ISSN 2278-7763

Web Security Attacks and Injection- A Survey*

Swarnaprabha Patil, Prof. Nitin Agrawal

[1]Department of Computer Science & Engineering, NRI Institute of Science & Technology, Bhopal, India; [2](Department of Computer Science & Engineering, NRI Institute of Science & Technology, Bhopal, India.
Email: swarnagle12@gmail.com, seonitin79@gmail.com

ABSTRACT

Over the last few decades, number of user may increased on using web application. So the web-based attacks have caused major harm to those users on using given application. Several of these possible attacks may occur through the utilizations of common security vulnerabilities in web-based programs. Generally these attacks are enormously essential to reduce some of the harmful consequences on web application. Web-based applications are full of vulnerabilities that can be exploited by attackers at client-side (Web browser) without the wounded (browser user's) knowledge. Web Services have become dependable platform for e-commerce and many convectional models. An increasing penetration of Web Services has enticed attackers which has made Web Services prone to various attacks. For this reason, it happen to essential to examine possible attacks and anticipate probable attacks targeted at the Web Services. When we identified those possible attacks, improvement techniques need to be formulated so as to protection Web Services from them.

Keywords : Web Service Attacks, Web Application, Vulnerabilities.

1 INTRODUCTION

The World Wide Web (WWW) has become an important day-to-day resource for the people of this internet technology. According to the Internet usage statistics trends, as of Jun 2010, the number of Web users has increased approx. to 1.9 billion users, more or less a fourth of the world population. Additionally to being the most important source of information for a popular of these users, the Web is day-to-day resource for the people being progressively more used for providing on a daily basis services such as various application like health care, education services, banking sectors and E-commerce areas. Due to the consequence of these services, the web has concerned participation from a dissimilar general population, regardless of barriers of age, gender, or geography. So the web based application vulnerability scanners are tools that are used by network administrators and security professionals to help detect flaws in web sites so that they can be repaired before criminals or adversaries take advantage of them. However, since these tools currently have many mistakes and drawbacks, they are an inappropriate method for detecting all of an application's vulnerabilities and preventing attacks. We rely on web-based programs or web applications to perform many essential activities. Web-based programs usually reside on a server-side and are accessed from its client-side. It is estimated that the web now comprises over one trillion pages; so many, in fact, that leading search engines do not bother indexing them all [1], and that the structure of the web (the way web pages link one to another) has warranted its own studies and modeling [2]. Besides its sheer dimensions, the web has changed dramatically in terms of the content it hosts. The initial set of static pages has largely been replaced by sophisticated web applications, whose content is generated dynamically depending on the users they serve and their requests. Unfortunately, most of the programs are not implemented in vulnerability-free ways. At the client-side, browsers provide some built-in security features such as the Same Origin Policy (SOP), which prohibit web pages from accessing each other contents based on dissimilar domain names, ports, and protocols [3]. However, these features are not always sufficient to protect users from being attacked. As a result, users are vulnerable to exploitations while performing functionalities (e.g., login) [4], [5]. The exploitations might result in stealing of session information and generation of anomalous runtime behaviors. The situation further worsens when many web-based programs are deliberately designed and deployed to mimic trusted websites (i.e., websites that have explicit authentication mechanisms and employ active session information to perform functionalities) for stealing personal information (e.g., phishing websites [6]) instead of providing legitimate functionalities. As a consequence, the mitigation of web-based security vulnerability exploitations is extremely important to reduce some of the consequences. In particular, the web offers two main incentives to evil-doers. Firstly, the web applications increasingly store sensitive data, such as military services, financial data, medical, and personal records. By successfully attacking web applications, criminals gain access to their data, which they can profitably monetize in the underground market or can leverage to perform additional attacks (e.g., identity theft) [7]. Second, the web is popular and successful web applications have millions of web users. By setting up malicious sites or by cooperate legitimate ones, criminals can effectively attack and infect a large population of web clients, which they can mine for sensitive data or use as drones of a botnet [8].

ISSN 2278-7763

2 SCRIPT INJECTION VULNERABILITIES: DEFINITION

Wherever web based applications are distributed applications consisting of components that execute either on a web server or on a user's client. Scripting vulnerabilities arise when content controlled by an adversary (i.e. Un-trusted data) flows into critical operations of the program (i.e. critical sinks) without sufficient security confirms. When the web data is entrusted then it parsed or calculated as trusted code by the web browser, a scripting vulnerabilities attack consequences. This causes an attacker to gain higher privileges than intended by the web based application, characteristically contributing entrusted data the same chances as the web application's code. Distinguished pattern category of such code-injection attacks includes cross-site scripting [9] and cross-channel scripting [10] attacks.

The definitions of critical sinks and untrusted data inputs are application-specific. The intended security policy for certain applications permit data taken from users or third-party web sites to be evaluated as script code. Alternatively, various area of applications do not intend untrusted data inputs (such as user-generated content) to be executed by the browser as code.

- **Script Injection Vulnerabilities at Server-side**

Script injection attacks in server-side applications [11] have been investigated in depth by prior work. We provide an example of a typical scripting attack for exposition. All the HTTP data inputs to the web application server are treated as untrusted data. In this web based application, the security policy prohibits untrusted data to be executed as scripts or HTML markup when processed by the web browser. Script injection vulnerability is one that allows injection of untrusted data into a victim web page which is subsequently interpreted in a malicious way by the browser on behalf of the victim web site.

- **Script Injection Vulnerabilities at Client-side**

Much prior vulnerability research has focused primarily on the server-side components of web applications. Scripting vulnerabilities can arise in client-side components, such as those written in JavaScript, as well [11]. Here they present examples of script injection vulnerabilities client-side, a subclass of scripting vulnerabilities which result from bugs in the client-side code. In client-side script injection vulnerability, critical sinks are procedures in the client-side code where data is used with exceptional opportunity, such as in a code assessment construction of web based application.

Script injection vulnerabilities client-side is different from server-side scripting vulnerabilities in a few ways. For example, one type of script injection vulnerability client-side involves data that penetrates the application through the browser's cross-window communication abstractions and is processed completely by JavaScript code, not including ever being sent back to the web server. An additional type of client-side script injection vulnerability is one where a web application server disinfects untrusted data sufficiently before embedding it in its initial HTML response, but does not sanitize the data sufficiently for its use in the JavaScript component.

3 ATTACK INJECTION AND POSSIBLE VULNERABILITIES

When a threat representative makes the most of a crafted malicious SQL input to commence an attack, the attack objective is the purpose that the threat representative tries to accomplish once the attack has been effectively implemented.

Recognizing Injectable Parameters [12]: Using these injectable parameters or the user input fields of the Web applications directly used by server-side program logic and to build SQL statements, which are vulnerable to SQLIA. With the intention of launch a flourishing attack, a threat representative must first determine which parameters are vulnerable to SQL injection attack.

Presenting database finger-printing [13]: Database finger-print is the information that identifies a precise type and edition of database system. Every database system makes use of a different proprietary SQL language dialect. For example, the SQL language occupied by Microsoft SQL server is T-SQL while Oracle SQL server uses PL/SQL. Consecutively for an attack to be succeeded the attacker must first find out the type of and version of database organized by a web application, and then craft malicious SQL input for that reason.

Influencing database schema [12]: Database schema is the organization of the database system. The schema defines the tables, the fields in each table, and the relationships between fields and tables. Database schema is used by threat representatives to create an accurate consequent attack with the purpose of extract or modify data from database.

Bypassing Authentication [12]: Authentication is a method utilized by web application to emphasize whether a user is who he/she maintained to be. Matching a user name and a password stored in the database is the most frequent authentication system for web applications. Bypassing authentication facilitates an attacker to masquerade as an additional relevance user to gain un-authorized access.

Extracting Data [12]: Most of the cases for extracting data used by web applications are enormously vulnerable and attractive to threat representatives. Attacks with objective of extracting data are the most common type of SQL injection attacks.

Adding together or transforming Data [12]: Database alteration provides a selection of increases for a threat agent, for illustration, a hacker can pay much less for an online acquire by modifying the price of a product in the database. Or, the threads in an online discussion environment can be modified by an attacker to commence succeeding Cross-Site-Scripting attacks.

Executing denial of service [12]: These attacks are completed to stop the database of a Web application, consequently contradicting service to other users. Attacks concerning locking or dropping database tables also fall under this class.

ISSN 2278-7763

Escaping detection [12]: This class passes on to certain attack methods that are utilized to keep away from auditing and detection by structure safeguard methods.

Executing Remote Commands [12]: Remote commands are executable code resident on the cooperation database server. Remote command execution allocates an attacker to run uninformed curriculums on the server. Various Attacks with this category of meaning could cause complete inside networks being cooperation.

Performing concession appreciation [12]: Privileges are illustrated in a set of rights or authorizations join together with users. Privilege appreciation agree to an attacker to gain un-authorized access to a exacting benefit by associating a higher privilege set of rights with a existing user or masquerade as a user who has higher privilege.

Downloading File: Downloading files from a cooperation database server allows an attacker to view file content accumulated on the server. If the objective web application be inherent in on the same host, sensitive data such as design information and source code will be revealed too.

Uploading File: Uploading files to a finding the middle ground database server allow an attacker to store any malicious code onto the server. The malicious code could be a Trojan horse, a back door or a worm that can be used by an invader to commence subsequence attack.

The vulnerability is present when user input is either incorrectly passes through a filtered for string accurate escape characters embedded in SQL statements or user input is not strongly typed and in this manner without warning executed:

- **Inadequate Input Validation:** when input validation is an effort to authenticate or filter any given input for malicious activities. Inadequate input validation will allocate code to be executed not including proper verification of its objective. Attacker's delightful benefit of inadequate input validation can utilize malicious code to conduct attacks.

- **Privileged Account:** A privileged account has a degree of independence to do what normal explanations cannot. Its actions may also be excused from auditing and validation. This presents vulnerability in view of the fact that a make vulnerable privileged account, such as an administrator account, can finding the middle ground much more than what a make vulnerable regular account.

- **Extra Functionality:** Extra functionalities intended to provide a broader range of vulnerability, in view of the fact that combinations of this functionality may outcome in unintentional actions. For example, xp_cmdshell is inevitable to make available users with a way of executing operating system commands, but commonly used to supplementary unauthorized users into the operating system.

4 INJECTIVE MECHANISMS

Malicious SQL declarations can be commenced into a vulnerable application using many unusual input methods. In this part of injection mechanism, we give explanation the most common methods are as follows:

Injection all the way through user input: In this type of injection, attackers inject SQL commands by providing correctly technique through user input. A Web application can read user input in a number of ways based on the situation in which the application is organized. In most SQLIAs that target Web applications, user input characteristically comes from form submissions that are sent to the Web application via HTTP GET or POST requests [13]. Web applications are usually able to access the user input surrounded in these requests as they would right of entry through any other variable in the environment.

Injection via cookies: Cookies are files that include state information produced by Web applications and accumulated on the client machine. When a user i.e. client returns to a Web application, cookies can be used to re-establish the client's state information. Since the client has managed over the storage of the cookie, a malicious client could corrupt with the cookie's substances. If a Web application uses the cookie's substances to generate SQL queries, an attacker could easily submit an attack by embedding it in the cookie [14].

Injection during server variables: Server variables are a gathering of variables that contain HTTP, network headers, and natural variables. Web applications use these server variables in a variety of ways, such as logging usage statistics and recognizing browsing growths. If these server variables are logged to a database without any improvement, this could generate SQL injection vulnerability [15]. Because attackers can counterfeit the values that are placed in HTTP and network headers, they can take advantage of this vulnerability by placing an SQLIA directly into the headers. When the query to log the server variable is subjected to the database, the attack in the counterfeit header is then triggered.

Second-order injection: In second-order injections, attackers beginning malicious inputs into a system or database to indirectly trigger an SQLIA when that input is used at an afterward time. The objective of this category of attack differs considerably from a regular (i.e., first-order) injection attack. Second-order injections are not undertaking to because the attack to occur when the malicious input at the start reaches the database. As an alternative, attackers rely on knowledge of where the input will be subsequently used and technique their attack so that it suggest

ISSN 2278-7763

itself during that usage. To illuminate, they present a characteristic example of a second order injection attack [16].

5 LITERATURE SURVEY

In research community have been actively involved in finding attacks targeted at Web Services and designing mitigation techniques for the same. The attacks documented in the literature can fall into one of two categories. First category of attacks are those which were observed when the Web Services running on the production boxes came under the scanner of attackers and were description to be concession. Various of the attacks fall into second category where attacks were stated as reasonable by researchers after systematic analysis of different Web Service frameworks. This survey discusses various ranges of Web Service attacks found in the literature and also mitigation techniques to thwart some of these attacks.

The approach of Louw et al. [17] is the closest to our work. They develop the "Blueprint" tool to mitigate XSS attacks that first generates response pages without any JavaScript node at the server-side. To eradicated script is accomplished at the browser side based on the content generation provided by the server-side with code instrumentation. Consequently, Blueprint has to rely on a particularly intended external JavaScript library at the client-side. Absolutely not, but their approach depends on compassionate HTML and JavaScript features and removes suspected malicious contents from the server-side. Furthermore their come within reach of does not impose any external library dependency. Blueprint transforms untrusted contents (e.g., an attacker supplied inputs) when sending them to the browser. On the contrary, their approach transforms the server-side script code based on the possible injection places, but not the injected contents themselves.

In this paper author et al. Jose Fonseca [18] proposes a methodology to automatically inject realistic attacks in web applications and a prototype tool to evaluate web application security mechanisms. The proposed work analyzing the web application and generating a set of potential vulnerabilities on the idea that injecting realistic vulnerabilities in a web application and attacking them automatically can be used to support the assessment of existing security mechanisms and tools to find each vulnerability in custom setup scenarios is injected and various attacks are mounted over each one is automatically assessed and reported. The proposed vulnerability attack injection methodology relies on the vulnerabilities injected derives from the use of the results of a large field study on real security vulnerabilities in widely used in present web applications. The VAIT was also used to estimate two viable and extensively used web application vulnerability scanners, concerning their capability to detect SQLi vulnerabilities in web applications. To exhibit the possibility of the proposed work, they build up a tool that mechanizes the whole process: the VAIT. Even though it is only a prototype, it highlights and overcomes implementation definite problems. It may highlighted the necessitate to match the consequences of the dynamic analysis and the static analysis of the web application and the need to synchronize the outputs of the HTTP and SQL probes, which can be executed as independent processes and in different computers. All these results must produce a single analysis log containing both the input and the output interaction results. The proposed methodology describes the implementation of the Vulnerability & Attack Injector Tool (VAIT) that allows the automation of the complete process. The VAIT prototype focused on the most significant fault type, the MFCE generating SQLi vulnerabilities. It emphasized the need to match the effects of the dynamic analysis and the static analysis of the web application and the need to synchronize the outputs of the HTTP and SQL probes, which can be performed as independent practices and in different computers. By injecting vulnerabilities and attacking them automatically the VAIT could find weaknesses in the IDS. These outcomes were very important in developing bug fixes. The VAIT was also used to evaluate two commercial and widely used web application vulnerability scanners, concerning their ability to detect SQLi vulnerabilities in web applications. Experimental results have shown that the proposed methodology can efficiently be used to estimate security mechanisms like the IDS, providing at the same time indications of what could be improved. The experiments include the evaluation of coverage and false positives of an Intrusion Detection System for SQL Injection attacks and the assessment of the effectiveness of two top commercial web application vulnerability scanners. These scanners were unable to detect most of the vulnerabilities injected, regardless of the fact that some of them give the impression to without any difficulty be probed and confirmed by the scanners. Experimental results show that the injection of vulnerabilities and attacks is certainly an efficient way to evaluate security mechanisms and for improvement in the SQLi detection capabilities of these scanners to point out not only their weaknesses but also ways for their advancement.

In this paper author has proposed [19] mechanism for assessing Web application security was constructed to analyze the design of Web application security assessment mechanisms in order to identify poor coding practices that render Web applications vulnerable to attacks such as SQL injection and cross-site scripting. Proposed technique come across on the real-world circumstances of web application security estimation of intended a crawler crossing point that incorporates and mimics Web browser functions with the intention of analysis Web applications.

During the first assessment phase, the crawler attempts to perform a complete reverse engineering process to identify all data entry points–possible points of attack– of a Web application. These entry points then serve as targets for a fault injection process in which malicious patterns are used to resolve the most vulnerable points. They also proposed the NRE algorithm to eliminate false negatives and to allow for "deep injection." In "deep injection", the IKM formulates an invalid input pattern to retrieve a negative reply page, and then uses a programmed form completion algorithm to originate the most likely injection patterns. After sending the injection, WAVES analyzes the resulting pages using the NRE algorithm, which is simpler, yet more accurate than the LITE approach. Real-world circumstances situations are used to test a device they named the Web Application Vulnerability and Error Scanner (WAVES) and to evaluate it with other tools. Experimental results show that WAVES is a feasible platform for assessing Web application security.

ISSN 2278-7763

Here et al. Halfond, W. G [20] present an extensive perform this evaluation, they first identified the various types of SQLIAs known to date. They then evaluated the considered techniques in terms of their capability to detect and/or prevent such type of attacks. For each type of attack, they make available explanations and examples of how attacks of that type could be performed. They also studied the different mechanisms and analyze existing detection through which SQLIAs can be introduced into an application and identified which techniques were able to handle which mechanisms and prevention techniques against SQL injection attacks. Many of the techniques have problems handling attacks that take advantage of poorly-coded stored procedures and cannot handle attacks that disguise themselves using interchange encodings. They also found a common dissimilarity in prevention abilities based on the difference between prevention-focused and general detection and prevention methods. For each method, they talk about its strengths and weaknesses in addressing the complete range of SQL injection attacks that prevention-focused techniques try to incorporate defensive coding best practices into their attack prevention mechanisms.

Here author et al. Manisha A. Bhagat [21] paper proposed alerts the people who are related to database maintenance, DBA and other people who are introducing their sites on Internet. This paper gives idea on the subject of the hole which can be secured either by code or protection security like firewalls. It is essential to check the code before commencing the site. SQL Injection Attacks are dangerous to the applications on Internet. The objective of the attacker is to gain right of entry to the database. They have examined all common attack techniques and provided design for each of them. They have proposed one answer for input validation. That is generate one table which contains special characters like; ,,,--, ". If the input contains such special characters, the SQL query is finished and is not allowed to be accomplished on database. When attacker gains control over web application maximum damage is caused.

In this paper [22] author has presented an enhanced Tainted Mode model that incorporates data flows which allows inter-module vulnerabilities detection. They also commenced a new approach to involuntary penetration testing by leveraging it with knowledge from dynamic analysis. Convectional approaches based on the Tainted Mode vulnerability model which cannot hold inter-module vulnerabilities. So the author will work focus on two main approaches. Initially generalization of the model will be building up to sustain investigation of data flows all the way through other data storage types or put into practiced by means of stored procedures and triggers. Second one, exceptional consideration to improvement of repeated crawling methods will be given. The main crisis here is routine form filling. They plan to sophisticate the following thought. First, static analysis can be used to become aware of data flows from HTTP parameters to the database fields. This data flow analysis results in data dependencies of the database columns upon values of exacting form fields. As a final point, the database can be queried to get values from the data storage which will be used in automatic form filling, consequently leveraging crawling progression.

In this paper [23] authors presents an authentication scheme for preventing SQL Injection attack using Advance Encryption Standard (PSQLIA-AES). Here this scheme required encrypted user name and password are to improve the authentication process with minimum overhead. The server has to sustain three parameters of every user: user name, password, and user's secret key. They have put into practiced and examination the proposed scheme with three different ways.

- Calculated with different key sizes (128, 256, and 512)
- Evaluated the processing the transparency (time needed for encryption of user name and password) of proposed scheme with presenting related schemes and AQE-PSQLIA.
- Measured up to the processing overhead of the proposed scheme by using different number of users (10, 20, 30, 40, and 50).

Experimental results of proposed scheme are more competent, it needs 3.144 ms for encryption or decryption and this can be insignificant.

In this paper [24] author presents a new methodology by developing a framework that makes attacks on networked servers and discovers security vulnerabilities in software systems and allows security administrators to determine the problems. To show this model application is built that a new methodology that takes protocol requirement aspects from server and carry out various attacks on the server and discovers vulnerabilities. The discovered vulnerabilities are then continued into database for extra steps to fix the bugs in the software in which vulnerabilities are originate. All the experiments are prepared on Linux server and the experimental results discovered that the proposed tool is proficient of discovering vulnerabilities successfully.

In this paper [25], they present a detailed review on various types of Structured Query Language Injection attacks, Cross Site Scripting Attack, vulnerabilities, and prevention methods. From the survey of various papers it is found that SQL Injection and Cross-site Scripting (XSS) Attacks are most powerful and easiest attack methods on the Web Application. This research presents a review of various current methods for protecting against SQL injection and XSS make use of it. The academic research field and industry has developed shows potential strategies for example AMNeSIA, ARDILLA etc. products which are rising and likely to become essential parts of widespread online data protection strategies. However, even though the efficiency of these products, they consider that their use should not justification for developers from pertaining defensive coding techniques, as these hold true potential when put into practiced properly. Staying visible the promising web technologies and widespread procedure of extremely interactive content over Internet, it is imperative for the software development dwellings and developers to frame and follow suitable security framework to build security during Software Development Life Cycle (SDLC). Moreover presenting our discovering from the analysis, they also propose future anticipations and possible development of countermeasures beside Structured Query Language Injection attacks.

6 Conclusion

Whenever web application was proposed to have both a secure and insecure description then find the possible outcomes could be

examined. The secure protection explanation of the web application is expected to come back. Since no vulnerabilities are consciously implemented in this enlightenment. As a result, the only vulnerabilities that would be reported would be imprecisely recognized the probable vulnerabilities. Since many researchers have shown that the existing methods used by web application are not competent of detecting all types of vulnerabilities, it is guessed that not all of the put into practiced vulnerabilities will be detected.

REFERENCES

[1] J. Alpert and N. Hajaj. We knew the web was big. http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html , 2008.

[2] A. Barabasi and R. Albert. Emergence of scaling in random networks. Science, 286(5439):509{512, 1999.

[3] JavaScript Security in Mozilla, Accessed from www.mozilla.org/projects/security/components/jssec.html , November 2011.

[4] Open Source Vulnerability Database (OSVDB), Accessed from http://osvdb.org , November 2011.

[5] Common Vulnerabilities and Exposures (CVE), http://cve.mitre.org , November 2011.

[6] D. Geer, "Security Technologies Go Phishing," Computer Archive, Volume 38, Issue 6, June 2005, pp. 18-21.

[7] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), pages 375{388, Alexandria, VA, USA, 2007.

[8] M. Polychronakis, P. Mavrommatis, and N. Provos. Ghost Turns Zombie: Exploring the Life Cycle of Web-based Malware. In Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), San Francisco, CA, USA, 2008.

[9] RSnake. XSS Cheat Sheet for filter evasion. http://ha.ckers.org/xss.html.

[10] Hristo Bojinov, Elie Bursztein, and Dan Boneh. \XCS: Cross Channel Scripting and its Impact on Web Applications". In: CCS. 2009.

[11] Amit Klein. DOM Based Cross Site Scripting or XSS of the Third Kind. Tech. rep. Web Application Security Consortium, 2005.

[12] W. Halfond, J. Viegas, and A. Orso. A Classification of SQL-Injection Attacks and Countermeasures. *Proceedings of the IEEE International Symposium on Secure Software Engineering (ISSSE)*, 2006.

[13] N. W. Group. RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1.Request for comments, The Internet Society, 1999.

[14] M. Dornseif. Common Failures in Internet Applications, May 2005. http://md.hudora.de/presentations/2005-common-failures/dornseif-common-failures-2005-05-25.pdf.

[15] T. M. D. Network. Request. Server variables collection. Technical report, Microsoft Corporation, 2005. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/iissdk/html/9768ecfe-8280-4407-b9c0-844f75508752.asp.

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

[17] M. Louw and V. Venkatakrishnan, "BLUEPRINT: Robust Prevention of Cross-Site Scripting Attacks of Existing Browsers," IEEE Security and Privacy, Oakland, California, USA, May 2009, pp. 331-346.

[18] José Fonseca, Marco Vieira, and Henrique Madeira "Evaluation of Web Security Mechanisms using Vulnerability & Attack Injection", pp. 1- 12, 2013.

[19] Huang, Yao-Wen, Shih-Kun Huang, Tsung-Po Lin, and Chung-Hung Tsai. "Web application security assessment by fault injection and behavior monitoring." In Proceedings of the 12th international conference on World Wide Web, pp. 148-159. ACM, 2003.

[20] Halfond, W. G., Jeremy Viegas, and Alessandro Orso "A classification of SQL-injection attacks and countermeasures", In Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA, pp. 13-15. 2006.

[21] Manisha A. Bhagat and Vanita Mane "Protection Of Web Application Against Sql Injection Attack", International Journal of Scientific and Research Publications, ISSN 2250-3153, Volume 3, Issue 10, October 2013.

[22] Petukhov, Andrey, and Dmitry Kozlov "Detecting security vulnerabilities in web applications using dynamic analysis with penetration testing." Computing Systems Lab, Department of Computer Science, Moscow State University (2008).

[23] Indrani Balasundaram and E. Ramaraj "An Authentication Mechanism to prevent SQL Injection Attacks", International Journal of Computer Applications, ISSN: 0975 – 8887, Volume 19, No. 1, April 2011.

[24] Dr. B. Raveendranath Singh "Vulnerability Discovery with Attack Injection Software Vulnerability Discovery", International Journal of Advanced Research in Computer Science and Software Engineering, ISSN: 2277 128X, Volume 3, Issue 9, September 2013.

[25] Rahul Johari and Pankaj Sharma "A Survey On Web Application Vulnerabilities (SQLIA,XSS)Exploitation and Security Engine for SQL Injection", 2012 International Conference on Communication Systems and Network Technologies, pp. 453 – 458, 2012.