Using Secure Search Engine to Counter Web-based Man-In-The-Middle and Phishing Attacks

Dijiang Huang and Vijayakrishnan Nagarajan
{dijiang,vnagara5}@asu.edu

***Abstract*—Security has become a big concern while browsing, as the number of malicious sites keeps increasing with the cost for hosting a site decreasing. Though most of the web servers use Secure Socket Layer (SSL) over HTTP (Hyper Text Transfer Protocol) to ensure trust between consumers and providers, SSL is vulnerable to Man-In-The-Middle (MITM) attack and becoming very common these days. Phishing is another problem which has increased rapidly over the years. In this paper we propose a Secure Search Engine (SSE) framework, which would resolve phishing and MITM attacks for web based applications.**

I. Introduction

World Wide Web (WWW) and the hyper linked web pages along with services like online chatting, electronic mail, and Internet services based on Cloud computing [1] have made web browser as a universal information portal for end users. Hyper Text Transfer Protocol (HTTP) [2] has become the most popular application-layer protocol to deliver information through Internet.

As people send private information through Internet, the web-based communications have become easy targets for attackers to compromise end-to-end communications using Man-in-the-Middle (MITM) attacks (e.g., SSLStrip [3]) and deploy malicious phishing web sites to delude Internet users to expose their private information. Cryptography-enhanced Internet protocols have widely been used to protect Internet users from being attacked. For examples, Secure Socket Layer [4], commonly known as SSL or TLS, makes use of Public Key Infrastructure (PKI), provides fundamental security services such as key agreement, authentication, and confidentiality for Internet applications. Moreover, for browser-based Internet applications, SSL supported Hyper Text Transfer Protocol (HTTPS) protocol was developed. As of now, SSL (or HTTPS) has become the defacto security standard for web browsing applications.

Often, we face the problems of using strong cryptography-based algorithms to construct weak security protocols. Many times, security protocol designers overlook the weakness of human beings in the type of human-in-the-loop security protocols. SSL supported web browsing is an example of this kind. In this paper, our research focuses on two major security problems: SSLStrip MITM attack and web based phishing attack, which require end users to be involved to make decisions on accepting or rejecting a potentially breached web link.

In SSLStrip attack, attackers explore the vulnerability that an end user may request a secure web site by initiating an insecure HTTP request. The attacker can intercept the request through ARP spoofing [5] or DNS poisoning [6] attacks and then impersonate the user to send the request. Once received the request, the secure web server usually sends an HTTP 302 message to ask the user to initiate an SSL session instead. The attacker can intercept the HTTP 302 redirect message and initiate an SSL session to the web sever without forwarding the redirect message to the end user. After setting up an SSL connection to the web server, the attacker sends un-encrypted web page to the user. Most users will not notice that the protocol name remains HTTP instead of HTTPS and a small lock logo should appear in the bottom of the web browser. If the end user types in his username and password into the web page, which is usually required by most financial web sites for authenticating the user, the username and password will be transmitted to the attacker in cleartext.

Phishing, is another way to explore the human weakness by using fake web sites. Similar to SSLStrip attack, Phishing attack also presents the exact same web content and page layout to users, except the web link address is different and the lock logo does not appear in browser's status bar. Many existing web browsers, e.g., Firefox, incorporate anti-phishing filters by checking several phising web site repositories, and then send alerts to end users if they happen to visit the phishing web sites. However, based on our testing, the false negative rate is very high. This is due to several reasons. First, as cost of hosting site has come down and there are tools like dyndns.org which would map the dynamic changing an IP address with a domain name, attackers can easily change their domain name and corresponding IP address. In last 6 months of 2008, there were 56,959 phishing attacks [7] and the average uptime of a phishing site was only 53 hours. Most of phishing-site repositories require users to send reports to them for phishing inspections, which introduces long delay and makes the information in repositories obsolete quickly. More severely, many phishing sites may not be detected previously, and thus cannot be detected by web browser's phishing filter.

To address the presented problems, we present a new secure referrer service, namely Secure Search Engine (SSE), which checks the user typed or clicked URLs for potential MITM or phishing attacks. SSE is designed as an automatic referrer service involving minimal interventions for humans for security decision. SSE returns deterministic security inspection answers "Yes or No" to users' browsers to remove potential human errors. To use SSE, a user just simply install a browser plug-in, which is downloadable at https://securesearch.eas.asu.edu. After enabling the plug-in, the browser will establish an SSL connection to SSE. Any typed in URLs or clicked links will be sent through the encrypted connection for security checking. Once the security checking is passed, SSE will inform the browser to initiate an SSL connection to the requested link if the web server supports HTTPS. In this way, the MITM attacker cannot intercept the user's secrets and the worst the MITM attacker can do is dropping the connection. In addition to countering MITM attacks, SSE will also perform realtime phishing site checking. We developed a phishing filter, which is especially useful to validate finance related websites. Our phishing filter can detect misspelling of URLs, check every existing phishing site repository, and check the validity of site certificates to identify a phishing site. Additionally, our phishing filter can differentiate phishing sites from wiki-type of sites that has phishing or financial information, which cannot be easily identified by many existing phishing filters. Our solution is non-intrusive for browser users. Only if an MITM attack or a phishing attack is identified, the browser will appear a pop-up window to alert the user and the user has the control to ignore the alert or discard the connection. Base on our experiments, the SSE only introduces about 25% to 40% delay for the first time visiting compared to web browsing without using SSE. By using caching techniques, the additional delay is not noticeable.

The rest of this paper is organized as follows. In Section II, we describe the related work done on MITM and phishing attacks. In Section III, we present the attack model for this paper. Section IV describes the detailed designs of SSE to counter MITM and phishing attacks. In Section V, we show how testing results from our implements. Finally, in Section VI, we conclude our work and describe the future work.

II. Related Work

Various solutions have been proposed to counter MITM attacks and phishing attacks. In [8], the authors explained how an MITM attack takes place, and they proposed three ways to counter ARP spoof based MITM attacks. In [7], the Anti-Phishing Working Group published the phishing scams in 2008. Authors of [9] used session-aware user authentication techniques to prevent MITM attacks, which allows a server to authenticate users based on their session information. In [10], authors proposed solutions for countering a web-based MITM attack by introducing context sensitive certificate verification and specific password warning-aware browsers. This technique validates a certificate and also checks if a password will be sent in an unsecured way. In [11] Jackson et al, proposed ForceHTTPS, which forces the browser to open a secure connection to the destination. If the destination does not support a SSL connection then the user should manually set a policy in ForceHTTPs. Among all existing MITM attacks, SSLStrip [3] is a tricky attack that exploits the security vulnerabilities of HTTPS. It first uses ARP spoof or DNS poisoning attacks to intercept users traffic. Then it partitions the end-to-end connection into two segments: one is un-encrypted from the victim to the attacker and another is an SSL connection between the attacker and a secure web server. Victim users usually doe not notice the HTTPS key word in the address bar or the security logo in the status bar, and send their private information unencrypted to the attacker.

Researchers at Stanford university developed anti-phishing toolbar [12] for web browsers. However, studies showed in [13], 23% of the people do not look at address bar, status bar, and security indicators when they browsing web sites. Techniques presented in [12] use heuristic approaches, e.g., checking URL, logo, referring page etc., to counter phishing web sites. The authors in [14], proposed a TrustBar to show the logo of the web server and the authority who has signed the web page in the trust bar. The TrustBar also shows if a web server does not support a secured connection. Firefox and opera browsers check phishing banks that update the reported phishing URLs every few hours. In [15], the authors compared effectiveness of different phishing tools including firefox2, ie7, spoof-guard [12], etc. Their results showed that all evaluated tools are less 90% effective in identifying phishing web sites. Based on users' browsing history, Cao et al. [16] proposed an automated individual white-list (AIWL) technique to defend pharming and phishing attacks. Machine learning techniques [17] were also proposed to identify phishing sites. Zhang et al, in [18], proposed a phishing filter "CANTINA", where they make use of robust hyperlinks and term-frequency - inverse document frequency ($tf - idf$), where $tf - idf$ is a technique to give less importance to the common occurring words. A sketch of $tf - idf$ is described in Appendix A. When a URL is fed into CANTINA, it first calculates the $tf - idf$ scores for the page, then it calculates the lexical signature using the top 5 $tf - idf$, and finally it uses Google Search engine to check if the web site is in top N results. The drawback with this technique is that CANTINA depends on the Google's crawler. When a new web site is up, it can stay online for approximately 53 hours [7] that cannot be crawled by Google's search engine immediately.

III. Attack Models

SSE is designed to identify unsafe web sites, especially web-based MITM and phishing attacks. In this section, we describe the attack models of MITM and phishing attacks.

A. *SSLStrip Man-In-The-Middle (MITM) Attack*

MITM attack is an active attack, where the attacker actively eavesdrops a communication, intercepts the packets from the source, and then forwards (w/o modifications) them to the destination. The attacker also ensures that both the source and destination believe that they are communicating with each other directly and no intermediate nodes in between to eavesdrop and modify the transmitted messages.

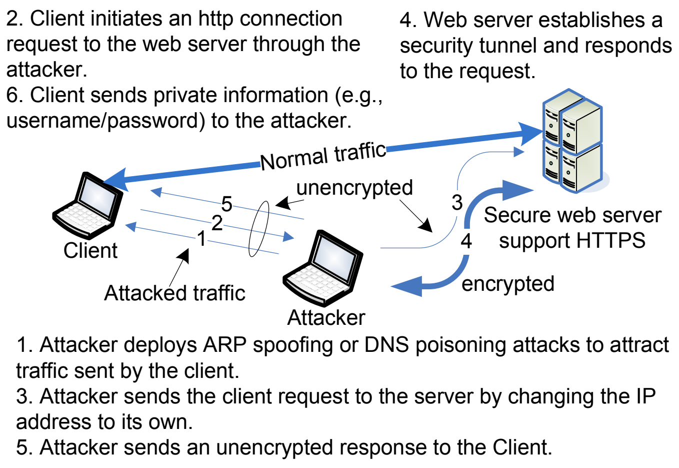

2. Client initiates an http connection request to the web server through the attacker.
6. Client sends private information (e.g., username/password) to the attacker.
4. Web server establishes a security tunnel and responds to the request.
Normal traffic
unencrypted
Client
5
2
1
Attacked traffic
Attacker
3
4
Secure web server support HTTPS
encrypted
1. Attacker deploys ARP spoofing or DNS poisoning attacks to attract traffic sent by the client.
3. Attacker sends the client request to the server by changing the IP address to its own.
5. Attacker sends an unencrypted response to the Client.

Fig. 1. Illustration of SSLStrip MITM Attack.

Based on SSL/TLS standard, when an user tries to view a web site that supports HTTPS connections, the browser needs to verify certificates of the web server. However, this does not means the user must initiate an HTTPS request to start the communication. SSLStrip MITM attack [3] explores this vulnerability to hijack into the user's SSL connections by partition an end-to-end SSL connection into two segments: one is encrypted and another is unencrypted. As shown in Fig. 1, the attacker first deploy ARP spoof [5] or DNS poisoning [19],[20], [21] attacks to direct the user's traffic through the attackers. Once the user initiates an HTTP request to an SSL enforce web server, the attacker can intercept the request and establish a secure connection to the server through an SSL tunnel. When receiving the replies from the server, the attacker decrypts the encrypted traffic and forwards it to the user. Since most secure web servers require their users to provide username and password for login, the user then send them unencrypted to the attacker. The problem is originated from the negligence of users since they should check if the HTTP is changed to HTTPS in the address bar and the security lock should appear in the status bar after loading the received web page. However, studies showed in [13], 23% of the people, including security professionals, do not look at address bar, status bar, and security indicators when they browsing a secure web site.

B. Web-based Phishing Attack

Web-based phishing is a criminally fraudulent process of attempting to acquire sensitive information by masquerading as a trustworthy web site. Phishing site is usually carried out in different forms through (i) instant Messenger or emails, (ii) social Networks (iii) phones or other communication means. One common way of phishing is to make a fraudulent web page look like a trusted web page and ask for sensitive information and the links from the phishing web-site will often lead to the original or other trusted web-sites. If the user is not careful enough to validate if the page is protected by SSL or can be validated, then he/she may unwittingly release private information to attacker. In general, the goal of phishing web sites is to get as much personal information as possible from the user. The information can be credit card details, address, date of birth, account information, password, social security number, etc. The attacker can use it for later frauds. Normally these phishing pages appear only for a very short span of time, i.e., average of 53 hours [7].

C. Assumptions of Attackers Capabilities

We assume that the attacker does not have a valid SSL certificate and corresponding private key of a spoofed website. The attacker cannot compromise our SSE server and interrupt the security validations between the SSE server to any inspected web site.

IV. Description of Secure Search Engine

In this section, we first present the architecture and functionalities of each component to build an SSE. Latterly, we explain the data flow in the SSE system. The architecture is based on the search engine structure presented in [22], which is an efficient and scalable architecture.

A. Secure Search Engine

1) Secure Search Engine Architecture: The SSE architecture is a layered architecture where the higher layer components makes use of the components at its immediate lower layer. Fig. 2 shows the main components of SSE architecture.

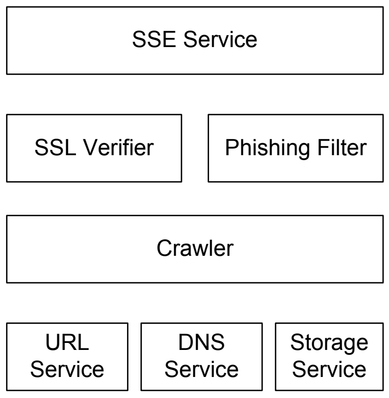


Fig. 2. Overview of the SSE architecture.

SSE service: SSE service answers all the user queries, which are generated by web browsers. The decision of SSE service is based on the inspection results derived from the SSL verifier and the Phishing filter.

SSL verifier: SSL verifier is one of the core parts of SSE. The SSL verifier module picks up an URL, collects the certificates for one or multiple domains running on the server. It also validates the certificate chain and stores it in the certificate repository of SSE.

Phishing filter: Phishing filter processes each URL and check if a site is a legitimate site or a phishing site based on the learning algorithm explained in Section IV-C.

Crawler: The Crawler is an automatic computer program that collects security and web page information of URLs for SSL verifier and phishing filter. In general, the crawler starts with a list of URLs (such as URLs for Banks) to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, and collect all the security information from each URL, such as public key certificates.

DNS service: DNS service is very critical for a search engine as mentioned in [23] that about 87% of a web crawling is just the DNS requests. The DNS service is actually a caching mechanism to make the crawler much efficient by not wasting requests on DNS. The DNS service is also used by phishing filter to get the IP addresses of the domains.

URL service: The URL service records URLs visited by the crawler whether are visited or not. To make the crawler scalable and recover from crashes. Thus, the URL service is responsible for restoring the state of the crawler when the system crashes.

Storage service: The storage service gets all collected and link inspection information, and stores them in a data base.

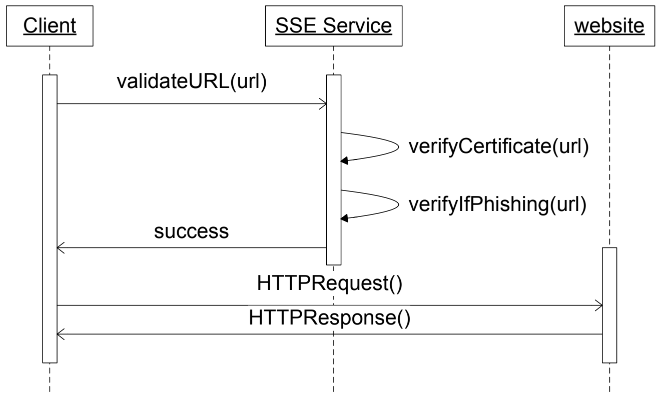
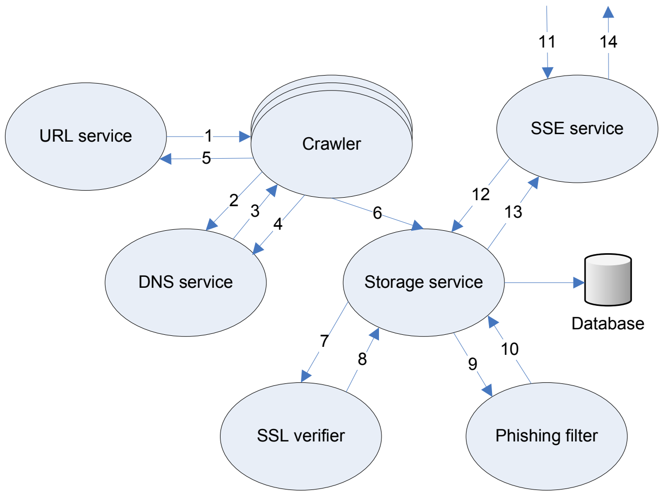


Fig. 3. Steps involved between a browser and the SSE service.

2) Procedures of Using SSE Service: The Fig. 3 explains the data flow among the user, SSE, and the web server. An SSE add-on component (or plugin/extension) must be installed for a browser at the client side to use SSE services. When the user types a URL or clicks on a weblink, the browser extension sends a URL inspection request to SSE through a pre-established SSL tunnel. SSE processes the URL inspection request, and inspect whether the requested URL links to a phishing site or not. The inspection includes two procedures through two filters: (a) SSL verifier validates if the web server provides valid certificates and supports HTTPS for the requested domain; and (b) phishing filter identifies phising properties of the inspected web page. The inspection result is sent to the user for decisions: (i) sending an HTTPS request if the web server provides valid certificates and SSL is enabled, (ii) sending an HTTP request if the web server does not provide a valid certificate and the web site had passed the inspection of phishing filter, and (iii) a visual popup alert will be shown to the user if the web site does not pass any one of filters of the SSE server.



(1) Crawler gets the unprocessed URLs from the URL service. (2 & 3) Crawler requests and gets the rewritten URL(with the corresponding IP-Address) (4 & 5) After crawling the new URL, crawler updates the DNS cache and URL service with the extracted URL. (6) The collected results are persisted using the storage service. (7 & 8) The SSL verifier collects, validates and store the certificates. (9 & 10) The phishing filter runs the algorithm to find out a website is a phishing site or not. (11) The browser extension request for verification of a website. (12 & 13) The SSE service uses the storage service to get the corresponding data to evaluate. (14) The SSE service answers the query

Fig. 4. Dataflow within the Secure Search Engine.

Fig. 4 presents the data flows within the SSE server. Initially, we populate the URL service with a set of seed URLs. The crawler picks up an unprocessed URL from the URL service and sends an HTTP request. From our current implementation, we only crawl banking web sites. In this process, the crawler uses the DNS service to retrieve the IP address of the given domain name in the URL. The crawler collects the new URLs from the HTTP responses and passes the same to the storage service. The stored website information will shorten the search responding time when the searched web sites have been already inspected. During the crawlering time window, new phishing web sites can be issued and they will be handled by SSE processing a realtime security inspection. The procedure is shown in Algorithm 1. To inspect a crawled web site, SSL verifier and phishing filter get the URLs from the storage service and run their algorithms to inspect potential phishing attacks of the web page. The inspection results are stored using the storage service and user replies are then generated.

Algorithm 1 *SSE Processing.*

```
1:  The browser retrieves $URL = getTheURL()$;
2:  if The $URL$ is available in browser caches and the cache expiry time is less than current time then
3:      Take actions based on the cached attributes;
4:  else
5:      The browser sends a redirect message $msg = redirectToSSEService(URL)$ to SSE;
6:      if SSE checks $msg$ is a phishing site in its cached database then
7:          SSE sends a Warning to the browser;
8:          if The user ignores warning then
9:              The browser sends normal $HTTPRequest(URL)$ to the web server;
10:         else
11:             The browser drops the HTTP request;
12:         end if
13:     else if SSL verifier returns “the web server supports HTTPS” AND phishing filter returns “the web server is not a phishing site” then
14:         SSE sends the certificate information to the browser;
15:         The browser sends secure $HTTPSRequest(URL)$ to the web server;
16:     else if SSL verifier returns “the web server supports HTTP“ AND phishing filter returns “the web server is not a phishing site“ then
17:         SSE informs the browser;
18:         The browser sends normal $HTTPRequest(URL)$ to the web server;
19:     else
20:         SSE sends a Warning to the browser;
21:         the browser goes to Step 8;
22:     end if
23: end if
```

3) SSE Processing Algorithm: In Algorithm 1, we present the SSE service processing algorithm. To make the understanding much easier, we would describe the algorithm ignoring the lines 2-4, which is explained in Section IV-D. The browser redirects all the new URL requests, including both inputs from the address bar and clicks from the browser window, to the SSE service. URLs are sent through pre-established HTTPS connections, which are only acceptable by the SSE service. SSE service checks with its database through the storage service on the received URLs to check if the site is already previously evaluated as a phishing site and if the site supports SSL and has a valid certificate. If the requested URL does not exist in the SSE's database, SSE calls SSL verifier and phishing filter processes to inspect potential MITM and phishing attacks. The evaluation results will be sent to the browser and it is up to the user to discard the HTTP request or ignore the Warning when receiving a negative reply. The corresponding HTTP or HTTPS request then will be sent out. If the requested web site information does not exist in SSE's database, SSE will start a realtime inspection which is straightforward, and the inspection steps are illustrated from line 13 to line 21.

B. Countering SSLStrip MITM Attacks

The goal of an MITM attacker is to get personal private information, such as a bank account associated username and password, etc. In the following presentations, we consider an example of a user going to a banking site, e.g., www.usbank.com, and illustrate how to defend the SSLStrip MITM attack using SSE.

The basic problem incurred by SSLStrip attack is originated by a human who uses a browser to access his/her bank account. The awareness of using security protocols to support online-banking system is very low for general users. It is the human nature to type as less number of letters to get access to a remote website. Often, a user just simply type usbank.com in the address bar without typing the secure protocol and service domain name, e.g., https://www.usbank.com, which enforces an SSL connection with requested port 443. When using HTTPS connections, an encrypted secure tunnel is built with the usbank web site, and the attacker trying to sniff will not get any information as the connection is secure. We also note that the MITM attacker can still cause a Denial-Of-Service attack by not forwarding the intercepted packets. However, this will disarm the attacker from deriving the users' private information.

To overcome the above mentioned drawbacks, a user can do a search in some search engines, from which we can extract the relevant results to verify the security protocols supported by the destination web site. However, extracting a relevant results is not a simple task since it requires users to verify the trustworthiness, which can be cumbersome and sometimes even impossible when the user is lack of security training. To address these issues, SSE automates the security inspection procedure and the user will be notified only if potential MITM attacks are detected.

We illustrate the steps involved in countering an MITM attack using the SSL verifier algorithm that is presented in Algorithm 2. The crawler collects information about the web site and stores it using the storage service. The SSL verifier gets the unprocessed URLs from the storage (use function $extractAllHttpsUrls()$). For each URL, the SSL verifier checks for an SSL connection to the server. If an SSL connection request is accepted by the server, then it inspects the certificate chain and validates each certificate in the chain towards the site certificate. The validation includes checking the certificate expiration date, and the basic constraint field specifying if the certificate can be used to to sign other certificates. If the web site rejects the HTTPS request, or the web site does not have a valid certificate, and thus no support for HTTPS. Additionally, the IP addresses used by the web site is also cached in the DNS service. This would serve two purposes: (i) protects from DNS spoofing as we send the trusted IP address of the web-site (ii) helps improving the time taken to load the page as there is no need of DNS requests. Finally, the browser initiates a HTTPS connection if it learns from the SSE that web site supports HTTPS connections.

Algorithm 2 SSL Verifier

```
1: $urls = extractAllHttpsUrls()$;
2: for each URL in urls do
3:   $certChain = getCertificate()$;
4:   if $certChain$ is valid then
5:     for each $certificate$ in the $certChain$ do
6:       $params = extractCertParameters()$;
7:       $store(params, certificate)$;
8:     end for
9:   else
10:    mark invalid certificate against the URL;
11:  end if
12: end for
```

C. Countering Web-based Phishing

At present, most browsers are incorporate phishing filters that utilize existing phishing site databases like PhishTank [24]. In these phishing site databases, phishing sites are manually fed by users who come across phishing sites. Phishing sites usually appear for only short span of period, and thus most of listed phishing sites in these databases are expired, which make the database less effective.

To address the described problem of existing phishing filtering techniques, we present a realtime phishing filter as part of SSE to inspect phishing sites in real time. Our solution is highlighted in Algorithm 3, which includes four components: IP checker ip_i, Bayes classifier P_i, Certificate checker $Cert_i$, and Google page rank G_{pr_i}.

Algorithm 3 Phishing filter

```
for web page $i$ do
  $ip_i = 1 - \frac{1}{logt}$;
  $P_i = getProbabilityForPhishing(URL)$;
  $Cert_i = hasValidCertificate(URL)$;
  $G_{pr_i} = getGooglePageRank(URL)$;
  if $confidence_i \leq \tau$ then
    return phishing site found;
  else
    return trustable site;
  end if
end for
```

First, the information of phishing site is collected from the other phishing banks and the Bayes classifier is trained based on these inputs. During training, we remove the common occurring words using $tf-idf$ values (illustrated in Appendix A). All $tf-idf$ values which are less than 0.5 are not considered for further processing. The remaining words in a page are used to train the Bayes classifier. Based on the training results, we calculate the P_i (probability of a page i being a phishing site) of each crawled web page. For each web page i, we also get the corresponding Google page rank, IP address checker value, and certificate checker value. Next, we calculate the $confidence_i$ value using (2), which will be described shortly, and store the result in SSE's database.

$Cert_i$ can be derived from our SSL verifier. We use t to represent the number of days that an IP address has not hosted a phishing site. The value of ip_i can be computed using the following formula:

$$ip_i = \begin{cases} 1 - \dfrac{1}{\log t}, & \text{if } t > 1; \\ 1, & otherwise. \end{cases} \tag{1}$$

When ip_i is calculated for the first time, t is set to a large number greater than 30 for a non-phishing site, or t is set to 1 for a phishing site collected from different phishing repositories.

To take a comprehensive consideration for the four evaluation components, the $confidence_i$ is computed by the following weighted function:

$$confidence_i = \alpha P_i + \beta Cert_i + \gamma ip_i + \delta G_{pr_i}, \tag{2}$$

where α, β, γ, δ are constants and $\alpha+\beta+\gamma+\delta = 10$, P_i is probability of URL being a phishing site derived from the Bayes classifier and $P_i \in [0, 1]$, $Cert_i$ is a boolean, whether the checking of SSL verifier is passed or not, ip_i is calculated from (1), and $G_{pr_i} \in [0, 1]$ is the value of Google page rank for the corresponding URL i.

To obtain the values of α, β, γ and δ, SSE uses an approach based on linear classification [25]. If the $confidence$ value is less than the threshold τ, then the site is phishing site. To determine the values of four constants and τ, we used 100 phishing web sites and 100 non-phishing web sites as the training sample to derive the following values for SSE phishing filter: $\alpha = 3$, $\beta = 4$, $gamma = 1$, $\delta = 2$, and $\tau = 3$.

D. Use Caching to Reduce the Delay incurred by SSE Service

Using SSE, additional delay will be added into the browsing. In order to reduce the introduced delay, we maintain 2 levels of caching both at the user side and SSE server side.

At the SSE server side, we maintain all phishing sites, whether a web site provides a valid certificate or not, in SSE's main memory. SSE applies bloom filter technique [26] to efficiently store phishing sites information. In this way, the processing time of the SSE server will be greatly reduced. At the user side, the browser can cache previously visited phishing site information derived from SSE service for later use. This is illustrated from lines 2 to 4 of Algorithm 1. The cached site has a expiry time associated with it. When the current time crosses the expiry time, the browser requests the SSE service to re-evaluate the site.

V. Performance Evaluations

In this section, we present the performance evaluation of SSE. First, we present the effectiveness of using SSE to counter MITM and phishing attacks; and then, we present the performance issues of SSE such as computations and delay.

A. Performance Analysis

1) SSLStrip MITM attack: To evaluate our solution under MITM attacks, we enumerate three web browsing scenarios and address the effectiveness of our solutions accordingly. In our evaluation, we requested 10 users to visit different banking websites, from a large pool of banking websites, for a span of 30 minutes with the SSE browser extension installed. We had two attackers, present in the same network, trying to perform a MITM attack using SSLStrip.

In the first scenario, when an HTTP connection is requested to the banking site like www.bankofamerica.com, the website sends an HTTP 302 redirect message to start a secure connection. In second scenario, when an HTTP connection is requested, the banking site like www.usbank.com replies back with the content in an unsecured manner. Both the above mentioned scenarios will respond to a secure SSL request following HTTPS standard when the user initiates the connection using HTTPS. In the last scenario, the banking websites like www.discovercard.com initiates the HTTPS connection only when the user hits submit button after filling up the username and password fields. The HTTPS is handled by additional javascript codes embedded in the html web page. Our current development of user-side extension does not inspect the html content and thus fails in this scenario. However, this problem can be fixed by improving the user-side browser extension to inspect the html content, which is scheduled for our future development.

2) Phishing filter: To evaluate our phishing filter we considered two parameters, false positives and false negatives. If a non-phishing site is blocked as a phishing site, then it is false positive; and if a phishing site is not blocked, then it is false negative. Both false positive and false negative value should be low for a good phishing filter. However, false negative is more important than a false positive for browsing.

We compared SSE phishing filter with built-in phishing filters of firefox 3.5, Chrome 2.0, and IE 8.0. All these browsers update their local phishing cache every 30 minutes from their own phishing sites reporsitories. In our experiment, we collected newest reported phishing sites from www.phishtank.com with respect to different time frames. The time frames, which we used in our evaluation are the phishing sites reported in last 30 minutes, 60 minutes, and 90 minutes. We collected 100 URLs from each time frame. We tried visiting those reported phishing sites using the compared browsers. The expected behavior of each browser is to warn the user before he/she accesses those phishing sites. If the page gets loaded without any warning messages, then we considered it as a false negative. We then use SSE phishing filter to evaluate the phishing sites. We then calculate the percentage of false negatives of each system considered. The Fig. 5 shows the percentage of false negative of each browser and the SSE phishing filter. The x-axis of the graph shows the time frame and the y-axis is the percentage of false negatives. In each time frame, we compare the false negative results obtained. The graph shows that the percentage of false negatives of the browser reduces as the time frame increases (from 30 minutes to 90 minutes). The percentage of false negative of the SSE phishing filter is lower in all time frames compared to the evaluated browsers.

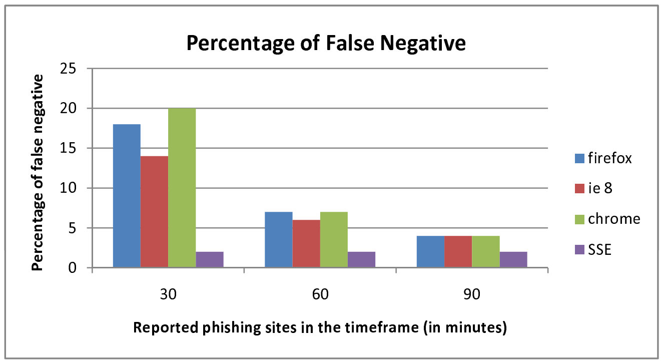


Fig. 5. Percentage of False Negative.

To evaluate the false positives of SSE phishing filter and the browsers, we randomly collected 100 non-phishing sites which are close to phishing site, from www.phishtank.com. These URLs were from unknown category of www.phishtank.com, which have been verified by us as non-phishing sites. We expected that the browsers do not block these non-phishing sites. We calculated the percentage of URLs that were blocked by the browsers and the SSE phishing filter. As shows in the fig 6, the false positives of all the techniques were very low. However, the percentage of false positives of SSE phishing filter is the lowest.

B. Delay and Resource Consumption of SSE

The current web browsers, on a user request to load a web page, will directly contact the web server of the corresponding domain given in the URL. The browser then waits for the response from the web server. Upon receiving the response, the browser renders the web page to the user. Comparing with this model, the SSE solution uses one extra validation request with SSE service. This validation request is done over an SSL connection. There is a delay

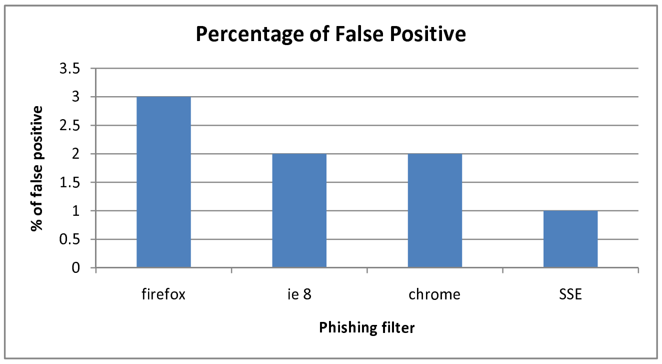


Fig. 6. Percentage of False Positive.

associated with this extra validation added into the normal browsing models. In this section, we discuss about how much percentage of the delay our system would cause and also analyze how much load SSE system can withstand with a given set-up.

Our experiment is based on the theoretical queuing model using little's law to generate user requests to the SSE server. Little's law states that the long term average number of customers (L) in a stable system is equal to the long-term average arrival rate (λ) multiplied by the average time taken to service (W), which is described as follows:

$$L = \lambda W.$$

The arrival rate in our system is the number of requests arrived at SSE server per second. Let the average arrival rate be λ per second. We assume that the arrival rate λ follows poisson distribution, in which we can approximately generate one request every $t = 1/\lambda$ seconds. In other words, a request would be generated $t - \Delta t$ or at $t + \Delta t$, which would average out close to t over a long period of time. In this way, we can emulate the real scenarios with multiple SSE service requests sent from different users.

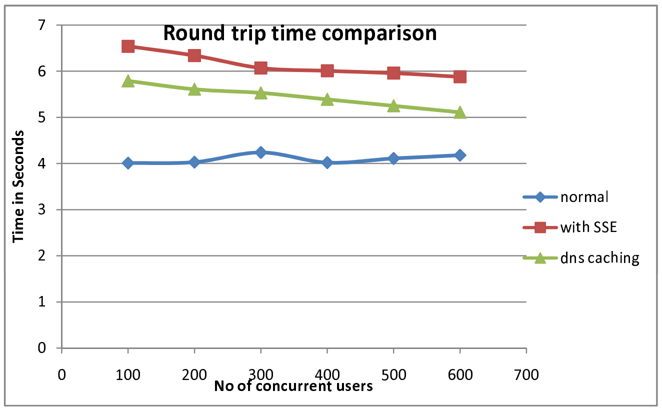


Fig. 7. Comparison of current web browsing loading time and SSE approach.

In our experiment, we allow each user to send 50 SSE requests for every 15 seconds. Out of those 50 URLs, some URLs were banking sites where an SSL connection to these banking sites is mandatory. The experiment was performed after clearing the cached URLs at SSE server. We emulated 100 to 600 users' behavior and studied the round trip time taken with different models. The results obtained are shown in the Fig. 7. The x-axis of the graph shows the number concurrent users in the system and the y-axis is the total time from the user requests the web site to the browser loads the request web page. The blue curve is the normal scenario that takes place without using SSE service, where the average time taken to load a page is about 4 seconds. The red curve shows the experiment results with SSE service. Initially, the SSE service takes more time to respond and builds up the cache. The successive request for validation takes less time as the URLs have already been verified and cached.

The average time saved by the rewriting the IP-address with domain name was around 9%. The graph shows the round trip time taken with rewriting IP address is shown in green color. During our experiment, we discovered that the average overhead caused by the secure connection to the SSE service was approximately 12% of the entire round trip time including SSE request, SSE response, web server request, and web server response.

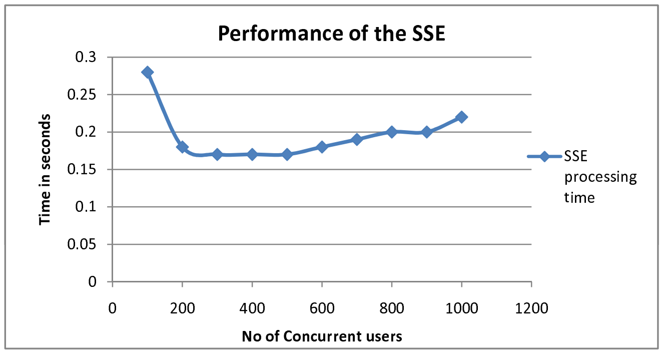


Fig. 8. Processing time taken by the SSE service.

Using the same experiment set-up, we also calculated the performance of the SSE. Particularly, we evaluate the processing time taken by the SSE service to respond after getting a validation request. The average time to respond is calculated by increasing the number of users accessing the SSE service. The Fig. 8 shows the values of average time taken by the SSE service to respond, with the x-axis representing the number of concurrent users and y-axis representing the time taken to process in seconds. The experiment was started with 100 users and the cache at the SSE server was cleared initially. With time going, the SSE will caches the inspected web sites. From the figure we notice that the processing time is high at the begining that confirms the advantage of using caching. Then the processing time drops down due to many requests can be answered by using SSE caching.

The SSE system distributes the work load among 3 servers. The crawler, DNS service, and URL service runs on different machines. The SSE service and Storage service, SSL verifier and phishing filter runs on two different machines. With the current set-up we have we would be able to support approximately few thousand concurrent users simultaneously.

VI. Conclusion and Future Work

In this paper we have proposed a SSE framework, which could counter web based MITM and phishing attacks. This service can be accessed by downloading the browser extension from https://www.securesearchengine.eas.asu.edu. The SSE involves minimum interventions of humans for security decisions. The performance of the SSE phishing filter has low false positives and false negatives. We will extend the browser extension by providing support for those websites which starts HTTPS request from a Javascript.

References

[1] B. Hayes, "Cloud computing," *Commun. ACM*, vol. 51, no. 7, pp. 9–11, 2008.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," 1999.

[3] M. Moxie, "Sslstrip software," http://www.thoughtcrime.org/software/sslstrip, 2009.

[4] A. Freier, P. Karlton, and P. Kocher, "The SSL protocol version 3.0," 1996.

[5] S. Whalen, "An introduction to arp spoofing," *Node99 [Online Document], April*, 2001.

[6] D. Sax, "DNS spoofing (malicious cache poisoning)," *URL: http://www.sans.org/rr/firewall/DNS_spoof.php November*, vol. 12, 2000.

[7] G. Aaron and R. Rasmussen, "Anti phishing working group - global phishing survey: Trends and domain name use in 2h2008," http://www.antiphishing.org/reports/APWG_GlobalPhishingSurvey2H2008.pdf, 2008.

[8] T. Chomsiri, "HTTPS Hacking Protection," in *Proceeding of the 21st International Conference on Advanced Information Networking and Applications Workshops*, vol. 1, 2007.

[9] R. Oppliger, R. Hauser, and D. Basin, "Ssl/tls session-aware user authentication," *Computer*, vol. 41, no. 3, pp. 59–65, 2008.

[10] H. Xia and J. Brustoloni, "Hardening web browsers against man-in-the-middle and eavesdropping attacks," in *Proceedings of the 14th international conference on World Wide Web*. ACM New York, NY, USA, 2005, pp. 489–498.

[11] C. Jackson and A. Barth, "ForceHTTPS: Protecting high-security web sites from network attacks," 2008.

[12] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, and J. Mitchell, "Client-side defense against web-based identity theft," in *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS04), San Diego*. Citeseer, 2005.

[13] R. Dhamija, J. Tygar, and M. Hearst, "Why phishing works," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM New York, NY, USA, 2006, pp. 581–590.

[14] A. Herzberg and A. Jbara, "Security and identification indicators for browsers against spoofing and phishing attacks," *ACM Trans. Internet Technol.*, vol. 8, no. 4, pp. 1–36, 2008.

[15] Y. Zhang, S. Egelman, L. Cranor, and J. Hong, "Phinding phish: Evaluating anti-phishing tools," in *Proceedings of the 14th Annual Network and Distributed System Security Symposium*. Citeseer, 2007.

[16] Y. Cao, W. Han, and Y. Le, "Anti-phishing based on automated individual white-list," in *Proceedings of the 4th ACM workshop on Digital identity management*. ACM New York, NY, USA, 2008, pp. 51–60.

[17] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair, "A comparison of machine learning techniques for phishing detection," in *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*. ACM New York, NY, USA, 2007, pp. 60–69.

[18] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: a content-based approach to detecting phishing web sites," in *Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA: ACM, 2007, pp. 639–648.

[19] R. Arends, R. Austein, M. Larson, and D. Massey, "DNS Security Introduction and Requirements," 2005.

[20] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Resource Records for the DNS Security Extensions," 2005.

[21] ——, "Protocol Modifications for the DNS Security Extensions," 2005.

[22] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[23] A. Heydon and M. Najork, "Mercator: A scalable, extensible web crawler," *World Wide Web*, vol. 2, no. 4, pp. 219–229, 1999.

[24] "Phishtank," www.phishtank.com.

[25] C. Bishop *et al.*, *Pattern recognition and machine learning*. Springer New York:, 2006.

[26] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

VII. Appendix

A. Computation of tf-idf

This is common technique used in information retrieval and text mining to give less importance to the commonly occuring word. There are different types of ways to look at a document. Bag of words, bag of letters, bag of shillings (a set of consecutive words) etc., Most of the current search engines uses the bag of words model i.e. a document is seen a set of words. The $tf - idf$ approach is based on the bag of words model.

Let document, d_i, have set of words say $< t_1, t_2, t_3..., t_w >$ and i ranges from 1 to n. The count of a word occurring in the document is called as term frequency,tf, of that word in that document. Let the t_{ij}, be the term frequency of j^{th} word (j is not the index of word in the document) in i^{th} document. Let number of documents in which a particular word has occurred be k_{t_j}, where t_j is the word. The inverse document frequency, idf, is defined as the log of total number of document, n, over k_t. The idf can be represented as follows

$$idf = \log(\frac{n}{k_{t_j}}). \tag{3}$$

The $tf - idf$ value can be computed using the following equation:

$$tf - idf = tfxidf. \tag{4}$$

Using 4, we can reduce the term frequency of an unimportant word in a document. Intuitively, if a word frequently occurs in many documents, then we can consider it to be an unimportant word.