Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.7, No.7, 2016
www.iiste.org

SQLSCAN: A Framework to Check Web Application Vulnerability

Narottam Chaubey[1] Sumit Sharma[2]
1.PG scholar, CSE, VITS, Bhopal, INDIA
2.HOD, CSE department , VITS, Bhopal, INDIA

Abstract
Security vulnerabilities in web applications that are being found today are much higher than in any operating systems. So it clearly means that threats intended at web applications are utilizing vulnerabilities at the application. Simultaneously, amount and impact of security vulnerabilities on web applications has increases as well. Almost in all online transactions user access is authorized before providing access to database of application. But organized injection could provide entry to unauthorized users and it almost achieved via SQL injection and Cross-site scripting (XSS). In this article we provide a web vulnerability scanning and analyzing tool of various kinds of SQL injection and Cross Site Scripting (XSS) attacks named as SQLSCAN. Our proposed method will work with web application developed on any technology like PHP, JAVA, ASP .NET. We evaluate our proposed scanner by experiments to calculate its performance. We also evaluate the performance of SQLSCAN with performance of parallel tools in the literature.
Keywords: Web Application security, Attack, Injection, SQL, XSS, Vulnerability, Scanner.

1. INTRODUCTION

Web applications have become an integral part of recent industry and our lives. Much of today's online business, including university account management, social networking, email, banking, and shopping, are available online with use of web applications. Since ecommerce has grown significantly, there has been an exponential increase in online transactions in the past few years. The data that web applications handle, such as credit card numbers and shopping activity information, typically is of considerable value to the users and the service providers. In order to be sustainable, web applications should protect the user's data from unauthorized access, use, disclosure, disruption, modification, perusal, inspection, and recording or destruction. But often, it fails to satisfy these requirements. The root cause of most security risks on the Web is based on vulnerabilities in web applications [1], [2].

Modem web applications deliver a wide range of useful Internet services. On the other hand, web applications are routinely exploited by cyber attackers [4]. Web-based cyber attacks are typically classified into two categories: attacks targeting web servers and attacks targeting web clients. Attacks against web servers often take the form of SQL injections [3], cross site scripting attacks (XSS), and remote file inclusion attacks (RFI). Common attacks against clients include hijacking of cookies, injecting malicious codes into web-browsers, click-jacking, and many others [3].

The authors in The Web Application Hacker's Handbook [5] evaluated the security of hundreds of server-side and client-side web applications. They found that 62% of the applications had authentication vulnerabilities, 71% had broken access control mechanisms, 33% contained SQL injection vulnerabilities, 94% had cross-site scripting vulnerabilities, 78% had information leakage, and 92% had Cross-site request forgery vulnerability. In view of these alarming trends, there is an urgent need for techniques for automatically detecting security vulnerabilities in web applications. This research is a step in that direction.

In era of computer security, vulnerability is a weakness that permits a hacker to decrease a system's information guarantee. Vulnerability is the merger of three components: first a system susceptibility or mistake, second attacker entrance to the mistake, and third attacker potential to utilize the mistake [6]. In order to utilize vulnerabilities, the attackers have to utilize least one appropriate method or tool that can connect to a system weakness security pitfall.

Web application vulnerability scanners are preferred by many security auditors and web application administrators because they are typically very easy to use, perform tests relatively quickly, and identify a wide variety of web application vulnerabilities. However, these tools lack the ability to detect many types of vulnerabilities that exist today because the level of artificial-intelligence currently implemented in their scanning engines is far below where it needs to be. Every web application scanner is faced with the same set of problems, these being that every web application is different, web application scanners operate based off of syntax, web application scanners do not improvise, and web application scanners are not intuitive [5]. Because of these problems, web application scanners are plagued with numerous false-positives and false-negatives. This research develops a secure and insecure version of a web application, so that a method to analyze the limitations of web application scanners can be utilized. Solutions to the problems discovered from this analysis can then be proposed in order to improve the techniques used by web application scanners to detect hard-to-find vulnerabilities.

2. WEB VULNERABILITY ATTACK THREATS

The Open Web Application Security Project (OWASP) has place together what is measured as the perfect standard list of top threats to Web applications. It is called ―The OWASP Top 10 Project and it signifies a common harmony on the major areas of threat by category. The Top 10 threats [Fig. 1] as they exist currently are as follows:

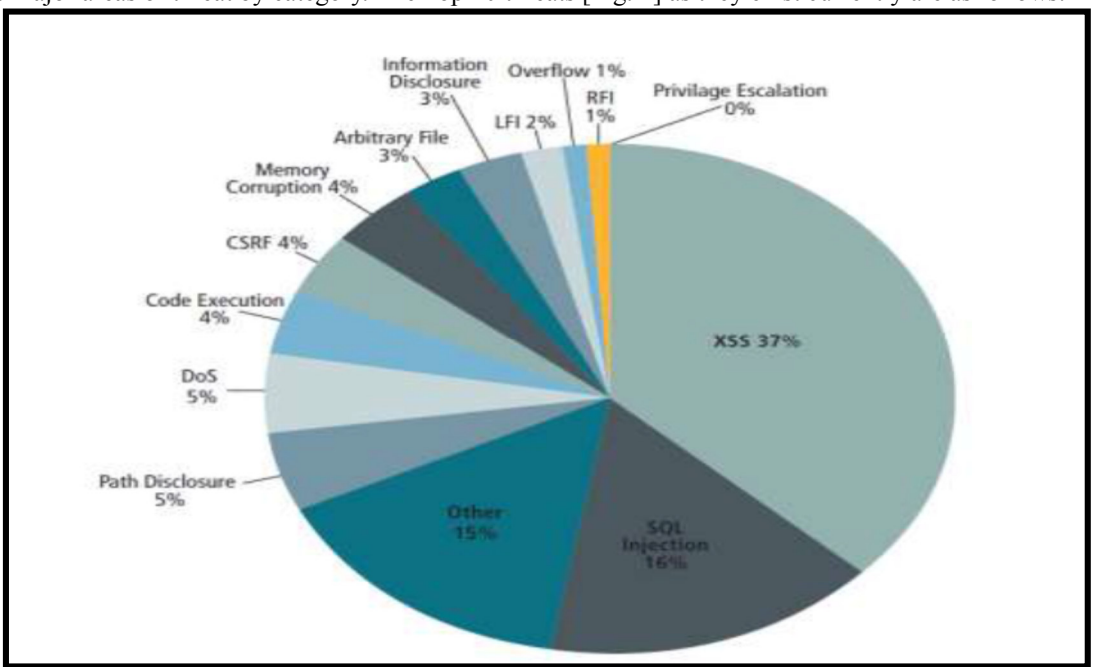

Fig.1: Top 10 Threats

- Injection (SQL, LDAP etc.)
- Cross Site Scripting (XSS)
- Broken Authentication and Session Management
- Insecure Direct Object References
- Cross-Site Request Forgery (CSRF)
- Security Misconfiguration
- Insecure Cryptographic Storage:
- Failure to Restrict URL Access:
- Insufficient Transport Layer Protection:
- Invalidated Redirects and Forwards

3. WEB APPLICATION VULNERABILITY SCANNERS

Presently number of web application vulnerability scanners that test for popular vulnerabilities in web servers and web applications. These tools can either be academic research projects, free/open-source applications, or commercial software products.

3.1 Web Application Scanners in Academia

Huang et al. developed a web application scanner called WAVES that attempts to reduce the number of potential side effects of black-box testing [8, 9]. The reviewing process of web application scanners can be reason of permanent alterations, or even harm, to the state of the application it is targeting. Their experimental results found that WAVES was unable to detect any new vulnerability that was not already detected by a static source code analyzer they had developed. Also, WAVES was unable to discover all of the vulnerabilities that the static source code analyzer had found (detected only 80% of the vulnerabilities found by the static analyzer). The authors believe their tool failed in part because it did not have complex procedures able to detect all data entry points, and because it was unable to observe HTML output.

Another academic black-box approach was developed by Antunes and Viera as described in [10]. Their web vulnerability scanner was used to identify SQL injection vulnerabilities in 262 publicly available web services. The first step in their approach was to prepare for the tests by obtaining information regarding the web service in order to generate the workload (valid web service calls). The second step was to execute the tests. This was accomplished by using a workload emulator that acted as a web service consumer, and by using an attack load generator that automatically generated attacks by injecting them into the workload test calls. The final step in their approach was to analyze the responses by using a set of well-defined rules which would identify vulnerabilities

and exclude potential false-positives. Their results showed that they achieved a detection coverage rate of 81% in the scenario where they had access to the known number of vulnerabilities, and maintained a false-positive rate of 18% in their optimistic interpretation. These results are better than those of the commercial tools that the authors analyzed, and suggest that it is possible to improve the effectiveness of vulnerability scanners [7].

3.2 Free/Open-Source Web Application Scanners

Many open-source and free web application scanners are available for black box testing and analysis. Some of these applications provide extensive functionality with the ability to be customized and expanded to meet the needs of users. Others however do not provide a great deal of usability and have a limited amount of functionality, and therefore can only test for a few web application vulnerabilities. Three of the more thorough and robust free/open source scanners, Grendel-Scan [11], Wapiti [12], and W3AF [13], is reviewed here.

Grendel-Scan [11] is an open-source web application security testing tool which has an automated testing module for detecting common web application vulnerabilities. It has the ability to find simple web application vulnerabilities, but its designers state that no automated tool can identify complicated vulnerabilities, such as logic and design flaws. Grendel-Scan tests for SQL injection, XSS attacks, and session management vulnerabilities, as well as other vulnerabilities.

Wapiti [12] is a free web application vulnerability scanner and security auditor. It performs black-box analysis by scanning the web pages of a web application in search of scripts and forms where data can be injected. After the list of scripts and forms is gathered, Wapiti injects payloads to test if the scripts are vulnerable. Wapiti scans for remote file inclusion errors, SQL and database injections, XSS injections, and other vulnerabilities.

W3AF [13] is exactly what it stands for, a Web Application Attack and Audit Framework. The goal of the project is to create a framework which can find and exploit web application vulnerabilities easily. The project's long term objectives are for it to become the best open source web application scanner, and the best open source web application exploitation framework. Also, the designers want the project to create the biggest community of web application hackers, combine static code analysis and black box testing into one framework, and become the NMAP [14] of the web. W3AF incorporates a great deal of plug-ins into its framework, and is capable of testing for SQL injection, XSS attacks, buffer overflow, malicious file execution, and session management vulnerabilities.

3.3 Commercial Web Application Scanners

Commercial web application scanners are generally licensed to companies or organizations that wish to test their web applications for vulnerabilities so that they can fix security holes before they are maliciously exploited. Some of the features of popular commercial web application scanners is discussed below

Cenzic [58] sells a web application scanner tool called Hailstorm which utilizes stateful testing. Stateful testing tools are designed to behave like human testers by taking what seem to be an application's insignificant or disparate weaknesses, and combining them together into serious exploits. The key benefits that Hailstorm claims are the ability to identify major security flaws in target applications, to help with internal compliance policies, to avoid vulnerabilities that lead to downtime, and to assess applications for commonly known vulnerabilities. Cenzic provides a 7-day free trial of Hailstorm Core which can detect vulnerabilities including SQL injection, XSS , and session management.

Acunetix Web Vulnerability Scanner [8] is another black-box tool which claims in-depth checking for SQL injection, XSS, and other vulnerabilities with its innovative AcuSensor Technology. This technology is supposed to quickly find vulnerabilities with a low number of false-positives, pinpoint where each vulnerabilities exists in the code, and report the debug information as well. Acunetix also includes advanced tools to allow penetration testers to fine tune web application security tests, and has many more features to scan websites with different scan options and identities. The only vulnerability that the free edition of the software detects is XSS, but a 30-day trial version of the product is available that also can detect SQL injection, file execution, session management, and manual buffer overflow attacks.

N-Stalker [59] provides a suite of web security assessment checks to enhance the overall security of web applications. It is founded on the technology of Component-oriented Web Application Security Scanning, and allows users to create their own assessment policies and requirements, enabling them to check for more than 39,000 signatures and infrastructure security checks. Vulnerabilities checked for include SQL injection, XSS attacks, buffer overflows, and session management attacks, but the evaluation edition only lasts for a 7-day period.

Netsparker [60] is a web application vulnerability scanner developed by Mavituna Security Ltd. Netsparker is focused on eliminating false-positives, and uses confirmation and exploitation engines to ensure that false-positives are not reported. The engines also allow the users to see the actual impact of the attacks instead of text explanations of what the attack could do. Because of the techniques Netsparker uses, Mavituna Security claims that it developed the first false-positive free web application scanner. Netsparker scans for all types of XSS injection, SQL injection, malicious file execution, and session management vulnerabilities.

Burp Scanner [61] is a web application vulnerability scanner that is part of Burp Suite Professional. Burp

Suite Professional is the commercial version of Burp Suite, which is an integrated platform for attacking and testing web applications. Burp Suite provides a number of tools, including an interception web proxy, web spider, application intruder, session key analyzer, and data comparer. The professional version includes Burp Scanner which can operate in either passive or active mode, or either manual scan or live scan mode. The vulnerabilities it searches for include SQL injection, XSS injection, and session management vulnerabilities.

Rational AppScan [62] is licensed by IBM for advanced web application security scanning. The AppScan tool automates vulnerability assessments and tests for SQL injection, XSS attacks, buffer overflows, and other common web application vulnerabilities. AppScan can generate advanced remediation capabilities in order to ease vulnerability remediation, simplify results with the Results Expert wizard, and test for emerging web technologies. Rational AppScan provides an unlimited evaluation period for its standard edition; however, with the evaluation license the software is only capable of testing a test web site provided by AppScan.

4. RELATED WORK

According to Curphey and Araujo [15], there are eight categories of web application security Assessment tools: source code analyzers, web application (black box) scanners, database scanners, binary analysis tools, runtime analysis tools, configuration management tools, HTTP proxies, and miscellaneous tools. The most common of these web application assessment tools are source code analyzers and web application scanners. Source code analyzers generally achieve good vulnerability detection rates, but are only useful if the web application's source code is available. There are two main approaches to test web applications for vulnerabilities [16]:

- White box testing: consists of the analysis of the source code of the web application. This can be done manually or by using code analysis tools like Ounce or Pixy. The problem is that exhaustive source code analysis may be difficult and cannot find all security flaws because of the complexity of the code.
- Black box testing: consists in the analyses of the execution of the application in search for vulnerabilities. In this approach, also known as penetration testing, the scanner does not know the internals of the web application and it uses fuzzing techniques over the web HTTP requests.

AMNESIA is a model-based method that merges static analysis and runtime observing [17]. In static stage, AMNESIA uses static analysis to construct models of the different types of queries an application can legally produce at each point of entrance to the database. In dynamic stage, AMNESIA seizes all queries before they are sent to the database and verifies each query against the statically built models. Queries that breach the model are recognized as SQLIAs and prevented from executing on the database. In their evaluation, the authors have shown that this technique performs well against SQLIAs. The primary limitation of this technique is that its success is dependent on the accuracy of its static analysis for building query models. Certain types of code obfuscation or query development techniques could make this step less precise and result in both false positives and false negatives.

SQLGuard [18] and SQLCheck [19] also check queries at runtime to see if they conform to a model of expected queries. In these approaches, the model is expressed as a grammar that only accepts legal queries. In SQLGuard, the model is deduced at runtime by examining the structure of the query before and after the addition of user-input. In SQLCheck, the model is specified independently by the developer. Both approaches use a secret key to delimit user input during parsing by the runtime checker, so security of the approach is dependent on attackers not being able to discover the key. Additionally, the use of these two approaches requires the developer to either rewrite code to use a special intermediate library or manually insert special markers into the code where user input is added to a dynamically generated query.

WebSSARI detects input-validation related errors using information flow analysis [20]. In this approach, static analysis is used to check taint flows against preconditions for sensitive functions. The analysis detects the points in which preconditions have not been met and can suggest filters and sanitization functions that can be automatically added to the application to satisfy these preconditions. The WebSSARI system works by considering as sanitized input that has passed through a predefined set of filters. In their evaluation, the authors were able to detect security vulnerabilities in a range of existing applications. The primary drawbacks of this technique are that it assumes that adequate preconditions for sensitive functions can be accurately expressed using their typing system and that having input passing through certain types of filters is sufficient to consider it not tainted. For many types of functions and applications, this assumption is too strong.

Livshits and Lam [21] use static analysis techniques to detect vulnerabilities in software. The basic approach is to use information flow techniques to detect when tainted input has been used to construct an SQL query. These queries are then flagged as SQLIA vulnerabilities. The authors demonstrate the viability of their technique by using this approach to find security vulnerabilities in a benchmark suite. The primary limitation of this approach is that it can detect only known patterns of SQLIAs and, because it uses a conservative analysis and has limited support for untinting operations, can generate a relatively high amount of false positives.

Huang and colleagues [22] propose WAVES, a black-box technique for testing Web applications for SQL injection vulnerabilities. The technique uses a Web crawler to identify all points in a Web application that can be

used to inject SQLIAs. It then builds attacks that target such points based on a specified list of patterns and attack techniques. WAVES then monitors the application's response to the attacks and uses machine learning techniques to improve its attack methodology. This technique improves over most penetration-testing techniques by using machine learning approaches to guide its testing. However, like all black-box and penetration testing techniques, it cannot provide guarantees of completeness.

Fu et al. suggested a Static Analysis approach to detect SQL Injection Vulnerabilities. The main aim of SAFELI approach is to identify the SQL Injection attacks during compile-time. It has a couple of advantages. First, it performs a White-box Static Analysis and second, it uses a Hybrid-Constraint Solver. On one hand where the given approach considers the byte-code and deals mainly with strings in case of White-box Static Analysis, on the other through Hybrid-Constraint Solver, the method implements an efficient string analysis tool which is able to deal with Boolean, integer and string variables. Its implementation was done on ASP.NET Web applications and it was able to detect vulnerabilities that were ignored by the black-box vulnerability scanners. This approach is an efficient approximation mechanism to deal with string constraints. However, the approach is only dedicated to ASP.NET vulnerabilities [23].

SQL-IDS approach has been suggested by Kemalis and Tzouramanis in [24] and it uses a novel specification-based methodology for detecting exploitations of SQL injection vulnerabilities. The method proposed here does query-specific detection which allows the system to perform concentrated analysis at almost no computational overhead. It also does not produce any false positives or false negatives. This is a very new approach and in practice it's very efficient; however, it is required to conduct more experiments and do comparison with other detection methods under a flexible and shared environment.

McClure and Krüger suggested a framework SQL DOM (strongly-typed set of classes with database schema). The existing flaws have been considered closely during access of relational databases from Object-Oriented Programming Language's point of view. The focus lies mainly in identifying hurdles in interacting with databases through Call Level Interfaces. The solution proposed here is based on SQL DOM object model to handle this kind of issues by creating a secure surrounding i.e., creation of SQL statement through object manipulation for communication. When this technique was evaluated qualitatively it showed many advantages for: testability, readability, maintainability and error detection at compile. Although this proposal is efficient, there still exists scope of further improvements with latest and more advanced tools such as CodeSmith [25].

Ali et al.'s approach has been adopted by Ali et al. in which a hash value technique has been followed to improve user authentication mechanism. Hash values for user name and password has been used. For testing this kind of framework SQLIPA (SQL Injection Protector for Authentication) was developed. Hash values for user name and password is created for the first time user account is created and they stored in the user account table in a database. The framework requires further improvement in order to minimize the overhead time which was 1.3 ms even though tested on few sample data. Hence simply minimizing the overhead time is not sufficient but also to test this framework with large amount of data is required [26].

Su and Wassermann implemented their algorithm with SQLCHECK on a real time environment. It checks whether the input queries conform to the expected ones defined by the programmer. A secret key is applied for the user input delimitation. The analysis of SQLCHECK shows no false positives or false negatives. Also, the overhead runtime rate is very low and can be implemented directly in many other Web applications using different languages. Table 3 shows the number of attacks attempted as well as prevented [27]. It also shows the number of valid uses attempted and allowed, and the mean and standard deviation of times across all runs of SQLCHECK for the application under check. It is a very efficient approach; however, once an attacker discovers the key, it becomes vulnerable. Furthermore, it also needs to be tested with online Web applications [27, 28].

Dynamic Candidate Evaluations approach has been proposed by Bisht et al. called CANDID (Candidate evaluation for Discovering Intent Dynamically). It is a Dynamic Candidate Evaluations technique in which SQL injection is not only detected but also prevented automatically. Mechanism behind this method is that it dynamically extracts the query structures from every SQL query location which is intended by the developer. So, basically it resolves the problem of manually changing the application to produce the prepared statements. Although tool using this mechanism has been shown to be efficient in some cases, it failed for many other cases. An example for its failure is when applied at a wrong level or when an external function is dealt with. Furthermore it also fails in many cases due to limited capability of this technique [29]

5. SQLSCAN: PROPOSED TOOL

Now we give details about our proposed tool SQLSCAN. First we explain about the basic terms. Every web application has developed to be utilized by some users. Therefore, from now onwards by the term "user" means the user of the web based application. On the other hand, the proposed tool SQLSCAN is also implemented for some users and we mention them as "tool-user". Also we call each possible SQLIA as "threat". A threat in line n of file index.php means, line number n of the file index.php executes an SQL statement and it may not be safe. Note that, threat does not mean that the SQL statement is always unsafe, it only tells that there may be a possibility

of attack. The detail verification has to be done by the tool-user manually. Later in this section we explained as why it is necessary to verify each threat manually.

Fig. 2: Basic Work flow of proposed tool SQLSACN

Fig. 2 explains the basic flow and functionality of the proposed SQLSCAN. It takes a whole web application as input, scans for each SQL statements and verifies them. All possible threats are displayed as output. The tool is developed in JAVA and works in any environment. The basic functionality of the tool is:

- It scans each file in the application one-by-one to check for any vulnerable SQL executions.
- Display all such SQL statements (threats) in user friendly interface with the file name, line number and also with the cause of vulnerability.
- It also advises the tool-users about what to do for each threat.
- Once the user resolves an issue (regarding threat) he can mark the threat as resolved so that it won't show in next run.
- It can be easily upgraded to meet the demands of rapid technology changes.

Working of tool

In this section we discuss about how the SQLSCAN works. The tool works for PHP, JSP and ASP based applications but the basic mechanism for all types of application is same.

1) Attack detection mechanism: After getting an application folder (say AppliFold) as input the tool starts processing the folder. First of all it extracts all the files inside the folder and in its sub-folders. There is a recursive function written to extract all the files from the folder. All the extracted files are stored in an ArrayList (a data structure defined in Java, see the Java documentation for details). After getting the list of all files, the tool searches each file one-by-one to collect all the variables used. The variables can be of three types:

- Dynamic: variables which stores information entered by the user of the application. For example, in PHP an expression like $name=$ POST["name"]; means $name is a variable which stores the data inserted by user via an HTML form.
- Composite: variables whose value depends on other variables. For example, in JSP an expression var1=var2+var3; means var1 is depend on var2 and var3. So var1 is of a composite type.
- Static: variable which contains a hard coded value without depending on any other variables.

The tool searches all the files and collects all the variables and stored in an ArrayList (say varList). The variable searching procedure is based on regular expression written on a configuration file. Note that, the syntax of variable declaration and variable assignment in PHP, JSP and ASP are different and hence we have to write different regular expression rules for each language. Each entry in varList represents one variable and the structure of each entry is:

- Variable: The name of the variable. For example, var1, $name etc.
- Value: Lexical value assigned to the variable. Note that, the value means the text value not the actual value of the variable. Actual value differs based on the type of the variable.
- File name: The name of the file.
- Line number: The line number of the variable in the file.
- Variable type: Either dynamic, compositor static.

Once the variable got extracted the next job is to find out all the SQL statements that have been called from the application. We do it with the help of some regular expressions same we do for variable searching. Each extracted SQL statement can be in:

- Static: Depends on no other variables. For example,"select name, roll from students" is a SQL statement statically written and not depends on any other variable.
- Composite: Depends on some variables. Those variables can be either static, dynamic or composite in nature.

All extracted SQL statements are stored in an ArrayList (say SQLList). The structure of each entry in SQLLIst is:
Statement: The SQL statement.
File name: Name of the file where the statement resides.
Line number: Line number in the file where the statement is.
The next task is to verify the SQL statements. We consider a SQL statement as safe if it has no dynamic information required. In other word the statement which does not depend on the values entered by the users. Such type of safe statements can be of two types.

- Direct-static: Means a static statement as described above.

- Indirect-static: Not directly static but none of its depending variables are connected any dynamic variables.

The unsafe statements have possibility to depend on dynamic variables. We have written a recursive function to check each SQL statement. The statement declares safe if all its depending variables (not just direct dependents, but all the hierarchical dependence) are eventually connected with static variables. Otherwise, a threat generates with the list of all dynamic variables that can be connected with this statement. All such threats are stored in Array List (say threat List). Once the threat detection phase is over we display the content of threat List with an interactive GUI (java JTable). Also we can mark each threat as verified after confirming it as safe. The marked threats will not be visible for the next run until we forcefully want to do so.

2) How to confirm validity: The validity confirmation is left as manual. Based on the information provided by SQLSCAN, the tool-user has gone to the exact location of the statement and ensures that the statement has been handled properly. Each dynamic variable connected with the statement must pass through a validation process before using into the statement. Since the validation rule and the validation procedure changes from application to application, it is not easy to propose a universal rule for such validations. There for, we left it on the tool-user.

Experimental Analysis

We have experimented SQLSCAN 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 grip.

Table.1: Details of the Applications Used As Input For SQLSCAN

Application	Area	Tech.	Description
EX-MNGR	Academic	ASP.NET	Online Portal For Examination Management.
E-Lib	Academic	ASP.NET	Library Automation
Hospital-Management	Medical	PHP	Online Hospital Management Application.
Online Shopping	E- Commerce	PHP	Online Shopping Web Application
Employee-Sys	Corporate	JAVA	Online Employee Portal.

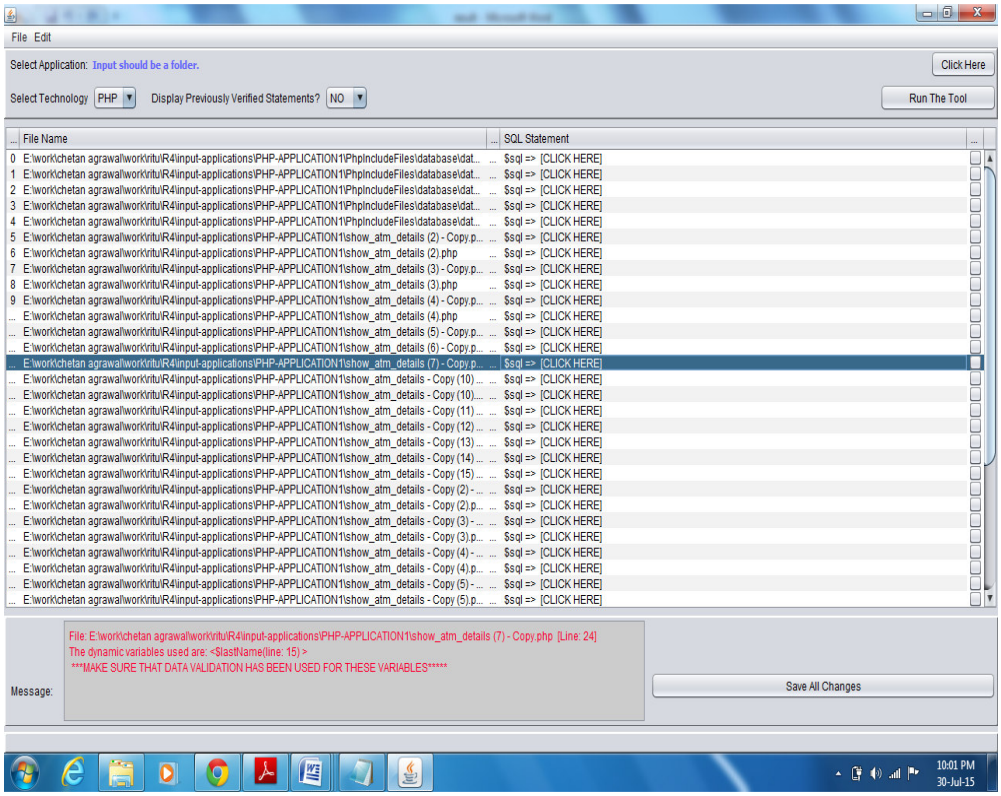

Fig.3: Snapshot of PHP application in SQLSCAN

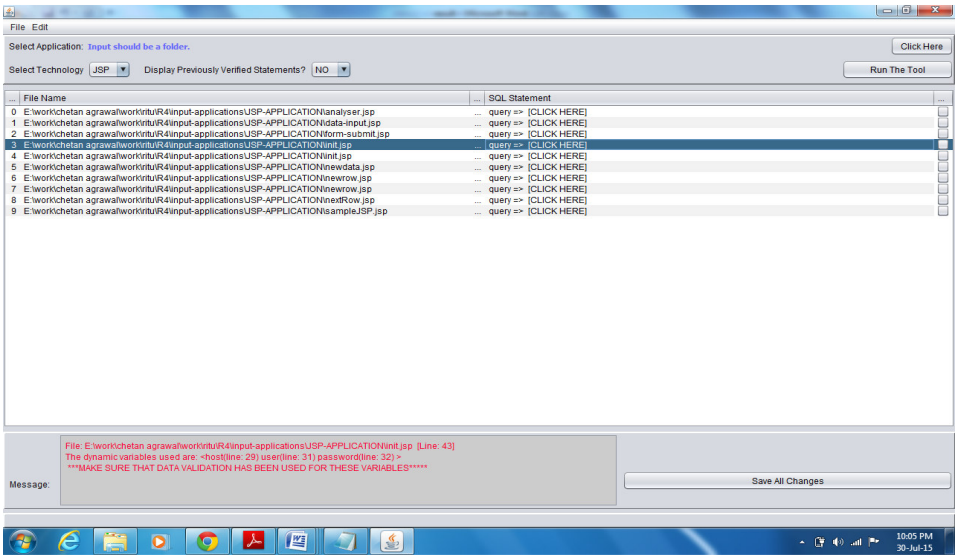

Fig.4: Snapshot of JAVA application in SQLSCAN

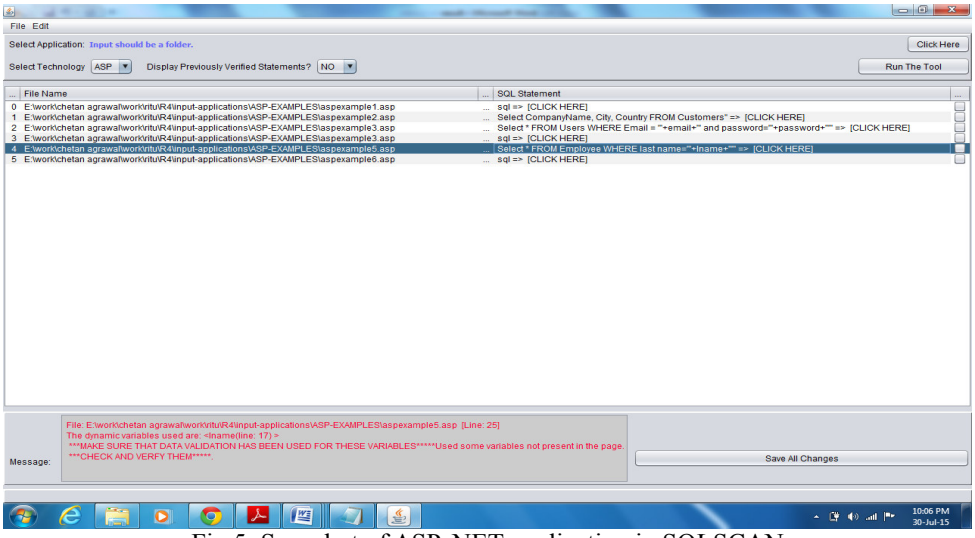

Fig.5: Snapshot of ASP. NET application in SQLSCAN

Conclusion

Database based applications might have security vulnerabilities that are not recognized to the developer of that web application. This paper describes a tool SQLSCAN that aims to the detection of security vulnerabilities. The proposed tool performs a scanning process for all kinds of website developed on any technology like JAVA, PHP, ASP .NET etc. Our scanner tool SQLSCAN relies on learning the source code of the application relaying on any server side scripting files and the code behind those files. A program written for this reason is to produce a statement that explains most leaks and vulnerabilities types (by declaration of the file name, leak description and its location on the system). The proposed SQLSCAN will help developers or organization to secure the vulnerabilities and get better the overall safety of web application. In the future, this work can increase to embrace an investigation of Other Web 2.0 technologies, such as Ruby, Flash, Python, and Ajax, can be included once sufficient web application vulnerability scanners are analyzed that include them in the list of technologies that they search for. The reputation of these Web 2.0 technologies should enhance in the near future, and will therefore participate major element in where vulnerabilities exist in web applications. By increasing the functionality and complexity of the test bed web server, it will more exactly imitate the design of web applications of today and of the future.

REFERENCES

[1] Nahari, H. & Krutz, R. L. “Web Commerce Security: Design and Development.” John Wiley & Sons, 2011.
[2] The Open Web Application Security Project (OWASP) Foundation. “Top Ten Web Application Security Risks”. 2011

[3] Roberts-Morpeth, Paul, and Jeremy Ellman. "Some security issues for web based frameworks." In Communication Systems Networks and Digital Signal Processing (CSNDSP), 2010 7th International Symposium on, pp. 726-731. IEEE, 2010.
[4] Silić, Marin, Jakov Krolo, and Goran Delač. "Security vulnerabilities in modern web browser architecture." In MIPRO, 2010 Proceedings of the 33rd International Convention, pp. 1240-1245. IEEE, 2010.
[5] Stuttard, Dafydd, and Marcus Pinto. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. John Wiley & Sons, 2011.
[6] "The Three Tenants of Cyber Security". U.S. Air Force Software Protection Initiative. Retrieved 2009-12-15.
[7] D. Shelly. Using a Web Server Test Bed to Analyze the Limitations of Web Application Vulnerability Scanners. Virginia Polytechnic Institute and State University. 2010.
[8] Y. Huang and D. Lee. Web Application Security-Past, Present, and Future. Pages 183–227. 2005
[9] Y. W. Huang, C. H. Tsai, D. Lee, and S. Y. Kuo. Non-detrimental web application security scanning. In Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on, pages 219–230, Nov. 2004.
[10] N. Antunes and M. Vieira. Detecting SQL Injection Vulnerabilities in Web Services. In Dependable Computing, 2009. LADC '09. Fourth Latin-American Symposium on, pages 17–24, Sept. 2009.
[11] D. Byrne and E. Duprey. Grendel-Scan. Available at: http://www.grendel-scan.com/.
[12] N. Surribas. Wapiti. Available at: http://www.ict-romulus.eu/web/wapiti/.
[13] A. Riancho. W3AF-Web Application Attack and Audit Framework. Available at: http: //w3af.sourceforge.net/.
[14] G. F. Lyon. NMAP.ORG. Available at: http://nmap.org/.
[15] M. Curphey and R. Arawo. Web application security assessment tools. Security & Privacy, IEEE, 4(4):32–41, July-Aug. 2006.
[16] M. Vieira, N. Antunes, and H. Madeira. Using Web Security Scanners to Detect Vulnerabilities in Web Services Using Web Security Scanners to Detect Vulnerabilities in Web Services. IEEE/IFIP Intl Conf. on Dependable Systems and Networks, Lisbon, Portugal, June 2009.
[17] W. G. Halfond and A. Orso. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In Proceedings of the IEEE and ACM International Conference on Automated Software Engineering (ASE 2005), Long Beach, CA,USA, Nov 2005.
[18] G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti. Using Parse Tree Validation to Prevent SQL Injection Attacks. In International Workshop on Software Engineering and Middleware (SEM), 2005.
[19] Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications. In The 33rd Annual Symposium on Principles of Programming Languages (POPL 2006), Jan. 2006.
[20] Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. 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), May 2004.
[21] V. B. Livshits and M. S. Lam. Finding Security Errors in Java Programs with Static Analysis. In Proceedings of the 14th Usenix Security Symposium, pages 271–286,Aug. 2005.
[22] Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. 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), May 2004.
[23] Fu, X., Lu, X., Peltsverger, B., Chen, S., Qian, K., and Tao, L., A Static Analysis Framework for Detecting SQL Injection Vulnerabilities. Proc. 31st AnnualInternational Computer Software and Applications Conference 2007 (COMPSAC 2007), 24-27 July (2007), 87-96.
[24] K. Kemalis and T. Tzouramanis. SQL-IDS: A Specification-based Approach for SQLinjection Detection. SAC'08. Fortaleza, Ceará, Brazil, ACM (2008), 2153 2158.
[25] R.A. McClure and I.H. Kruger, SQL DOM: compile time checking of dynamic SQL statements. 27th International Conference on Software Engineering (ICSE 2005),15-21 May (2005), 88- 96.
[26] S. Ali, S.K. Shahzad, and H. Javed. SQLIPA: An Authentication Mechanism Against SQL Injection. European Journal of Scientific Research, Vol. 38, No. 4 (2009), 604-611.
[27] Z. Su and G. Wassermann, ―The essence of command Injection Attacks in Web Applications,‖ In Proceeding of the 33rd Annual Symposium on Principles of Programming Languages, USA: ACM, January 2006, pp. 372-382.
[28] A. Tajpour, M. Masrom, M.Z. Heydari, and S. Ibrahim. SQL injection detection and prevention tools assessment. Proc. 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT'10) 9-11 July (2010), 518-522
[29] P. Bisht, P. Madhusudan and V.N. Venkatakrishnan. CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. ACM Transactions on Information and System Security, Volume 13 Issue 2, (2010).