DRC: A Detection Tool for Dangling References in PHP-Based Web Applications

Hung Viet Nguyen, Hoan Anh Nguyen, Tung Thanh Nguyen, and Tien N. Nguyen
Electrical and Computer Engineering Department
Iowa State University, USA
{hungnv, hoan, tung, tien}@iastate.edu

Abstract—PHP is a server-side language that is widely used for creating dynamic Web applications. However, as a dynamic language, PHP may induce certain programming errors that reveal themselves only at run time. A common type of error is dangling references, which occur if the referred program entities have not been declared in the current program execution. To prevent the run-time errors caused by such dangling references, we introduce Dangling Reference Checker (DRC), a novel tool to statically detect those references in the source code of PHP-based Web applications. DRC first identifies the path constraints of the program executions in which a program entity appears and then matches the path constraints of the entity’s declarations and references to detect dangling ones. DRC is able to detect dangling reference errors in several real-world PHP systems with high accuracy. The video demonstration for DRC is available at http://www.youtube.com/watch?v=y_AKZYhLiu4.

I. INTRODUCTION

To support multiple configurations and enhance user interactivity in a Web application, developers often write Web programs in a dynamic scripting language to generate different versions of the client page at run time. PHP is a widely-used language for creating such dynamic Web applications. However, being an interpreted language, PHP may induce certain types of programming errors that are difficult to detect before run time. In this paper, we are interested in the static detection of dangling reference errors in PHP programs. A reference to a program entity (e.g., a variable or a function call) becomes dangling at run time if the entity has not been declared in the current execution [1], resulting in run-time errors. Let us illustrate this type of error through an example.

Figure 1 shows an example of a PHP-based Web application. The file Login.php is used to generate a login page with an HTML form named loginform (line 12), an input field userid (line 16 or 21), and a button (line 17 or 22) for a user to submit a user ID to the page VerifyUser.php for verification. The JavaScript (JS) function validate () (line 6) is used to catch an empty input on the onsubmit event of the form (line 12). The text in the login form can be displayed in different languages depending on different configurations (defined via the variable $lang on line 13). If the language option is 'de' (line 14) and a dictionary for German is available (line 15), the dictionary variable $dict_de will be used to display the text in German. Otherwise, if the language option is 'en' (line 20), the text will be displayed in English. When the Login.php program is executed on the server side, the corresponding client-side code in HTML/JS will be generated. Figure 2 shows the client-side HTML/JS code, which is the PHP program’s output, corresponding to the German language option.

When that client-side code is run on the client’s browser, the user could enter his/her user ID. The user ID is first validated by the JS function validate () (Figure 3). The PHP variable $REQUEST (line 1) contains the submitted user ID. Then, a query is sent to the database (by a call to mysql_query at line 3) to look up the user’s information (using the fields firstname and lastname in the SELECT part of the SQL query). If that information is found, the user’s first name, middle initial, and last name will be displayed (line 6).
**Dangling Reference Errors.** The above program contains several dangling references that will be exposed in only certain program executions. In Figure 1, the PHP variable $input is declared either on line 16 (in the execution when the conditions $lang == ‘de’ (C1) and isset($dict_de) (C2) both evaluate to true) or on line 21 (in the run when the condition C1 is false and $lang == ‘en’ (C3) is true). It is then used on line 24 in all program executions. Thus, the PHP reference $input on line 24 will be undefined in the two cases of execution satisfying: 1) C1 is true and C2 is false, or 2) C1 is false and C3 is also false.

Two other dangling errors are 1) the reference to the HTML input userid from JS code (line 7, Figure 1), and 2) the reference to the same input but from PHP code (line 1, Figure 3). These references exist in all executions whereas their corresponding declaration is defined for only certain configuration options (lines 16 and 21, Figure 1). In this case, we call them embedded dangling references since they are embedded in PHP string values (line 7, Figure 1 and line 1, Figure 3). Another error is that the reference to the database entity middleinitial (line 6, Figure 3) is undefined since the field middleinitial is not selected from the SELECT part of the SQL query (line 3). We also call it an embedded dangling reference.

Those aforementioned dangling errors can lead to unexpected behavior of the program at run time: The first error results in an empty login form, the second and third errors change the behavior of the JS function validate and the file VerifyUser.php, and the fourth one either displays missing user information or produces unwanted error messages on the client page. In general, the issues caused by such dangling references can range from warning messages, incorrect behavior, unexpected run-time errors, to security vulnerabilities.

Since dangling references in PHP programs can cause run-time errors, it is desirable to detect them early. However, as there exist numerous execution paths in a program, it is non-trivial to determine what execution paths contain a reference that does not have its corresponding declaration. In addition, some dangling references are embedded within PHP string literals or variables’ values. To identify such references, a tool will need to understand the semantics of the embedded code, which is often incomplete code fragments. Moreover, the declarations and references could be cross-language (e.g., a JS variable referring to an HTML input, or a PHP reference referring to an SQL entity as in the example).

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Type</th>
<th>D/R Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$input</td>
<td>Login.php@L16</td>
<td>PHP variable</td>
<td>Decl C1 AND C2</td>
</tr>
<tr>
<td>$input</td>
<td>Login.php@L21</td>
<td>PHP variable</td>
<td>Decl (NOT C1) AND C3</td>
</tr>
<tr>
<td>$input</td>
<td>Login.php@L24</td>
<td>PHP variable</td>
<td>Ref TRUE</td>
</tr>
<tr>
<td>$input</td>
<td>Login.php@L16</td>
<td>HTML input</td>
<td>Decl C1 AND C2</td>
</tr>
<tr>
<td>$input</td>
<td>Login.php@L21</td>
<td>HTML input</td>
<td>Decl (NOT C1) AND C3</td>
</tr>
<tr>
<td>$user Id</td>
<td>Login.php@L7</td>
<td>JS ref to HTML</td>
<td>Ref TRUE</td>
</tr>
<tr>
<td>$user Id</td>
<td>VerifyUser.php@L1</td>
<td>PHP ref to HTML</td>
<td>Ref TRUE</td>
</tr>
<tr>
<td>middleinitial</td>
<td>VerifyUser.php@L6</td>
<td>PHP ref to SQL</td>
<td>Ref C4</td>
</tr>
</tbody>
</table>

**Fig. 4. Overview of DRC’s dangling reference detection algorithm**

**II. DRC’S APPROACH**

We introduce **Dangling Reference Checker (DRC)**, a tool to statically detect PHP and embedded dangling references in PHP-based Web applications. The key ideas in DRC are as follows. Given a PHP program, all declarations and references in the program are first collected into an Entity Table (Table I). Each declaration or reference has an associated constraint representing the conditions of the program executions in which it will appear. For example, the first entry in Table I shows that the declaration of the PHP variable $input is found on line 16 of Login.php with the constraint C1 AND C2 (Figure 1). In the second step, with the Entity Table, the constraints of all the declarations and references are compared to identify dangling references. Let us describe the detailed steps via Figure 4.

**Extracting PHP entities (Step S1).** DRC uses PhpSync [3] to symbolically execute a PHP program and identify PHP declarations and references together with their constraints. In PHP, a variable is declared when it is assigned with a value in a program execution such as in an assignment expression, a parameter of a function call, or a running variable of a for statement [1]. The variables that appear in non-declaring expressions are considered as references. For example, the PHP variable $input has two declarations on lines 16 and 21 (for the two language options) and one reference on line 24 of Login.php (Figure 1). When a declaration or reference is detected, DRC associates it with the current path constraint of the symbolic execution and stores that information into the Entity Table.
Extracting embedded entities (Steps S2-S4). To detect entities that are embedded in PHP code, DRC uses PhpSync’s D-model [3], a tree-based representation, to approximate all possible string values of the variables and the program’s output during the symbolic execution of a PHP program (step S2). Figure 5 shows the D-model representing the value of the PHP variable $input at line 24 in Figure 1. The Select nodes in the D-model represent the alternative values associated with the executions’ path constraints. A Concat node represents a concatenation of constituent string values or D-models.

Given the D-models of the program’s output (containing HTML/JS code) and SQL queries (containing SQL code), DRC uses specialized parsers to extract HTML/JS entities (step S3) and SQL entities (step S4). The parsers operate on a D-model tree and parse the fragmented code contained in the D-model’s leaf nodes. For example, in Figure 5 since the string node userid appears after the HTML code fragment <input name=’>, DRC recognizes that userid is the declared name of an HTML input entity. Then, DRC associates each detected embedded entity with the constraints of all Select nodes that are the ancestors of the leaf node containing that entity. In this example, the HTML entity userid has two declarations whose corresponding constraints are C1 AND C2, and (NOT C1) AND C3.

Matching entities to detect dangling references (Step S5). Using the Entity Table, DRC matches each reference against their declarations to check if it is dangling. There are two cases in which a reference becomes dangling: 1) there exist no declarations with the same name and type as the reference; and 2) such declarations are found, but the combined constraint of all the matched declarations is stricter than the constraint of the reference (i.e., there exist some program executions in which the reference exists while its declarations do not). For example, when matching the reference $input (entry 3 in Table I) with its declarations (entries 1 and 2), DRC finds that the combined constraint of the declarations is (C1 AND C2) OR (NOT C1 AND C3), which does not cover the constraint TRUE of the reference. Thus, it detects that the reference is a dangling one with either (C1=true, C2=false) or (C1=false, C3=false). Those two conditions correspond to the execution paths when the variable $input is used but undefined.
If a reference cannot be matched to any declaration by names/types/constraints, DRC reports it as a dangling error. These dangling references are shown as a special entity group named “[Dangling References]” (Figure 7). As seen, DRC detects two dangling references in the file Login.php: the PHP variable $input on line 37 and the (embedded) JS reference to the HTML input userid on line 17 (not visible in Figure 7).

B. Case Studies

Figure 8 shows an example in which DRC detects that middleinitial (line 13) is an SQL embedded dangling reference, since middleinitial was missing from the database query (line 9). Figure 9 displays DRC’s detection result on SquirrelMail at revision 11,338. The PHP variable $query is defined only when $response == 'NO' (line 397). In the else branch of the if statement on line 396 ($response == 'NO' evaluates to false), the variable $query is not defined, and thus its reference on line 405 is a dangling one. The commit log from the fix at the next revision confirms this error detected by DRC: “$query is also used when $response != 'NO'. Fixed undefined notice error...”

C. Discussion

We have analyzed DRC’s detection results and found a few causes of false positives/negatives. First, our symbolic execution in some cases could not resolve the variables’ values (e.g., the name of an included file), resulting in incorrect detection of dangling references. Second, the approximation of the program’s output and SQL queries using symbolic execution might lead to some missed embedded entities. Finally, our heuristic to solve the constraint satisfaction problem using boolean formulas is another source of inaccuracy.

IV. RELATED WORK

Despite the success of fault localization techniques in dynamic Web code, none of the existing approaches addresses the static detection of PHP and embedded dangling references. However, DRC is related to those techniques for fault localization in Web applications. In Apollo [4], a variation of Tarantula is applied together with a dynamic output mapping technique using an instrumented interpreter for a PHP program, allowing the PHP statements that affect the output to be given higher suspiciousness scores. Clark et al. [5] proposed a method to localize faults inside SQL commands in Java database applications. Both of those approaches are dynamic and do not aim to detect dangling PHP and embedded references. DRC is also related to TypeChef [6], a type checking tool with variability-aware parsing for C code in the presence of conditional compilation. Constraints are associated to code portions, and their parser derives the combinations for correct parsing. In comparison, due to the dynamism in PHP, DRC must deal with all conditions in a PHP program, while TypeChef needs to handle conditional compilation in C code. Moreover, DRC uses symbolic execution to detect embedded entities in HTML/JS/SQL code, which is the output of a PHP program, while TypeChef deals with only one language.

V. CONCLUSIONS

In dynamic PHP-based Web applications, dangling reference errors occur when the referred program entities have not been declared in the current program execution, resulting in run-time failures. We introduced DRC, a tool for statically detecting dangling reference errors by matching the constraints of the entities’ declarations and references. When run on several real-world PHP systems, DRC was able to detect both PHP and embedded dangling references with high accuracy.

ACKNOWLEDGMENT

This work was supported in part by NSF grants CCF-10-18600 and CNS-12-23828.

REFERENCES