Abstract. A wide range of websites in World Wide Web are deemed invalid as they do not adhere to the HTML standards defined under W3C. Websites which are not well-formed may cause performance degradation for some applications and poses compatibility issues with some browsers. Furthermore, it is harder to render a malformed webpage using state of the art web browsers. To validate a HTML page, a wide range of HTML document validation tools have been developed to validate HTML pages but these tools is not effective in a realm of dynamic web applications such as PHP due to its diversified and complex nature. These tools typically analyze PHP in a static manner which failed to provide meaningful accurate validation on real world applications. In this paper, we present a novel tool for validating dynamic PHP web application in a robust manner which takes into consideration the syntax as well as operation of PHP application.

Keywords: Web Validation, Server Side Scripts, Dynamic Web Generation

1 Introduction

Nowadays, web applications have become a crucial element in the society. A website must be interactive, dynamic, as well as capable of performing on-line transaction in order for it to strive in the competitive environment today. PHP was designed to cater for such requirement and has been an Internet programming language of choice for such a long time. As a matter of fact, according to TIOBE Programming Community Index, it is the 4th most popular programming language in the world. [1]

In order to guarantee the growth of high-quality and robust web applications, a collection of standards have been laid out by the World Wide Consortium (W3C). One of the criterions for having a superior web application is the markup validity, which includes the adherence to official DTD for HTML 4.01 or XHTML 1.0 of a web document in HTML as well as other client-side markup web languages with respect to their respective grammar, vocabulary and syntactical rules.

Despite the fact that state-of-the-art web browsers are capable of handling the parsing and rendering of malformed HTML web documents, a number of bugs or defects in web applications cannot be identified easily every time because of the dynamic nature of PHP web applications. One of the ways that could facilitate the process of identifying bugs and debugging during the development state is to check for HTML validation errors. In a survey conducted by W3C, a large number of web professionals mentioned that HTML validation errors are the first thing they check whenever then encounter scripting or web styling bug. This is because most of the time, errors found are not handled in a similar fashion by different software on different platforms, hence resulting difficulties in applying or styling the layout in a consistent manner.

Web applications that adhere to official DTD for HTML 4.01 or XHTML 1.0 set by W3C are easier to maintain and grow regardless whether maintenance and growth are performed by the same developers because the standards such as HTML and CSS are a form of “coding-style” that has been agreed upon internationally.

To appreciate the significance of markup validity for Web documents, a number of organizations as well as individuals have developed HTML validators to check for the conformance of official DTD for HTML and XHTML. A handful of them also provide automatic support for fixing markup errors to convert a malformed HTML document into a well-formed one that adhere to HTML grammar and syntax.

Unfortunately, these tools do not address the new challenges in web development because it only functions well on static HTML pages. First challenge in current web development is that the client-side HTML documents are often generated at run time from the server-side codes. These server-side codes are written in different languages, for instance PHP. The HTML code is embedded within the string literals or the values of variables in the PHP server-side code. Furthermore, these HTML codes are often scattered in multiple locations across various server pages. This is especially true when the web application follows the Model-View-Controller (MVC) architecture. Most importantly, since different client pages are produced at run time by server-side codes, it is challenging for the web developer to manually map the buggy segment of code(s) back to the server-side code for any validation error found and reported.

Dynamic web applications such as PHP produce interactive web pages where structures, layout and content are made dynamically during runtime depending on user response and behavior as well as contextual condition where a system is capable of demonstrating. As a matter of fact, professionally developed and tested PHP web applications encompass multiple faults which can only be discovered through unprecedented application behavior. Due to this reason, in an
ideal environment, validation is needed for all execution path web documents that are generated in an ad-hoc manner in a PHP application.

2 Related Work

Current techniques in PHP applications employ static analysis and target security vulnerabilities such as SQL injection, Cross-Site Request Forgery (CRSF), Cross-Site Scripting (XSS) and etc. Artiz et al. proposed an approach for discovering HTML faults in PHP web application that depends on integration of concrete as well as symbolic execution. [8] Unfortunately, their tool is not available for public access. Minamide developed a static analysis for discovering HTML malformedness faults in PHP applications. Unfortunately, this tool yields a higher overhead for checking discrepancy between regular expression and context-free languages with an expensive algorithm. Firstly, Minamide’s analysis tool estimate the string output of program against a context-free grammar. Then it locates unclosed tags by intersecting this grammar with the regular expression of matched pairs of delimiters (open/closed tags). [9]

Wassermann et al introduces a concolic testing tool for PHP which is hinge on Minamide’s PHP static analyzer. It utilizes a finite state transducers and a distinctively designed constraint solver. Wassermann suggest an automated analysis technique which targets security vulnerabilities as a result from injecting malicious strings into SQL commands.[7] Xie and Aiken construct an analogous analysis but traded off some precision for scalability thought the usage of block and function summaries. A genuine 3-tier architecture to acquire information at decreasing levels of granularity at intra-block, intraprocedural and interprocedural level is incorporated in their analysis. Such architecture makes it possible to manage and deal with distinctive dynamic features to scripting languages such as dynamic typing and code inclusion. [10] Javonovicet. al. contemplate the issue of vulnerable web applications in terms of the analysis of static source code. They implemented Pixy for identifying cross-site scripting vulnerabilities in PHP scripts.

Since most of the work that has been done previously concentrated on static analysis as well as security vulnerabilities of PHP applications, our tool distinct from the others in several aspects. First, our tool is able to validate HTML Tags at PHP scripts level. Validating PHP scripts is a non trivial task as it is highly dynamic and therefore it is harder to predict the flow of execution in PHP scripts to generate HTML codes during run time. Secondly, due to the enormous and wide range of libraries and tools provided by PHP language, it is very hard for users to formulate validation techniques to validate dynamic PHP scripts, as the possibilities for generating HTML codes are far greater than HTML codes. Finally, since validation is carried out at PHP scripts level, one may need to consider for PHP plugins and their relevant supporting files, unlike HTML codes, which is plainly HTML Tags and front end scripts such as Javascripts, and CSS. For all these reasons, we are able to develop a novel PHP validation tool which could robustly validate PHP scripts, taking into considerations all the factors mentioned previously. To the best of our knowledge, our tool is the first tool to validate PHP scripts. Our PHP validation tool is certainly useful for the design of future HTML Validation Tool.

3 PHP Web Applications

Overview

This section gives an overview of PHP scripting language. PHP is an open source server-side scripting language that is specifically tailored for web development which can also be used as a general-purpose programming language. Unlike Java, PHP can be used as a general-purpose programming language because it’s not Object-Oriented (OO). As compared to its competitor ASP server side script engine and languages, PHP is installed on more than 20 million websites and 1 million web servers. [5] User interaction can only occur in a back-and-forth fashion (server replies with pages generated at runtime based on requests sent by users) since PHP code is triggered when a page is requested from the server.

One of the techniques used by PHP in devising user interactivity is through HTML forms. Users are able to send data to the server in a convenient manner through web forms, which makes the entire web application much more interactive. Depending on the method use in the form, which in turn influence the request method used by the browser, these data can be retrieved with the $_GET and $_POST super global arrays.

To illustrate how data is passed, consider the following form:

```
<form action="process.php" method="post">
  <input type="text" name="firstName">
  <input type="text" name="lastName">
</form>
```

As shown above, the value can be referenced as $_POST[‘firstName’] and $_POST[‘lastName’]. However, if ‘get’ is being used as the method, data is passed as the query string of the URL instead of the content of the request, where the value can only be reference with $_GET[‘firstName’] and $_GET[‘lastName’]. Apart from $_POST and $_GET, PHP
provides other arrays such as $REQUEST, $COOKIE, $SERVER, $SESSION and etc which composing all useful information about an http request.

**Strength and Weakness**

A number of leading benefits that PHP has to offer are as follows. Firstly, since PHP is a server-side scripting language, any PHP code embedded in a file is executed by a PHP interpreter at runtime. With such feature, it maintains the actual source code of the web application in a secure manner as end user does not see the source code as they do with client-side scripting such as JavaScript. Secondly, it can be deployed on most web servers (Apache, IIS, Zend, Xampp, Wamp and etc) across a wide range of operating systems as well as platforms (Windows, Linux, Unix, Mac OS X and etc). Thirdly, PHP has support for over 20 relational database management systems (RDBMS) including MySQL, Oracle as well as Ms Access, just to name a few. Apart from that, PHP provided by PHP Group is completely free where a complete collection of source codes can be downloaded from its official page, www.php.net for developers to build, customize and extend for their own use. Moreover, PHP eliminates client configuration problems because there is no need to worry whether client has the appropriate software installed on their side, since the web application runs seemingly on the server side.

On the other hand, PHP poses some weaknesses which are yet to be resolved in recent releases. Such weaknesses are as follows. Firstly error handling in PHP is still primitive as it does not provide sufficient detail to help diagnose errors that occurs within the framework. [2] Such common errors can only be improved by intercepting its condition and providing a redirection for common HTTP status codes such as 404 Not Found by the web server. Furthermore, a number of core modules of PHP are not thread safe, and because of this PHP Group doesn’t recommend running PHP on Apache 2 server in a multithreaded environment. [3] Additionally, PHP has a hard time handling Unicode, which is an industry standard for representing and handling text expressed in most of the world’s writing systems. However, such trivial feature remains one of the enhancements to be implemented for PHP 6 [4]

**Version History**

The first version of PHP/FI, was known as Personal Homepage Tools/Form Interpreter, was a collection of Perl scripts in 1995. One of its basic features was Perl-like language for handling form submission. [5][11] A rewrite came with PHP/FI2 somewhere in 1997, when PHP/FI was revealed to be not as powerful as it seemed due to the lack of many common features, for instance, the way while loops were implemented. When PHP3 was released, it also came along with a new name, Hypertext Preprocessor and implemented a new extension API which made it possible to support extra extensions.[11] In the late 1998, PHP 4 was introduced which came with a new paradigm of “compile first, execute later”. In terms of performance, it was so much better than that of PHP 3 because it compiles PHP scripts into machine code (instead of byte code) which will then be executed by Zend Engine. However, it comes with a tiny issue in terms of backward compatibility.[5][11] In 2004, PHP 5 was released, powered by the new Zend Engine II. One major feature that was incorporated includes support for object-oriented programming with the PHP Data Object (PDO) extension. Apart from that, with the new SimpleXML extension, XML documents are made easier to be manipulated as compared to previously which put PHP5 on par or overtakes with other web technologies in some areas.[5]

**Strategy in Securing PHP Applications**

About 30% of all vulnerabilities listed on National Vulnerability Database are related to PHP. Such vulnerabilities are often resulted from not following best practice of programming rules, only 23 in year 2008, which was approximately 1% in total is caused by security flaws of the language itself. [5] Since security in PHP web application encompasses remote as well as local security concerns, it is indispensable for PHP developers to inculcate certain habits in order to make sure their web application are as secure as possible despite the security issues regarding the actual platform as well as operating systems. Such habits are described as follows. Firstly, a primary habit where a PHP developer should adopt is to validate the input data. Considering where your input data is going meticulously will allow you to build a robust and secure application. [6]

4 **Problem Formulation**

The architecture of PHP is different as compared to other commonly used web applications languages such as J2EE and ASP.NET. Nevertheless, it is a lot more complex as compared to static web pages and comprises more than just the front end graphical user interface that users see. Quality of PHP web applications are complex and remain as multidimensional attribute since its functioning in a non-clustered environment. In order to improve the quality of PHP web applications, a number of aspects need to be vetted carefully. Nonetheless, this paper focuses on the quality of PHP
applications in terms of validity (adherence to official DTD for HTML 4.01 or XHTML 1.0 set by W3C) of HTML documents generated by it. Validating dynamic PHP web application is a non trivial task as it is difficult to predict the state and flow of the execution of PHP scripts for webpage generation. Secondly, unlike HTML tags which are tree based, the complexity provided by the PHP language is a lot more due to the functionalities and syntax available in that language. One could write a script which is recursive as well as write a nested loop which may eventually evaluate to further expressions, thus making it hard to represent it using a simple heuristic rule.

5 Motivation

Web applications currently make up one of the largest growth areas in software. PHP is one of the most widely used server-side programming language. A lot of platforms are built on PHP including Drupal, WordPress, Joomla, Magneto and etc. Web applications that are built on top of these platforms may have high amount of visitors navigating to their website, where each of them may be using different web browsers such as Mozilla Firefox, Safari, Internet Explorer, Google Chrome and etc. In order for the dynamically generated web documents to render properly across a wide spectrum of web browsers, it is vital for these documents to adhere to the official DTD for HTML 4.0.1 or XHTML 1.0 set by W3C. Unfortunately, there are still a lot of invalid websites in the World Wide Web. Apart from being motivated for improving the quality of PHP web applications, having a well-formed or valid HTML documents also portrays an aura of professionalism since there is little or almost no certification for web professionals.

6 Proposed Solution

In this paper, we proposed a novel PHP Validation Tool for dynamic web application. To validate HTML codes in PHP scripts, we need to check the opening and closing tags in PHP syntaxes. To achieve this, we need to study and examine carefully PHP language, particularly its data type, variables, methods, and classes. Our proposed solutions are divided into several fundamental steps. Details of these steps are described below:

Step 1: Conversion of variables to their types

Due to the fact that PHP language is not type safe, it is harder to determine the type of variables in PHP. Therefore, we develop a type conversion tool for each of the PHP variable. Type conversion tool is needed as knowing the type of variables in PHP will help us to determine the output of HTML tags through print command such as echo. Numeric variables are given the type of int, whereas variables with false or true content will be assigned Boolean type. On the other hand, variables with alphanumeric content will be assigned string type. We use List data structure to store all these variables with their associated types.

Step 2: Logical and Condition Comparison

Logical comparison in PHP occurs when for, while, and if statement is issued. For such a case, we check for logical operators such as <, >, &&, and ||. However, logical comparison in PHP is far more complicated than the usual simple logical comparison. PHP language supports nested logical comparisons, where logical operators are encapsulated within themselves. To handle this scenario, we use a stack data structure where operator such as ( and ) are pushed and popped when necessary. Whenever our tool encounters ‘(‘ character, we pushed the stack. Whenever a stack is popped due to the ’)’ character, we evaluate the current logical condition of that operator and update the logical condition accordingly. This procedure is repeated recursively until the stack is empty. Checking the logical condition is important as it is necessary to check the flow of the print statement (particularly those print statements that output HTML tags) in the logical statement.

Step 3: Method Checking

At the method level, all the attributes associated with methods are given due consideration for validation. This includes attributes such as return type, parameters, and variables passing. These attributes are given due attention as they contain important information related to HTML tags creation. A method may be called locally in a PHP file or it may be called across different classes. In the case of local method call, it may not be necessary to link the method to the calling class and duplicate method naming may create conflicts. On the other hand, it may be possible to call the same method name from different classes. For such a case, the calling class may have to be associated with the method in order to differentiate them.
Step 4: Object Level Validation

For Object Level Validation, due considerations are given for the creation of local and global variables. Local variables are created at the method level, whereas global variables are created at the class level. Handling local and global variables are important as their values are changed at different level. Furthermore, it is possible to create two similar local variables within different scope in a method whereas global variable is allowed to be created once. We use the conversion type tool in Step 1 to convert variables declared locally and globally. A variable and method in a particular class may called different class located in different location. For such a case, a mapping and parameter passing to another class is handled using the same convention provided in languages such as PHP, Java, and .NET. On the other hand, PHP provides feature where another PHP file can be included in a PHP file. To handle this scenario, we check for the PHP file associated with the include statement and have them included in the processing in the main PHP file.

Step 5: Further handling of variables

Since PHP language is not type safe, it is therefore difficult to parse the PHP files. A developer may accidently omit ; statement in his codes and PHP is still able to render and call the appropriate functions correctly. PHP also does not differentiate ' and " characters, which is widely used in String statement. This is a huge disadvantage as one is not able to determine whether a string of characters are actually a PHP statement or a collection of more than one PHP statements. In order to parse through the PHP codes effectively, we first screen through the existing PHP files and determine the variables, methods, and objects creation. Every variables, methods, and objects creation are stored in a list. Variables are stored in a list which is a reference to the methods list. The method list, on the other hand, is stored in the object list. When a PHP codes are parsed, we use this list to determine the number of statements.

7 Experimental Tests

We conduct our experimental tests on a wide range of datasets. We collect a sample of 200 pages from the deep web repositories. Due to the fact that our system is the first to validate PHP scripts, we are not able to benchmark our system against other state of the art systems. Secondly, our system needs to validate against PHP scripts, not static HTML pages. For such a case, special attention is taken to collect data from sites which contain server side scripts written in PHP. To measure the effectiveness of our algorithm, we use precision and recall which are formulated as follows:

Recall=\frac{Correct}{Actual} \times 100
Precision=\frac{Correct}{Extracted} \times 100

Correct depicts the number of pages correctly validated. Actual is the actual number of pages to be validated. Extracted depicts the number of pages validated.

As shown in Table 1, our system is able to validate PHP scripts effectively. A number of PHP scripts failed to be validated by our system due to the fact that they contain complicated loops for logic processing. Secondly, PHP scripts are not type safe, for some of the cases our validator fails to check for the type of variables correctly. Fortunately, our validator tool is able validate most of the PHP scripts successfully. Our validator works for nested loops and well defined scripts, and for some cases even scripts which has object level programming works well on our validator.

<table>
<thead>
<tr>
<th>Terms</th>
<th>Our system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td>200</td>
</tr>
<tr>
<td>Extracted</td>
<td>195</td>
</tr>
<tr>
<td>Correct</td>
<td>176</td>
</tr>
<tr>
<td>Recall</td>
<td>88.00%</td>
</tr>
<tr>
<td>Precision</td>
<td>90.26%</td>
</tr>
</tbody>
</table>

8 Conclusions

In this paper, we developed a novel PHP Validation Tool to validate dynamic web applications. To the best of our knowledge, our tool is the first tool to validate PHP scripts. Validating PHP scripts is a non trivial task due to the ambiguity and complexity presented in that language. Besides, the language lacks uniformity in its design. Validating PHP scripts requires careful consideration over the various syntaxes provided in that language. Dynamic web applications are useful as well formed HTML pages will be more compatible across browsers, and it is also easier to render a well formed HTML page. Our PHP validation tool is certainly useful for the design of future HTML Validation Tool.
9 References