
25:    end if
26: end for
```

Once having the value of t, we apply Algorithm 2 to update the rating of each website. After finding the 50% regression point, we set a score of each website to a value found from the heuristics applied on that website and award t number of samples initially. After that, we iterate through each entry in the database and update the score rating for each website. If the entry indicates the user respecting the error message, then we add that as one full sample. However, if the user bypasses the error, then we award a fraction of a full sample, depending on the warning level. If a high warning level was shown, and the user still bypassed, then we give that action 80% weight.

For a medium warning level, we give 50% weight, and for a low level, we give 20%.

The rating update equation can be written as:

$$rating_{new} = \frac{rating_{old} \times n_{sample} + n_{accept}}{n_{sample} + n_{accept} + weight \times n_{reject}} \quad (2)$$

Here, n_{sample} is the number of current samples, n_{accept} is the number of instances where the user accepted the warning message, n_{reject} is the number of instances where the user rejected the warning, and $weight$ is a variable that depends on the current warning level of the site. If the current warning level is high, then its value is 0.8; if the current level is medium, then its value is 0.5; and if the current level is low, then its value is 0.2.

The rationale behind the different weights for different user actions is explained by the habit of users when confronted with warning dialogs. Users usually ignore security warnings. Hence, the false negative rate is very high. However, when a user accepts a security warning, this is likely to be a genuine action. Therefore, the false positive rate is low. Now, different warning levels in our system pose different levels of difficulty to bypass. If the highest warning level is shown, then the user has to manually add it to the whitelist. Therefore, this action is given more weight than the medium warning where the user simply has to click a button to bypass.

We keep using this algorithm iteratively based on data received from the client side, and as the sample size of user data increases, the noise goes down, and the security rating of each website more truly reflects the actual vulnerability of that site. In this way, we have combined heuristics and machine learning to provide end users with robust protection against SSL stripping attacks. In the next section, we demonstrate our experimental setup and the application of our algorithms on sample data.

IV. Experimental setup and demonstration

We use a simulator to demonstrate how our system works with some real-world websites. For this, we chose five websites of various level of importance to apply our Naive Bayes algorithm and then update their rating using the simulated user behavior.

In our test setup, we use a transparent proxy called `Squid` on Ubuntu Linux operating system. The Squid proxy software allows filtering and redirecting network traffic in a transparent way, that is, without the client software knowing about it. We wrote a redirector script for this in Python.

First, we used a sample of 100 websites of various categories to train our Naive Bayes classifiers – both content-based and URL-based. We assigned 25% weight to URL-based classifier and 75% weight to the content-based classifier. We arrived at those ratings through an empirical approach and found them to give the best results.

We plan to explore using alternate algorithms to Naive Bayes for improved website classification, as we gather more real-world data. Furthermore, we decided to only use text content and URL of websites for our classification purpose. We tried incorporating the HTML tags of pages to see if they helped in providing better results. However, the HTML tags of web pages did not improve the classification; rather, it provided worse accuracy. Therefore, we used the `dump` flag of UNIX tool `w3m` to extract only the text content of web pages for classification.

After training our classifiers, we take 5 sample websites of various categories and apply our classifier on them. The initial ratings for these sites are shown in Table II.

TABLE II
Initial ratings

Website	Class	Security Level	Rating
dutchbanglabank.com	Banking	High	1.0
gmail.com	Education/Email	Medium	0.7
buet.ac.bd	Education/Email	Medium	0.7
facebook.com	Social Media	Medium	0.5
stackoverflow.com	Miscellaneous	Low	0.1

We then simulate user behavior towards warning messages for these websites. The result of the simulation is summarized in Table III.

TABLE III
Result of user behavior simulation

Website	Accepted Warning	Rejected Warning
dutchbanglabank.com	156	44
gmail.com	122	78
buet.ac.bd	147	53
facebook.com	68	132
stackoverflow.com	11	189

After that, we figure out the point of 50% regression by partitioning the user data for each website into two sets of equal size (100 samples each) and find out the correlation coefficient r between the two sets. This technique is known as *Split-half correlation*. Table IV shows the data used for correlation analysis.

We explored an alternative to the split-half correlation technique, called *Cronbach's Alpha* to gain a better measure of the correlation coefficient by reducing potential errors caused by random sample variance. Cronbach's Alpha can be thought of as the average of all split-half correlations from all possible splits in the data. This helps reduce errors but is computationally more expensive. Therefore, we decided to stick with simple split-half correlation. Once having enough real-world data, we will revisit Cronbach's Alpha to see if the extra computational time is worth it to achieve better accuracy. We can find out the correlation coefficient $r = 0.916$ from this

TABLE IV
Partitioned data for correlation analysis

Website	Sample 1 Accept %	Sample 2 Accept %
dutchbanglabank.com	70	86
gmail.com	69	53
buet.ac.bd	78	69
facebook.com	40	28
stackoverflow.com	5	6

data. The average sample size here was 100. From this we find out the point of 50% regression.

$$t = \frac{1 - 0.916}{0.916} 100 = 9.17$$

Therefore, we can treat nine samples as a point of 50% regression. Finally, we can apply our rating update algorithm to each of the websites as shown in Table V. Here, we can see that the security level of `buet.ac.bd` moved from initial Medium level to updated High level and `stackoverflow.com` moved from initial Low level to updated Medium level. In this way, we update the ratings based on the user behavior data. Finally, we can improve our Naive Bayes classifier by applying it on the updated database.

TABLE V
RATING UPDATE FOR EACH WEBSITE

Website	Updated rating	Security Level
dutchbanglabank.com	$r = \frac{9*1.0+156}{9+156+0.8*44} = 0.82$	High
gmail.com	$r = \frac{9*0.7+122}{9+122+0.5*78} = 0.75$	Medium
buet.ac.bd	$r = \frac{9*0.7+147}{9+147+0.5*53} = 0.84$	High
facebook.com	$r = \frac{9*0.5+68}{9+68+0.5*132} = 0.51$	Medium
stackoverflow.com	$r = \frac{9*0.1+11}{9+11+0.2*189} = 0.21$	Medium

V. FUTURE PLANS

In the real-world setup, we plan to use a Google Chrome extension to provide the functionality described in the previous section. In the real world, we will also ensure privacy for users.

A. Browser extension

We plan to build an extension for the Google Chrome browser using JavaScript. which will provide the functionality required for intercepting the connection and showing various levels of warning based on the security rating of each site and the filter lists. By using an extension, the users do not have to go through the hassle of manually having to update the program, and are also ensured security and privacy.

B. Ensuring privacy

User privacy is a crucial factor here. As this involves the sensitive issue of users browsing habit being monitored, it is important to ensure the data remains strictly anonymous. In our system, no identifiable user data will be stored at the server end. That way, the user's browsing habit cannot be tracked down by any party.

VI. CONCLUSION

In this paper, we have described the SSL Stripping based Session Hijacking threat. We have analyzed existing works that solves such threats. We have proposed an intelligent system to prevent SSL Stripping based session hijacking attacks. The experimental setup of our model is demonstrated. Our system is designed to strike a delicate balance between security and user-friendliness. Common user behavior towards security warnings is taken into account and combined with well-known machine learning and statistical techniques to build a robust solution against SSL Stripping.

We can conclude that our system provides a good balance between getting out of the users' way when not required, ensuring security in most crucial situations, and at the same time educate users on secure browsing habits. Our system learns over time from user behavior, and thus gradually improves as more and more user data becomes available. For these reasons, we believe this system scales very well to real-world usage.

REFERENCES

[1] Z. Lu, F. Chen, G. Cheng, and S. Li, "The best defense strategy against session hijacking using security game in SDN," in *International Conference on High Performance Computing and Communications*, Bangkok, Thailand, 18-20 Dec., 2017, pp. 419–426.

[2] S. Calzavara, R. Focardi, and M. S. ans M. Tempesta, "Surviving the web: A journey into web session security," *ACM Computing Surveys*, vol. 50, no. 1, 2017.

[3] D. Rupprecht, A. Dabrowski, T. Holz, E. Weippl, and C. Popper, "On security research towards future mobile network generations," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 3, pp. 2518–2542, 2018.

[4] W. Lee, H. Chen, S. Chang, and T. Chen, "Secure and efficient protection for http cookies with self-verification," *International Journal of Communication Systems*, vol. 32, no. 2, 2019.

[5] S. Calzavara, A. Rabitti, and M. Bugliesi, "Sub-session hijacking on the web: Root causes and prevention," *Journal of Computer Security*, vol. 27, no. 2, pp. 233–257, 2019.

[6] "Session Management Cheat Sheet, from the Open Web Application Security Project (OWASP)," https://www.owasp.org/index.php/Session_Management.

[7] A. Gavrichenkov, "Breaking HTTPS with BGP hijacking," *Briefings Black Hat*, Aug 2015.

[8] P. Burkholder, "SSL man-in-the-middle attacks," *The SANS Institute*, 2002.

[9] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-Middle Attack to the HTTPS Protocol," *IEEE Security & Privacy*, vol. 7, no. 1, pp. 78–81, 2009.

[10] M. S. Hossain, A. Paul, M. H. Islam, and M. Atiquzzaman, "Survey of the protection mechanisms to the SSL-based session hijacking attacks," *Journal of Network Protocols and Algorithms*, vol. 10, no. 1, pp. 83–108, 2018.

[11] K. Cheng, M. Gao, and R. Guo, "Analysis and research on HTTPS hijacking attacks," in *2nd International Conference on Networks Security Wireless Communications and Trusted Computing*, Wuhan, China, 24-25 Apr., 2010, pp. 223–226.

[12] M. Prandini, M. Ramilli, W. Cerroni, and F. Callegati, "Splitting the HTTPS stream to attack secure web connections," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 80–84, 2010.

[13] Y. Joshi, D. Das, and S. Saha, "Mitigating man in the middle attack over secure sockets layer," in *IEEE International Conference on Internet Multimedia Services Architecture and Applications*, Bangalore, India, 9-11 Dec., 2009, pp. 1–5.

[14] H. S. Narman, M. S. Hossain, and M. Atiquzzaman, "Multi class traffic analysis of single and multi-band queuing system," in *IEEE GLOBECOM*, Atlanta, GA, USA, Dec 9-13, 2013.

[15] N. Nikiforakis, Y. Younan, and W. Joosen, "HProxy: Client-side detection of SSL stripping attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA).* LNCS, Springer, 2010, vol. 6201, pp. 200–218.

[16] 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, 13-16 April, 2010.

[17] K. Cheung and A. P. Fung, "HTTPSLock: Enforcing HTTPS in unmodified browsers with cached JavaScript," in *International Conference on Network and System Security*, Melbourne, Australia, 1-3 Sept, 2010.

[18] S. Puangpronpitag and N. Sriwiboon, "Simple and Lightweight HTTPS Enforcement to Protect against SSL Striping Attack," in *Int. Conference on Computational Intelligence, Communication Systems and Networks*, Phuket, Thailand, 24-26 July, 2012.