www.ioirp.com
International Journal of Innovative Research in Computer Science and Engineering (IJIRCSE)
ISSN: 2394-6364, Volume – 1, Issue – 6. May 2015

Appraisal of Web Security Contrivances by Means Of Vulnerability & Attack Injection: A Survey

[1]Abdur Rahman.A, [2]Chitra Devi.S
[1]PG Scholar of Software Engineering, Indian Institute of Information Technology, Srirangam, India
[2]Assistant Professor of CSE, Anna University Chennai BIT Campus, Trichy, India
[1]rahmanrar@gmail.com, [2]Chitra_tau@yahoo.co.in

***Abstract*— Web applications can be personal websites, blogs, news, social networks, web mails, bank agencies, forums, e-commerce applications, etc. There is an escalation in craving on web applications, ranging from personalities to outsized officialdoms. This paper designates the security requirements for web application which are effortlessly predisposed to more attacks. This paper expounds the methodology which is based on the inkling that injecting realistic vulnerabilities in a web application and confronting them inevitably that can be used to upkeep the impost of persisting security mechanisms and tools in routine setup circumstances. The experiments include the estimation of coverage and false positives of an intrusion detection system for SQL Injection attacks and the valuation of the efficacy of two top commercial web application vulnerability scanners. Results show that the instillation of vulnerabilities and attacks is undeniably an effective way to appraise security mechanisms and to point out not only their flaws but also ways for their enhancement.**

Keywords*— Web Applications, SQL Injection, Vulnerabilities.

I. INTRODUCTION

Web application security is a branch of Information Security that covenants explicitly with security of websites, web applications and web services. At a high level, Web application security lures on the ideologies of application security but smears them specifically to Internet and Web systems. Stereotypically web applications are technologically advanced using programming languages such as PHP, Java EE, Java, Python, Ruby, ASP.NET, C#, VB.NET or Classic ASP. With the advent of Web 2.0, enlarged information partaking over social networking and increasing business adoption of the Web as a means of undertaking business and conveying service, websites are habitually attacked directly. Hackers either pursue to compromise the corporate network or the end-users accessing the website by endangering them to drive-by downloading. As a result, industry is paying improved courteousness to the security of the web applications themselves in addition to the security of the underlying computer network and operating systems. The mainstream of web application outbreaks transpire through cross-site scripting (XSS) and SQL injection attacks which classically result from blemished coding, and catastrophe to sterilize input to and output from the web application. These are ranked in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors. Phishing is another conjoint threat to the Web application.

The Security Division of EMC, announced the findings of its January 2013 Fraud Report, reckoning the global fatalities from Phishing at $1.5 Billion in 2012. Two of the well-known phishing methods are Covert Redirect and Open Redirect. OWASP is the emerging standards body for Web application security. In particular they have issued the OWASP Top 10 which designates in detail the foremost threats against web applications. The Web Application Security Consortium (WASC) has fashioned the Web Hacking Incident Database and also created open source best practice documents on Web application security. While security is primarily based on people and processes, there are a number of technical elucidations to contemplate when scheming, edifice and testing secure web applications. At a high level, these solutions include: First, Black box testing tools such as Web application security scanners, vulnerability scanners and penetration testing software. Second, White box testing tools such as static source code analyzers Fuzzing Tools used for input testing. Third, Web application security scanner (vulnerability scanner). Fourth, Web application firewalls (WAF) used to provide firewall-type protection at the web application layer. Fifth, Password cracking tools for testing password strength and implementation.

II. SECURITY REQUIREMENTS IN WEB APPLICATIONS

In web security, there are many security requirements which are to be considered that makes the web applications more secure and resist the attacks from the hackers.

A. Authorization & Access control

Authentication expresses a user "I distinguish you as a user." Authorization says "Now that I know who you are I also know what you are permitted to do; what data you are permissible to see and modify." Access control delineates where a user can connect from; what time they can connect, and the type of encryption required. The goalmouth is to advance a security strategy to protect back-end and front-end data and systems. This can be skilled through the use of roles, credentials, and sensitivity labels [2]. During the design phase, user roles should be defined based on a "least privilege" model. If a user role will not be amending data, then the role should not be given any chance to edit, delete, or add data to the critical database. Document the user roles during development and determine unsanctioned page by entering the location into the URL.

B. Session Management

A common vulnerability of web applications is instigated by not defending account credentials and session tokens [1]. There are four types of session id attacks: interception, prediction, brute-force, and fixation. In each attack, an unauthorized user can hijack a session and assume the valid user's identity. Encrypting sessions is actual against interception; indiscriminately assigned session ids protect against prediction; long key spaces render brute-force attack less efficacious and constraining assignment and frequent regeneration of session ids make fixation less tricky [3]. HTTP is a stateless protocol. It will answer any http request. HTTP does not, by itself, keep conversations in any specific order. It is imperative to use a state appliance to distinct and sustain an individual user's activities within a session. A cookie is a common vehicle used to preserve state in HTTP sessions. The cookie consents a user to make plentiful HTTP transactions in one conversation. The session id keeps the various requests unruffled in one exchange [4].

C. Cross Site Scripting (XSS)

When a web application generates output from user input without validating the data, the output can include malevolent code. An attacker looks for instances in code where there is no validation and inserts the attack at that point. The output that the user obtains may well carry malevolent code. The user may receive a "click here" message, and trusting the "known site," abide by the hackers desire. The result is focused at the end user rather than the application's infrastructure. This could transmit corporate confidential data to an outside site. It can result in program fixings or revelation of end user files [6].

D. Data & Input Validation

Cross-Site Scripting and Command Injection take advantage of a "violation of trust" [5] between a user retrieving a known and trusted site and an attacker. The attacker bypasses security mechanisms by adding malevolent code to open parameters in an application. An open parameter could be a URL, Query String, Header, Cookie, Form Field, or a Hidden Field. It is any parameter that does not guarantee that the data entered is data that would ordinarily be expected. For example, if the parameter is a date field, and the input "injected" into it is a script file, then the attacker has been efficacious in finding and using an open parameter. Well-written code would abandon the script. The importance of knowing and documenting what is known as valid data cannot be frazzled enough.

E. Buffer Overflows

"The buffer overflow attack encompasses sending large amounts of data that outstrip the quantities anticipated by the application within a given field [7]. Such attacks cause the application to abandon its customary behavior and begin implementing commands on behalf of the attacker." Attackers find buffer overflow susceptibilities by incisive for system calls and functions that do not confine the length and type of input. This can be done automatically or electronically with a code inspection tool. The attacker can also run a brute force attack against the program in the hope of finding vulnerabilities in the code. Once the attacker catches vulnerability, custom code is interleaved that does not crash the system, rather inculcates it to execute other commands or programs of the attackers desire.

F. Error Handling

Errors are inexorable. Errors can be instigated by user, programs, or perhaps they are errors between two systems. During development and testing an effort is made to categorize all potential errors and applicable error messages are developed for the end user. There will also be errors that are unexpected. The application must have protocols for these errors as well. Left "unhandled," the administrator has no idea that an error has occurred. The procedure for handling the unanticipated needs to include what the error was, when it occurred, and where it occurred [8].

G. Logging

Logging is crucial to officialdom's ability to track unlawful access and to conclude if any access attempt was successful. Logs provide individual accountability. They are vigorous to renovation of events leading to a program failure. Logs are often obligatory in any legal proceedings [9].

III. ATTACKS IN WEB SECURITY

In general, there are numerous types of attacks that threaten web security which empowers the hackers to make use of web applications illegitimately.

A. Union Query

It mainly focuses on bypassing Authentication, extracting data. In union-query attacks, an attacker abuses a vulnerable parameter to change the data set returned for a given query. With this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer. Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query>. Because the attackers completely control the second/injected query, they can use that query to retrieve information from a specified table. The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the results of the injected second query.

Example: Referring to the running example, an attacker could inject the text "' UNION SELECT cardNo from CreditCards where acctNo=10032 - -" into the login field, which produces the following query:
SELECT accounts FROM users WHERE login='' UNION
SELECT cardNo from CreditCards where
acctNo=10032 -- AND pass='' AND pin=

Assuming that there is no login equal to "", the original first query returns the null set, whereas the second query returns data from the "CreditCards" table. In this case, the database would return column "cardNo" for account "10032." The database takes the results of these two queries, mergers them, and returns them to the application. In many applications, the effect of this operation is that the value for "cardNo" is displayed along with the account information.
References: [10, 17, 15]

B. Piggy Backed Queries

It intends on extracting data, accumulation or amending data, performing denial of service, executing remote commands. In this attack type, an attacker attempts to inject additional queries into the original query. We discriminate this type from others because, in this case, attackers are not trying to change the original intended query; instead, they are trying to include new and dissimilar queries that "piggy-back" on the original query. As a result, the database obtains multiple SQL queries. The primarily is the intended query which is executed as normal; the successive ones are the injected queries, which are executed in addition to the first. This type of attack can be tremendously detrimental. If efficacious, attackers can supplement virtually any type of SQL command, including stored procedures,1 into the additional queries and have them accomplished along with the original query. Vulnerability to this type of attack is habitually reliant on having a database configuration that permits multiple statements to be contained in a single string.

Example: If the attacker inputs "'; drop table users - -" into the *pass* field, the application generates the query:
SELECT accounts FROM users WHERE login='doe' AND
pass='; drop table users -- ' AND pin=123

After completing the first query, the database would recognize the query delimiter (";") and execute the injected second query. The result of executing the second query would be to drop table users, which would likely extinguish valuable information. Other types of queries could supplement new users into the database or implement stored procedures. Note that many databases do not necessitate a special character to detached dissimilar queries, so simply scanning for a query separator is not an effective way to prevent this type of attack.
References: [10, 17, 14]

C. Alternate Encodings

It mainly focuses on evading detection. In this attack, the injected text is revised so as to avoid detection by wary coding practices and also many mechanized prevention techniques. This attack type is used in aggregation with other attacks. In other

words, alternate encodings do not provide any unique way to outbreak an application; they are simply an enabling technique that allows attackers to elude detection and prevention techniques and feat vulnerabilities that might not otherwise be exploitable. These evasion techniques are often obligatory because a common defensive coding practice is to scan for definite known "bad characters," such as single quotes and comment operators. To evade this defense, attackers have engaged alternate methods of encoding their attack strings (e.g., using hexadecimal, ASCII, and Unicode character encoding). Communal scanning and detection techniques do not try to estimate all specially encoded strings, thus allowing these attacks to go undetected. Contributing to the problem is that different layers in an application have different ways of handling alternate encodings. The application may scan for certain types of escape characters that symbolize alternate encodings in its language domain. Another layer (e.g., the database) may use different escape characters or even entirely altered ways of encoding. For example, a database could use the expression char(120) to represent an alternately-encoded character "x", but char(120) has no special meaning in the application language's context. An effective code-based defense in contradiction of alternate encodings is difficult to contrivance in practice because it necessitates developers to consider of all of the possible encodings that could distress a given query string as it passes through the different application layers. Therefore, attackers have been very efficacious in using alternate encodings to obscure their attack strings.

Example: Because every type of attack could be represented using an alternate encoding, here we simply provide an example (see of how esoteric an alternatively-encoded attack could appear. In this attack, the following text is injected into the *login* field:
"legalUser'; exec(0x73687574646f776e) - - ". The resulting query generated by the application is:
SELECT accounts FROM users WHERE login='legalUser';
exec(char(0x73687574646f776e)) -- AND pass='' AND pin=

This example makes use of the char () function and of ASCII hexadecimal encoding. The char() function takes as a parameter an integer or hexadecimal encoding of a character and returns an instance of that character. The stream of numbers in the second part of the injection is the ASCII hexadecimal encoding of the string "SHUTDOWN." Therefore, when the query is interpreted by the database, it would result in the execution, by the database, of the SHUTDOWN command. References: [10, 14]

IV. Recognition and Prevention

There are many techniques available to check the attacks and vulnerabilities in web applications. These techniques empower the users to exploit the web applications in secure environment.

A. Black Box Testing

The black-box technique [20] is used for testing Web applications for SQL injection vulnerabilities. The technique uses a Web crawler to recognize all points in a Web application that can be used to inoculate SQLIAs. It then builds attacks that aim such points based on a specified list of patterns and attack techniques. WAVES then screens the application's reaction to the attacks and uses machine learning techniques to progress its attack methodology. This technique mends over most penetration-testing skills by using machine learning approaches to monitor its testing and cannot afford assurances of completeness.

B. Static Code Checker

JDBC-Checker is a technique for statically testing the type perfection of dynamically-generated SQL queries [18, 19]. This technique was not developed with the intent of spotting and avoiding general SQLIAs, but can nevertheless be used to avert attacks that take advantage of type incongruities in a dynamically-generated query string. JDBC-Checker is able to perceive one of the derivation causes of SQLIA vulnerabilities in code— indecorous type checking of input. However, this technique would not catch more general forms of SQLIAs because most of these assaults consist of syntactically and type precise queries. Wassermann and Su recommend an approach that practices static analysis united with automated reasoning to corroborate that the SQL queries created in the application layer cannot contain a tautology [23]. The primary shortcoming of this technique is that its scope is limited to perceiving and thwarting tautologies and cannot distinguish other types of attacks.

C. Intrusion Detection System

The IDS system [22] is centered on a machine learning technique that is qualified using a set of typical application queries. The technique forms models of the typical queries and then screens the application at runtime to recognize queries that do not match the model. In evaluation, the system is able to distinguish attacks with a high rate of success. However, the major constraint

of learning based techniques is that they can provide no pledges about their detection capabilities because their success is reliant on the worth of the training set used. A poor training set would cause the learning technique to engender a large number of false positives and negatives.

D. Proxy Filters

Security Gateway [21] is a proxy filtering system that enforces input validation rules on the data flowing to a Web application. Using their Security Policy Descriptor Language (SPDL), developers convey constraints and instruct transformations to be applied to application parameters as they drift from the Web page to the application server. Because SPDL is highly sensitive, it permits developers considerable freedom in expressing their policies. However, this approach is human-based and, like defensive programming, necessitates developers to know not only which data needs to be filtered, but also what patterns and filters to apply to the data.

E. Vulnerability Attack Injection Tool (VAIT)

The VAIT [24] exemplar targets Linux, Apache, MySQL and PHP web applications, which is currently one of the most commonly used solution stack to advance web applications. Future progresses of the prototype may include other attacks types (e.g., XSS) and application technologies (e.g., Java). The VAIT inserts vulnerabilities into the web application code and spasms them in a faultlessly manner. The process of attacking the web application comprises of the preparation stage, the vulnerability injection stage, the attack load generation stage and the attack stage. All this vulnerability and attack injection process is accomplished with least human intrusion. The VAIT is configured with the web application folder location. Then the preparation stage is accomplished while the web application is being interacted. Latterly, the vulnerability injection stage continually generates the vulnerabilities, followed by the attack load generation stage that engenders the attack payloads. At this point, the attack stage can be instigated to outbreak the vulnerabilities, accumulate the results and estimate the attack success.

V. CONCLUSION

In recent years, Web applications have gained much reputation among people. The survey of web security is immense with innumerable attack models and counter measures suggested by researchers. The dissimilar attacks that pamper the functioning of the web application and degrade the performance have been discussed. Among the various countermeasures, the Vulnerability Attack Injection Tool (VAIT) technique progresses the web application security to the greater extent. In conclusion there is a better view on security requirements with attacks and their countermeasures in web applications.

References

[1] Aspect Security, “Common Vulnerabilities for Web Applications,” http://www.aspectsecurity.com/comvuln.html, February 18, 2004.

[2] Curphey, M. page 49\.

[3] Kolsek, M. (December 2002), Session Fixation Vulnerability In Web-Based Applications, ACROS, http://www.acros.si/papers/session_fixation.pdf, February 4, 2004.

[4] Moore, K., (October 2000), “RFC 2964 - Use of HTTP State Mechanisms”, http://www.faqs.org/rfcs/rfc2964.html, February 8, 2004.

[5] CERT Coordination Center, DoD-CERT, the DoD Joint Task Force for Computer Network Defense (JTF-CND), the Federal Computer Incident Response Capability (FedCIRC), and the National Infrastructure Protection Center (NIPC), (February 3, 2000), “Malicious HTML Tags Embedded in Client Requests”, http://www.cert.org/advisories/CA-2000-02.html, p.2.

[6] Cook, Stephen, (January 11, 2003), “A Web Developer’s Guide to Cross-Site Scripting”, http://www.giac.org/practical/GSEC/Steve_Cook_GSEC.pdf , February 24, 2004.

[7] MultiNet, Inc, (2001), Web Vulnerabilities and Security Solutions, http://elitesecureweb.com/dta/solutions/wvuln1.html, February 11, 2004

[8] Tuliper, Adam, (January 2, 2003), “Web Application Error Handling in ASP.NET,” http://www.15seconds.com/issue/030102.htm, February 23, 2004.

[9] Curphey, M. page 53.

[10] C. Anley. Advanced SQL Injection In SQL Server Applications. White paper, Next Generation Security Software Ltd., 2002.

[11] F. Bouma. Stored Procedures are Bad, O’kay? Technical report, Asp.Net Weblogs, November 2003. http://weblogs.asp.net/fbouma/archive/2003/11/18/38178.aspx.

[12] E. M. Fayo. Advanced SQL Injection in Oracle Databases. Technical report, Argeniss Information Security, Black Hat Briefings, Black Hat USA, 2005.

[13] P. Finnigan. SQL Injection and Oracle - Parts 1 & 2. Technical Report, Security Focus, November 2002. http://securityfocus.com/infocus/1644

[14] M. Howard and D. LeBlanc. Writing Secure Code. Microsoft Press, Redmond, Washington, second edition, 2003.

[15] S. Labs. SQL Injection. White paper, SPI Dynamics, Inc., 2002. http://www.spidynamics.com/assets/documents/ WhitepaperSQLInjection.pdf.

[16] C. A. Mackay. SQL Injection Attacks and Some Tips on How to Prevent Them. Technical report, The Code Project, January 2005. http://www.codeproject.com/cs/database/SqlInjectionAttacks.asp.

[17] S. McDonald. SQL Injection: Modes of attack, defense, and why it matters. White paper, GovernmentSecurity.org, April 2002.

[18] C. Gould, Z. Su, and P. Devanbu. JDBC Checker: A Static Analysis Tool for SQL/JDBC Applications. In Proceedings of the 26th International Conference on Software Engineering (ICSE 04) – Formal Demos, pages 697–698, 2004.

[19] C. Gould, Z. Su, and P. Devanbu. Static Checking of Dynamically Generated Queries in Database Applications. In Proceedings of the 26th International Conference on Software Engineering (ICSE 04), pages 645–654, 2004.

[20] Y. Huang, S. Huang, T. Lin, and C. Tsai. Web Application Security Assessment by Fault Injection and Behavior Monitoring. In Proceedings of the 11th International World Wide Web Conference (WWW 03), May 2003.

[21] D. Scott and R. Sharp. Abstracting Application-level Web Security. In Proceedings of the 11th International Conference on the World Wide Web (WWW 2002), pages 396–407, 2002.

[22] F. Valeur, D. Mutz, and G. Vigna. A Learning-Based Approach to the Detection of SQL Attacks. In Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), Vienna, Austria, July 2005.

[23] G. Wassermann and Z. Su. An Analysis Framework for Security in Web Applications. In Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems (SAVCBS 2004), pages 70–78, 2004.

[24] D. Avresky, J. Arlat, J.C. Laprie, and Y. Crouzet, "Fault Injection for Formal Testing of Fault Tolerance," IEEE Trans. Reliability, vol. 45, no. 3, pp. 443-455, Sept. 1996.