IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727
PP 41-44
www.iosrjournals.org

Taint Analysis and False Positive Prediction for Web Application Vulnerability Deportation

Ameena.S [1], Devi Dath [2]
[1](*Department of Computer Science, College of Engineering Perumon, India*)
[2](*Department of Computer Science, College of Engineering Perumon, India*)

Abstract: *Web applications are increasing day by day and they have a prominent role in the life of a common man. It is required to keep the web application secure from the security loopholes. Web applications that reside in the web server have access to the database for executing the user queries. Along with the increase in security protection alternatives, hackers are trying all the possible ways to breach the security. Hackers are much more interested in attacks like SQLI, (SQL Injection) which can make alterations in the source code and can access the database; it will affect the entire web application security. The various attacks involves XSS (cross-site scripting),SCD(Source Code Disclosure),RFI(Remote File Inclusion),DT-PT(Directory Traversal-Path Traversal),LFI (Local File Inclusion),OSCI (Operating system command injection) and PHPCI (PHP Code Injection).Static analysis plays a great role in detecting and removing these attacks, a large research has been going on that. Static analysis will often result in a great number of false positives. They will adversely affect the accuracy of the system. Our approach is to build a frame work which will detect and remove all the specified vulnerabilities, together with classification algorithms for removing false positives.*
Keywords: *False positives, Sanitization function, Static analysis, Taint analysis, Web Application.*

I. Introduction

Vulnerabilities are often rare and hidden in a source code, thus locating the code for testing and inspection is a challenging task .Often the method used against this is the performance evaluation of code based on metrics that include the categories like code churn complexity and the developer activity. It is difficult to identify which slice will make a source code vulnerable. As a counter against this during the application development itself AJECT system creates a large number of attacks using the communication protocol specification of the server and test case generation algorithms. Based on these attacks the system will observe the execution of the server and the response to the clients [2]. The main factor that is considered for application vulnerability is the information flow policy which is ensured through the static program analysis [3]. Early attacks were mainly aimed at the integrity of data, now attack is possible in every aspect. Static analysis which is a program debugging method without executing the code makes each vulnerable entry as tainted and syntax tree is traced for the conditional branching. WASP, which is conceptually and practically efficient tool, is used to implement this technique [5]. Classification of each attribute to its corresponding vulnerability class is done for future prediction and assessments. Selection of suitable classifier regarding the superiority of one over other is a challenging task [6].SQL injection attacks and cross site scripting (XSS) are two major types of vulnerabilities among the top 10 OWASP attacks. SQLI occurs by injecting some input string result in illegal queries to database. XSS is against the victim's browser itself. Nowadays it's vital for every organization to deploy the strongest possible set of application security to reduce risk of sensitive data. Protection of web application against the attackers is a challenging task.

II. Existing System

A prediction model is followed in which instead of going through all the modules, possible modules that contain error are predicted and the security experts will go through such modules. Identifying such a predictive factor for differentiating vulnerability from fault is cumbersome. Metrices based on CCD is applied on open source projects like Mozilla Firefox browser and Linux kernel [1].

Some attacks are injected against a target system, and any abnormal behavior marks that, attack can trigger an existing flaw. After that, the problem and its location is identified for debugging. For implementing this method a tool named, AJECT is used and the server is the target system. AJCET only requires the specification of the protocol used in the server communication [2].

It is difficult to analyze the confidentiality of the system especially when there is error during the design and implementation. Most of the modern computing system includes some untrusted codes which makes the assurance of confidentiality more difficult. Access control only check the release of information not its

propagation. Confidentiality control restricts the information flow only to where some policies are satisfied. The mechanism used to enforce the confidentiality policy is termed as information flow control. Information flow policy applies the principle of end to end design to the specification of security needs. Annotations that specifying the policy of typed data will be augmented with the program variables and expressions in security typed languages [3].

Conditional branching is an indication of vulnerability and any technique that ignore control flow path is prone to error. Data dependence graphs are used for the conditional branching evaluation but they don't have predicates, thus it makes it difficult to consider input validation code pattern. Thus the system opted for the attributes which are collected from backward static program slices for sanitization code patterns and input validation code patterns. Here it mainly deals with XSS and SQLI attacks in web applications. Backward static slice Sk, according to slicing criterion <k,v> contains all nodes which may affect the value of v at k,v is the variable set used in k. Here they used three classifiers Naïve Bayes (NB), Multi-Layer Perceptron (MLP) and C4.5 for SQLI and XSS. Fivefold cross validation technique is used. The performance is measured using probability of detection (pd), probability of false alarm (pf), precision (pr) and accuracy (acc).The best attributes were measured based on gain ratio [4].

Insufficient validation of input strings results in the SQLI attacks. Attackers include specially encoded commands in the input string and during the execution of query using these strings within the database attack succeeds. Even though Defensive coding practices are employed they are less successful due to the implementation and enforcement difficulty and they can address only a subset of all the possible attacks. Retrofitting existing code is also problematic for legacy software's. Our approach works by allowing only trusted strings and dynamic tainting. Otherwise lot of human effort is required in rewriting the parts of applications. Dynamic tainting is highly automated with minimal intervention. Positive tainting mainly goes for trusted data .Trusted data can be easily and accurately identified compare to the untrusted data sources results in better automation. Failure in detection of untrusted data will result in false negatives while trusted data only results in false positives. Syntax aware evaluation helps the developers to control the string usage based on its source as well as the syntactical role. The web application must be deployed with the metastring library, for applying this approach [5].

For the prediction of software defect, large bench marks of 22 classification models are used. For validating performance variation among the classifiers, hypothesis testing methods were used [6]. AMNESIA consist of three modules analysis, instrumentation and runtime monitoring modules. The insufficient input validation results in SQLIAs even though defensive programming exists; they require a lot of human effort which is impractical in a legacy system. Other analysis can't address most of the specifications in SQLIA [7].

Ranking performed on the basis of vulnerable component helps to identify which component needs further investigation. Mozilla projects maintain such a vulnerability database but unaware of the distribution of vulnerability. The suggested vulnerability tool vulture can mine from the database and can map to a specific component. They considered the two phenomena, the vulnerability itself and the imports that is the range of services. The range of service has a high impact on risk associated with the vulnerability. Here mainly consider the Mozilla projects which is greatly different from JAVA projects. The vulture performed all this automatically and gives an idea to programmer about where past vulnerabilities were located, other components likely to be vulnerable and effective allocating quality assurance effort [8].

In WebSSARI system using 230 open source web application on SourceForge.net resulted in reduction in runtime overhead by 98.4%.Here web application vulnerability is considered as a fall in secure information flow, and used lattice based static analysis algorithm derived from type streams and type state. The use of runtime guards in code, considered to be vulnerable, reduced the user intervention. Here insecure information flow in terms of data integrity violation is considered and non-interference policies are enforced. Noninterference policy is verified and specified through Denning's axioms and Strom's type state [9].

In case of AFEX system there uses a technique and tool for automatically injecting the faults and categorizing the result of test. AFEX is mainly for MySQL, Apache http, UNIX. It will cover more bugs than black box testing. Triggering a fault in a system requires great knowledge in source code and sometimes these codes may not be available. So they may interfere the system with simple faults which is a poor coverage testing. It is replaced by LFI with a wide variety of fine grained faults. With fine grained injection there lies the problem of what to inject, where to inject and when to inject. Only the best choice can result in expected outcome. The hard choice problem can be solved using brute force exhaustive testing. It also has a problem of millions of possible injection for a single fault.

Randomly selected subset is a solution for this, but may result in poor coverage so holding on to both inherent structure oblivious to random and exhaustive testing chosen. AFEX is a cluster based parallel testing system with feedback mechanism for unknown structure. AFEX implemented fitness driven fault exploration algorithm which can explore the fault space of a system and can efficiently categorize the fault into their corresponding classes with less human effort. It succeeded in real system like MySQL and Apache [10].

III. Proposed System

The vulnerability check is performed in the source code and the semantic check for a programing language without making any alteration to source code have importance [1]. Most of the system comes up with the symptoms of fault rather than the cause behind it, if we have to make alterations and correct the source code we need to know the reason behind that.

Taintedness propagation problem is another serious issue[2],taintedness propagation occurs in between the entry point and the sensitive sink presence of any of the attributes or the sanitization functions can make the variable untainted, sometimes it may again alter it back to tainted, so we require a better way to trace out the taintedness propagation. Vulture tool mainly focused on producing a result which can be evaluated by a statistical package or spread sheet [4], system involving programmer interaction can give better result. Static analysis has a problem of false positives, by using the dynamic analysis which can produce zero false positives, but it makes the system much more complex [5].

Our system, Taint analysis and false positive prediction for vulnerability deportation's input will be PHP source code. The source code will be parsed using the lexer and the parser. It will generate the AST tree for the source code. For analyzing the taintedness of each variable in the code we will create TST in which each of its cells is a sub tree of AST. Each cell will contain variable name, its line number in the source code and value of the tainted variable. The TST will be used to analyze the taintedness propagation and if there is a presence of sanitization function then it is not propagated to the variable that depends on it. Together with the TST creation, TEPT table will also be created, which contain the location where the variable get tainted and also about the location to which it is propagated further. Whenever the variable becomes untainted it will be inserted into the untainted table. A symbol from TST which belongs to TEPT but not to UD will be considered for propagation.

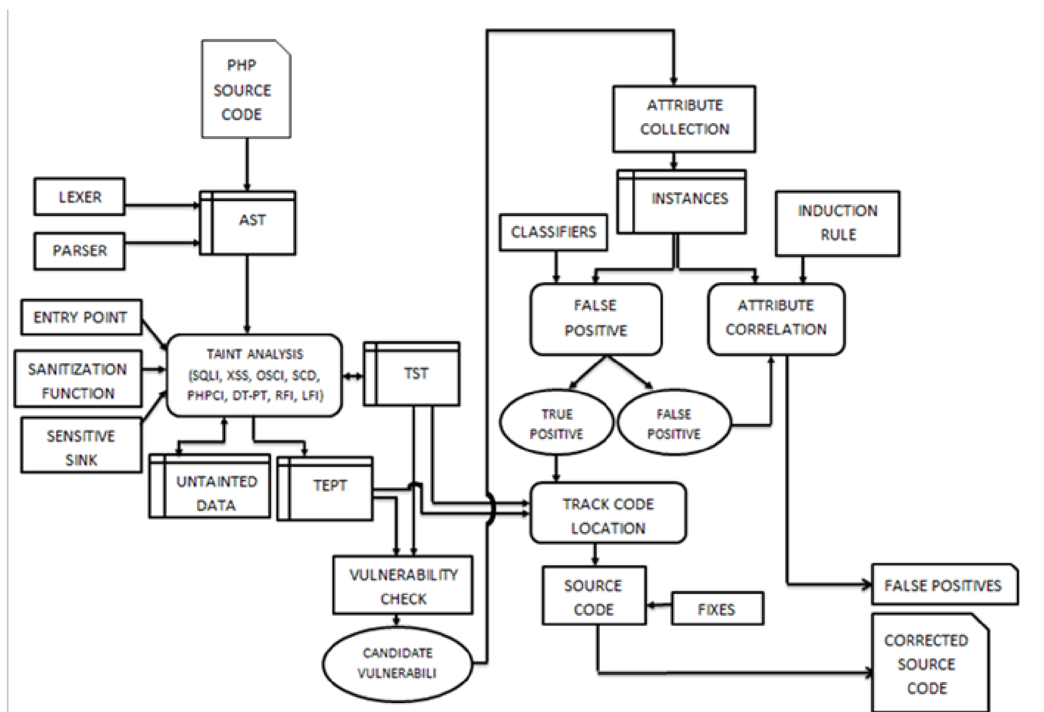

Fig.1 architecture for taint analysis and false positive prediction for web application vulnerability deportation

The slice of vulnerabilities given by the taint analyzer will be considered for false positive prediction. For that different attributes that result in false positive will be identified and classifiers will be used for categorizing each of the vulnerability that is reported as false positive in the known label set and its efficiency is predicted. Top classifiers among these will be selected and will be used for analyzing new dataset for false positives. Induction rule to identify the correlation of the combination of attribute with the false positive is also implemented.

The TEPT(Tainted execution Path table), TST(Tainted Symbol Table) tree are analyzed to point out the location of the real vulnerabilities and the code will be corrected in the corresponding location by keeping the programmer in the loop which will give rise to interaction with the programmer rather than a statistical package. The real vulnerabilities will be removed using the fixes which include the sanitization function that will sanitize the tainted variable.

IV. Conclusion

We presented a system which can reduce the false positives in static analysis. TEPT and TST will help to trace out the taintedness propagation in a better way. Data mining approach for false positive prediction is selected after considering alternative algorithms using the metrices, which will ensure the choice of our algorithm. Rather than a Web application Protection Tool, a framework approach will help programmer to build application and fork code according to their need.

Acknowledgements

We thank members of Computer Science and Information Technology department of College Of Engineering, Perumon for their valuable support and feedback.

References

[1] A. Sabelfeld and A. C. Myers, Language-based information-flow security, *IEEE J. Sel. Areas Commun.,21(1)*,2003, 5–19

[2] Y.-W. Huang et al., Securing web application code by static analysis and runtime protection, *Proc. 13th Int. Conf. World Wide Web*, 2004, 40–52

[3] W. Halfond and A. Orso, AMNESIA: analysis and monitoring for neutralizing SQL-injection attacks, *Proc. 20th IEEE/ACM Int.Conf. Automated Software Engineering*, 2005, 174–183

[4] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, Predicting vulnerable software component, *Proc. 14th ACM Conf. Computer and Communications Security*, 2007, 529–540.

[5] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, *IEEE Trans. Softw. Eng., 34(4)*, 2008, 485–496.

[6] W. Halfond, A. Orso, and P. Manolios, WASP: protecting web applications using positive tainting and syntax aware evaluation, *IEEE Trans. Softw. Eng., 34(1)*, 2008, 65–81.

[7] J. Antunes, N. F. Neves, M. Correia, P. Verissimo, and R. Neves, Vulnerability removal with attack injection, *IEEE Trans. Softw.Eng.,.36(3),*2010, 357–370.

[8] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, Evaluating complexity, code churn, developer activity metrics as indicators of software vulnerabilities, *IEEE Trans. Softw. Eng., 37(6),* 2011, 772–787.

[9] R. Banabic and G. Candea, Fast black-box testing of system recovery code, *Proc. 7th ACM Eur. Conf. Computer Systems*, 2012, 281–294

[10] L. K. Shar et al., Predicting common web application vulnerabilities from input validation and sanitization code patterns, *Proc. 27th IEEE/ACM Int. Conf. Automated Software Engineering,* 2012,310–313.