Isolating JavaScript in Dynamic Code Environments

Antonis Krithinakis Elias Athanasopoulos Evangelos P. Markatos

Institute of Computer Science,
Foundation for Research and Technology - Hellas
{krithin,elathan,markatos}@ics.forth.gr

Abstract

We analyze the source code of four well-known large web applications, namely WordPress, phpBB, phpMyAdmin and Drupal. We want to quantify the level of language intermixing in modern web applications and, if possible, we want to categorize all coding idioms that involve intermixing of JavaScript with a server-side programming language, like PHP. Our analysis processes more than half of a million of LoCs and identifies about 1,000 scripts. These scripts contain 163 cases, where the source code is mixed in a way that is hard to isolate JavaScript from PHP. We manually investigate all 163 scripts and proceed in a classification scheme of five distinct classes. Our analysis can be beneficial for all applications that apply operations in the client-side part of a web application, various XSS mitigation schemes, as well as code refactoring and optimization tools.

Categories and Subject Descriptors D.2.3 [*Software Engineering*]: Coding Tools and Techniques

General Terms Languages, Security

Keywords JavaScript, Web Security

1. Introduction

Cross-site scripting (XSS) is one of the most well known techniques for compromising web applications. It is considered a very popular exploitation technique nowadays [13, 14]. Many schemes for mitigating XSS attacks are based on isolating all scripts that are trusted [6–8, 15]. All such schemes require that all trusted scripts (e.g., JavaScript) must be isolated from the untrusted ones (i.e. possible code injections). So far, the most prominent techniques for *marking* code are based on tainting, static analysis and flow tracking methods [11, 12, 16]. However, these strategies experience often dramatic computational overheads. On the other hand, modern web applications are based on frameworks that combine multiple server-side and client-side technologies for producing dynamic content. In these applications, isolating one technology from the other is not considered a trivial task. For example, consider a script written in PHP (a server-side language) which dynamically produces JavaScript source code. This is a *code-mixing* case where PHP and JavaScript are intermixed. The JavaScript source code structure is not complete prior the execution of the PHP script.

To the best of our knowledge, there has been no systematic effort for identifying how client-side and server-side languages intermix together in modern web applications and how hard is to isolate the one from the other. In this paper we try to identify the level of intermixing between different programming languages in web applications that enable real-world web sites. More precisely, we analyze the source code of four popular applications, namely phpBB, WordPress, phpMyAdmin and Drupal. Our findings suggest that all these web applications are experiencing mixing of PHP and JavaScript. The way that each framework generates dynamic content varies and it can be considered as a result of a series of coding idioms. However, according to our empirical study, all coding idioms can be classified in five major classes. For each of the five classes we proceed and present alternative coding styles in order to reduce the mixing. In short, we analyze more than half of a million LoCs, identify less than 1,000 scripts which include about 163 scripts where JavaScript and PHP are mixed in a way that is hard to automatically isolate one from the other.

Benefits and applications. Automatically isolating the client-side part, which is most often expressed in JavaScript, from the rest of the code of the web application is vital for security schemes that a) consider all server-generated JavaScript trusted [7, 8], or b) apply operations to all server-generated JavaScript in order to isolate it from code injections [6]. All the analysis of this paper has been carried out during an attempt for applying Instruction Set Randomization [9, 10] to JavaScript. Furthermore, our analysis can be beneficial for all applications that perform operations in the JavaScript source corpus of a web application. For example, consider refactoring, optimization, and code analysis tools.

APLWACA '10 June, Toronto, Canada.

Contribution. a) We perform an extensive analysis in the source code of four real-world web applications and we identify all cases where JavaScript is intermixed with PHP. and b) we classify all cases in five main categories; for each case we provide an alternative strategy to reduce the mixing.

Web App	LoCs	Scripts	Non-mixed	Mixed
WordPress	143,791	187	136	51
phpBB	213,681	539	512	27
phpMyAdmin	178,447	263	183	80
Drupal	44,780	8	3	5

Table 1. Summary of scripts that experience mixing in four real-world web applications.

2. Methodology

In this paper we aim on finding cases in the source of real-world web applications where JavaScript is mixed with a server-side programming language and isolate the JavaScript code. We process the source code of four web applications, which happen to use PHP. We assume that JavaScript and PHP are intermixed only in files and not in a database. A web application may store JavaScript in a database and generate content via SQL queries issued by PHP, in order to produce JavaScript code. This case is beyond the scope of this paper.

In order to isolate JavaScript we use the following methodology. For each web application we attempt to randomize all scripts contained in files included in the application's distribution. We have built a custom tool which has the following functions. The first processing unit takes as input a file and attempts to identify all possible JavaScript source code occurrences. It first removes PHP and HTML comments and then searches for JavaScript scripts. That is, all code inside a `<script>` tag, as well as all code in HTML events such as `onclick`, `onload`, etc. Notice the identified code may contain PHP language elements. Finally, the mixed JavaScript source code is stored for further processing.

The second processing unit is a parser, based on the Mozilla SpiderMonkey JavaScript engine [4]. The original spider monkey parser has been modified to randomize all identifier tokens as they derive from the JavaScript syntax analysis. The randomization process follows ideas introduced by Keromytis et al. [9, 10], implemented specifically for JavaScript. For an example of the randomization process output please refer to Figure 1. For every stored source code from the previous phase, the parser is invoked to produce the randomized output. Apparently there are cases the tool fails to randomize the script and produces a syntax error. This derives from the mixing between JavaScript and PHP. In other words the parser is confused when processing PHP language elements and tries to identify them as JavaScript language elements. Apart from this, a syntax error can be produced in other cases where PHP is not involved. This occurs because the input the parser is given, may not be a complete JavaScript code but a fragment. For example the syntax analysis fails when a return statement exists out of a function block. This mostly occurs in code from HTML events. Every event has a default action. When an event handler is also defined (like `onclick`), the event handler can return a boolean to tell the browser whether or not to take the default action. Another example is when the parser has to deal with escaped string quotes which come from PHP string variable content extractions. In our implementation, such cases are successfully identified without producing any error messages. An example of the parser input is given in Figure 2. Successfully identified cases do not mean that the input is not mixed. In many cases PHP language elements contained in a script pass the syntax checking. This occurs in JavaScript assignment expressions where the right value is a string constant containing PHP code. Whenever the tool fails to randomize a script we record the file and the LoC of the script and then we analyze each case manually. The randomization tool performs a best effort for isolating the JavaScript source. A failure indicates that manual intervention is needed for the isolation of the JavaScript code.

Web Application	Files	With JavaScript	Pass	Fail
WordPress	279	48	28	20
phpBB	542	137	117	20
phpMyAdmin	304	65	28	37
Drupal	143	5	1	4

Table 2. Statistics for files contained in each web application. The third column specifies how many files include scripts. The fifth column indicates how many files include at least one mixed case.

We manage to identify 163 scripts in all four web applications in which the tool fails to randomize them. We further proceed and investigate each of the 163 mixed scripts manually. We manage to create a taxonomy with five different classes. We, finally, assign each script to the corresponding category. In the following section we describe each of the five categories. For the overall statistics of all web applications refer to Table 1 and 2.

2.1 Web Applications

Our analysis includes four popular web applications. We now provide a short description for each one of these.

WordPress. A popular web application for constructing

```
1  <!-- Original Document. -->
2  var s = "Hello World!";
3    document.getElementById("welcome").text = s;
4
5  <!-- Randomized Document. -->
6  var s0x78 = "Hello World!";
7    document0x78.getElementById0x78("welcome")
8                                  .text0x78 = s0x78;
```

Figure 1. A randomization example. All JavaScript identifiers get appended with a random key.

blogs. It is estimated that WordPress is used by over 202 million web sites worldwide [5].

phpBB. A web application for developing interactive forums. It is written in PHP and it supports multiple database management systems and unlimited levels of sub-forums [2].

phpMyAdmin. This web application is intended to handle the administration of MySQL databases over a web front-end. Through the web interface a user can create, modify, or delete databases, tables, etc. phpMyAdmin contains a significant amount of JavaScript intermixed with PHP [3].

Drupal. Drupal [1] is a content management system (CMS) written in PHP. It allows the system administrator to organize the content, automate administrative tasks and manage site visitors.

3. Classification

The analysis of 163 different scripts, where JavaScript is mixed with PHP, produces the following five categories. For each category we provide an example from one of the four web applications that take part in the analysis. The example may be slightly simplified for presentation reasons.

Case 1. *Partial injection of non-mixed JavaScript source using the PHP built-in function* `echo()`. This is the case where non-mixed JavaScript source code is injected in the web document via PHP, by using the built-in function `echo()`. We present a example of a JavaScript snippet, which is injected using multiple calls to `echo()` in Figure 3. In such a case, isolating the JavaScript part is hard, since the parser's input has many extra quotes and `echo()` occurrences.

Case 2. *String concatenations.* In this case the final JavaScript code is generated by the concatenation of string literals and values of PHP variables. All dynamic generated strings are printed using the PHP built-in function `echo()`. The difficulty here occurs because both quotes (single and double) are used as part of a complex string concatenation operation. Isolating the JavaScript part is hard in this case, since both quoting styles are significant for both programming languages, JavaScript and PHP. In Figure 4 we present a string concatenation example.

Case 3. *Partial JavaScript code generation by PHP scripting blocks.* This is the most frequently occurring case of JavaScript and PHP intermixing. In this case PHP scripting blocks are put inside JavaScript. These scripts contain calls of `echo()`. After PHP is invoked, these blocks are evaluated and the final JavaScript source is generated. The parser fails the syntax analysis at the time it consumes the PHP scripting block. An example of this case is in Figure 5.

Case 4. *JavaScript code generation by using frameworks' meta languages.* In this case a framework uses a meta language to build JavaScript dynamically. This case exists only in phpBB. The framework generates dynamic content by transforming static HTML pages containing the meta languages elements. It loads the page, uses patterns to locate all meta language elements and then substitutes them with PHP code as the meta language semantics order. Secondly, the generated PHP code must be evaluated by using PHP function `eval()` to produce the final JavaScript code. In addition, the meta language is supposed to be more expressive than ordinary substitutions. It gives the programmer the opportunity to make easy conditional assignments. In Figure 6 we present an example which is used in order to assign a value in the `lang` and the `index` field. In such a case, isolating the JavaScript code is hard, since the complete source code is not known in advance. The parser fails at the time it consumes the meta language elements.

```
<!-- Original Document. -->
.... onsubmit="return checkPassword(this)"> ....
<!-- Parser Input -->
//Start Of Input
return checkPassword(this)
//End Of Input
```

Figure 2. Parser input example from phpMyAdmin.

```
/* Case 1. */
echo "<script>\n";
echo "document.write(...);"
echo "</script>\n";

/* Parser Input */
//Start Of Input
\n";
echo "document.write(...);"
echo "
//End Of Input
```

Figure 3. Example for Case 1: Partial injection of non-mixed JavaScript source using the PHP built-in function `echo()`.

```
/* Case 2. */
$actions['quickedit'] =
 'onclick=
"commentReply.open(\'' .$post->ID. '\',\'edit\');"';

/* Parser Input */
//Start Of Input
commentReply.open(\'' .$post->ID. '\',\'edit\');
//End Of Input
```

Figure 4. Example for Case 2: String concatenation.

```
/* Case 3. */
tinyMCEPreInit = {
 ...
 mceInit:{<?php echo $mce_options; ?>},
 ...
};
```

Figure 5. Example for Case 3: Partial JavaScript code generation by PHP scripting blocks.

Application	Scripts	C1	C2	C3	C4	C5
WordPress	51	3	12	36	0	0
phpBB	27	1	0	0	26	0
phpMyAdmin	80	0	43	34	0	3
Drupal	5	0	2	3	0	0

Table 3. Categorization of all mixed scripts in the four web applications.

```
1  /* Case 4.  */
2  var RecaptchaOptions = {
3   lang : {L_RECAPTCHA_LANG},
4   index :
5    <!-- IF $CAPTCHA -->{$CAPTCHA}
6    <!-- ELSE -->10<!-- ENDIF -->
7  };
```

Figure 6. Example for Case 4: JavaScript code generation by using frameworks' meta languages.

```
1  /* Case 5.  */
2  onsubmit=
3    "return
4    (emptyFormElements(this , 'table')
5    &amp;&amp;amp; checkFormElementInRange(
6    this , 'num_fields', ...);"
```

Figure 7. Example for Case 5: Markup injections.

```
1  /* Case 1 Alternative Idiom.  */
2  ... ?>
3  <script type='text/javascript'>
4  document.write(...);
5  </script>
6  <?php ...
```

Figure 8. Alternative approach for case 1.

```
1  /* Case 2 Alternative Idiom.  */
2  $content = '\'' . $post . '\',\'edit\'';
3  $actions['quickedit'] =
4   'onclick=
5    "commentReply.open('.$content.');"';
```

Figure 9. Alternative approach for case 2.

Case 5. *Markup injections.* Finally, the last case includes scripts where HTML special characters (like `&`) are injected in JavaScript expressions. These HTML characters are translated from a PHP filter before the script reaches the web browser. Before the translation the JavaScript expression is invalid. For example, the most frequently occurring case is when the sequence of HTML entities `&&` is translated to `&&`, which is the logical `AND` in a JavaScript expression. For an existing example of such a case, please refer to Figure 7. We are not aware of the goal of the web programmers that use this coding tactic.

Results. We now present the results from the classification of all 163 scripts of the four web applications. In Table 3 we list each application with all identified mixed scripts, categorized in each one of the five categories we present above. Notice that for all four application most of the mixed scripts fall in the third case. The meta-language case, Case 4, occurs only in phpBB. In addition, only three scripts fall in Case 5 and occurs only in phpMyAdmin. Case 1 cases are limited as well. Our observation suggests that there are only a few coding idioms, in order to mix PHP and JavaScript, that are used in real-world web applications. The dominant idioms is string concatenation, partial injection using PHP scripting blocks and custom meta-language technologies.

4. Mixing Reduction

In this section we try to reduce the mixing between JavaScript and PHP code and successfully isolate JavaScript in failed

```
1  /* Case 3 Alternative Idiom.  */
2  tinyMCEPreInit = {
3   ...
4   mceInit: <?php $mce_options_s = "{".$mce_options."}";
5            echo $mce_options_s; ?> ,
6   ...
7  };
```

Figure 10. Alternative approach for case 3.

```
1  /* Case 4 Alternative Idiom.  */
2  <!-- IF $CAPTCHA -->
3  var RecaptchaOptions = {
4   lang: {L_RECAPTCHA_LANG}, index: {CAPTCHA}
5  };
6  <!-- ELSE -->
7  var RecaptchaOptions = {
8   lang: {L_RECAPTCHA_LANG}, index: 10
9  };
10 <!-- ENDIF -->
```

Figure 11. Alternative approach for case 4.

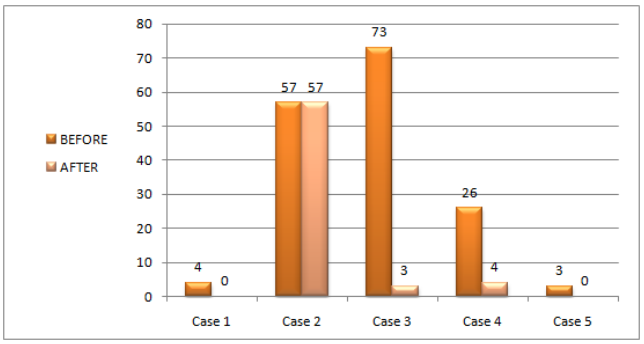

Figure 12. Script occurrences for each category for all four web applications before and after our alterations. Notice, that most scripts fall in Case 3.

cases of Section 2. This is possible by altering the mixed code or extending the parser to support some individual cases.

Case 1. In this case it is easy to give an alternative coding idiom to avoid the parser failing. Instead of using PHP `echo()` calls to generate the code in the final document, the code can be injected as it is. The programmer can finalize the PHP scripting block , inject the JavaScript code and then start the PHP scripting block again. The alternative code is depicted in Figure 8.

Case 2. This is the most difficult case to address. Consider that in our implementation the snippet should be modified to be syntactically correct by both the PHP interpreter and our JavaScript parser. Mix reduction in this case can be achieved by making less confusing use of quotes and by using as less as possible concatenation parts. The alternative code is depicted in Figure 9.

Case 3. In this case we try to address failed cases by both rewriting the code and extending the parser. The parser identifies the start and the end of a PHP scripting block in the syntax analysis, consumes it and then handles it as a valid JavaScript identifier. However, there are cases where an identifier is not expected due to JavaScript semantics and

the analysis fails again. These cases can be addressed by code rewriting. The alternative code for this case is depicted in Figure 10. Now all the PHP scripting blocks are identified as JavaScript identifiers.

Case 4. During the syntax analysis we use the same patterns phpBB uses to identify the meta language elements. After a successful identification the parser handles them as JavaScript identifiers, as above. In cases where the meta language elements are straight substituted this approach works. In cases where the meta language is quite complicated we propose an alternative writing to make it more simple. This strategy makes code maintenance more difficult but helps the parser to successfully isolate the JavaScript. The alternative code is depicted in Figure 11, where the meta language elements, {L_RECAPTCHA_LANG} and {CAPTCHA}, are treated as JavaScript identifiers.

Case 5. In this last case the parser is extended to recognize HTML entities (like &) and simply consume and ignore them in the syntax analysis.

Results. In cases where the failed instances are few, like in Case 1 and in Case 5, we rewrite the code as proposed. In addition we extend the parser semantics to successfully handle Case 3 and Case 4. We then rerun the extended parser with the methodology described in Section 2. We further investigate the current failed cases. In Figure 12 we plot the overall script occurrences we recorded for each of the five different cases before and after our interference. In Case 1 and Case 5 all failed cases are eliminated. In Case 3 and Case 4, the parser extensions managed to strongly reduce the failing rates. In Case 2 the failed cases remain because neither code rewriting nor extensions are applied in the second run.

5. Conclusion

In this paper we present a systematic analysis in the source code of four large web applications. The analysis aims on identifying possible cases where JavaScript is intermixed with PHP. During our analysis we process more than half of a million of LoCs. We identify about 1,000 scripts, which contain 163 scripts, where JavaScript is intermixed with PHP. We manually investigate all 163 scripts and create a classification scheme of five distinct classes. Moreover, we try to address failed cases and reduce the mix by proposing alternative code idioms or extensions to the processing tool. We further analyze the results. Our analysis can be beneficial for all applications that apply operations in the client-side part of a web application and various XSS mitigation schemes.

Acknowledgements. Elias Athanasopoulos, Antonis Krithinakis and Evangelos P. Markatos are also with the University of Crete. Elias Athanasopoulos is also funded by the Microsoft Research PhD Scholarship project, which is provided by Microsoft Research Cambridge.

References

[1] Drupal: An open source content management platform. http://drupal.org/.

[2] phpBB: One of the most widely used Open Source forum solution. http://www.phpbb.com/.

[3] phpMyAdmin: Application that handles the administration of MySQL over the World Wide Web. http://www.phpmyadmin.net/.

[4] SpiderMonkey (JavaScript-C) Engine. http://www.mozilla.org/js/spidermonkey/.

[5] WordPress Usage: 202 Million Worldwide 62.8 Million US. http://andrewapeterson.com/2009/09/wordpress-usage-202-million-worldwide-62-8-million-us/.

[6] E. Athanasopoulos, V. Pappas, A. Krithinakis, S. Ligouras, and E. P. Markatos. xJS: Practical XSS Prevention for Web Application Development. In *Proceedings of the 1st USENIX WebApps Conference*, Boston, US, June 2010.

[7] M. V. Gundy and H. Chen. Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS)*.

[8] T. Jim, N. Swamy, and M. Hicks. Defeating Script Injection Attacks with Browser-Enforced Embedded Policies. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*.

[9] G. Kc, A. Keromytis, and V. Prevelakis. Countering Code-Injection Attacks with Instruction-Set Randomization. In *Proceedings of the 10th ACM conference on Computer and Communications Security*.

[10] A. D. Keromytis. Randomized Instruction Sets and Runtime Environments Past Research and Future Directions. *IEEE Security and Privacy*, 2009.

[11] L. C. Lam and T.-c. Chiueh. A General Dynamic Information Flow Tracking Framework for Security Applications. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference*.

[12] J. Newsome and D. Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *Proceeding of the 13th Annual Network and Distributed System Security Symposium (NDSS)*, 2005.

[13] SANS Insitute. The Top Cyber Security Risks. September 2009. http://www.sans.org/top-cyber-security-risks/.

[14] Symantec Corp. April 2008. 1-3. Retrieved on 2008-05-11. Symantec Internet Security Threat Report: Trends for July-December 2007 (Executive Summary).

[15] M. Ter Louw, P. Bisht, and V. Venkatakrishnan. Analysis of hypertext isolation techniques for XSS prevention. In *Web 2.0 Security and Privacy 2008*, May 2008.

[16] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In *Proceeding of the 14th Annual Network and Distributed System Security Symposium (NDSS)*, 2007.