International Journal of Engineering Research (ISSN:2319-6890)(online),2347-5013(print)
Volume No.3, Issue No.2, pp : 126-131 01 Feb. 2014

Web Vulnerability Scanner (WVS): A Tool for detecting Web Application Vulnerabilities

Shivam Swarup, Dr. R.K Kapoor

National Institute of Technical Teacher & Research, Bhopal
shivam.swarup@gmail.com, rkkapoor@nitttrbpl.ac.in

Abstract: In recent years, internet applications have became enormously well-liked, and today they're habitually employed in security-critical environments, like medical, financial, and military systems. Because the use of internet applications has increased, the amount and class of attacks against these applications have also matured. Moreover, the research community primarily targeted on detecting vulnerabilities, which results from insecure information flow in internet applications like cross-site scripting and SQL injection have also increased. Injection Attacks exploit vulnerabilities of websites by inserting and executing malicious code (e.g., information query, JavaScript functions) in unsuspecting users, computing surroundings or on a web server. Such attacks compromise user's information, system resources and cause a significant threat to private and business assets. We tend to investigate and develop a tool Web Vulnerability Scanner (WVS) which queries the vulnerable fragments of applications (written in query and application languages) and are then identified and analyzed offline (statically). Results show the effectiveness of our Tool, compared to the present ones in dimensions alike, it has been observed that vulnerabilities go undetected once the existing ways of area unit used; it makes offline analysis of applications time efficient; and finally, it reduces the runtime observation overhead.

Key Words: web vulnerability, SQL injection, XSS,

1. INTRODUCTION

Today, the web may be an international network infrastructure that covers the complete world and connects many a lot of users. The bulk of the in public offered info on the web is formed offered via the planet Wide internet (WWW) or "the Web". As a result of the simplicity of its use and its high accessibility, the online has become the dominant method for folks to go looking for info, socialize, perform money transactions, etc.

Today, the web may be an international network infrastructure that covers the complete world and connects many users. The bulk of the information offered in public domain on the web is formed by World Wide Web (WWW) or "the Web". As a result of the simplicity of its use and its high accessibility, the Web has become the dominant method for people to go looking for information, to socialize with friends and to perform financial transactions, etc.

The internet applications square measure habitually utilized in security-critical environments, like medical, financial, and military systems. Sadly, because the use of internet applications for crucial services has exaggerated, the amount and class of attacks against internet applications has also grown rapidly. The interaction of code and information from numerous sources within the internet applications and browsers has resulted in the emergence of behaviors that might threaten the confidentiality, integrity, and convenience properties of the infrastructure on that the online applications and the internet browsers. Injection, cross-site scripting (XSS), cross-site request forgery (XSRF), etc. are some examples of vulnerabilities that might permit malicious attacks on internet applications and their infrastructure. Descriptions of those vulnerabilities are often found within the list of ten highest internet application security vulnerabilities printed by Open internet Application Security Project (OWASP) in 2010 [1]. Among all the vulnerabilities listed, OWASP ranks injection vulnerabilities as the most rife.

The OWASP list describes injection vulnerabilities as follows: Injection flaws, like SQL, OS, and LDAP injection, occur once untrusted information is distributed to AN interpreter as a part of a command or question. The attacker's hostile information will trick the interpreter into capital punishment unplanned commands or accessing unauthorized information.

Injection attacks occur once input passes from the browser to the server application, probably even back to the user's browser; and this input contains malicious values/scripts which will alter the behavior of the online application and cause unexpected results. A typical unfortunate attack begins within the client-side browser wherever an internet application is rendered, giving A wrongdoer chance to input malicious information via the browser. This information is then sent to the server, where it's going to reach the back-end information, leading to a SQL Injection attack, or it's going to be sent back to the attacker's client-side browser for execution, leading to a Cross-Site Scripting Attack. These attacks can acquire sensitive information or offer unauthorized access to system resources right away, giving act as an initial Order attack. Second Order attacks will

victimize on AN unsuspecting user once the antecedent hold on malicious information is retrieved and becomes a part of a question or the rendered web content. One extreme step to counter such attack would be to remove any user input to internet applications; such an action is impractical for any internet application that interacts with end-users. An internet application while not taking any input has been restricted in its use, and it also doesn't have the flexibility to question the user as to show any useful information. Thus, there's no access to back-end databases to retrieve personal information (e.g., account balance) neither there is the potential to perform transactions (e.g., on-line bill pay). Therefore, the first step for countering injection attacks involves distinguishing potential malicious inputs and sterilizing them, so that any kind of rendering can be avoided. This can be mentioned as the sanitation methodology.

The rest of the paper is organized as follows. Web vulnerability is discussed in the next section. Section III explains related Work. Our proposed tool (WSV) is described in section IV. Result analysis is explained in Section V. Finally we conclude our paper in Section VI.

2. WEB VULNERABILITIES

SQL Injection: SQL injection attacks occur once the inputs to internet applications attack the back-end layers of the net servers. Typical web application architecture is shown in Figure 1. The internet applications generate websites that are displayed on the internet browser. These applications allow the user and the host machine to interact with each other. Most internet applications permit their users to input information, which further determines the management flow and also the output of the net application. An internet server sends this information to the back-end servers containing database. The back-end servers generates the results and returns them to the net application, which then displays the results to the user's using an application program. A malicious user of the net application will manipulate the inputs in order to make that application vulnerable, thereby generating SQL injection attacks.

We can illustrate SQL injection mistreatment with an example. Think about a simple on-line phone book management application that permits users to look at or modify their phone book entries. Phone book entries are non-public, and are protected by passwords. To check any entry, the user enters his user name and password using HTML scripts provided at the back-end of online application. When the application receives any input the HTML script using some procedure generates a SQL question. Thus, if the user provides input as the string "shivam" for the username and "correct-password" as the password, in the query which would be generated as follows: select * from telephone directory where username='shivam' AND password='correct-password' .Now to attack the online application, a malicious user will input the string "shivam' OR 1=1 - -" for the username, and "not-needed" as the password as inputs, which will create an SQL question as: select * from telephone directory wherever username='shivam' OR 1=1 - - password='not-needed'. The command 'select' contains the tautology 1=1, and uses the "OR" operator, as a result the select condition will evaluate to TRUE. The sub-string "--"will act as the comment operator in SQL, and therefore the portion of the query that checks the password doesn't work. And as a result this query the malicious user will currently read all the phone book entries of all the users. Using similar kind of queries, the offender can initiate attacks that could delete telephone number entries or modify existing entries with spurious values. An application liable to any kind of SQL injection attack is usually exploitable, as the offender can mistreat the information, which he can exploit to enormous levels (for example mistreatment of command-shell scripts in hold-on procedures within the SQL).

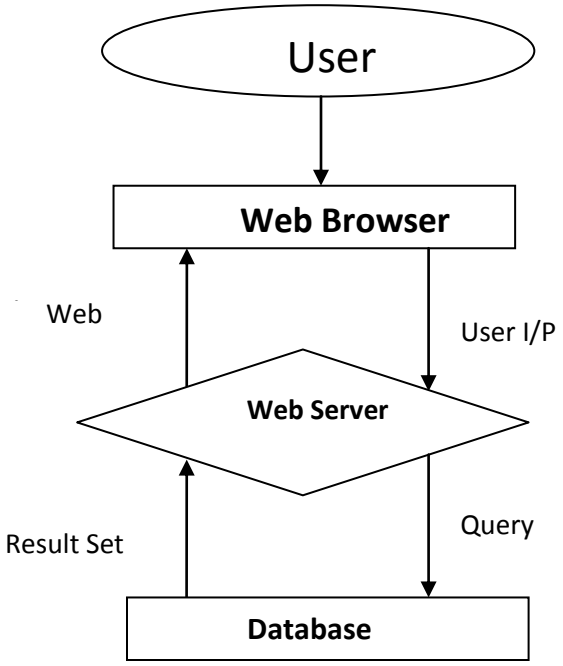

Figure 1: Web application Architecture

Cross-site Scripting (XSS): Vulnerabilities arise from the application's failure to properly validate user input before it comes to the back-end for processing. By taking advantage of this vulnerability, the offender will force a consumer, like a user application, to execute attacker-supplied code, like JavaScript, within the context of a trustworthy computing machine [2]. As a result, the attacker's code is granted access to security-critical data that was issued by (or is associated with) the trusty website. The aim of XSS attack is to bypass the same-origin policy enforced by web browsers while executing the client-side code. This policy doesn't permit scripts and documents loaded from one website to access properties of documents, like cookies, issued by different sites. This prevents malicious net applications from viewing and modifying security-sensitive data related to different sites.

The two main varieties of Cross website Scripting [3] are:

1) Store Cross website Scripting

2) Reflected Cross website Scripting

The stored (or persistent) Cross site Scripting happens once the information provided by the offender is saved by the server, and then displayed as "normal" pages to front-end users. Hold on XSS needs specific reasonable vulnerability within the application wherever the information is placed in somewhere (ex. information base) and later feedback is send to the user, this could be through Forum, Blog, etc. The offender will send or to the application rather than the traditional input to be hold on within the information base, later once the victim involves the computing machine he/she can transfer the or situated there. The application here acts as associate attack website and works according to the offender [4].

Reflected (or non-persistent) Cross website Scripting will occur once the information provided by an internet user, in an ordinary manner using HTTP parameters or using the HTML script, it is employed directly by server-side scripts to get a page of results and then mirrored back to the user, while not taking action on the request [5]. For instance, if we've got a user Log-In prompt (User-Id, Password) and the user has provided his Log-In information as:

User-Id: Shivam

Password: ********

Suppose the user writes his password incorrectly, he should receive a message like ("Sorry, Invalid Log-In"), instead he will get a message like ("Sorry Ahmed, Invalid Log-In") and the (user name) given by the user will be mirrored back to the output. If there's no input validation for Log-In text boxes, the offender will exploit this vulnerability to inject his malicious input 'XSS' rather than User-Id. The offender will create an email containing a link request from the user and asking to update personal information. This is termed as "Phishing", using this offender sends emails with the hope that somebody will click on the link. Once the victim clicks on the link he will provide information to application which will be mirrored back to the victim. The offender may erase the initial Log-In page and place precisely the same original one, once the victim clicks on "Submit" button it'll sends cookie data to the offender.

3. RELATED WORK

Our work is expounded to many areas of active analysis, like explanation and victimization specifications for bug finding, vulnerability analysis, and attack detection for net applications. During this section, we have a tendency to describe the foremost seminal and fascinating works in these areas with the intent of highlight the main trends and achievements in these areas of analysis, and to place the work represented during this paper within the context of previous work.

Paper [3] describes associate rule that aims within the detection of security vulnerabilities. The prompt rule performs a scanning method for all website/application files. Our scanner tool depends on finding out the ASCII text file of the appliance counting on ASP.NET files and therefore the code behind files (Visual Basic VB and C sharp C#). A program written for this purpose is to come up with a report that describes most leaks and vulnerabilities varieties (by mentioning the file name, leak description and its location). The prompt rule can facilitate organization to repair the vulnerabilities and improve the general security.

A tool named WebSSARI [6], revealed in 2004, and is one in every of the primary works that applies static taint propagation analysis to finding security vulnerabilities in PHP applications. WebSSARI targets 3 specific varieties of vulnerabilities: cross-site scripting, SQL injection, and general script injection. The tool uses flow-sensitive, intra-procedural analysis supported a lattice model and sort state. Above all, the PHP language is extended with 2 type-qualifiers, specifically tainted and unblemished, and therefore the tool keeps track of the type-state of variables. The tool uses 3 user-provided files, known as prelude files: a file with preconditions to any or all sensitive functions (i.e., the sinks), a file with post conditions for acknowledged cleansing functions, and a file specifying all attainable sources of untrusted input. So as to undamaged the contaminated information, the information needs to be processed by a cleansing routine or to be forged to a secure kind. Once the tool determines that tainted information reaches sensitive functions, it mechanically inserts runtime guards, that area unit cleansing routines.

Another approach conjointly supported static taint propagation analysis, to the detection of input validation vulnerabilities in PHP applications is delineated in [7,8]. A flow sensitive, inter-procedural and context-sensitive information flow analysis is employed to spot intra-module XSS and SQL injection vulnerabilities. The approach is enforced in a very tool, known as Pixy that is that the most complete static PHP analyzer in terms of the PHP options shapely. To the simplest of our information, it's the sole publicly-available tool for the analysis of PHP-based applications

The work by Xie and author [10], revealed in 2006, describes a three-level approach to search out SQL injection vulnerabilities in PHP applications. First, symbolic execution is employed to model the impact of statements within the fundamental blocks of intra-procedural management Flow Graphs (CFGs). Then, the ensuing block outline is employed for intra procedural analysis, wherever a typical reachability analysis is employed to get a perform outline. In conjunction with alternative data, every block outline contains a group of locations that were

unblemished within the given block. The block summaries area unit composed to come up with a perform outline, that contains

the pre- and post-conditions of perform. The preconditions for perform contain a derived set of memory locations that have to be compelled to be alter before the perform invocation, whereas the post conditions contain the set of parameters and international variables that area unit alter within perform. To model the results of cleansing routines, the approach uses a programmer-provided set of attainable cleansing routines, considers sure types of casting as a cleansing method, and, additionally, it keeps an information of sanitizing regular expressions, whose effects area unit specific by the technologist. Once perform summaries area unit computed, they're utilized in inter-procedural analysis to look for attainable SQL injections.

The work by Su associated Wassermann [9] is another example of an approach that uses a model of "normality" to find injection attacks, like XSS, XPath injection, and shell injection attacks. However, this implementation, known as SqlCheck is meant to find SQL injection attacks solely. The approach works by trailing substrings from user input through the program execution. The trailing is enforced by augmenting the string with special characters, which mark the beginning and therefore the finish of every substring. Then, dynamically-generated queries area unit intercepted and checked by a changed SQL programmed. Mistreatment the meta-information provided by the substring markers, the program is ready to work out if the question syntax is changed by the substring derived from user input, and, therein case, it blocks the question.

4. PROPOSED TOOL

In this section we tend to explain the small print relating to our tool. However before discussing concerning the tool it's necessary to outline some terms. Every web-based application is developed to use by some users. Hence, from currently forward by the term "user" means that the user of the net primarily based application. On the opposite hand, the projected tool Web Vulnerability Scanner (WVS) is additionally developed for a few users (normally for the lead developers who do final code verification of Associate in an application) and those we mention them as "tool-user". Additionally we tend to decision every doable SQLI attack as "threat". A threat in line n of file index.php means, line range n of the file index.php executes Associate in Nursing SQL statement and it's going to not be safe. Note that, threat doesn't mean that the SQL statement is often unsafe, it solely tells that there could also be a clear stage of attack. The detail verification has got to be done by the tool-user manually. Later during this section we tend to explained as why it's necessary to verify every threat manually.

The tool is developed in JAVA and works in any surroundings. The fundamental practicality of the tool is:

- It scans every get into the appliance one-by-one to examine for any vulnerable SQL executions.
- Display all such SQL statements (threats) in user friendly interface with the file name, line range and additionally with the reason for vulnerability.
- It additionally advises the tool-users concerning what to try and do for every threat.
- Once the user resolves a problem (regarding threat) he will mark the threat as resolved so it won't show in next run.
- It is often simply upgraded to satisfy the stress of speedy technology changes.

Working of tool

In this section we tend to discuss concerning how the WVS works. The tool works for PHP, JSP and ASP primarily based applications; however the essential mechanism for every type of application is same.

1) Attack detection mechanism: when obtaining associate degree application folder (say AppFold) as input the tool starts process the folder. Initial of all it extracts all the files within the folder and in its sub-folders. There's an algorithmic operate written to extract all the files from the folder. All the extracted files are kept in associate degree ArrayList (a system outlined in Java, see the Java documentation for details). When obtaining the list of all files, the tool searches every file one-by-one to gather all the variables used. The variables will be of 3 types:

- Dynamic: variables that stores data entered by the user of the applying. For instance, in PHP associate degree expression like $name=$ POST ["name"]; suggests that $name may be a variable that stores the info inserted by user via associate degree hypertext mark-up language kind.
- Composite: variables whose price depends on alternative variables. For instance, in JSP associate degree expression var1=var2+var3; suggests that var1 is rely upon var2 and var3. Therefore var1 is of a composite kind.
- Static: variable that contains a tough coded price while not betting on the other variables.

The tool searches all the files associate degreed collects all the variables and keep in an ArrayList (say varList). The variable looking out procedure is predicated on regular expression written on a configuration file. Note that, the syntax of variable declaration and variable assignment in PHP, JSP and ASP are completely different completely and thus we've to jot down different regular expression rules for every language. Every entry in varList represents one variable and also the structure of every entry is:

- Variable: The name of the variable. For instance, var1, $name etc.
- Value: Lexical price allotted to the variable. Note that, the value suggests that the text value not the particular value of the variable. Actual price differs supported the sort of the variable.

- File name: The name of the file.
- Line range: the road number of the variable within the file.
- Variable type: Either dynamic, setter static.

Once the variable got extracted future job is to seek out all the SQL statements that are known as from the applying. We tend to hump with the assistance of some regular expressions same we tend to do for variable looking out. Every extracted SQL statement will be in:

- Static: Depends on no alternative variables. For instance,"select name, roll from students" may be a SQL statement statically written and not depends on the other variable.
- Composite: Depends on some variables. Those variables will be static, dynamic or composite in nature.

All extracted SQL statements are kept in associate degree ArrayList (say SQLList). The structure of every entry in SQLLIst is:

- Statement: The SQL statement.
- File name: Name of the file wherever the statement resides.
- Line range: Line number within the file wherever the statement is.

The next task is to verify the SQL statements. We tend to think about a SQL statement as safe if it's no dynamic data needed. In alternative words, the statements that doesn't rely upon the values entered by the users. Such sort of safe statements will be of 2 sorts.

- Direct-static: suggests that a static statement as delineate on top of.
- Indirect-static: Circuitously static however none of its relying variables are connected any dynamic variables.

The unsafe statements have risk to rely upon dynamic variables. We've written an algorithmic operate to visualize every SQL statement. The statement declares safe if all its relying variables (not simply direct dependents, however all the hierarchic dependence) are eventually connected with static variables. Otherwise, a threat generates with the list of all dynamic variables which will be connected with this statement. All such threats are keep in Array List (say threat List). Once the threat detection part is over we tend to show the content of threat List with associate degree interactive GUI (java JTable). Additionally we will mark every threat as verified when confirming it as safe. The marked threats won't be visible for future run till we tend to forcefully wish to try and do therefore.

2) The way to ensure validity: The validity confirmation is left as manual. Supported the knowledge provided by SQL-TOOL, the tool-user has gone to the precise location of the statement and ensures that the statement has been handled properly. Every dynamic variable connected with the statement should tolerate a validation method before victimization into the statement. Since the validation rule and also the validation procedure changes from application to application, it's tasking to propose a universal rule for such validations. There for, we tend to left it on the tool-user.

5. RESULT ANALYSIS

We have experimented WVS on all kind of applications. Table I gives the list of applications on which we have tested our tool. Experiments found that our proposed tool saves 40-50% of time as compared to manual verification with the help of some text processing commands like grep. Figure 2 shows a look of our proposed tool.

Application Name	Technology	Area	Description
project-uploader	PHP	Academic	Online project management for university
hospital-management	PHP	medical	Online hospital management application
employee-sys	JSP	corporate	Online employee portal
EX-MNGR	ASP	academic	Online portal for examination management

Table I: Details of The Applications Used As Input For WVS.

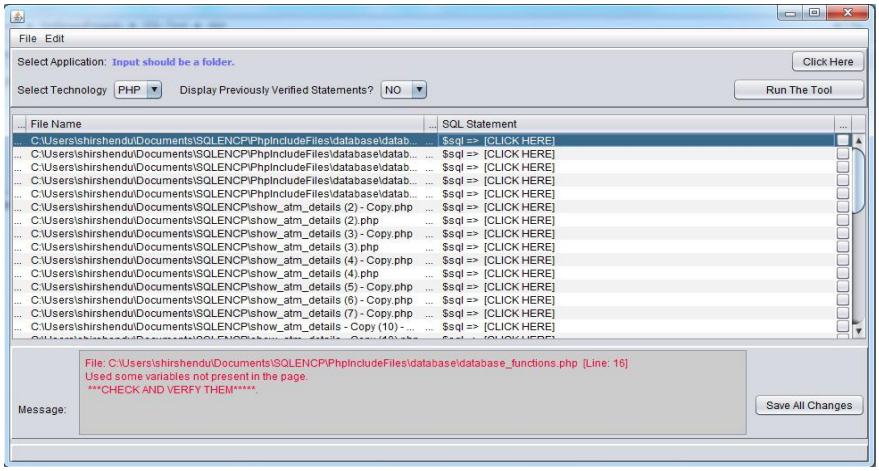

Figure 2: Screen shot for WVS

6. CONCLUSION

In recent years, internet applications have become hugely ubiquitous, and these days they're habitually utilized in numerous security-critical environments. Because the use of internet applications for essential services has accumulated, the amount and class of attacks against these applications have full-grown. Moreover, the analysis communities primarily targeted on effort vulnerabilities that result from insecure information flow in internet applications, like cross-site scripting and SQL injection. Whereas relative success was reached in characteristic appropriate techniques and approaches for managing this kind of vulnerabilities, very little has been explored regarding vulnerabilities that result from blemished application logic.

The work bestowed during this paper represents the primary step toward higher understanding of logic vulnerabilities and finding appropriate techniques for the error detection. Results show the effectiveness of our Tool, compared to the present ones in dimensions alike, it has been observed that vulnerabilities go undetected once the existing ways of area unit used; it makes offline analysis of applications time efficient; and finally, it reduces the runtime observation overhead. Also, the projected approaches have variety of limitations (such as scalability), which require to be self-addressed before they'll be effectively applied to real-world applications. Thus, there still stay additional queries than answers on however logic vulnerabilities are often detected mechanically, effectively, and expeditiously. We tend to believe that there three main directions for future research: 1) achieving a better understanding and etymologizing more precise characterization metrics for logic vulnerabilities, 2) finding appropriate vulnerability detection techniques, and 3) developing ascendible analysis techniques.

References

i. *The Open Web Application Security Project (OWASP), "OWASP top 10 web application security risks in year 2010," https://www.owasp.org/index. php/Top 10 2010-Main.*

ii. *A. Klein. "Cross Site Scripting Explained" Technical report, Sanctum Inc., June 2002.*

iii. *Al-Amro, Huyam, and Eyas El-Qawasmeh. "Discovering security vulnerabilities and leaks in ASP. NET websites." In Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on, pp. 329-333. IEEE, 2012.*

iv. *Safelight of security advisors, "Cross Site Scripting (Stored XSS) demo." [Online] http://www.youtube.com/watch?v=7MR6U2i5iI. Jan 2009.*

v. *Safelight of securityadvisors, "Cross Site Scripting (Reflected XSS) demo." [Online] http://www.youtube.com/watch?v=V79Dp7i4LRM. Jan 2009.*

vi. *Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D. Lee, and S.-Y. Kuo. Securing Web Application Code by Static Analysis and Runtime Protection. In Proceedings of the 12th International World Wide Web Conference (WWW'04), pages 40–52, May 2004.*

vii. *N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities. In Proceedings of the IEEE Symposium on Security and Privacy, May 2006.*

viii. *N. Jovanovic, C. Kruegel, and E. Kirda. Precise Alias Analysis for Static Detection of Web Application Vulnerabilities. In Proceedings of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS'06), June 2006.*

ix. *Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications. In Proceedings of the 33rd Annual Symposium on Principles of Programming Languages (POPL'06), pages 372–382, 2006.*

x. *Y. Xie and A. Aiken. Static Detection of Security Vulnerabilities in Scripting Languages. In Proceedings of the 15th USENIX Security Symposium (USENIX'06), August 2006.*