2013 IEEE Seventh International Symposium on Service-Oriented System Engineering

# Secure Web Referral Services for Mobile Cloud Computing

Le Xu*, Li Li†, Vijayakrishnan Nagarajan*, Dijiang Huang*, and Wei-Tek Tsai*

*Arizona State University, Email: {le, dijiang, vnagara5, wtsai}@asu.edu
†Wuhan University, Email: lli@whu.edu.cn

***Abstract*—Security has become a major concern for mobile devices when mobile users browsing malicious websites. Existed security solutions may rely on human factors to achieve a good result against phishing websites and SSLStrip-based Man-In-The-Middle (MITM) attack. This paper presents a secure web referral service, which is called Secure Search Engine (SSE) for mobile devices. The system uses mobile cloud-based virtual computing and provides each user a Virtual Machine (VM) as a personal security proxy where all Web traffics are redirected through it. Within the VM, the SSE uses web crawling technology with a set of checking services to validate IP addresses and certificate chains. A Phishing Filter is also used to check given URLs with an optimized execution time. The system also uses private and anonymously shared caches to protect user privacy and improve performance. The evaluation results show that SSE is non-intrusive and consumes no power or computation on the client device, while producing less false positive and false negative than existing web browser-based anti-phishing solutions.**

**Keywords**: Mobile Cloud, Security, Web

## 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.

Using mobile devices, the web-based communications have become easy targets for attackers to compromise end-to-end communications, e.g., 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 (e.g., Secure Socket Layer [4] based HTTP, or HTTPS) have been widely used to protect Internet users from being attacked. However, strong cryptography-based algorithms could not prevent security protocol designers from overlooking the weakness of human beings in the human-in-the-loop types of security protocols. SSL supported web browsing is an example of this kind. In this paper, our research focuses on two major security problems incurred due to human errors: e.g., SSLStrip MITM attack and web based phishing attack, which require users to be involved to make decisions on accepting or rejecting a potentially breached web site.

In SSLStrip attack, attackers explore the vulnerability that a 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 then intercepts the HTTP 302 redirect message and initiate an SSL session to the web sever without forwarding the redirect message to the user. After setting up an SSL connection to the web server, the attacker sends un-encrypted web page to the user. It is quite common for users overlooking the protocol name, which should be changed from HTTP to HTTPS, and a small lock logo should appear in the bottom of the web browser. This problem becomes even severe when lightweight small-screen mobile devices are used. If the user types his username and password in 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 un-encrypted.

Phishing is another way to explore the human weakness by using fake web sites. Similar to SSLStrip attack, Phishing attacks also present the exact same web content and page layout to users, except the web address is different and the lock logo does not appear in browser's status bar. To counter phishing attacks, many existing web browsers, e.g., Firefox, incorporate anti-phishing filters by checking several phishing-web site repositories, and then send alerts to users if they happen to visit a phishing web site. However, based on our testing, the false negative rate is very high. This is due to several reasons. First, as cost of hosting sites has come down and there are tools like dyndns.org, which can map the dynamic changes of an IP address with a domain name, attackers can easily change their domain name and corresponding IP address. In addition, 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, this paper presents a

978-0-7695-4944-6/12 $26.00 © 2012 IEEE
DOI 10.1109/SOSE.2013.94

secure web referral system, named Secure Search Engine (SSE), for mobile devices. The system uses mobile cloud-based virtual computing [7], [8], [9] and provides each user a VM as a personal security proxy where all Web traffics are redirected through it. Each VM is isolated from other VMs to protect the user's privacy. Within the VM, the SSE checks the Web traffics for potential MITM or phishing attacks. SSE is designed as an automatic web referral service involving minimal interventions from humans for security decisions. SSE returns deterministic answers of security inspections to either accept or reject the Web request, and thus it can remove potential human errors. SSE also includes a Phishing Filter that can check several existing phishing-site repositories, and validate site certificates to identify a phishing site.

The rest of this paper is organized as follows. Section II describes related works. Section III describes the detailed designs of the system. Section IV shows testing results from the implemented system. Finally, section V concludes and describes future work.

## II. Related Work

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 is 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 an SSL connection, then the user needs to manually set a policy in ForceHTTPs. This approach does not prevent MITM attacks since attackers can intercept the HTTPS request and return a "no-HTTPS-support" message and force the web browser to initiate HTTP sessions.

Researchers in [12] developed an anti-phishing toolbar for web browsers. However, studies showed in [13], 23% of the people do not look at the address bar, status bar, and security indicator when they browsing web sites, and let alone the small screen size when using mobile devices. In [14], the authors compared the effectiveness of using different phishing tools such as Firefox2, IE7, Spoofguard [12], etc. Their results showed that all evaluated tools are less 90% effective in identifying phishing web sites. Zhang et al, in [15], proposed a Phishing Filter "CANTINA", where they make use of robust hyperlinks and *t*erm-*f*requency - *i*nverse *d*ocument *f*requency ($tf - idf$), where $tf - idf$ is a technique to give less importance to the common occurring words. 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 the 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 [16] that cannot be crawled by Google's search engine immediately.

To address the MITM and phishing issues, a number of proxy-based security models have been proposed. Tahoma [17] uses a browser OS running on top of a client-side Xen-managed virtual machine to serve as a local proxy to scan web applications. Flashproxy [18] targeted the performance and security of Flash object browsing on mobile devices. WebShield [19], SpyProxy [20], AjaxScope [21] and BrowserShield [22] proposed similar proxy-based web security models where they use a sandbox on the remote proxy to execute and render the web application while detecting security threats by monitoring the application behaviors. These approaches incur overheads when switching from one virtual machine to another. Moreover, since they share virtual machines among users, the user's privacy may be an issue. The system is deployed on centralized server and may suffer from single-point-failure problems. Zscaler [23] proposed a policy-based, secure Internet access model for mobile device that is built upon cloud services. It allows administrators to enforce user level policies without any software or hardware deployment on the devices. However, Zscaler lacks the data isolation and privacy protection for individual users.

## III. Architecture and Implementation

This section presents the architecture and components of the mobile cloud-based SSE system. The terminology and description are based on [24], which describes the basic components and models to establish an efficient and scalable search engine.

The overall system architecture is shown in Fig. 1. The major components of the system are described as follows.

• **Xen Server Cluster**: The Xen server cluster is a set of Xen servers [25] that provisions VM resource pools. The Xen platform provides XenAPIs, which are used by the web server to provide VM management and configuration functions to the mobile users.

• **Web Server**: The web server hosts a website to provide a management portal for users and system administrators. A database within the web server manages the DNS names, IP addresses and VM software configurations. The web server also works with Xen other servers in the cloud to perform system instructions. For instance, it works with DNS/DHCP servers to assign domain names and IP addresses to the newly created VMs.

• **Mobile User VM**: Each mobile cloud user has a dedicated VM that incorporates several components to provide features such as http proxy, caching and logging. The VM also works with the SSE service to detect and counter MITM and phishing attacks. Fig. 2 shows the components of a mobile user VM.

• **Portal Gateway & Portal Network**: The Portal Gateway is the access point for mobile users to access internal VMs and web services. The Portal Network serves as a data exchanging channel between internal servers.

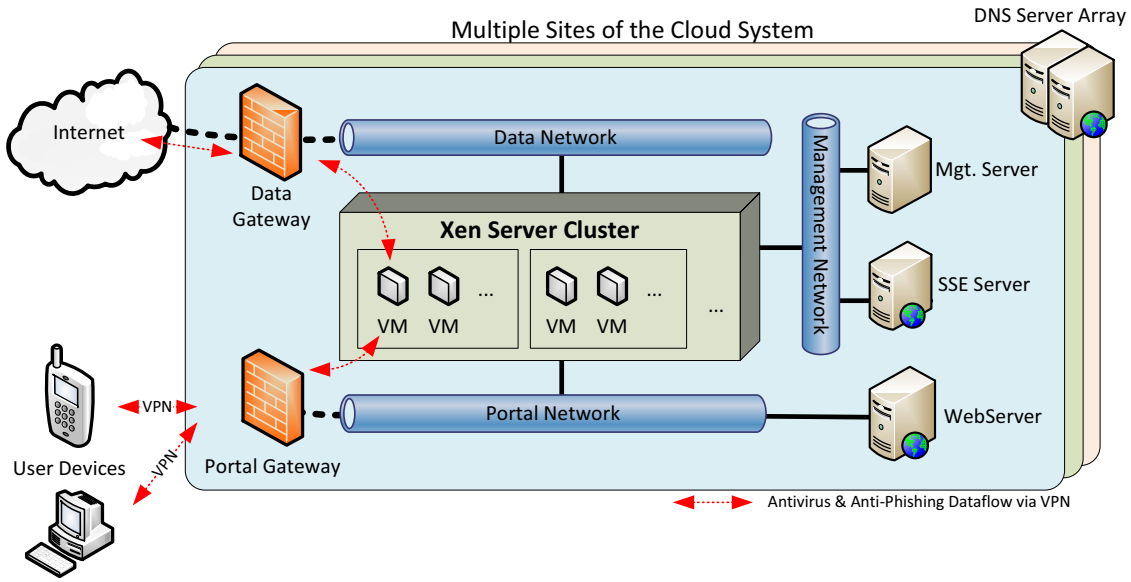


Fig. 1. System Components of the SSE Mobile Cloud System.

• **SSE & Management Server**: SSE server provide the SSE service proposed in this article, and the Management Server is in charge of the system resource allocation.

• **Data Network & Data Gateway**: The networks that are used by VMs and SSE fetch web data from the Internet and send to the user mobile devices.

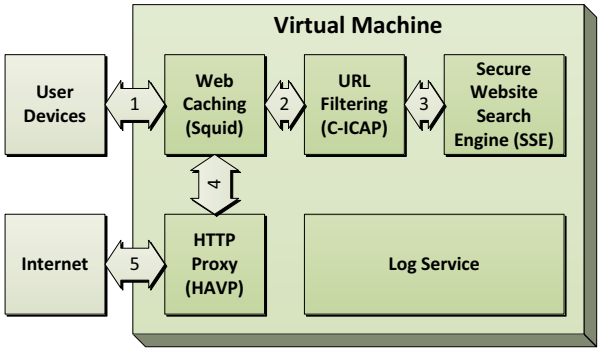


Fig. 2. Components of the user VM.

### A. Secure Search Engine

*1) Secure Search Engine Architecture:* The SSE is a cloud service that can be used by each mobile user VM. To provide web proxy and caching functions, a mobile user VM incorporates multiple components, as shown in Fig. 2. The implementation of SSE is through a layered service architecture, where the higher layer services make use of the services at its immediate lower layer. Fig. 3 shows the main services of the SSE architecture.

• **SSE service**: SSE service answers all the user queries generated by web browsers. Replies of the SSE service are made from the inspection results from the SSL verifier and the Phishing Filter.

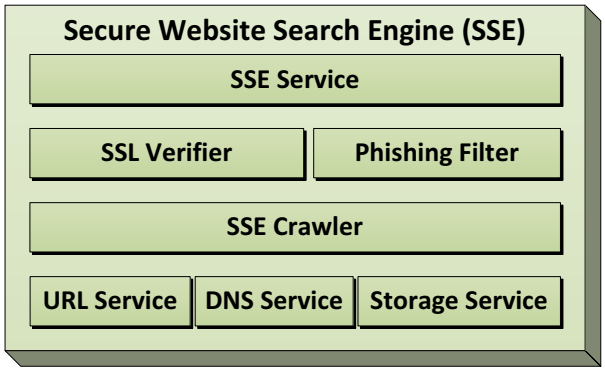


Fig. 3. SSE Service Models.

• **SSL verifier**: SSL verifier is one of the core services in 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 the validation results in the certificate repository maintained by the SSE.

• **Phishing Filter**: Phishing Filter is another core service in SSE, in which it inspects each web page linked through the inspecting URL. Using the learning algorithm explained in Section III-C, the Phishing Filter checks if a web site is a legitimate site or a phishing site

• **SSE Crawler**: The SSE 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, which are called “seeds”. As the SSE Crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs.

The SSE Crawler collects all the security information from each URL, such as public key certificates.

• **DNS service**: DNS service is critical for a search engine as mentioned in [26], where about 87% of web crawling traffic is just due to DNS traffic. The DNS service is a caching mechanism to make the Crawler more efficient without causing too much repeated DNS traffic. The DNS service is also used by Phishing Filter to get the IP addresses of inspecting domain names.

• **URL service**: The URL service records URLs visited by the Crawler including both visited URLs and to be visited ones. Additionally, to make the Crawler scalable and allow recovery from crashes, the URL service is responsible for restoring the state of the Crawler in case of the system crashes.

• **Storage service**: The storage service stores all the information from SSE Crawler and link inspection results.

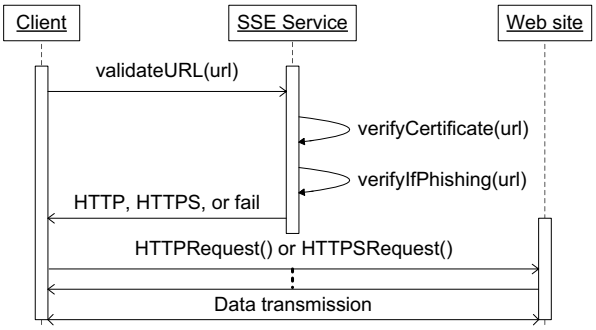


Fig. 4. Steps involved between a VM and the SSE service.

*2) Procedures of Using SSE Service:* The Fig. 4 explains the data flow among the user, the SSE, and the web request destination. When the SSE receives a web request, it inspects 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 phishing 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) sending an access denial webpage to the user if the web site does not pass any one of filters of the SSE server.

Fig. 5 presents the data flows within the SSE server. Steps: (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 then stores the

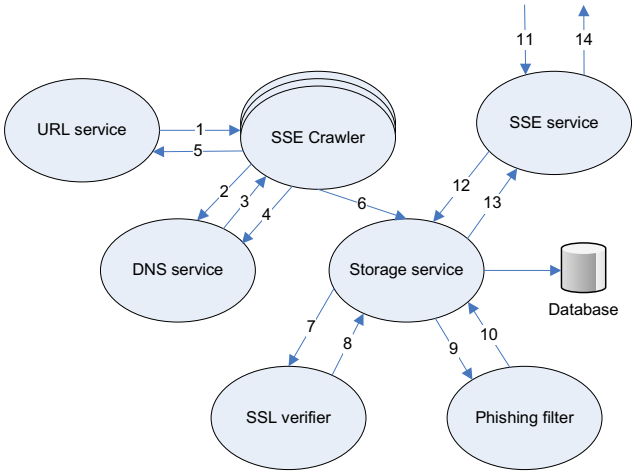


Fig. 5. Dataflow within the Secure Search Engine.

certificates. (9 & 10) The Phishing Filter runs the algorithm to find out whether a web site is a phishing site or not. (11) The user VM requests the SSE to verify a web site. (12 & 13) The SSE service uses the storage service to get the corresponding data to evaluate. (14) The SSE service answers the query.

Initially, the URL service is populated with a set of seed URLs. The Crawler picks up an unprocessed URL from the URL service and sends an HTTP request. 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 new URLs from the HTTP responses and passes them to the storage service. The stored web site information will shorten the search responding time when the searched web sites have been already inspected. Crawling services are usually operated periodically based on a certain time interval. Thus, between the crawling time window, new phishing web sites can be issued. To handle the phishing sites that have not been crawled, the SSE needs to process a real-time security inspection.

*3) SSE Processing Algorithm:* Algorithm 1 presents the SSE service processing algorithm. Lines 2-4 represent the inspection based on cached information, which is explained in Section III-D. The browser redirects all 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 when a web browser connects to the Internet. SSE service checks with its database through the storage service to see (i) if the site is already previously evaluated as a phishing site, and (ii) 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 if the user encounter MITM or 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

**Algorithm 1** *SSE Processing.*

1: The browser retrieves $URL = getTheURL()$;
2: **if** The $URL$ is available in VM's cache and the cache expiration timer is not expired **then**
3: Take actions based on the cached attributes;
4: **else**
5: The browser sends a request 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 message 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 message to the browser;
21: the browser goes to Step 8;
22: **end if**
23: **end if**

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 real-time 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, the paper considers an example of a user accessing a banking web site, e.g., www.usbank.com, and illustrate how SSE works to defend the SSLStrip MITM attack. In this example, 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

**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**

to a remote web site. For example, a mobile user usually just types usbank.com in the address bar. By default, the browser will interpret the request as an HTTP request, i.e., http://www.usbank.com, which sets the destination web server's port number to 80. By intercepting the user's request, the attacker can establish an SSL session to the web server and replies to the user with an unencrypted web page using HTTP protocol. To counter the described SSLStrip MITM attack, the SSE automates the security inspection procedure and the user will be notified only if potential MITM attacks are detected. This minimizes the human factors in the process.

The SSL verifier algorithm (Algorithm 2) can be used to illustrate the steps involved in countering an MITM attack. 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 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 an HTTPS connection if it learns from the SSE that web site supports HTTPS connections.

Once an HTTPS connection is initiated, the HTTPS protocol will enforce an SSL connection with requested port number 443. When initiating an HTTPS connection, a user expects to receive a valid certificate and uses its public key to derive a shared key for securing future communications. Since MITM attackers do not have a valid private key of the web server's certificate, they cannot derive the shared

**Algorithm 3** Phishing Filter

**for** web page $i$ **do**
  Bayes classifier will return with a probability of a web site being a phishing site $P(i) = getProbabilityForPhishing(URL)$;
  $Cert(i) = hasValidCertificate(URL)$, checks if the URL has a valid certificate;
  $GPR(i) = getGooglePageRank(URL)$, gets the Google page rank which will be normalized to 0-1 scale;
  Compute $IP(i) = 1 - \frac{1}{\log t}$, which is illustrated in (1);
  Compute (2) to get the *confidence* value for page $i$;
  **if** $Confidence(i) \leq confidenceThreshold\tau$ **then**
    return phishing site found;
  **else**
    return trustable site;
  **end if**
**end for**

key. As a result, MITM attackers cannot intercept and decrypt the ciphertext sent to U.S. Bank's web site. Thus, the MITM attack fails. Although the MITM attacker can cause a Denial-Of-Service attack by not forwarding the intercepted packets, this still prevents the attackers from compromising the users' private information.

### C. Countering Web-based Phishing

At present, many browsers incorporate phishing filters that utilize existing phishing-site databases like PhishTank www.phishtank.com. In these databases, phishing sites are manually fed by users who come across them during web surfing. Phishing sites usually appear for only short span of period, and thus most of listed phishing sites in these databases are expired, which make them less effective.

To address the described problem of existing phishing filtering techniques, the paper presents a real-time Phishing Filter as part of SSE to inspect phishing sites. The solution is highlighted in Algorithm 3, which includes four independent components: IP checker $IP(i)$, Bayes classifier $P(i)$, Certificate checker $Cert(i)$, and Google page rank $GPR(i)$.

The phishing sites often resemble like a financial web site and the content of the web page is also similar to the financial web page. Thus, a Phishing Filter must take a comprehensive and systematic approach, which is adaptable to their changes. To this end, we utilize machine-learning techniques that could learn phishing site properties based on the changes of phishing sites. In Algorithm 3, we use a supervised learning-based classification technique to improve the accuracy of detecting phishing sites. This supervised learning technique adapts to the changes made by the phishing sites to get a better result. The supervised learning is based on Bayes classification [27], which learns from a training set given about the phishing sites and uses words commonly existing in phishing sites as the features for Bayes classification. The training samples are extracted from phishing databases such as PhishTank.com. The Basis of Bayes classification is presented in [28].

Apart from the Bayes classification, the Phishing Filter checks with the SSE database and see if the IP address has hosted any phishing site in the past. The filter also checks if the site has a valid certificate and its Google page rank. These components are chosen for following reasons. (i) Phishing sites usually do not provide valid certificates since the process for obtaining a certificate is not desired for phishing attackers. (ii) Google page rank is calculated using many parameters to evaluate the popularity of a web page. One important parameter is the propagation of trust among web sites. A trusted web page should always refer to other trusted web pages. Intuitively, this would bring down the rank of a phishing web site since no trusted web pages refer to it. More importantly, phishing sites are hosted for a short span of time and the probability of crawling the web page would be less. In our established prototype, a weighted average based on the presented four parameters is calculated and normalized to a range of [0,10]. If the final value, $confidence$, is less than an allowable threshold then the web page is reported as phishing web site.

As shown in Algorithm 3, first, the information of phishing site is collected from 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 [15]). 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 $i$, the system also gets the corresponding Google page rank, which is normalized in scale of 0 to 1 and the value of GPR is assigned. From the SSL Verifier we also get the certificate information, whether a domain has a valid certificate or not.

Finally to get the value of $IP(i)$ we use the following computation. Let $t$ 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:

$$IP(i) = \begin{cases} 1 - \frac{1}{\log t}, & \text{if } t > 1; \\ 0, & otherwise. \end{cases} \quad (1)$$

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. As the $t$ increases the $\frac{1}{\log t}$ decreases and the value of $IP(i)$ increases. Thus, increase in value of $IP(i)$ would increase the value of $confidence_i$.

To take a comprehensive consideration for the four evaluation components, the $Confidence(i)$ is computed by

**Algorithm 4** Crawler

1: **while** Get an unprocessed $URL$ from URL service ($URL = getURLFromURLservice()$)! $= NULL$ **do**
2: $URL = replaceDomainWithIP(URL)$, rewrite the domain name with IP address(es).
3: $pageContent = getPage(URL)$, send and receive the content of the URL.
4: $URLS = extractURLS(page)$, extract all the URL links in the web page with new domain names.
5: $DNSService(URLS)$, update the DNS service with the extracted URLs. The DNS service would try to cache the IP Address(es) of the un-cached domain name.
6: $setNewUrlsInURLService(URLS)$, update the extracted URLs to URL Service, which maintains the state by having visited and not visited URLs classes.
7: **end while**

the following weighted function:

$$Confidence(i) = \alpha \cdot P(i) + \beta \cdot Cert(i) + \gamma \cdot IP(i) + \delta \cdot GPR(i), \quad (2)$$

where $\alpha$, $\beta$, $\gamma$, $\delta$ are constants and we set $\alpha + \beta + \gamma + \delta = 10$, $P(i)$ is probability of URL being a phishing site derived from the Bayes classifier and $P(i) \in [0, 1]$, $Cert(i)$ is a boolean value whether the checking of SSL verifier is passed or not, $IP(i)$ is calculated from (1), and $GPR(i) \in [0, 1]$ is the value of Google page rank for the corresponding URL $i$.

To obtain the values of $\alpha$, $\beta$, $\gamma$ and $\delta$, SSE uses an approach based on linear classification [29]. If the $confidence$ value is less than the threshold $\tau$, then the site is phishing site. To determine the values of four constants and $\tau$, 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: $\alpha = 4$, $\beta = 3$, $\gamma = 1$, $\delta = 2$, and $\tau = 3$.

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

Using SSE, additional delay will be added into the browsing processes while the SSE Service is checking the web URL. To reduce it, two levels of caching are maintained at both the VM side and the SSE server side.

At the SSE server side, history for checked phishing sites are maintained in database. The SSE uses Bloom filter [30] data structure to efficiently store the phishing sites information. The paper has configured the bloom filter to have 5% false positive for about one million URLs.

At the VM side, the http proxy can cache previously visited phishing site information derived from SSE service for later use. Before sending a URL to the SSE, the VM checks whether the URL is present in the cache and its timestamp is not expired. Only when the current time exceeds the expiration time, the VM requests the SSE service to re-evaluate the site. This step is illustrated from lines 2 to 4 of Algorithm 1.

## IV. Performance Assessment

This section presents the effectiveness of SSE to counter MITM and phishing attacks and computations and delay performance.

### *A. System Setup*

*1) SSE Crawler:* The SSE crawler is developed as a multi-threaded application as shown in Algorithm 4. It fetches web pages from Internet, extracts the URLs present in the page and iterates them while checking. To start with, the Crawler is fed with a few seed URLs from banking web sites. The Crawler gets an unprocessed URL from the URL service. Then, the Crawler gets the IP address(es) of the domain from the DNS Service. The DNS Service returns the IP address(es) of the domain if it has already cached the IP address(es). If the IP address(es) is not available in the DNS Service's cache, the DNS Service sends out a request to get the IP address(es) of the domain. The Crawler then starts fetching the web pages. For each fetched web page, the Crawler parses the page to extract the new URLs in it. The new URLs that contain new domains are given to DNS service and URL service to repeat the process. The URL service classifies these new URLs into visited and not-visited categories comparing with both the categories. The DNS service will check if new URLs domain-IP addresses mapping is available in its cache. If not available, the DNS service tries to cache the IP Address of the un-cached domain name. The content of the page, received from web server through the crawler, will be persisted in the local storage.

To obtain the optimized results, the crawler is configured with certain parameters like maximum depth to crawl for a given domain, number of concurrency, how often the crawler should take back-ups, size of the cache storage for DNS service and URL service, etc. Both the URL service and the DNS service make use of Least Recently Used (LRU) algorithm [31] for caching the data. The URL service makes use of Bloom Filter[30] to store the URL. The URL service also takes backup at regular intervals, which can also be configured.

*2) User Traffic Generator:* Table I lists the notations that will be used in the user traffic generator.

The theoretic traffic models are used to generate mobile user traffic for load and delay experiments. Particularly, the 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 ($\lambda$) multiplied by the average time taken to service ($W$), which is described as follows:

$$L = \lambda W.$$

TABLE I
USER TRAFFIC GENERATOR NOTATIONS

| List of notations used | |
|---|---|
| *Notations* | *Name* |
| $\lambda$ | arrival rate. |
| $u$ | total number of URLs in pool. |
| $RT_{t_i}$ | round-trip time of $i^{th}$ request. |
| $RT\_DNS_{t_i}$ | round-trip time of $i^{th}$ DNS request. |
| $RT\_WS_{t_i}$ | round-trip time of $i^{th}$ web service request. |
| $RT_t$ | summation of the time spent by each packet in network and the service time. |

The arrival rate in the system is the number of requests arrived at SSE server per second. Let the average arrival rate be $\lambda$ per second. Assuming that the arrival rate $\lambda$ follows Poisson distribution, the experiment approximately generates 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, the experiment can emulate the real scenarios with multiple SSE service requests sent from different users.

To compare the current browsing behavior with SSE model, a client in Python is developed using multi-threaded programming to generate SSE service requests. The client can be configured to simulate the current browsing behavior and also the SSE Model. For comparison purpose, client 1 represents the current browsing model simulation and client 2 represents the SSE model simulation.

Client 1 picks up a random URL from a pool of URLs and sends out an HTTP request and waits for the response. After receiving the response from the web server, client 1 waits for $t$ seconds before sending another request. The total number of requests sent by a client or the number of URLs in the pool is represented as $u$. For each request, represented by $i$, the client records the time $t_{i_1}$ when the request was sent, and the time $t_{i_2}$ when the client receives the response, and then we can calculate the round-trip time $RT_{t_i} = t_{i_2} - t_{i_1}$. $RT_{t_i}$ comprises of two entities: $RT_{t_i} = RT\_DNS_{t_i} + RT\_WS_{t_i}$, where $RT\_DNS_{t_i}$ is the round-trip time for the DNS request and $RT\_WS_{t_i}$ is the round-trip time for a web service request, and the $RT_t$ is summation of round-trip time of all the $i$ requests. The total round-trip time for all requests is represented as follows:

$$RT_t = \sum_{i=1}^{u} RT_{t_i}, \tag{3}$$

and the average $RT_{t_{avg}}$ for each request is

$$RT_{t_{avg}} = \frac{RT_t}{u}. \tag{4}$$

Client 2 sends the request to the SSE service, confirms if it is a trusted site and then sends a request to the URL by the mobile user. Again, the URL is randomly picked up from the pool of URLs and round-trip time is noted separately for the SSE service request/response and web service request/response, $RT\_SSE_{t_i}$ and $RT\_WS_{t_i}$ respectively. As our SSE service accepts only secure connection, the $RT\_SSE_{t_i}$ also comprises of the SSL overhead. A modified version of our SSE service and client uses unsecured connection to find out the overhead caused due to SSL connection. The secure SSE round-trip time can be represented as $RT\_SSEsec_{t_i}$ and $RT\_SSE_{t_i}$, for unsecured connection. The time taken for a DNS request is not taken into consideration, as the SSE service will rewrite the requested URL with the corresponding IP addresses. Using this user traffic generation model the performance overhead induced by the framework can be measured.

### B. Performance Evaluation of Phishing Filter

Two parameters are used to evaluate the Phishing Filter: false positives and false negatives. If a non-phishing site is blocked as a phishing site, then it is a false positive; and if a phishing site is not blocked, then it is a false negative. Both false positive and false negative value should be low for a good Phishing Filter. Generally speaking, false negative of detecting phishing sites is more important than a false positive for web browsing.

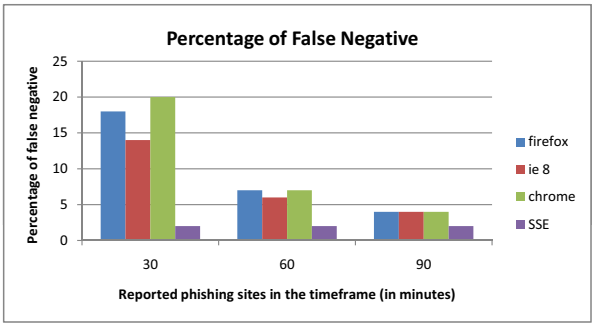


Fig. 6. Percentage of the False Negative.

To show the effectiveness of SSE Phishing Filter, the paper compares it 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 repositories. In the experiment, newest reported phishing sites from www.phishtank.com are collected with respect to different time frames: 30 minutes, 60 minutes, and 90 minutes. 1000 URLs are collected for each time frame and are tested using the compared browsers. The expected behavior of each browser is to warn the mobile user before he/she can access those phishing sites. If the page gets loaded without any warning messages, then it is considered a false negative. The results are compared with the SSE Phishing Filter on calculated percentage of false negatives of each system in Fig. 6. The $x$-axis of the graph shows the time frame and the $y$-axis is the percentage of false negatives. The graph shows that the percentage of false negatives of the browser reduces as

the time frame increases (from 30 minutes to 90 minutes). This is because more reported phishing sites are entered in phishing site repositories with time going. The percentage of false negative of the SSE Phishing Filter remains a low level, and it is lower in all time frames compared to the evaluated browsers. This performance evaluation shows that SSE filter achieves the best performance through active phishing scan. The false negatives of SSE Phishing Filter are mainly from phishing sites using figures to present statements.

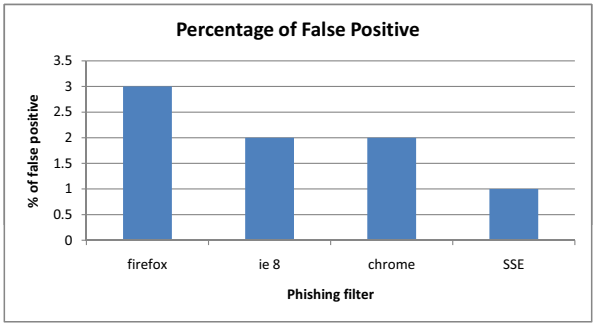


Fig. 7. Percentage of the False Positive.

To evaluate the false positives of SSE Phishing Filter and the compared browsers, the paper randomly collects 100 non-phishing sites that are close to a phishing site. These URLs were from unknown category of www.phishtank.com, which have been verified manually as non-phishing sites. It is expected that the SSE Phishing Filter does not identify these web sites as phishing sites. The paper calculates the percentage of URLs that are identified as phishing sites by the SSE Phishing Filter. As shown in the Fig. 7, the false positives of all the techniques are low and the results from SSE Phishing Filter are the lowest. Most of the phishing web sites are based on banking sites. Almost all of these banking sites have a proper certificate through which they prove their genuineness. Once a web site has a proper certificate the SSE Phishing Filter will have $\beta = 3$ which is enough to cross the threshold, i.e., $\tau = 3$. This is the reason why the SSE's false positive is less.

### *C. Performance Evaluation of Delay and Resource Consumption*

This section discusses the percentage of the delay of the SSE service for a web request and analyzes how much load SSE system can withstand with a given system setup.

The experiment allows each mobile user using the traffic generator to send 50 SSE requests every 15 seconds. Out of those 50 URLs, some are banking sites where an SSL connection to these banking sites is mandatory. The experiment is performed after clearing the cached URLs at SSE server. The paper emulates 100 to 600 users' behaviors and studies the round-trip time taken with different models. The results obtained are shown in the Fig. 8. The $x$-axis of the graph

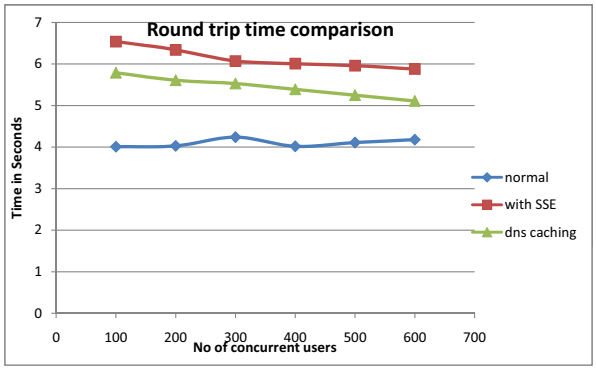


Fig. 8. A comparison between current web browsing loading time and the presented SSE solution.

shows the number of concurrent users in the system and the $y$-axis is the delay of a user from requesting the web site to loading the requested 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 requests for validation take less time as the URLs have already been verified and cached.

The average time saved by 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 the experiment, the average overhead caused by the secure connection to the SSE service is approximately 12% of the entire round-trip time including the processing and communication delay of SSE requests, SSE responses, web server requests, and web server responses.

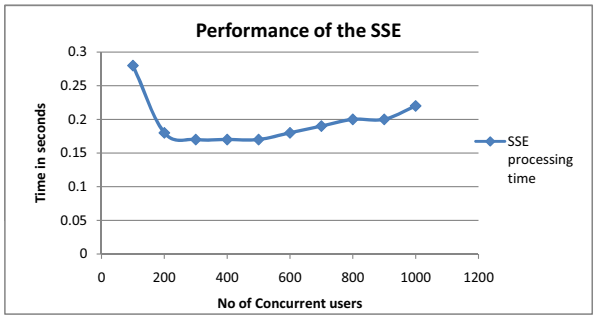


Fig. 9. Processing time taken by the SSE service.

Using the same experiment settings, the paper also calculates the performance of the SSE, particularly, 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 mobile users accessing the SSE service. Fig. 9 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 cache the inspected web sites. The figure shows that the processing time is high at the beginning that confirms the advantage of using caching to handle large amount of traffic, where the processing time drops down due to many requests can be answered latterly by using SSE caching.

The SSE system distributes the workload among 3 servers: The crawler, DNS service, and URL service. The SSE service and Storage service, SSL verifier and Phishing Filter run on two different machines. Based on the current set-up, SSE is able to support about 5,000 concurrent users simultaneously.

## V. Conclusion and Future Work

This paper presents a mobile cloud-based secure web referral service to counter web-based MITM and phishing attacks on the mobile devices. In the central of the system, SSE serves as the foundation of the presented secure web referral framework and involves minimum interventions of humans for security decisions. SSE Phishing Filter produces low false positives and false negatives. In the future, SSE can be extended to counter other web attacks, such as Cross-site Scripting (XSS) attacks.

## Acknowledgement

This work is supported by Office of Naval Research (ONR) Young Investigator Program (YIP) award.

## 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] Dijiang Huang et. al., "Mobile cloud computing," http://mobicloud.asu.edu, 2010.

[8] D. Huang, "MobiCloud: A Secure Mobile Cloud Computing Platform," *E-Letter of Multimedia Communications Technical Committee (MMTC), IEEE Communications Society (invited paper)*, 2011.

[9] D. Huang, Z. Zhou, L. Xu, T. Xing, and Y. Zhong, "Secure data processing framework for mobile cloud computing," in *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, april 2011, pp. 614 –618.

[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] 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.

[15] 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.

[16] 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.

[17] R. Cox, J. Hansen, S. Gribble, and H. Levy, "A safety-oriented platform for web applications," in *Security and Privacy, 2006 IEEE Symposium on*, may 2006, pp. 15 pp. –364.

[18] A. Moshchuk, S. D. Gribble, and H. M. Levy, "Flashproxy: transparently enabling rich web content via remote execution," in *Proceeding of the 6th international conference on Mobile systems, applications, and services*, ser. MobiSys '08. New York, NY, USA: ACM, 2008, pp. 81–93. [Online]. Available: http://doi.acm.org/10.1145/1378600.1378611

[19] Z. Li, Y. Tang, Y. Cao, V. Rastogi, Y. Chen, B. Liu, and C. Sbisa, "Webshield: Enabling various web defense techniques without client side modifications," in *NDSS*, 2011.

[20] A. Moshchuk, T. Bragin, D. Deville, S. D. Gribble, and H. M. Levy, "Spyproxy: execution-based detection of malicious web content," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2007, pp. 3:1–3:16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1362903.1362906

[21] E. Kiciman and B. Livshits, "Ajaxscope: a platform for remotely monitoring the client-side behavior of web 2.0 applications," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 17–30, October 2007. [Online]. Available: http://doi.acm.org/10.1145/1323293.1294264

[22] C. Reis, J. Dunagan, H. J. Wang, O. Dubrovsky, and S. Esmeir, "Browsershield: Vulnerability-driven filtering of dynamic html," *ACM Trans. Web*, vol. 1, September 2007. [Online]. Available: http://doi.acm.org/10.1145/1281480.1281481

[23] Zscaler Inc., "Zscaler cloud services overview," July 2011. [Online]. Available: http://www.zscaler.com/cloudservicesoverview.html

[24] 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.

[25] Xen, "Xen Virtualization Open Source Project." [Online]. Available: http://xen.org

[26] A. Heydon and M. Najork, "Mercator: A scalable, extensible web crawler," *World Wide Web*, vol. 2, pp. 219–229, 1999, 10.1023/A:1019213109274. [Online]. Available: http://dx.doi.org/10.1023/A:1019213109274

[27] T. Mitchell, "Bayesian learning," *Machine learning*, pp. 154–200, 1997.

[28] P. Langley, W. Iba, and K. Thompson, "An analysis of bayesian classifiers," in *Proceedings of the National Conference on Artificial Intelligence*. JOHN WILEY & SONS LTD, 1992, pp. 223–223.

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

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

[31] T. Johnson and D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm," in *Proceedings of the 20th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1994, pp. 439–450.