Memento: A Framework for Hardening Web Applications

Karthick Jayaraman Grzegorz Lewandowski Steve J. Chapin

Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, NY 13210
{kjayaram, grlewand, chapin}@syr.edu

Abstract

We propose a generic framework called Memento for systematically hardening web applications. Memento models a web application’s behavior using a deterministic finite automaton (DFA), where each server-side script is a state, and state transitions are triggered by HTTP requests. We use this DFA to defend against cross-site request forgery (CSRF) and cross-site-scripting (XSS) attacks. The client web browser and the application server each maintain a view of the application state. XSS and CSRF attacks either create an interaction that does not conform to the interaction model or force the web application’s view of the state to diverge from the user’s view. Memento derives behavior models directly from the application, and limits all run-time interactions to the derived interaction models, flagging any state divergence as an attack. We implemented Memento for the Apache web server and evaluated it using 8 open source web applications. We created Memento instances for the 8 web applications and verified Memento’s defense on 14 CSRF and 46 XSS attacks. Memento was able to detect all the attacks with zero false positives and a performance overhead of 28%. Memento does not require any modifications in the web applications it protects.

1. Introduction

An upsurge of web application vulnerabilities has paralleled the trend toward widespread deployment and use of web applications [8]. Cross-site scripting (XSS) and cross-site request forgery (CSRF) were among the top 10 types of web attacks in 2007 [29]. In XSS attacks, an attacker abuses vulnerabilities in a trusted-web site to inject malicious JavaScript programs into the web page delivered to victim users. The JavaScript program is part of the trusted web page and can access user credentials stored in the trusted web-site’s context in the web browser. In a CSRF attack, an attacker injects a malicious HTTP request into a victim user’s active session with a trusted-web site.

The behavior model of a web application provides a better way for understanding the behavior and effects of web-based attacks and also for unifying protection approaches. A web application’s behavior can be represented in the form of a deterministic finite automaton (DFA) [40]. In the simplest form, each server-side script or static HTML page is a state and the application transitions between its states in response to HTTP requests it receives. The requests are typically sent by a web browser which in turn receives and displays the responses produced by the web application. The responses delivered to the user determine what further requests a user may issue. The client web browser and the application server each maintain a view of the application state. In each state, a server-side script determines the contents of the generated response. In effect, there is a response model attached to each state.

In the DFA model, the web application expects only certain requests in each state. A set of access control rules for HTTP requests can be collected from the DFA model for a web application. A CSRF attack not only may violate the access control rules implicit in the DFA, but also causes the web application’s view to diverge from the user’s view of state in the session. An XSS attack injects a JavaScript program that does not conform to the response model in the corresponding state.

We propose a generic framework called Memento for systematically hardening web applications using their behavioral models. Memento consists of the DFA model, the response model, the Request Checker, and the Response Checker. The DFA and response models are extracted from the web application. The Request Checker verifies the validity of each incoming

request against the DFA model and also detects any divergence between the application’s view and the user’s view of state in the session. The Response Checker verifies the conformance of each web page to the response model before delivering it to the browser.

Existing CSRF protection methods detect an attack by checking whether a request originated from a web page delivered to the user. Clickjacking-based CSRF attacks coerce victim users into clicking on URL or forms in a genuine web page and so can evade all existing CSRF protection schemes. Memento’s combination of enforcing the access control rules implicit in the DFA model and detecting state divergence between the application’s view and user’s view can detect all CSRF attacks, including clickjacking-based CSRF attacks, that affect a victim user’s session integrity. Memento’s response model can detect malicious JavaScript injections and is also flexible enough to accommodate certain genuine forms of dynamic JavaScript programs commonly found in web applications.

We implemented the Memento framework for the Apache web server and evaluated it on 8 open source web applications. We created Memento instances for the 8 web applications and evaluated Memento’s defense on 14 CSRF attacks and 46 XSS attacks. Memento detected all the attacks with zero false positives and false negatives. The average performance overhead of using Memento instances for a web application is 28%. Memento does not require any changes in the web application it protects.

The key contributions of the paper can be summarized as follows:

1. A generic framework called Memento for systematically hardening web applications using their behavior models for defending against XSS and CSRF attacks. The key feature of Memento is the approach for understanding the effects of attacks via modeling web applications’ behavior using deterministic finite automata.
2. A prototype implementation of Memento for the Apache web server and an evaluation of Memento on eight open source web applications.

**Organization.** Section 2 describes the threat model. Section 3 describes the Memento framework. Section 4 describes our implementation on Apache web server. Section 5 contains results of our experimental evaluation. We provide a discussion of certain key issues in section 6. Section 7 contains an analysis of related work. We conclude in section 8.

2. Web-Attacker Threat Model

We assume a threat model in which web applications are equally accessible to both attackers and genuine users. The objective of a web attacker is to steal user credentials stored in the web browser in the context of a trusted web site or forge sensitive requests on behalf of victim users to trusted web sites. The attacker may use attacks such as cross-site scripting, cross-site request forgery, and clickjacking. The key feature of these web-based attacks is the ability to maliciously affect a large-user base without any control over their network resources and web browsers. Threat models in which the attacker controls network resources and web browsers are outside the scope of this paper. Memento is a server-side framework and so we do not address client-side attacks such as DOM-based XSS attacks.

2.1. Server-side XSS Attacks

Server-side XSS attacks come in two types: reflected and persistent [10]. Reflected attacks require a vulnerable server-side script that copies user input onto the response without any validation. An attacker may provide a malicious program in the input and the server-side script would copy that onto the response. Persistent attacks require a server-side script that copies data from a database onto the response without any validation. An attacker may store a malicious program in the database of a message forum by posting messages containing JavaScript programs; users who later read these messages will execute the JavaScript programs in their browsers.

2.2. Cross-site Request Forgery

There are two steps in a CSRF attack [41]. First, the attacker steals the session credentials of the user, and then injects a request into the victim user’s session. There are several ways to do this. An attacker may inject a malicious JavaScript program that either reports the user’s session cookie to the attacker or may forge sensitive requests to the web application using the session cookie stored in the web browser. Also, an attacker may identify session identifiers by brute force techniques and inject HTTP requests using those session identifiers.
2.3. Clickjacking-based CSRF

In a clickjacking attack [12], an attacker lures a victim user into visiting an attacker-controlled web site that stealthily places the contents of a trusted web page in an invisible frame behind the victim user’s mouse pointer. When the user tries to click on a URL or button in the attacker’s web site he actually ends up sending a request, crafted by the attacker, to the trusted web site. If the user has an active session with the trusted web site, the web browser automatically attaches the session cookie associated with the trusted web site to the malicious request, thereby compromising the integrity of the user’s session with the trusted web site. For example, if the victim user has an active session with http://www.paypal.com, the attacker may translate all clicks on the attacker’s web site into a sequence of submissions that transfers money from the victim user’s account to the attacker’s account.

3. The Memento Framework

An instance of the Memento framework for a web application consists of the DFA and response models of the web application, the Request Checker, and the Response Checker. The Request Checker verifies the conformance of each user request against the DFA model and also detects any divergence between the web application’s and user’s views of the state. The Response Checker verifies conformance of each response delivered to the user against the response model.

3.1. The DFA Model for a Web Application

The behavior of a web application can be represented in the form of a deterministic finite automaton (DFA). Yuen et al. [40] contains a formal representation for a DFA model to describe the behavior of a web application. A state in the DFA is a server-side script and the values for local parameters of the script (if any). All web pages generated in a state are equivalent. State transitions are invoked by HTTP requests. The hyper links and forms contained in each web page constructed in a state determine the set of valid requests a user may issue in that state. The DFA model for a simple message board application is shown in figure 1. The web application is made of four PHP scripts and a database. All messages in the forum are stored in the database.

1. **index.php**: The index.php script, also called a landing page, constructs the first web page whenever a user visit the website. The web page contains hyper links through which a user can issue HTTP GET requests for scripts such as viewforum.php and login.php.
2. **login.php**: The login.php script is responsible for user login and logout operations and accepts HTTP GET requests to retrieve the login form and logout from an existing session. The login form prepares an HTTP POST request for login.php using the username and password supplied by the user. After a successful login, the login.php script internally redirects the user to view messages in the forum using an HTTP GET request for viewforum.php.

3. **viewforum.php**: The viewforum.php script displays all the messages in the forum. Using the web page delivered by viewforum.php, a user may issue HTTP GET requests to retrieve an edit form to prepare a message for posting, or may logout of an existing session.

4. **posting.php**: The posting.php script is responsible for all message posting functions. An HTTP GET request for posting.php delivers an edit form. The edit form prepares an HTTP POST message for submitting a new message using the user-supplied data. After posting the message, posting.php redirects the user to viewforum.php to view the message he just posted.

There are several implicit access control rules for HTTP requests that can be collected from the DFA of a web application. In the simple message board application, a user is expected to submit a new message only when the user is in the posting.php state. A user is expected to submit a login request only when the user is in the login.php state. A user is not expected to post a message when in states such as viewforum.php, login.php, and index.php. These implicit access control rules are not enforced by current web applications and servers. Cross-site request forgery attacks abuse this flexibility. For instance, the Samy worm was deftly hidden in the attacker’s profile in MySpace.com and issued sensitive requests to manipulate the friends list and heroes list of victim users when they viewed the attacker’s profile page. However, those sensitive requests were not expected to be issued by a user viewing the profile page.

Since the HTTP protocol is by its nature stateless, web applications use a session management scheme to group all user requests in a session. The most recent request in the session that was processed by the web application determines the web application’s view of the application’s state. The most recent response delivered to the user determines the user’s view of the state. Memento uses a web application’s DFA to protect the integrity of user sessions. Memento’s Request Checker keeps track of the application’s state for each session and verifies whether each incoming request is valid in the corresponding session’s current state.

In addition, Memento detects any divergence of the web application’s view from the user’s view of the state in a session. Memento associates a nonce with each successive state in a session at the web server. Whenever a web page is delivered to the user, the Request Checker inserts a JavaScript program in the web page that sends a synchronization message to the web server at a configurable frequency (by default, every 100 milliseconds). The objective of the synchronization message is to report the user’s view of the state to the web server. Web 2.0 applications use similar mechanisms to update the content in the web page delivered to a user. The synchronization message is an HTTP GET request with the nonce specified as a parameter. The JavaScript program stops sending the message whenever the user clicks on a URL or on forms in the web page to initiate a new HTTP request. Any malicious request that was not issued by the user and is accepted in the web server will cause the web application’s view of the state to diverge from the user’s view of the state. Hence, a new nonce will be associated with the session’s state at the web server, while the user-side JavaScript synchronization program will continue to send the old nonce. The Request Checker detects the divergence on the arrival of the next synchronization message and will invalidate the session associated with the request forcing the victim user to reauthenticate.

### 3.2. Response Model

Memento’s response model characterizes the contents of a web page constructed in each state of the DFA and helps in identifying malicious JavaScript programs injected into a response. For each state, the response model consists of a model for the contents of script-inducing constructs, which are constructs that can invoke the JavaScript interpreter in the browser, and a URL-URI whitelist. We collected a list of script-inducing constructs based on observations from current work [7, 36] and the XSS-cheat sheet [11]. The response model is automatically extracted from the web application and the Response Checker enforces the response model on each response delivered to the user.

Our model of script-inducing constructs is motivated by BEEP’s [14] model of whitelisting JavaScript programs. A straightforward whitelisting of JavaScript programs will break some web applications. In a survey of 70 web sites, we found that 56 web applications use JavaScript programs, in which certain portions of the program text are dynamically updated using a server-side script. For instance, figure 2 contains a JavaScript program to display an advertisement. The advertisement to be displayed is specified by initializing a string array and the string literals are dynamically updated using a server-side
Table 1. Invariant Structural Features Of Dynamically Updated JavaScript Programs

<table>
<thead>
<tr>
<th>Structural Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function definitions in the body.</td>
</tr>
<tr>
<td><code>eval()</code> function invocations.</td>
</tr>
<tr>
<td>Accesses to <code>document.cookie</code>.</td>
</tr>
<tr>
<td>Accesses to <code>document.location</code>.</td>
</tr>
<tr>
<td>Accesses to <code>win/.location</code>.</td>
</tr>
<tr>
<td><code>XMLHttpRequest()</code> objects.</td>
</tr>
</tbody>
</table>

These JavaScript programs that are dynamically updated by a server-side script are a form of software clones whose program text varies in a predictable manner. Also, several structural features did not vary for the dynamically updated JavaScript programs we collected from the 70 web sites (Table 1). Koschke [21] contains a review of several methods for detecting general software clones that arise from actions such as copy & paste and modification of programs by developers. The predictable variation in program text and the structural similarity of these JavaScript programs make them a simpler form of clones that are easy to distinguish from attack scripts.

We use a JavaScript similarity algorithm that uses a combination of text similarity and structural heuristics to measure the similarity between two JavaScript programs. Algorithm 1 describes our JavaScript similarity algorithm. First, if the lengths of two JavaScript programs vary beyond a threshold we report them as dissimilar programs. Second, we check whether the two programs match in the structural heuristics. Finally, if the text similarity between the two JavaScript programs is greater than a threshold, we report the two JavaScript programs as being similar. We used three standard text similarity measures, F-Measure, Cosine, and Dice [30], from the information retrieval (IR) literature and all of them gave identical results. Two similar JavaScript programs may vary in their string and numeric literals. To disregard the variation from the literals, we extract a stoplist consisting of string and numeric literals from the JavaScript programs being compared. The text-similarity algorithms ignore the literals from the stop list. Ignoring the string literals may present a problem in the case of JavaScript clones that use `eval()`, a function that executes JavaScript program contained in a string constant. We can visualize contrived examples where an attacker can inject malicious code in the string constants used by `eval()`. In the set of JavaScript clones we collected from 70 web sites, none of them used `eval()`. Memento can be configured to selectively use straight string matching for JavaScript programs that use `eval()`, when verifying conformance to the response model.

Certain script-inducing constructs (HTML attributes such as `src`, `href`, and `url`) can be whitelisted using plain string comparison. These script-inducing constructs contain URL or URI and may not vary dynamically for some web applications. Memento can be configured to use plain string comparison for such constructs and verifies their contents against an URL-URI whitelist associated with the state. Additionally, Memento verifies that all URL in string literals of JavaScript programs are also from the whitelist.

### 3.3. Detecting XSS Attacks

The essence of an XSS attack is the addition of a new script-inducing construct or the modification of an existing one. The Response checker ensures that the contents of each script-inducing construct matches those in the database either by plain
Algorithm 1 JavaScript similarity algorithm

1: `procedure JAVASCRIPTSIMILARITY(X,Y)`
2:   if comparelengths(X,Y) <> 1 then
3:     return 0
4:   end if
5:   if comparestructure(X,Y) <> 1 then
6:     return 0
7:   end if
8:   stoplist ⇐ getliterals(X,Y)
9:   if textsimilarity(X,Y,stoplist) > threshold then
10:     return 1
11:   end if
12:   return 0
13: `end procedure`

Figure 2. A JavaScript Program Used to Display Advertisements

```javascript
var images = new Array()
var links = new Array()

images[0] = '/img/backtocampus08.jpg'
links[0] = 'http://xyz.com/backtocampus/

images[24] = '/img/except_watch.jpg'

var j = 0
var p = images.length - 1;
var index = Math.round(Math.random()*(p));

function showPromo()
{
  document.write('<a href="' + links + '">
<img src="' + images + '/></a>');
}
```
string matching or the JavaScript similarity algorithm. Memento’s notion of JavaScript similarity requires an exact match of chosen structural heuristics and a high text similarity. So, an attacker can succeed in injecting a JavaScript program only if the injected program has high similarity with a genuine JavaScript program. From our observations, such a program cannot do anything significantly different from a genuine JavaScript program in the web page.

3.4. Detecting CSRF Attacks

In the absence of XSS opportunities, the attacker may either try to discover the session identifiers by brute force or try a clickjacking-based CSRF attack. In either case, the attacker does not know the web application’s view of the session’s state. So, the attacker is limited to making some probabilistic guesses of the state. In addition, the attacker needs to direct the application to a specific state to forge a sensitive CSRF request. For instance, in the message board application described in figure 1, an attacker needs to direct the application’s state to posting.php to be able to post a message. Otherwise, the Request Checker would drop the request. Any successful effort by the attacker to direct the application’s state to diverge from user’s view of the state will be detected by Memento when the next synchronization message arrives from the response page delivered to the user. Following the detection, Memento will invalidate the current session and request the user to reauthenticate.

4. Implementation

We implemented Memento for the Apache web server using two mod_perl modules, Request Checker and Response Checker, to extend Apache to validate each request and response. Also, we implemented a model extractor to extract the DFA and response models from web applications. The Request Checker and the Response Checkers are configured using the DFA and response model for each web application hosted in the web server. Memento does not require any changes in the web applications it protects.

4.1. Model Extraction

We used a web spider for building the DFA model of web applications [26, 40]. Beginning from a starting page, the web spider collects URL from tags such as `<a>` and `<form>` and visits all web pages a user may potentially visit using the web application. For each web page, the name of the server-side script and the parameter values (if any) describe the corresponding state in the DFA that produced the web page. The URL extracted from the web page describe all the valid GET and POST requests a user may issue in the corresponding state. At the end of the traversal, the web spider writes the DFA model of the web application in a text file. The Request Checker is configured using the DFA described in the text file.

We implemented a mod_perl output filter to extract the model of script-inducing constructs and the URL-URI whitelist for each state of a web-application’s DFA in a training phase. During the training phase, we generate several web pages for each state in the DFA by executing the corresponding server-side script using non-malicious inputs. We modified an open-source HTML parser [1] to extract all the script-inducing constructs. Memento is configured with the list of script-inducing constructs that will use the JavaScript algorithm and their contents are stored in the model of script-inducing constructs. The contents of other script-inducing constructs are stored in a URL-URI whitelist. The tag and attribute for each of them is also stored to facilitate precise checking by the Response Checker. The structural heuristics are extracted by comparing similar JavaScript programs produced in a state during the training phase and can be specified either globally or locally for the JavaScript programs in each state of the DFA. The response model is stored in the database.

A false positive in the model produced from the automatic extraction process will translate into a loss of functionality when using the application and can be fixed by updating the model. In the future, we plan to extend the model extractor to use dynamic test-generation techniques [3, 19]. The DFA and the response model may also be produced as a byproduct of the software engineering process in the early stages of design.

4.2. Request Checker

The Request Checker is a mod_perl module that keeps track of the state of each user in a session and validates user requests before forwarding them to the web application. The web applications that we used in our experiments use a cookie-based session management scheme, in which the session identifier is stored in a cookie in the web browser. All HTTP requests carry the cookies associated with the web site in the cookie header. Whenever a new session is created, the web application creates
a session cookie in the user’s browser using the set-cookie header. The Request Checker intercepts the response headers and stores each new session identifier and the corresponding state in a database. The name of the session cookie is specified in a configuration file. Whenever an HTTP request is received, the request checker uses the session cookie to retrieve the current state and verifies whether the request is allowed in the current state. Valid requests are forwarded to the web application and invalid requests are dropped. In addition, the request checker also issues an internal redirect for an HTTP request to terminate the session associated with invalid requests. Only requests for landing pages such as index.php are exempt from the validation process requiring a session cookie. Such requests are allowed because the web application creates a new session cookie in the user’s browser using the set-cookie header in the response.

Each time a response is delivered to the client, the Request Checker associates a nonce with the corresponding state in the web application. Also, a JavaScript program is inserted into the web page to periodically send a synchronization message to the web server. The synchronization message is an HTTP GET request for a file named synch.php with the nonce associated with the state as a parameter. The JavaScript programs stops sending the synchronization message before the user issues a new HTTP request to facilitate the transition to the new state. The synchronization message is not forwarded to the web application. If the nonce specified in the message is associated with the current state of the user, the synchronization message is simply dropped. Otherwise, the request checker takes the additional step of issuing an internal redirect for an HTTP request to terminate the session associated with the incorrect synchronization message.

4.3. Response Checker

The Response Checker verifies the contents of each script-inducing construct present in a response against the response model either by plain string matching or the JavaScript similarity algorithm before delivering it to the user. Non-conforming constructs discovered by the checker are removed from the response. We used the Text-Similarity [30] package for measuring text similarity and modified JavaScript-Squish [27] package to extract the literals from JavaScript programs. The structural features are extracted by searching through the body of the JavaScript programs.

5. Experimental Evaluation

We evaluated Memento’s defense on CSRF and XSS attacks using eight open source web applications: phpBB, OsCommerce, Web Calendar, SchoolMate, WebChess, EVE, FaqForge, and gecchblite. We created CSRF attack vectors for phpBB, OsCommerce, and Web Calendar and used the XSS attacks detected by a tool called Ardilla [19] for the other web applications. We measured the false positives by manually operating the web application. Table 2 contains the results of our evaluation on XSS and CSRF attacks. Memento detected both XSS and CSRF attacks with 0 false positives and 0 false negatives. A false negative in the request model will still be detected because it causes a state divergence between the web application and the user’s view of the state.

We measured the performance overhead of using Memento instances using the Apache JMeter. We measured the response times of the web applications both with and without Memento instances in several test settings. Web applications protected with Memento incurred an average performance overhead of 28%. Currently, our prototype Request Checker and Response Checker are implemented in Perl and so a key component of the cost is in interpreting the modules at run-time. We could reduce the overhead further using compiled modules.

We evaluated the accuracy of the JavaScript similarity algorithm on a testing set of similar and dissimilar JavaScript programs. We collected 208 similar JavaScript programs, which are similar based on the criteria we described in section 3.2, from 70 web sites to form the similar set. We created a dissimilar set of 5754 JavaScript programs using 822 JavaScript programs extracted from the same 70 web sites and 8 JavaScript worms (7 JavaScript worms collected from XSSing [38] and a generic 2 line cookie stealing worm). We created 8 dissimilar JavaScript programs for each of the 822 JavaScript programs by appending each of the 8 worms to the JavaScript program.

Recall that the JavaScript similarity algorithm uses both structural heuristics and one of three configurable text similarity measures (F-measure, Cosine, and Dice). We evaluated the JavaScript similarity algorithm using each of the three measures at two thresholds, 0.90 and 0.95, both in the presence and absence of structural heuristics. Table 3 contains the results of our evaluation. There are 7 false positives at a threshold of 0.95 for all the three similarity measures because the genuine JavaScript updations have similarity ranging between 0.90 to 1.0. The structural heuristics are effective in reducing the false negatives. A false negative in the JavaScript similarity will arise if an attacker is able to inject a JavaScript program into the web page that is similar to one of the JavaScript programs in the state. Such a program cannot do anything significantly
Table 2. Evaluating Memento on XSS and CSRF Attacks

<table>
<thead>
<tr>
<th>Web Application</th>
<th>Attack Type</th>
<th>Attacks</th>
<th>Attacks Detected</th>
<th>False Positives</th>
<th>False Negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>phpBB</td>
<td>CSRF</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WebCalendar</td>
<td>CSRF</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OsCommerce</td>
<td>CSRF</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Schoolmate</td>
<td>Reflected XSS</td>
<td>14</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Schoolmate</td>
<td>Persistent XSS</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Webchess</td>
<td>Reflected XSS</td>
<td>13</td>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Webchess</td>
<td>Persistent XSS</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Faqforge</td>
<td>Reflected XSS</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Faqforge</td>
<td>Persistent XSS</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Geccblite</td>
<td>Persistent XSS</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3. Evaluation of the JavaScript Similarity Algorithm

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Errors</th>
<th>F-Measure</th>
<th>Cosine</th>
<th>Dice</th>
<th>F-Measure</th>
<th>Cosine</th>
<th>Dice</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>False Positives</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>False Negatives</td>
<td>111</td>
<td>115</td>
<td>111</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.95</td>
<td>False Positives</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>False Negatives</td>
<td>85</td>
<td>85</td>
<td>85</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

We conducted a whitelisting experiment on the index pages of seventy web sites (shopping and news web sites) to verify usability of the JavaScript similarity algorithm. We extracted the contents of all script-inducing constructs from the index pages for a few days and stored them in a database. Later, we compared the contents of script-inducing constructs in the web pages fetched from the web site against those in the database using both plain string comparison and the JavaScript similarity technique. The script-inducing constructs in the web site can be whitelisted if all them in the web page match with those in the database using either plain string comparison or the JavaScript similarity algorithm. Table 4 contains the results of our white-listing experiment. Fourteen web sites could be whitelisted using plain string matching and sixty four web sites could be whitelisted using the JavaScript similarity technique.

6. Discussion

We discuss a few scenarios where Memento might interfere with a web-application’s functionality and also the possibility of deploying Memento for mashups.

Table 4. Results of White Listing Experiment Using JavaScript Similarity

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Total number of web sites used in the survey</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of web sites that could be white listed using plain string comparison</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Number of web sites that could be white listed using JavaScript similarity</td>
<td>64</td>
</tr>
</tbody>
</table>
6.1. Back Button

Memento expects that the browser’s view of the application’s state corresponds to the actual state of the application. Therefore, if the browser changes its view of the application without notifying the application, problems might occur. For example, such a scenario can be triggered when the user navigates back to previously visited pages using the browser’s Back button. If the browser renders the visited pages without the application’s knowledge, the browser’s view of the application will be out-of-sync with what Memento expects. As a result, Memento might interpret subsequent requests as violating its application behavior model and invalidate the user’s session. Disabling HTTP caching is a way to prevent browsers from rerendering previously visited pages without communicating with the application. In fact, web applications frequently resort to cache disabling in order to stop the browser from displaying stale content. Memento can use the same technique to ensure synchronization between the browser and the application.

6.2. Multi-page access

Multi-page access happens when a user opens the same web application in several browser windows or instances. Depending on the session tracking mechanism employed by the web application, the application might or might not be able to assign separate sessions to separate browsers. If the application is able to track separate browser instances as separate sessions, Memento works as intended. However, if the web application is unable to distinguish between browser instances, all instances will participate in a single session. The session compounded of several actual sessions will likely not conform to the behavior that DFA model expects. Consequently, Memento will invalidate the session.

6.3. Mashups

A mashup is a web application that creates a new, distinct web service by combining content provided by several web applications. As opposed to a standard web application, an attack against a mashup might be executed either against the mashup application itself, or against any of the applications that provide integrated content. However, as long as a given mashup aggregates a fixed set of external applications, a Memento-enabled system can prevent attacks executed against it. The cost of defending a mashup application is likely to be larger than for a standard web application as the finite state machine representing the mashup would have to encompass the mashup itself and at least a subset of aggregated applications’ states. An alternative Memento mashup deployment scenario would be to deploy several Memento instances hierarchically to defend mashup itself and its components separately. The main advantage of such approach would be reduced complexity of individual Memento instances.

7. Related Work

Current work has proposed several anomaly detection methods that build models of normal application behavior using probabilistic models of low-level features such as contents of HTTP requests [13, 22] and internal state variables [9] of applications. Valeur et al. [33] describes an approach for managing false positives. In contrast, Memento directly uses the behavior model of web applications which is more abstract and deterministic. Prior work has used the behavior model of web applications for generating test cases and model checking against specifications [2, 26, 40]. However, Memento uses the behavior model of web applications for building run-time mitigation techniques for defending against CSRF and XSS attacks.

Existing approaches for preventing CSRF attacks can be classified into secret token validation techniques [15, 16], HTTP referrer header validation [18], and proposals for new headers [6]. Current CSRF protection schemes detect an attack by checking whether a request originated from a web page delivered to the user. Clickjacking-based CSRF attacks coerce victim users to click on a URL or forms in a genuine web page and so can evade all existing CSRF protection schemes. Memento not only checks whether a request belongs to web page delivered to the user but also ensures that the web application’s state does not diverge from the user’s view of the state. Clickjacking attacks will cause the web application’s state to diverge from the user’s view and can be detected by Memento. Barth et al. [6] describes the login CSRF attack, in which an attacker forges a login request on behalf of the victim user to a trusted web site using the attacker’s username and password, and proposes strict referrer header validation over HTTPS as a solution. Login CSRF does not affect the integrity of an existing user session and so Memento cannot detect it. Login CSRF attacks can be tried using clickjacking methods and in such cases
referrer header validation methods will not detect the attack. The best solution is to ensure login pages cannot be displayed inside iframes using de-framing JavaScript programs [12] in combination with referrer header validation.

There are several XSS vulnerability identification methods that use static-analysis techniques [17, 35, 36, 37], dynamic-analysis techniques [5], and a combination of both techniques [4]. Also, there are methods for creating SQL injection and XSS attack vectors for a web application [19, 24, 25]. These techniques are meant to help the developer identify potential problems prior to deployment. On the contrary, Memento is a run-time mitigation technique. Client-side XSS mitigation techniques such as Noxes [20] and Vogt et al. [34] depend on user involvement to create rules to permit or deny outgoing HTTP connections and transfer of sensitive information. Memento is a server-side mitigation method that is completely transparent to both the end user and the web application it protects.

There are several server-side taint-tracking methods [28, 31, 39] that work on specific platforms. In contrast, Memento does not make any assumptions about the platform used to build the web applications. BrowserShield [32] is a server-side method to rewrite web pages and embedded scripts into safe-equivalents based on known vulnerabilities, while Memento’s response model characterizes the contents of a genuine web page and detects non-conforming contents in the web page. Madou et al. [23] describes a server-side method that learns invariants in terms of occurrences of certain HTML patterns in each output location in a dynamically constructed response during a training period. Each response is checked for the invariants in the specific locations before delivering it to the user. The approach may be able to detect additions of new script-inducing constructs but cannot detect attacks that modify existing script-inducing constructs.

XSS-Guard [7] compares the JavaScript programs in each dynamically generated web page against those in a shadow page generated using benign inputs. The programs are compared for exact matches or syntactic equivalence that requires an exact match of lexical entities in the corresponding parse trees. So, the method cannot accommodate the genuine JavaScript updates. Also, XSS-Guard cannot detect persistent XSS attacks because the malicious program will also be part of the shadow page in such cases. BEEP [14] is a client-server method, in which a server component adds a whitelist of JavaScript programs for each page, and a browser component verifies each JavaScript belongs to the whitelist before the browser executes it. Memento’s response model is motivated by BEEP. Memento differs from BEEP in extracting the whitelist automatically during a training period and accommodating the genuine JavaScript updates.

8. Conclusion

We presented a generic framework, Memento, for hardening web applications against CSRF and XSS attacks and its prototype implementation for the Apache web server. The Memento framework uses applications’ behavior models to enforce the expected request and response behaviors at run-time. Moreover, the framework monitors the current web application’s state and the user’s view of the state and flags any divergence as an attack. Memento can detect all CSRF attacks, including clickjacking-based CSRF attacks, that affect a victim user’s session integrity. In addition, Memento can detect server-side XSS attacks and also accommodate certain genuine dynamic JavaScript programs we commonly found in web applications. We evaluated Memento using 8 open source web applications. We created Memento instances for the web applications and verified Memento’s defense on 46 XSS and 14 CSRF attacks. Memento detected all the attacks with zero false positives and an average performance overhead of 28%.

References


