International Journal of Engineering Research in Computer Science and Engineering Engineering (IJERCSE) Vol 2, Issue 3, March 2015

Finding Security Vulnerabilities in Web-Applications with Static Analysis

[1]S.Senthilkumar,[2]V. Vignesh,[3]M.Sathya Prakash ,[4]K.Rejini
[1][2][3]B.E, IV year students , Department of Computer Science and Engineering
[4]Assistant Professor of Computer Science and Engineering, Anand Institute of Higher Technology, Chennai.

Abstract — **The security of Web applications has become increasingly important in the last decade. The Web based enterprise applications deal with sensitive financial and medical data. Therefore the web applications are created by giving major preference to security since highly confidential data need to be secured and it is also crucial to protect these applications from hacker attacks. A recent study has exploited that attackers has been using two vulnerabilities methods to hack any web application that is SQL INJECTION(SQLi) and XSS CROSS SCRIPTING(XSS).SQL injections are caused by unchecked user input being passed to a back-end database for execution and Cross-site scripting occurs when dynamically generated Web pages display input that has not been properly validated. The existing system finds all vulnerabilities matching a specification in the statically analyzed code. The Results of our static analysis are presented to the user for assessment in an auditing interface and We also propose a static analysis approach based on a scalable and precise points-to analysis. The extensive experimental results are congruent with the theoretical analysis**

Index Terms — **SQL Injection, Cross-Site Scripting, Web Application Security, Detection of security attacks, Application Security Prevention Techniques.**

I. INTRODUCTION

A typical Web application accepts input from the user browser and interacts with a back-end database to serve user requests at same time they are implemented using a number of server-side ex-cutable components, such as CGI programs and HTML-embedded scripting code. The research says that SQL (Structured Query Language) Injection and Cross-Site Scripting (XSS) are the two top most attacks identified to implement the attacks. The users keep on inserting, querying and updating data in these databases. For all these operations a well-designed user interface is very important. Sometimes there are possibilities of existence of unchecked or unused input fields. The attackers exploit through this way to attack a web application. Such an attack may cause serious security violations such as account hijacking and cookie theft. XSS usually affects victim's web browser on the client-side where as SQL injection occurs in server side. These vulnerabilities could be exploited by SQL injection or XSS to gain control over the online web application database. The feasible solution for preventing SQL Injection requires keeping un-trusted data separate from commands and queries. The preferred option is to use a safe API (Application Program Interface) which avoids the use of the interpreter entirely or provides parameterized interface. The solution adapted for XSS CROSS SCRIPTING attack prevention is to keep un-trusted data separate from active

browser content. In existing system the user neither can identify which part of web-application has been attacked nor which method has been used to attack the web-application. But in the proposed system the user themselves can identify where and how the attack has happened. An efficient solution has been identified to prevent attacks and it is enforced to provide security in all environment of web applications. But The current problem lies in the integration of these attack prevention techniques in a practical environment and the developers' familiarities with injection attacks and XSS attacks, and the use of these techniques.

I. RELATED WORK

It is unfeasible to produce complex applications without defects, and even when this occurs, it is impossible to know it, prove it, and repeat it systematically. Software developers cannot assure code scalability and sustainability with quality and security, even when security is defined from the ground up. One

of the aspects that contribute to security problems seems to be related to how bad different programming languages are in terms of propensity for mistakes. Clowes discussed common security problems related to the easiness in programming with PHP and its features, but this affects many other programming languages. The choice of the type system (strong or weak) and the type checking (static or dynamic) of the programming language also affects the robustness of the software. For example, a strong typed language with a static type checking can help deliver a safer application without affecting its performance. Scholte et al. presented an empirical study on a large set of input validation vulnerabilities developed in six programming languages. However, that work focused on the relationship between the specific programming language used and the vulnerabilities that are commonly reported, not going into details in what concerns the typical software faults that originate vulnerabilities, like we do in the present work. One of the best practices to find software faults is to perform a static analysis to the code. This is a labor intensive job, usually done with automated tools, although they lack the precision of the manual counterpart. To improve them and to help predict software failures, a new defect classification scheme was proposed. Another research work proposed a security resources indicator that seems to be strongly correlated with change in vulnerability density over time. Web application vulnerabilities have been addressed by recent studies from several points of view, but without any code analysis. To overcome the low level of detail of existing vulnerability databases, some researchers proposed approaches based on the market, instead of on software engineering. The attacker's perspective has also been of some focus in the literature, but mainly through empirical data gathered by the authors highlighting social networking and what could be obtained from attacking specific vulnerabilities. Some studies analyzed the attacks from the victim's perspective, including the proposal of a taxonomy to classify attacks based on their similarities and the analysis of attack traces from Honey Pots to separate the attack types. There is, however, a lack of knowledge about existing exploits and their correlation with the vulnerabilities. To improve software quality, developers need a deeper knowledge about the software faults that must be mitigated. The underlying idea is that knowing the root cause of software defects helps removing their source, therefore contributing to the quality improvement. Researchers at IBM developed a classification scheme of software faults, intended to improve the software design process and, consequently, reduce the number of faults. It is the ODC and it is typically used to classify software faults or defects after they have been fixed and it is also broadly used by the industry and researchers outside IBM.

A. Vulnerabilities And Programming Languages

The Open Web Application Security Project Report listed the 10 most critical web application security risks, having SQLi at the top, followed by XSS. Other studies also found XSS and SQLi as the most prevalent vulnerabilities. Fig. 1 depicts the yearly percentage of disclosed XSS and SQLi among all the causes of web application vulnerabilities showing that they are increasing over time. SQLi attacks take advantage of unchecked input fields in the web application interface to maliciously tweak the SQL query sent to the back-end database. By exploiting an XSS vulnerability, the attacker is able to inject into web pages unintended client-side script code, usually HTML and JavaScript. SQLi and XSS allow attackers to access unauthorized data (read, insert, change, or delete), gain access to privileged database accounts, impersonate other users (such as the administrator), mimic web applications, deface web pages, view, and manipulate remote files on the server, inject and execute server side programs that allow the creation of botnets controlled by the attacker, and so on. Details on the most common vulnerabilities, including SQLi and XSS, along with the reasons of their existence, attacks,
best practices to avoid, detect, and mitigate them can be found in many referenced works, such as. Many programming languages are currently used to develop web applications. Ranging from proprietary languages (e.g., C#, VB) to open source languages (e.g., PHP, CGI, Perl, Java), the spectrum of languages available for web development is immense. Programming languages can be classified using taxonomies, such as the programming paradigm, the type system, the execution mode, and so on. The type system, particularly important in the context of the present work, specifies how data types and data structures are managed and constructed by the language, namely how the language maps values and expressions into types, how it manipulates these types, and how these types correlate. Regarding the type system, they can be typed versus untyped, static versus dynamic typed, and weak versus strong typed. In particular, strong typed languages provide the means to produce more robust software, since a value of one type cannot be treated as another type (e.g., a string cannot be treated as a

number), as in weak typed languages. One of the contributions of this work is to help understanding the impact of the type system in the security of web applications. This is of particular significance, as critical security vulnerabilities like XSS and SQLi are strongly related to the way the language manages data types. For example, it is common to find attacks that inject SQL code by taking advantage of variables that supposedly should not be strings (e.g., numbers, dates) as the type of the variable is determined based on the assigned value. On the other hand, in strong typed languages, this is not possible because the type of variables is determined before runtime and the attempt to store a string in a variable of another type raises an error. However, this does not prevent the occurrence of vulnerabilities in strong typed languages, but only by taking advantage of string variables. In fact, although Java is intrinsically a safe programming language and it is a strong typed language, vulnerabilities can be found in Java programs due to implementation faults.

B. Results And Discussion Of The Vulnerability Field Study

This section presents and discusses the results of the field study. We used the Pearson product-moment correlation (statistically significant when P < 0:05) to see the strength and direction of the relationship of two variables. A positive correlation (positive r) indicates that when one variable increases so does the other and a negative correlation (negative r) indicates that when one variable increases the other decreases. Strong correlation is when r is between 1 and 0.5; medium correlation when r is between 0.5 and 0.3; weak correlation when r is lower than 0.3. The number of samples is n.

II. SYSTEM REALISATION

A. *Existing Approach*

The security of web applications becomes a major concern and it is receiving more and more attention from governments, corporations, and the research community. Cross-site scripting (XSS) and SQL injection (SQLi), as these are two of the most common and critical vulnerabilities found in web applications. SQL input injection attacks may serve a number of ends. Generally, they are preferred by malicious users as a way to obtain restricted data from a back end database or to embed malicious code onto a web server that will in turn serve up malware to unsuspecting clients.

B.Proposed Approach

In this paper, we propose an analysis of the source code of the scripts used to attack the malicious users. And developers to know about how these vulnerabilities are really exploited by hackers. Its can be used to make software developers and code inspectors in the detection of such faults and are also the foundation for the research of realistic vulnerability and attack injectors that can be used to assess security mechanisms, such as intrusion detection systems, vulnerability scanners and static code analyzers.

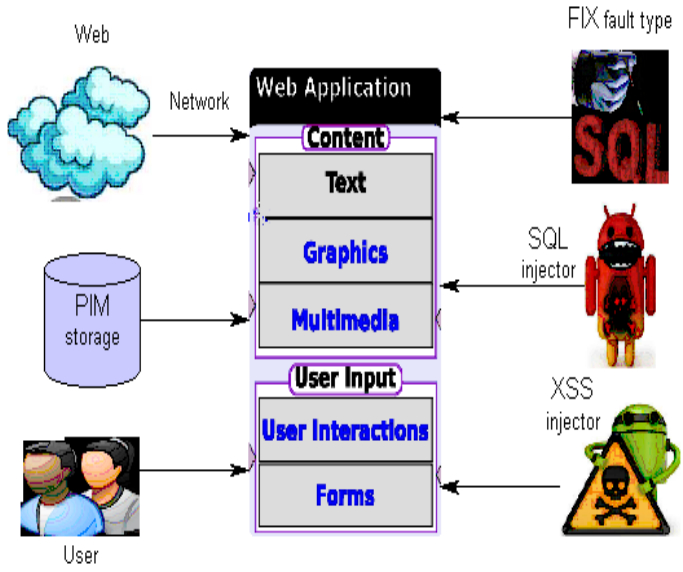

Fig 3.1 Vulnerability Analysis Mechanism

III. IMPLEMENTATION

A. Weblog Construction

Most information systems and business applications built nowadays have a web front end and they need to be universally available to clients, employees, and partners around the world, as the digital economy is becoming more and more prevalent in the global economy. So, when we develop web application,

we consider the security on that business sites. The security of web applications becomes a major concern and it is receiving more and more attention from governments, corporations, and the research community. Here, we develop the organization's site with that secure information, success formulae, account details, partners secure information, employee details, etc... And we want more security on this site.

B. Vulnerability Analysis

The Open Web Application Security Project Report listed the 10 most critical web application security risks, having SQLi at the top, followed by XSS. Other studies also found XSS and SQLi as the most prevalent vulnerabilities on web applications. SQLi attacks take advantage of unchecked input fields in the web application interface to maliciously tweak the SQL query sent to the back-end database. By exploiting XSS vulnerability, the attacker is able to inject into web pages unintended client-side script code, usually HTML and Java script. SQLi and XSS allow attackers to access unauthorized data (read, insert, change, or delete), gain access to privileged database accounts, impersonate other users (such as the administrator), mimic web applications, deface web pages, view, and manipulate remote files on the server, inject and execute server side programs that allow the creation of botnets controlled by the attacker, and so on.

C. Defect Classification

This section presents the methodology to obtain and classify the source code and the security patches of the web applications of our field study. PHP is the most widely used language present in web applications, we used it for the weak typed programming language study. Due to time constraints, other programming languages like PERL could not be considered. Given the high number of security problems found, we only used six web applications: PHP-Nuke (phpnuke.org), Drupal (drupal.org), PHP-Fusion (phpfusion.co.uk), Word Press (wordpress.org), phpMyAdmin (phpmyadmin.net), and phpBB (phpbb.com). For the strong typed programming languages, for which we found less security problems, we used 11 web applications developed in Java, C#, and VB: JForum (jforum.net), OpenCMS (opencms.org), BlojSom (sourceforge.net/projects/blojsom), Roller WebLogger (rollerweblogger.org), JSPWiki (jspwiki.org), SubText (subtextproject.com), Dot-NetNuke (dotnetnuke.com), YetAnotherForum (yetanotherforum.net), BugTracker.NET (ifdefined.com/bugtrackernet.html), Deki Wiki (developer.mindtouch.com), and ScrewTurn Wiki (screwturn.eu).

D. Attack Malicious Injector

We assumed that the information publicly disclosed in specialized sites is accurate and that the fix available by the developer of the web application solves the stated problem. When the patch can fix both XSS and SQLi, the corresponding fault type is counted for both vulnerabilities. To correct a single vulnerability several code changes may be necessary. We consider all the changes as a series of individual fault type fixes, because missing any of them makes the application vulnerable. When a particular code change corrects several vulnerabilities, each vulnerability corrected is counted. When a single vulnerability affects several versions of the application and the patch is the same for all, then it accounts for a single fix.

CONCLUSION

We investigate the vulnerability of the web application and analyze the strength of the programming language used in the web application. Using this analysis technique we can eliminate weak areas of the web application. The major security threats faced in the current scenario are SQL Injection and XSS attack and this technique makes the web application more secured towards those attacks.

FUTURE ENHANCEMENT

The future enhancement of this project will focus on developing better models and on using additional event streams (such as the system calls executed by server-side ex-ecutables) to more completely characterize the behavior of web-based systems.

REFERENCES

[1]Acunetix Ltd., "Is Your Website Hackable? Do a Web Security Audit with Acunetix Web Vulnerability Scanner," http://www.acunetix.com/security-audit/index/, May 2013.

[2]G. Alvarez and S. Petrovic, "A New Taxonomy of Web Attacks Suitable for Efficient Encoding," Computers and Security, vol. 22,no. 5, pp. 435-449, July 2003.

[3]P. Anbalagan and M. Vouk, "Towards a Unifying Approach in Understanding Security Problems," Proc. Int'l Symp. Software Reliability Eng., pp. 136-145, 2009.

[4]A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," IEEE Trans. Dependable and Secure Computing, vol. 1, no. 1, pp. 11-33, Jan.-Mar. 2004.

[5]US-CERT Vulnerability Notes Database, "Homepage," http://www.kb.cert.org/vuls/, May 2013.

[6]R. Chillarege, I.S. Bhandari, J.K. Chaar, M.J. Halliday, D. Moebus, B. Ray, and M. Wong, "Orthogonal Defect Classification—A Concept for In-Process Measurement," IEEE Trans. Software Eng.,vol. 18, no. 11, pp. 943-956, Nov. 1992.

[7]S. Christey, "Unforgivable Vulnerabilities," Proc. Black Hat Briefings, 2007.

[8] J. Christmansson and R. Chillarege, "Generation of an Error Set That Emulates Software Faults," Proc. IEEE Fault Tolerant Computing Symp., pp. 304-313, 1996.

[9] S. Clowes, "A Study in Scarlet, Exploiting Common Vulnerabilities in PHP Applications," http://www.securereality.com.au/studyinscarlet.txt, 2013.

[10] T. Manjaly, "C# Coding Standards and Best Practices," http://www.codeproject.com/KB/cs/c__coding_standards.aspx, May 2013.

[11] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, second ed., Lawrence Erlbaum, 1988.

[12] M. Cukier, R. Berthier, S. Panjwani, and S. Tan, "A Statistical Analysis of Attack Data to Separate Attacks," Proc. Int'l Conf. Dependable Systems and Networks, pp. 383-392, 2006.

[13] A. Adelsbach, D. Alessandri, C. Cachin, S. Creese, Y. Deswarte, K. Kursawe, J.C. Laprie, D. Powell, B. Randell, J. Riordan, P. Ryan,W. Simmonds, R. Stroud, P. Verissimo, M. Waidner, and A. Wespi, "Conceptual Model and Architecture of MAFTIA," Project IST-1999-11583, https://docs.di.fc.ul.pt/jspui/bitstream/10455/2978/1/03-1.pdf, 2003.

[14] Dotnet Spider, "C# Coding Standards and Best Programming Practices," http://www.dotnetspider.com/tutorials/BestPractices.aspx, May 2013.

[15] J. Dura˜es and H. Madeira, "Emulation of Software Faults: A Field Data Study and a Practical Approach," Trans. Software Eng.,vol. 32, pp. 849-867, 2006.

IFERP connecting engineers... developing research