American Journal of Engineering Research (AJER)
e-ISSN : 2320-0847 p-ISSN : 2320-0936
Volume-02, Issue-09, pp-76-80
www.ajer.org

Research Paper Open Access

Client-Side Validation and Verification of PHP Dynamic Websites

K.Sowjanya, N. Deepika, P.V.S.Srinivas
[1]*PG Scholar, Department of Computer Science and Engineering, TKR College of Engineering and Technology Hyderabad, A.P-500 097, India*
[2]*Assistant Professor, Department of Computer Science and Engineering, TKR College of Engineering and Technology Hyderabad, A.P-500 097, India*
[3]*Professor& Head, Department of Computer Science and Engineering TKR College of Engineering and Technology Hyderabad, A.P-500 097, India*

***Abstract*:** In contemporary, populace are interested in fault-localization techniques which are based on statistical analysis of program statements executed by going through and unexpected executions. Here we explore three fault localization algorithms which are referred as Tarantula, Ochiai and Jaccard can be evaluated to localize faults perfectly in web applications which are developed in PHP. These algorithms uses source mapping for conditional and function-call statements. We developed different fault localization techniques and test-generation cases in Apollo, and explored them on many open-source PHP apps. Then the ultimate output of tests are that Ochiai algorithm contains all our enhancements which localizes more than 80% of all faults in executed statements, compared to unenhanced Ochiai algorithm where it gives only quarter percent amount. And also we checked that , the strategies which are used for test generation taken under part are efficient to know the algorithm perfectness. And also these algorithms are cost efficient. Moreover, normally, a straight forward concept which depends upon similarities of path-constraint gets high range perfectness in less than 10 test generation, when compared to other strategies where we have to do more like almost half century.

Keywords: - *Fault localization, statistical debugging, program analysis, web applications, PHP.*

I. INTRODUCTION

Web application provides a user friendly interface to the users. Proper functionality and security testing should be done on the web application, prior to the web application release. The scope of the paper is to identify bugs in the usage of web apps in the network with the help of dynamic test generation. Web applications can be developed using server side programming languages (JSP, PHP, ASP etc,) or client side programming languages or with static web pages (HTML) and with combination of the languages. In the dynamic web sites, developed with server side programming languages accepts inputs from the remote users [9]. The remote user can pass web application expected inputs or can pass malicious inputs. If malicious inputs are passed to the web application, the behavior of the web application may change. The web application may crash at some instance. The web application response may be an unexpected that causes the browser to crash. The web application should be thoroughly tested for the functionality and in the security aspect. The draw backs in existing system are that it approaches are for testing the webpage validation. The existing approaches cannot identify the dynamic pages. The approaches cannot test the web application with the dynamic inputs. These cannot identify the dynamic pages and cannot generate test cases dynamically [3].

It identifies the expected response using state model maintenance. The state model is the request verves response combinations in the normal scenario. Should identify the security flaws of the web application. The proposed system is only for the web application developed with the server side scripting language PHP. Bug report contains a failure, a set of path constraints exposing the generates a report of a set of bugs. Each failure and a set of inputs exposing the failure. The path constraint minimization algorithm. The

method intersects returns the set of conjuncts that are present in all given path constraints and the method shortest returns the path constraint with fewest conjuncts.

In detail EB apps are developed based upon various languages of programming, such as Java- Script which was on interface of user, and PHP on mostly server side in embedded Structured Query Language (SQL) commands on the server side[7]. Such applications evaluate output in structured format with dynamic HTML pages those were helped in the execution of additional scripts. As with any program, programmers make mistakes and introduce faults. In the domain of web applications, some faults manifest themselves as web-application crashes and as malformed HTML pages that are not displayed properly in a web browser. While malformed HTML failures may seem trivial, and indeed many of them are at worst minor annoyances, they have on occasion been known to create serious vulnerabilities, e.g., via denial-of-service attacks.

Furthermore, such failures in the HTML code may be difficult to localize because HTML code is often dynamically generated by server side code written, for example, in PHP or Java and so, when a failure is detected, there really is no HTML file or line number to point the developer to. In this paper, we present Apollo, the first fully automatic tool that efficiently finds and localizes malformed HTML and execution failures in web applications that execute PHP code on the server side [6].

II. ARCHITECTURE

In particular, 1) it integrates an HTML validator to check for failures that manifest themselves by the generation of malformed HTML, 2) it automatically simulates interactive user input, and 3) it keeps track of the interactive session state that is shared between multiple PHP scripts[3]. However, our previous work focused exclusively on finding failures by identifying inputs that cause an application to crash or produce malformed HTML. We did not address the problem of pinpointing the specific web application instructions that cause these failures, and fixing the underlying faults can be very difficult and time consuming if no information is available about where they are located.

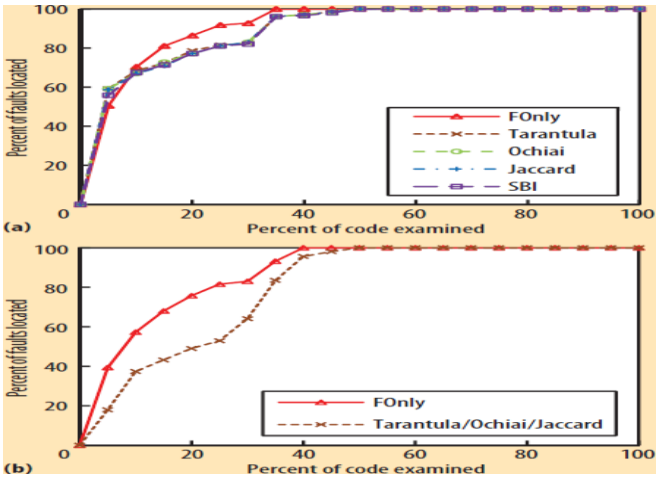

Fig.1 Fault localization technique

This paper addresses the problem of determining where in the source code changes need to be made in order to fix the detected failures. This task is commonly referred to as fault localization, and has been studied extensively in the literature [8]. The fault- localization algorithms explored in this paper attempt to predict the location of a fault based on a statistical analysis of the correlation between passing and failing tests and the program constructs executed by these tests. In particular, we investigate variations on three popular statistical fault-localization algorithms, known as Tarantula Ochiai and Jaccard. These algorithms predict the location of a fault by computing, for each statement, the percentages of passing and failing tests that execute that statement [7]. From this, a suspiciousness rating is computed for each executed statement. Programmers are encouraged to examine the executed statements in order of decreasing suspiciousness.

The effectiveness of a fault localization technique can be measured by determining how many statements need to be inspected, on average, until the fault is found. Our work differs from most previous research on fault localization in that it does not assume the existence of a test suite with passing and failing test cases. Instead, we rely on combined concrete and symbolic execution to generate passing and failing runs. This paper advances the state of the art in fault localization in two ways. First, we present enhancements to previous

statistical fault- localization techniques that make them significantly more effective at localizing the faults responsible for execution failures, and for the generation of malformed HTML pages in PHP web applications [10]. These enhancements include: The use of an extended domain. We apply existing statistical fault-localization techniques on an extended domain of pairs, in which the runtime value serves to differentiate occurrences of the statement at runtime. Applying this technique to conditional statements helps fault localization with the identification of errors of omission such as missing branches, and applying the technique to function calls enables us to differentiate normal return values from values such as null that are often correlated with erroneous [4]. The use of a source mapping [9]. We modified the PHP interpreter to maintain a source mapping that records the statement in the PHP application that produced each fragment of output at runtime; thus, it is a map from print statements to regions of the HTML file. This mapping when combined with the report of the HTML validator, which indicates the parts of the HTML output that are incorrect provides an additional source of information about possible fault locations, and is used to fine-tune the suspiciousness ratings of existing fault- localization techniques[5]. A second main research topic explored in this paper has to do with the fact that existing fault-localization approaches assume the existence of a test suite.

To summarize, the contributions of this paper are as follows:

1. We present two mechanisms, the use of an extended domain and the use of a source mapping, that significantly enhance the effectiveness of existing fault-localization techniques such as Tarantula, Ochiai, and Jaccard.
2. To evaluate these fault-localization techniques in Apollo, we implemented each of the fault localization techniques; localized 115 randomly selected faults in five PHP applications, and compared the technique‘s effectiveness. Our findings show that, using our best technique, an enhanced version of Ochiai, 87.8 percent of the faults are localized to within 1 percent of all executed statements, compared to only 37.4 percent for the unenhanced Ochiai algorithm [1]. Similar improvements were obtained for enhanced versions of the Jaccard and Tarantula algorithms.
3. We present a family of directed test-generation techniques, based on combined concrete and symbolic execution that are capable of generating small test suites with high fault- localization effectiveness. These techniques overcome the important limitation of many previous fault-localization methods that a test suite be available upfront [10].
4. To evaluate directed test generation, we implemented these directed test-generation techniques in Apollo. Our evaluation shows that a directed technique based on path-constraint similarity reduces test suite size by 86.1 percent and generation time by 88.6 percent when compared to an existing undirected test-generation technique, without compromising fault-localization effectiveness [1].
   These findings show that automated techniques for fault localization, which were previously primarily evaluated on programs with artificially seeded faults, are effective at localizing real faults in open-source PHP web applications.

The remainder of this paper is organized as follows:

Section 2 reviews the PHP language and the kinds of failures that may arise in PHP programs. Section 3 reviews the Tarantula, Ochiai, and Jaccard fault-localization algorithms and presents our extensions to these algorithms [6]. In Section 4, we present algorithms for test generation that are directed by similarity criteria toward the generation of test suites with high fault-localization effectiveness. Section 5 presents an implementation of our work in the context of the Apollo tool. Section 6 presents an evaluation of our fault localization and test-generation algorithms on a set of open source PHP programs.

Fault localization

In this section, we first review the basic fault-localization algorithms we implemented and extended: Tarantula, Jaccard, and Ochiai [4]. We then present an alternative technique that is based on source mapping and positional information obtained from an oracle. Next, a technique is presented that combines the former with the latter. Finally, we discuss how the use of an extended domain for conditional and return-value expressions can help improve the basic algorithm‘s effectiveness [2].

III. IMPLEMENTATION

Step 1: state manager
The module keeps track of all the pages to be visited.
If finds the absolute urls that are possible on the site under test.
It maintains the state of the url.

Step 2: Shadow Interpreter
It executes the server side program for testing
It interprets each url that are to be processed.
It executes programs with, without, negative values, and large values as input arguments
It gathers the web pages produced for each interpretation.

Step 3: UI Option analyser
It gathers all the input variables that are used in each server side program.
It finds the arguments names such as "name","id"etc,.
It works based on syntactic analysis on the program.

Step 4: symbolic driver
It consists of all the syntactic symbols and their states for identifying the input arguments.

For example:
- GET_[' -> symbol
- '] -> symbol

Step 5: constraint solver
It arranges the absolute urls with all possible combinations.
Eg: login.php has id, pwd as input args.
URL 1: http://www.xyz.com?id=
URL 2: http://www.xyz.com?id=asdf&pwd=123

Step 6: input value generator
Based on the input argument type, an integer can have negative, positive values.
It produces and assigns null values for the input arguments.
It produces large values for integer, float, char etc,

Step 7: bug finder

The outputs of shadow interpreter are a web page.
The page is validated against W3C validator.
If W3C validator confirms with any bug, the URL and the page is considered as reproducible scenario.

Step8: path constraint minimization
It tries to eliminate the unnecessary urls that are causing same bug.
The minimized urls set and its value causes the reduced and optimal bug report.

IV. RELATED WORK

This section reviews the literature on fault localization, focusing primarily on fault localization techniques that predict the location of faults based on the analysis of data from multiple tests or executions. In addition, we discuss research that explores the impact of test-suite composition on fault-localization effectiveness.

Fault Localization
In previous works these fault localization was used for slicing purpose in programming. As inverse here are the two people who used these fault localization for the sake of dicing which was used for combining those slices of programming [2]. Those scientists are Lyle and Weiser. The way of these fault localization works in order to find these faults in information. For example if we take A and B values if we get perfect result for a and some wrong for B then we have to check with respect B variation.[5] These variations are later invented by Pan and Spafford and Agrawal et al.

Trace comparisons
Renieris and Reiss use set-union, set-intersection, and nearest neighbor methods for fault localization; these all work by comparing execution traces of passing and failing program runs. Set-union. It computes the bunch of each and every statement that which are executed by sending test cases and cutoffs when it was executed by failing test case these from bunch of statements. The set which was the outcome of

failing test includes statements which are suspicious. This is the where programmer interacts first.

Here faulty statement was not included an report, based upon System Dependence Graph (SDG) scientists Renieris and Reiss proposed a ranking technique. Here the ranking technique involves additional statements [8]. These statements are set of things based upon distance and Set-intersection where the previous reports are edge touched them. It discovers the statements which are went successfully through all test cases, not only by single case failing test, and address errors of omission testing attempts don‘t under take or won‘t consider failing test case in the execution of statement. It chooses the passing test case that contains feature of execution spectrum that has common similarities with the failing test case distance criteria. In the event the faulty statement does not contain in the report, based upon System Dependence Graph scientists Reiner‘s and Reiss proposed a ranking technique. Here the ranking technique involves additional statements [11]. These statements are set of things based upon distance. Nearest Neighbor was evaluated on the Siemens suite, and was found to be superior to the set-union and set intersection techniques.

V. CONCLUSION

Until now, statistical fault-localization techniques that analyze execution data from multiple tests have been applied primarily in the context of traditional imperative programming languages such as C and Java. In this paper, we have shown how such fault localization techniques can be made effective in the domain of PHP web applications. We have presented two enhancements to the existing Tarantula, Jaccard, and Ochiai fault-localization techniques that greatly increase their effectiveness:

The use of an extended domain of pairs for conditional statements and function calls. The use of source mapping that correlates statements in the PHP application with the fragments of output that they produce at runtime. The former helps with the localization of certain common kinds of failures such as missing branches in conditional statements and the return of unexpected values by functions due to corner cases that are not handled properly[11]. The latter increases the precision of fault localization by leveraging the fact that PHP applications are expected to produce syntactically valid HTML output, and that the location of errors in these generated pages can often be determined quite precisely.

Briefly the existing systems cannot analyze the dynamic pages, and cannot generate dynamic test cases. The proposed approach can generate dynamic test case for the dynamic web pages. The proposed approach can generate test cases for finding the security bugs.

REFERENCES

[1] R. Abreu, P. Zoeteweij, and A.J.C. van Gemund, “An Evaluation of Similarity Coefficients for Software Fault Localization,”Proc. 12th Pacific Rim Int‘l Symp. Dependable Computing, pp. 39-46, 2006.

[2] R. Abreu, P. Zoeteweij, and A.J. van Gemund, “On the Accuracy of Spectrum-Based Fault Localization,’’ Proc. Testing: Academic and Industry Conf. Practice and Research Techniques, pp. 89-98, Sept. 2007.

[3] R. Abreu, P. Zoeteweij, and A.J.C. van Gemund, “An Evaluation of Similarity Coefficients for Software Fault Localization,’’ Proc. 12th Pacific Rim Int‘l Symp. Dependable Computing, pp. 39-46, 2006.

[4] H. Agrawal, J.R. Horgan, S. London, and W.E. Wong,”Fault Localization Using Execution Slices and Dataflow Tests,’’Proc. Int‘l Symp. Software Reliability Eng., pp. 143-151, 1995.

[5] S. Artzi, J. Dolby, S. Jensen, A. Møller, and F. Tip, “A Framework for Automated Testing of JavaScript Web Applications,’’ Proc. Int‘l Conf. Software Eng., 2011.

[6] S. Artzi, J. Dolby, F. Tip, and M. Pistoia, “Directed Test Generation for Effective Fault Localization,’’ Proc. 19th Int‘l Symp. Software Testing and Analysis, pp. 49-60, 2010.

[7] S. Artzi, J. Dolby, F. Tip, and M. Pistoia, “Practical Fault Localization for Dynamic Web Applications,’’ Proc. 32nd ACM/ IEEE Int‘l Conf. Software Eng., vol. 1, pp. 265-274, 2010.

[8] S. Artzi, A. Kie_zun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M.D. Ernst, “Finding Bugs in Dynamic Web Applications,’’ Proc. Int‘l Symp. Software Testing and Analysis, pp. 261-272, 2008.

[9] S. Artzi, A. Kie_zun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M.D. Ernst,”Finding Bugs in Web Applications Using Dynamic Test Generation and Explicit State Model Checking,”IEEE Trans. Software Eng., vol. 36, no. 4 pp. 474-494, July/Aug. 2010.

[10] P. Arumuga Nainar, T. Chen, J. Rosin and B. Liblit, “Statistical Debugging Using Compound Boolean Predicates,’’Proc. Int‘l Symp. Software Testing and Analysis, S. Elbaum, ed., July 2007.

[11] P. Arumuga Nainar and B. Liblit, ―Adaptive Bug Isolation,‖ Proc. 32nd ACM/IEEE Int‘l Conf. Software Eng., pp. 255-264, 2010.