www.ijprems.com
editor@ijprems.com

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

Vol. 03, Issue 04, April 2023, pp : 28-34

e-ISSN : 2583-1062

Impact Factor : 5.725

EFFICIENCY AND EFFECTIVENESS OF WEB APPLICATION VULNERABILITY DETECTION

S. Hari Prasad [1], Dr. V. Sai Shanmuga Raja[2], Dr. S. Geetha[3]

[1]M. Tech - CFIS, Department of Computer science and Engineering, Dr.M.G.R Educational and Research Institute, Chennai 600 089, Tamilnadu, India.

[2]Professor, Department of Computer science and Engineering, Dr.M.G.R Educational and Research Institute, Chennai 600 089, Tamilnadu, India.

[3]Head of Department, Department of Computer science and Engineering, Dr.M.G.R Educational and Research Institute, Chennai 600 089, Tamilnadu, India.

ABSTRACT

Malicious users can attack Web applications by exploiting injection vulnerabilities in the source code. This work addresses the challenge of detecting injection vulnerabilities in the server-side code of Java Web applications in a scalable and effective way. We propose an integrated approach that seamlessly combines security slicing with hybrid constraint solving; the latter orchestrates extract minimal program slices relevant to security from Web programs and to generate attack conditions. We then apply hybrid constraint solving to determine the satisfiability of attack conditionsand thus detect vulnerabilities. The experimental results, using a benchmark comprising a set of diverse and representative Web applications/services as well as security benchmark applications, show that our approach is significantly more effective at detecting injection vulnerabilities than state-of-the-art approaches, achieving 98% recall, without producing any false alarm. We also compared the constraint solving module of our approach with state-of-the-art constraint solvers, using six different benchmark suites; our approach correctly solved the highest number of constraints (665 out of 672), without producing any incorrect result, and was the one with the least number of time-out/failing cases. In both scenarios, the execution time was practically acceptable, given the offline nature of vulnerability detection.

1. INTRODUCTION

Symbolic execution and constraint solving represent a state-of-the-art approach usedin security analysis to identify vulnerabilities in software systems. Symbolic execution executes a program with symbolic inputs and at the end generates a set of path conditions. Each of them corresponds to a constraint imposed on the symbolic inputs to follow a certain program path, i.e., a constraint characterizing a possible execution. By solving these constraints with a constraint solver, one can determine which concrete inputs can cause a certain program path to be executed. In the context of security analysis this approach is used to detect injection vulnerabilities, i.e., program locations in which certain malicious inputs can alter the intended program behaviour. Roughly speaking, this approach consists of solving the constraints obtained by conjoining the path conditions (generated by the symbolic execution) with attack specifications provided by security experts. The main strength of this approach is that vulnerability detection yields a limited number of false positives, since the concrete inputs determined with constraint solving prove the existence of vulnerabilities.

However, the effectiveness and precision of this approach are challenged by two main problems that affect symbolic execution and constraint solving 1) path explosion and 2) solving complex constraints (e.g., constraints involving regular expressions or containing string/mixed or integer operations). Notice that while these problems are independent from the context in which symbolic execution and constraint solving are applied, the solutions to mitigate them can be tailored for a specific context. Nevertheless, existing proposals in the context of vulnerability analysis have not fully addressed these problems. The path explosion problem is triggered by the huge number of feasible program paths that symbolic execution has to explore in large programs. To mitigate this problem in the context of vulnerability analysis, in previous work we proposed an approach to extracting security slices from Java programs. A security slice contains a concise and minimal sequence of program statements that affect a given security sensitive program location (sink), such as an SQL query statement. Symbolic analysis can then be performed on security slices instead of the whole program; in this way path conditions are analyzed only with respect to the paths leading to sinks instead of every path in the program. Since, according to our experience, the number of sinks ina program is low1 and security slices are much smaller (approx. 1%) than the programcontaining them, this approach can effectively mitigate the path explosion problem.

The path explosion problem is triggered by the huge number of feasible programpaths that symbolic execution has to explore in large programs. To mitigate this problem in the context of vulnerability analysis, in previous work

we proposed an approach to extracting security slices from Java programs. A security slice contains a concise and minimal sequence of program statements that affect a given security sensitive program location (sink), such as an SQL query statement. Symbolic analysis can then be performed on security slices instead of the whole program; in this way path conditions are analyzed only with respect to the paths leading to sinks instead of every path in the program. Since, according to our experience, the number of sinks ina program is low1 and security slices are much smaller (approx. 1%) than the programcontaining them, this approach can effectively mitigate the path explosion problem.

2. Literature Survey

Ajjarapu Kusuma Priyanka et.al., web application security has become a major challenge due to the common vulnerabilities found in web applications. Attackers possess a never-ending list of vulnerabilities and payloads to exploit them in order to gain access over various web applications maliciously. Each time when there are any changes made at some layer of web-application architecture, there exists a chance of creating novel vulnerabilities. In this paper, we brief out our analysis on common and familiar vulnerabilities like Sql Injection, Cross site Scripting and Cross site Request Forgery (CSRF) and demonstrate the exploitation of these vulnerabilities by considering DVWA, a highly vulnerable web application designed for education purpose. We carry out exploitation both manually and through automated tools. We conclude our research by inferring some preventive mechanisms to be adopted while designing the web applications to mitigate such types of attacks.

Bogdan Korniyenko et.al., developed web application protection system by using modern technologies NET Framework, ASP. NET Core, EF, SSMS, Swagger. The system is resistant to changes and outside interference, able to prevent unauthorized access. The main types of vulnerabilities in web applications are considered. The most popular ready-made services for the implementation of the appropriate protection are described. The white list model of developing secure web applications and the main steps of the model implementation is defined. Implement a white list model for a web application by using a system of roles and access. The server part of the web application has been developed, which includes the built-in functionality of the basic methods of hacking prevention. Impact of SQL injection through project architecture is not possible. A method for accessing private user information has been developed by using the Rijndael encryption algorithm.

Rizki Agung Muzaki et.al., the use of web applications has been undergoing rapid increase. Many individuals, groups, organizations or governments use web applications as a means to exchange information or support business-related tasks. Despite the increased adoption, web applications use is however directly associated with comparable threats and attacks. With the increasing threats and attacks on web applications, organizations require a more effective concept of web application security. Web Application Firewall (WAF) is a security concept that can be used to prevent various threats and attacks on web applications. WAF has the ability to filter packets, block dangerous HTTP requests, and also do logging. This paper demonstrates and proposes the implementation of WAF on a web-based application using Mod Security and the Reverse Proxy method. From the tests carried out e.g. cross-site scripting, SQL injection and unauthorized vulnerability web scanning, all threats were successfully thwarted by Mod Security and reverse proxy method implemented in the WAF.

Jeom-Goo Kim et.al., the expansion of the Internet has made web applications become a part of everyday life. As a result, the numbers of incidents which exploit web application vulnerabilities are increasing. A large percentage of these incidents are SQL Injection attacks which are a serious security threat to databases with potentially sensitive information. Therefore, much research has been done to detect and prevent these attacks and it resulted in a decline of SQL Injection attacks. However, there are still methods to bypass them and these methods are too complex to implement in real web applications. This paper proposes a simple and effective SQL Query removal method which uses Combined Static and Dynamic Analysis and evaluates the efficiency through various experiments.

Giovanni Agosta et.al., the automatic identification of security vulnerabilities is a critical issue in the development of web-based applications. We present a methodology and tool for vulnerability identification based on symbolic code execution exploiting Static Taint Analysis to improve the efficiency of the analysis. The tool targets PHP web applications, and demonstrates the effectiveness of our approach in identifying cross-site scripting and SQL injection vulnerabilities on both NIST synthetic benchmarks and real-world applications. It proves to be faster and more effective than its main competitors, both open source and commercial.

3. PROBLEM STATEMENT

The vulnerable user inputs within the web page or web application. A web page or web application that has SQL Injection vulnerability uses such user input directly in an SQL query. The attacker can create input content. The unauthorized viewing of user lists, the deletion of entire tables and, in certain cases the attacker gaining administrative rights to a database, all of which are high detrimental to a business.

INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

www.ijprems.com
editor@ijprems.com

Vol. 03, Issue 04, April 2023, pp : 28-34

e-ISSN : 2583-1062

Impact Factor : 5.725

4. PROPOSED SYSTEM

We proposed a fallback mechanism to extend existing string constraint solvers for dealing with constraints with unsupported string operations. This mechanism, implemented in the ACO-Solver tool, used an off-the-shelf automata-based string constraint solver combined with a search-driven constraint solving procedure based on the Ant Colony Optimization meta-heuristic. The goal of the work presented in this paper is to provide a scalable approach, based on symbolic execution and constraint solving, to effectively find injection vulnerabilities in source code, which generates no or few false alarms, minimizes false negatives, and overcomes the path explosion problem and the one of solving complex constraints.

5. METHODOLOGY

Motivation

The challenges in adopting an approach based on symbolic execution and constraint solving in the context of vulnerability detection. Although we crafted this example for illustrative purposes, it can be considered realistic since it contains typical operations that are commonly found in modern Web applications. Moreover, it contains vulnerabilities that embody the patterns tracked in the CWE dictionary. The codes at vulnerable to XSS because of the inadequate sanitization procedure applied to variable sid, which contains a user input the codes at line is vulnerable to XPathi because the variable sid, containing a user input, is not sanitized properly before using it in the XPath query. Indeed, the standard sanitization procedure from OWASP [10] applied to variable sid only escapes meta-characters.

Existing Challenges

The path explosion problem is triggered by the huge number of feasible program paths that symbolic execution has to explore in large programs. To mitigate this problem in the context of vulnerability analysis, in previous work we proposed an approach to extracting security slices from Java programs. A security slice contains a concise and minimal sequence of program statements that affect a given security sensitive program location (sink), such as an SQL query statement.

Architecture

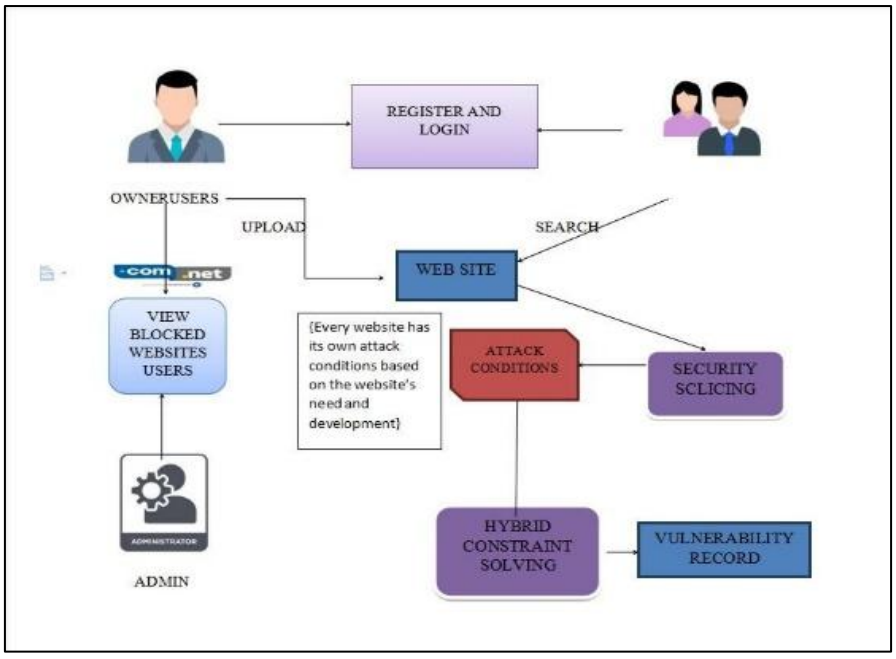

Figure 1. Architecture

Modules

User

- Register
- Login
- Search
- View the website
- Make right click in website

Admin

- Login
- Approve the web site
- Block unwanted website
- View user details

Owner

- Register
- Login
- Register Web site
- Upload and ready to host Website
- View blocked user account

Register

Register Your Account. A check register, also called a cash disbursements journal, is the journal used to record all of the checks, cash payments, and outlays of cash during an accounting period.

Login

A login is a set of credentials used to authenticate a user. Most often, these consist of a username and password. However, a login may include other information, such as a PIN number, password, or passphrase. Some logins require a biometric identifier, such as a fingerprint or retina scans.

Search

A web search engine or Internet search engine is a software system that is designed to carry out web search (Internet search), which means to search the World Wide Web in a systematic way for particular information specified in a textual web search query

View the website

In this module, user can view the website which is uploaded by website owner.

Make right click in website

In this module, if user tries to make right click for get a source code that account will blocked automatically.

Admin

Login

A login is a set of credentials used to authenticate a user. Most often, these consist of a username and password. However, a login may include other information, such as a PIN number, password, or passphrase. Some logins require a biometric identifier, such as a fingerprint or retina scans.

Approve the web site

This module, the administrator ready to confirm the site and if the site is authenticable administrator will favor to dynamic the site

Block unwanted website

On the Internet, a block or ban is a technical measure intended to restrict access to information or resources. Blocking and its inverse, unblocking, may be implemented by the owners of computers using software. Privileged users can apply blocks that affect the access of the undesirables to the entire website.

Owner

Register

Register Your Account. A check register, also called a cash disbursements journal, is the journal used to record all of the checks, cash payments, and outlays of cash during an accounting period.

Login

A login is a set of credentials used to authenticate a user. Most often, these consist of a username and password. However, a login may include other information, such as a PIN number, pass code, or passphrase. Some logins require a biometric identifier, such as a fingerprint or retina scans.

Register Web site

Web site registration is the process of registering a Web site name, which identifies one or more IP addresses with a name that is easier to remember and use in URLs to identify particular Web pages. The person or business that registers Web site name is called the Web site name registrant.

Upload and ready to host Website

The owner can allow posting their web site after them web site getting approval from the admin. and the website visible to all the users.

View blocked user account

In this module, owner can view the blocked account.

Implementation

The security slicer first extracts a security slice for each sink. It then explores the paths in the slice that lead to the sink in a depth-first manner, extracting the path conditions and the context information. The latter is used to generate the attack condition, by conjoining the path condition with the appropriate threat model. For scalability reasons, when encountering loops and recursive function calls, the slices iterate through them only once. The *constraint solver* comprises three modules: constraint preprocessor, an automata-based and interval constraint solver and a search-based constraint solver. The constraint preprocessor makes use of the J GraphT library a Java class library that provides mathematical graph-theory objects and algorithms, in order to generate a constraint network from the attack condition. The constraint network is then passed to the constraint solver to prove the presence/absence of vulnerability. Our automata-based and interval constraint solver handles string and integer constraints with supported operations, as described in. It is built on top of JSA and Sushi. JSA models a set of Java string/mixed operations using finite state automata; Sushi adds supports for string replacement and regular expression replacement operations using finite state automaton and transducer operations. In this component, we also defined the recipes for additional string operations such as the security APIs provided by two popular security libraries (OWASP and Apache). The search-driven constraint solver is invoked when a constraint contains unsupported operations.

We use six different benchmarks, obtained from different sources, to evaluate JOACO: JOACO-Suite, Stranger J-Suite, Pisa-Suite, AppScan-Suite, Kausler-Suite, and Cashew-Suite. JOACO-Suite is our homegrown benchmark, composed of 11 open-source Java Web applications/services and se- curity benchmark applications that have been used in the literature, with known XSS, XMLi, XPathi, LDAPi, and SQLi vulnerabilities. It is an extended version of the benchmark used in our previous work enriched with two new applications: Bodgeit and OMRS-LUI. WebGoat and Bodgeit are deliberately insecure Web applications developed for the purpose of teaching security vulnerabilities in Web applications. Roller and Pebble are blogging applications that also expose Web service APIs. WebGoat, Roller and Pebble have been already used as benchmarks in the vulnerability detection literature. Openmrs module legacyui (shortened as OMRS-LUI) [61] is the user interface pack- age of Open MRS, a widely used, open-source medical record system that manages highly sensitive medical data. Regain is a search engine, known to be used in a production-grade system by one of the biggest drugstore chains in Europe. The pubsubhubbub-java (shortened as PSH) tool [64] is the most popular Java project related to the PubSubHubbub protocol in the Google Code archive. The rest-auth-proxy (shortened as RAP) microservice is one of the most popular LDAP-based Web service Java projects returned by a query on Github.com with the search string ldap rest. TPC-APP, TPC-C, and TPC-W are the standard benchmarks provided by for evaluating vulnerability detection tools for Web services; the set of Web services they provide has been accepted as representative of real environments by the Transactions processing Performance Council, this benchmark contains in total 129 paths to sinks (and as many constraints): 86 paths vulnerable to XMLi, XPathi, XSS, LDAPi, or SQLi, and 43 non-vulnerable ones. Note that a vulnerable path corresponds to a single vulnerability.

StrangerJ-Suite is a security benchmark distilled from five real-world PHP web applications (MyEasyMarket, proMan- ager, PBLguestbook, aphpkb, and BloggIT). It has been used for assessing the effectiveness of the stranger tool in the context of automatically detecting and sanitizing security vulnerabilities in PHP Web applications. We have manually translated every program of this benchmark from PHP to Java so that we could use it in our evaluation. As shown in the bottom part of Table 7, this benchmark contains in total 9 paths which are all vulnerable to XSS. PisaSuite contains 12 constraints generated from sanitizers detected by PISA [68]; these constraints have been used in the experimental evaluation reported in. AppScan-Suite contains 8 constraints derived from the security warnings emitted by IBM Security AppScan, a commercial vulnerability scanner tool, when executing on a set of popular websites.

6. CONCLUSION

This work addresses the challenge of analyzing the source code of a Java Web application for detecting injection vulnerabilities in a scalable and effective way. We have proposed an integrated approach that seamlessly combines

static analysis- based security slicing with hybrid constraint solving, that is constraint solving based on a combination of automata-based solving and meta-heuristic search (Ant Colony Optimization). We use static analysis to extract minimal program slices from Web programs relevant to security and to generate the attack conditions, i.e., conditions necessary for the slices to be vulnerable. We then apply a hybrid constraint solving procedure to determine the satisfiability of attack conditions and thus detect vulnerabilities. This work addresses the challenge of analyzing the source code of a Java Web application for detecting injection vulnerabilities in a scalable and effective way. We have proposed an integrated approach that seamlessly combines static analysis-based security slicing with hybrid constraint solving, that is constraint solving based on a combination of automata-based solving and meta-heuristic search (Ant Colony Optimization). We use static analysis to extract minimal program slices from Web programs relevant to security and to generate the attack conditions, i.e., conditions neces- sary for the slices to be vulnerable. We then apply a hybrid constraint solving procedure to determine the satisfiability of attack conditions and thus detect vulnerabilities.

The experimental results, using a benchmark comprising a set of diverse and representative Web applications/services as well as security benchmark applications, show that our approach (implemented in the *JOACO* tool) is significantly more effective at detecting injection vulnerabilities than state-of-the-art approaches, achieving 98% recall, without producing any false alarm. We also compared the constraint solving module of our approach with state-of-the- art constraint solvers, using six different benchmarks; our approach correctly solved the highest number of constraints (665 out of 672), without producing any incorrect result, and was the one with the least number of time-out/failing cases. In both scenarios, the execution time was practically acceptable, given the offline nature of vulnerability detection. As part of future work, we plan to extend our integrated vulnerability detection approach with support for widely used Java Web frameworks such as Spring. We also plan to incorporate dynamic symbolic execution to further enhance our approach.

7. REFERENCES

[1] Kiezun, P. Guo, K. Jayaraman, and M. Ernst, "Automatic creation of SQL injection and cross-site scripting attacks," in Proceedings of ICSE'09. IEEE, 2009, pp.199–209.

[2] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song, "A symbolic execution framework for JavaScript," in Proceedings of S&P'10. IEEE, 2010, pp. 513–528.

[3] X. Fu, M. Powell, M. Bantegui, and C.-C. Li, "Simple linear string constraints," Form. Asp.Comput., vol. 25, no. 6, pp. 847–891, 2013.

[4] Y. Zheng and X. Zhang, "Path sensitive static analysis of Web applications for remote code execution vulnerability detection," in In Proceedings of ICSE'13. IEEE, 2013, pp. 652–661.

[5] Cadar and K. Sen, "Symbolic execution for software testing: Three decades later," Commun. ACM, vol. 56, no. 2, pp. 82–90, 2013.

[6] J. Thomé, L. K. Shar, D. Bianculli, and L. Briand, "Security slicing for auditing common injection vulnerabilities," *J. Syst. Softw.*, 2017, (in press) https://doi.org/10.1016/j.jss.2017.02.040.

[7] J. Thomé, L. Shar, D. Bianculli, and L. Briand, "Search-driven string constraint solving for vulnerability detection," in *Proceed-ings of ICSE'17*. IEEE, 2017, pp. 198–208.

[8] M. Dorigo and K. Socha, "An introduction to ant colony opti- mization," IRIDIA, Tech. Rep. TR/IRIDIA/2006-010, 2006.

[9] Apache, "StringEscapeUtils," https://commons.apache. org/proper/commons-lang/javadocs/api-3.1/org/apache/ commons/lang3/StringEscapeUtils.html, 2017.

[10] OWASP, "OWASP ESAPI," https://www.owasp.org/index. php/Category:OWASP Enterprise Security API, 2017.

[11] W. Huang, Y. Dong, and A. Milanova, "Type-based taint analysis for Java Web applications," in *Proceedings of FASE'14*. Springer, 2014, pp. 140–154.

[12] P. M. Pérez, J. Filipiak, and J. M. Sierra, "LAPSE+ static analysis security software: Vulnerabilities detection in Java EE applica- tions," in *Proceedings of FutureTech*. Springer, 2011, pp. 148–156.

[13] T. Liang, A. Reynolds, C. Tinelli, C. Barrett, and M. Deters, "A DPLL(T) theory solver for a theory of strings and regular expressions," in *Proceedings of CAV'14*. Springer, 2014, pp. 646–662.

[14] M. Berzish, Y. Zheng, and V. Ganesh, "Z3str3: A string solver with theory-aware branching," *CoRR*, vol. abs/1704.07935, 2017.

[15] J. Thomé, "JOACO: Vulnerability analysis through security slic- ing and hybrid constraint solving,"

https://sites.google.com/ site/joacosite/home, 2017.

[16] OWASP, "OWASP Top 10," https://www.owasp.org/index. php/Category:OWASP Top Ten Project, 2017.

[17] CWE, "Common weakness enumeration," http://cwe.mitre. org/, 2017.

[18] O. Tripp, M. Pistoia, P. Cousot, R. Cousot, and S. Guarnieri, "ANDROMEDA: Accurate and scalable security analysis of Web applications," in *Proceedings of FASE'13*. Springer, 2013, pp. 210– 225.

[19] O. Tripp, M. Pistoia, S. J. Fink, M. Sridharan, and O. Weisman, "TAJ: Effective taint analysis of Web applications," in *Proceedings of PLDI'09*. ACM, 2009, pp. 87–97.

[20] Kiezun A, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst, "HAMPI: A solver for string constraints," in *Proceedings of ISSTA'09*. ACM, 2009, pp. 105–116.

[21] Y. Zheng, V. Ganesh, S. Subramanian, O. Tripp, J. Dolby, and X. Zhang, "Effective search-space pruning for solvers of string equations, regular expressions and length constraints," in *Proceedings of CAV'15*. Springer, 2015, pp. 235–254.

[22] Shannon, S. Hajra, A. Lee, D. Zhan, and S. Khurshid, "Ab- stracting symbolic execution with string analysis," in *Proceedings of TAICPART-MUTATION'07*. IEEE, 2007, pp. 13–22

[23] K. Sen, D. Marinov, and G. Agha, "CUTE: A concolic unit testing engine for C," in *Proceedings of ESEC/FSE'05*. ACM, 2005, pp. 263–272.

[24] S. Horwitz, T. W. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," *Trans. Program. Lang. Syst.*, vol. 12, no. 1, pp. 26–60, 1990.

[25] Qi, H. D. T. Nguyen, and A. Roychoudhury, "Path exploration based on symbolic output," *Trans. Softw. Eng. Methodol.*, vol. 22, no. 4, pp. 32:1–32:41, 2013.