RESEARCH ARTICLE OPEN ACCESS

Web Application Vulnerability Detection and Mitigation with Static Exploration

Uduakobong Asuquo Ituen [1], M. B. Mukeshkrishnan [2]

M.Tech [1] Information Security and Cyber Forensics

Assistant Professor [2], Department of Information Technology

Faculty of Engineering and Technology, SRM university

Tamil Nadu – India

ABSTRACT

The number of reported web application vulnerabilities is increasing dramatically. Security vulnerabilities in web applications may result in stealing of confidential data, breaking of data integrity or affect web application availability. Thus the task of securing web applications is one of the most urgent for now: according to Acunetix survey 60% of found vulnerabilities affect web applications. These web applications, which can be accessed from anywhere, become so widely exposed that any existing security vulnerability will most probably be uncovered and exploited by hackers. The most common way of securing web applications is searching and eliminating vulnerabilities therein. This paper explains a field study on two of the most widely spread and critical web application vulnerabilities: SQL Injection and XSS. This paper presents how these confidential data are really exploited by hackers in web applications, and its presents an analysis of the source code of the scripts used to attack hackers.

Keywords:- SQL Injection, Cross Site Scripting, Vulnerabilities, Vulnerability Exploration, Web Application Security.

I. INTRODUCTION

The issue of web application security vulnerability has grown significantly in recent times. This has been mainly due to the increase in the use of the internet. A good number of organizations these days depend on the use of the internet to make their business reachable by their clients and users. Looking critically at the history of the internet, the internet has been of great help to these industries and organizations, it has caused increase in the productivity of these businesses, but just as the internet has positive effects it also has its negative effects too.

In recent times the numbers of people who have access to computers have increased thereby causing the possible number of internet users to also increase. According to web resources the internet by definition is a global computer network providing a variety of information and communication facilities consisting of interconnected networks using standardized communication protocol, this simply means that the internet can be accessed from anywhere and at any time. Most of the people who have access to computers know little or nothing about securing these computers. Since these computers are what makes up the Web therefore the web id made vulnerable through these computers. The users who have access to the internet are served using a huge amount of data which is stored in databases that are connected to a number of web applications all over the world.

Furthermore, these applications play a dynamic role in maintaining security of data stored in these databases. Web applications which are not very secured allow well-made injection to perform unwanted operations on back-end database. An unauthorized user may be able to exploit this situation by destruction or with theft of trusted users sensitive data stored here. The maximum amount of damage is caused when an attacker gains full control over a database or web application as there is a possibility of system being fully destroyed. The sensitive data on one's system could be transferred to any intermediary by injecting a class of vulnerability in trusted site's dynamically generated pages. This intermediary could be an attacker's server. This mechanism could also be used in avoiding cookie protection mechanism or same-origin-policy.

SQL (Structured Query Language) Injection and Cross-Site Scripting (XSS), the two top most attacks according to OWASP (Open Web Application Security Project), has been used frequently to implement the attacks.

SQL injection is a method used by attackers to attack data-driven applications here SQL statements are inserted into an

entry field for execution. For an SQL injection attack to be possible the application would have vulnerability. It is this security vulnerability that SQL injection must exploit in an application's software, for example, when user input is either incorrectly filtered

SQL Injection attempts to pass SQL commands (statements) through a web application for execution by the backend database. If these inputs are not sanitized properly, web application may result in SQL Injection attacks that allow hackers to view information from the database, wipe data out, or change data.

When the authorized user submits his details, an SQL query is created from these details and submitted to the database for verification. If the details are valid, the user is allowed access. The web application which controls the login page communicates with the database through a sequence of planned commands so as to verify the username and password combination. After verification, the authorized user is granted appropriate access. Using SQL Injection, the hacker may input SQL commands with the intent of bypassing the login form security and seeing what lies behind it. This is only possible if the inputs are not properly sanitized. SQL Injection vulnerabilities provide a hacker with the means to communicate directly to the database. Web servers owned by an organization is primary target of these kinds of attack.

Cross site scripting (XSS) vulnerability arises when a web application makes use of inputs received from users in web pages without properly checking them. When an attacker gets a user's browser to run a specific code, the code will run within the security zone of the hosting web site. At this privilege level, the code is able to read, modify and transmit or delete any sensitive data accessible by the browser. A user that has been attacked in this manner could have his/her account controlled by the attacker and their browser could also be redirected to another location. Cross-site Scripting attacks compromise the trust that exits between a user and the web site.

Current methods to mitigate this problem mainly focus on effective detection of XSS vulnerabilities in the programs or avoidance of real time XSS attacks. As more refined attack vectors are being discovered, vulnerabilities if not removed could be exploited anytime. XSS usually affects victim's web browser on the client-side where as SQL injection occurs in server side. These vulnerabilities could be exploited by SQL injection or XSS to gain control over the online web application database. In this paper we focus on various SQL injection and XSS attacks and approaches for their detection and prevention.

II PROBLEM DEFINITION

The security of Web applications has become increasingly important in recent year. More and more Web based enterprise applications have sensitive financial, medical, and personal data which, if compromised, in addition to lost time can mean millions of dollars (or any currency which such organization operates) in damages. It is important that these applications are protected from hacker attacks. However, dues to the current state of application security it leaves much to be desired. The 2002 Computer Crime and Security Survey conducted by the Computer Security Institute and the FBI revealed that, on a yearly basis, over half of all databases experience at least one security breach and an average episode results in close to $4 million in losses. A recent penetration testing study performed by the Imperva Application Defense Center included more than 250 Web applications from e-commerce, online banking, enterprise collaboration, and supply chain management sites.

III. EXISTING SYSTEM

As the security of web applications has become a major concern and it is receiving a lot of attention from governments, corporations, and the research community. Cross-site scripting (XSS) and SQL injection (SQLi), are two of the most common and critical vulnerabilities found in web applications. SQL input injection attacks may serve a number of ends. Generally, malicious users prefer method as a way to obtain restricted data from the backend of a database or in embedding malicious codes onto a web server that will serve up malware to unsuspecting clients.

According to OWASP [16], the most efficient way of finding security vulnerabilities in web applications is to review code manually. This method has proven to be effective but has the disadvantage of being very time-consuming, for this method to actually be worthwhile it requires expert skills, and also the possibility of codes with errors to be overlooked is high; therefore, security societies have developed automated methodologies to finding security vulnerabilities. These approaches are divided into two wide categories:

Black-box testing
White-box testing

Black-box testing: It is also called functionality testing. It could be described as a method of testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. With black box testing, the software tester does not (or should not) have access to the source code itself. The code is likened to a "big black box" to the tester. This simply means the tester can't see inside the box. The tester only knows that information can be as input to the black

box, and the black box will send output. Based on the requirements knowledge, the tester knows what the black box is expected to send out and therefore tests to make sure the black box sends out what it's supposed to send out (functions properly)

White-box testing:

The first method is based on web application exploration from the user side, assuming that source code of an application is not available [17]. The idea is to submit various malicious patterns into web application forms and to analyze its output afterwards. If any application errors are observed an assumption of possible vulnerability is made. This approach does not guarantee accuracy or completeness of the obtained results. The existing approaches for mitigating dangers to web applications can be divided into client-side and server-side solutions. The only client-side tool known to the authors is Noxes [15], an application-level firewall offering protection in case of suspected cross-site scripting (XSS) attacks that attempt to steal user identifications. Server-side solutions have the advantage of being able to discover a higher range of vulnerabilities, and the advantage of a security flaw fixed by the service provider is instantly spread to all its clients. These server-side techniques can be further classified into dynamic and static approaches.

IV PROPOSED SYSTEM

This paper, proposes an analysis of the source code of the scripts used to attack the malicious users. And helps developers to know about how these vulnerabilities are really exploited by hackers. It could be used to aid software developers and code inspectors in the exposure of such faults and are also the foundation for the research of accurate vulnerability and attack injectors that can be used to assess security mechanisms, such as intrusion detection systems, vulnerability scanners and static code analyzers.

A. *ADVANTAGE:*

Through analysis, strong and weak type languages in developing web applications are detected. And a method to improve the weak type languages to decline malicious users is suggested.

From other studies, we know weak typed languages contain more vulnerabilities than strong typed languages which makes the weak typed languages a constant point of attack.

i. *SOFTWARE REQUIREMENTS:*

Front End	:	Java
Environment	:	Eclipse/Net Beans
Back End	:	MySQL
Operating System	:	Windows XP

ii. *HARDWARE REQUIREMENTS:*

Processor	:	Pentium IV
RAM	:	512 MB
Hard Disk	:	80 GB

V. IMPLEMENTATION

A. construction of test web application
B. determination of web application vulnerabilities
C. classification of defect
D. inject malicious code

A. CONSTRUCTION OF TEST WEB APPLICATION

In the attempt to analyze web security vulnerabilities a sample web application is created using open source tools which are available on the internet. The web application created is an imitation of an organization's web page.

Most information systems and business applications built nowadays have a web front end and they need to be universally available to clients, employees, and partners around the world, as the digital economy is becoming more and more prevalent in the global economy. So, when we develop web application, we consider the security on that business sites. The security of web applications becomes a major concern and it is receiving more and more attention from corporations, and the research community as a whole. Here, the organization's site is developed with secure information, success formulae, account details, partners secure information, employee details, etc. having such information present on the website and in public view makes it imperative that security of the website should be the first thing that should be considered.

B. *DETERMINATION OF WEB APPLICATION VULNERABILITIES*

The Open Web Application Security Project Report listed the 10 most critical web application security risks, having SQLi at the top, followed by XSS. Other studies also found XSS and SQLi as the most prevalent vulnerabilities on web applications. It is important to emphasize that each vulnerability discovered opens a door for hackers to successfully attack anyone of the millions of web sites developed with a given version of the web application. SQLi attacks take advantage of unchecked input fields in the web application interface to maliciously tweak the SQL query sent to the back-end database. By exploiting XSS vulnerability, the

attacker is able to inject into web pages unintended client-side script code, usually HTML and Java script.

SQLi and XSS allow attackers to access unauthorized data (read, insert, change, or delete), gain access to privileged database accounts, impersonate other users (such as the administrator), mimic web applications, deface web pages, view, and manipulate remote files on the server, inject and execute server side programs that allow the creation of botnets controlled by the attacker, and so on. This module deals with checking the web application to see it is vulnerable to known attacks on the database or the user input form.

C. *CLASSIFICATION OF DEFECT*

This section presents the methodology to obtain and classify the source code and the security patches of the web applications of our field study. PHP is the most widely used language present in web applications; we used it for the weak typed programming language study. [1] Due to time constraints, other programming languages like PERL could not be considered. Given the high number of security problems found, we only used six web applications: PHP-Nuke (phpnuke.org), Drupal (drupal.org), PHP-Fusion (phpfusion.co.uk), Word Press (wordpress.org), phpMyAdmin (phpmyadmin.net), and phpBB (phpbb.com). For the strong typed programming languages, for which we found less security problems, we used 11 web applications developed in Java, C#, and VB: JForum (jforum.net), OpenCMS (opencms.org), BlojSom (sourceforge.net/projects/blojsom), Roller WebLogger (rollerweblogger.org), JSPWiki (jspwiki.org), SubText (subtextproject.com), Dot-NetNuke (dotnetnuke.com), YetAnotherForum (yetanotherforum.net), BugTracker.NET (ifdefined.com/bugtrackernet.html), Deki Wiki (developer.mindtouch.com), and ScrewTurn Wiki (screwturn.eu).

Drupal, PHP-Fusion and phpBB are Web Content Management Systems (CMS). A CMS is an application that allows an individual or a community of users to easily create and administrate web sites that publish a variety of contents. The created sites can go from personal web pages and community portals to corporate and e-commerce applications. Drupal won first place at the 2007 Open Source CMS Award [19]. PHP-Fusion was one of the five award overall winner finalists at the 2007 Open Source CMS Award [19] and has a large community of users working with it. Finally, phpBB is the most widely used Open Source forum solution. phpBB was the winner of the 2007 SourceForge Community Choice Awards for Best Project for Communications [18]. PHP-Nuke is a well-known web based news automation system built as a community portal. The news can be submitted by registered users and commented by the community. WordPress is a personal blog publishing platform that also supports the creation of easy to administrate web sites. A Google search of WordPress pages using the text "Proudly powered by WordPress", which is at the bottom of WordPress based sites, finds over about 7 million pages or more which are currently powered by WordPress. phpMyAdmin is a web based MySQL administration tool. It is one of the most popular PHP applications and has a very large community of users. phpMyAdmin is available in 47 languages, is included in many Linux distributions and was the winner of the 2007 SourceForge Community Choice Awards for Best Tool or Utility for SysAdmins [18].

D. *INJECT MALICIOUS CODE*

The first approach is based on web application analysis from the user side, assuming that source code of an application is not available [17]. The idea is to submit various malicious patterns (implementing for example SQL injection or cross-site scripting attacks) into web application forms and to analyze its output afterwards. If any application errors are observed an assumption of potential vulnerability is made. This tactic does not guarantee neither exactness nor completeness of the obtained results.

• Another tactic to model input validation vulnerabilities is to model syntactic configuration for sensitive operations arguments. The notion behind this is that the web application is susceptible to an injection attack, if syntactic configuration for searching operation arguments depends on the user input. This approach was implemented by means of string analysis in static [16, 17] and it was applied to detect SQL injection [19] and XSS [18].

Architecture Diagrams

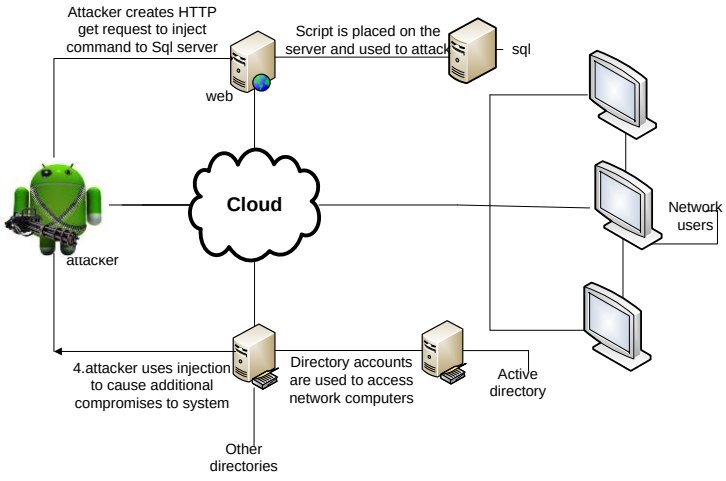

Fig. 1 A sample SQL injection diagram showing how an attack takes place.

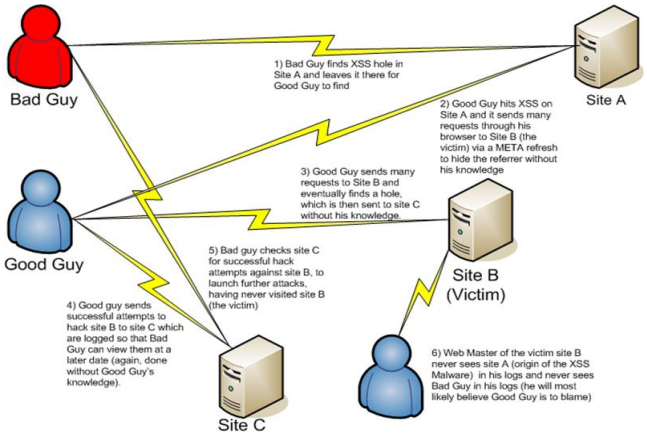

Fig. 2 Sample diagram showing how cross site scripting occurs

VI. SQL INJECTION EXPLAINED

A SQL injection attack as the name implies, involves the changes that are made to SQL statements which are used within a web application through the use of data supplied by an attack. This is made possible due to insufficient input validation and inappropriate construction of SQL statements in web applications which expose them to SQL injection attacks.

SQL injection is a prevalent and could be potentially destructive. The Open Web Application Security Project (OWASP) lists it as the top most threat to web applications.

Implications of Successful SQL Injection Attacks

The effects of a successful SQL injection attack may differ based on the victim application and the methods used by that application in processing data supplied by users, generally SQL injection can be used to for the following types of attacks:

•***Information Leak:*** This attack allows an attacker to acquire, either sensitive information in a database directly or indirectly.

•***Compromised Data Integrity:*** This is an attack which involves the modification of the contents of a database. An attacker could use this attack to ruin a web page or more likely to introduce malicious content into otherwise safe web pages.

• ***Avoid Authentication***: This is an attack which allows an attacker to log on to an application, possibly with administrative rights, without providing a valid username and password.

•***Hinder Availability of Data***: An attacker is able to delete information with the intention causing harm or delete log and review information in a database.

An Example of SQL Injection for Authentication Bypass

One of the many possible uses for SQL injection involves evading an application login process. The following example shows the general operation of a SQL injection attack. The following HTML form asks login information from an application user.

```
<form method='post' action="User_login">
        <tr>
        <td><b><font color="#2a2040"> USER NAME :
</font></b></td>
        <td><font style="" color="#00c7cc" size="7"><input
type="text" name="u1" placeholder='USER
NAME"'/></font></td>

</tr>

<tr>
        <td><b><font color="#2a2040">PASSWORD :
</font></b></td>
        <td><input type="password" color="#00c7cc"
name="p1" placeholder='PASSWORD"'/></td>
        </tr>

        </table>
<input class="submit full-width" type="submit"
value="USER LOGIN"/>

</center>
    </form>
```

When a user enters his or her information into this form and clicks submit, the browser sends a string to the web server that contains the user's credentials.

An application with a vulnerable login procedure may accept the submitted information and use it as part of the following SQL statement, which locates a user profile that contains the submitted username and password:

*select * from Users where (username = 'submittedUser' and password = 'submittedPassword');*

If an application does not use strict input validation methods, the application could be vulnerable to a SQL injection attack. For instance, an application which accepts and processes data supplied by user without any validation, an attacker can submit a maliciously constructed username and password.

Consider the following string sent by an attacker:

username=admin%36%29+--+&password=+

When this string is received and the URL is decoded, the application will attempt to build an SQL statement using the username admin') -- and a password that contains of a single space. The SQL statement produces:
select * from Users where (username = 'admin') -- and password = ' ')
As seen above, the attacker-constructed username changes the logic of the SQL statement to successfully remove the password check. In the above example, an attacker could effectively log in to the application using the admin account without any information about the password to that account.
The string of two dash characters (--) that appears in the input is very significant; it indicates to the database server that the continuing characters in the SQL statement are a comment and should be ignored. This skill is one of the most vital tools that is available to an attacker and without which, it would be difficult to guarantee that the malicious SQL statements were syntactically correct

A. PREVENTING ATTACKS ON WEB APPLICATIONS

Due to the level to which SQL injection and Cross site scripting (XSS) affect web applications it is of great importance to be aware of the means by which these attacks can be prevented in web applications.
There are various methods which have been proposed by researchers for the prevention of these attacks, such methods include;

B. SQL INJECTION PREVENTION:

i. Input validation:

SQL injection can prevented if you adopt an input validation technique whereby user's input is authenticated against a set of distinct rules for length, type and syntax and also against business rules. This can be done in the form of whitelisting or blacklisting, and the structure of SQL statements such that data supplied user cannot influence the logic of the statement.

blacklisting and whitelisting

Within an application itself, there are two approaches to input validation that can defend against SQL injection attacks:
- blacklisting
- Whitelisting

A. Blacklisting

The method of blacklisting involves the removal of known and specific malicious characters from the user input, or simply replacing them. This method is usually applied due to the ease at which it is accomplished, but it is not as effective as whitelisting. The approach of blacklisting could fail because it is impossible to have knowledge of all the malicious characters that could be used to attack an application; it is made worse by the fact that new characters are created by attackers every now and then, and if not properly handled could allow an attacker to disrupt filters and potentially inject SQL statements.

A. Whitelisting

Whitelisting in turn examines each piece of user input against a list of characters permitted. This method is more effective in alleviating the risk of SQL injection, as it is more restraining concerning which types of input are allowed in the form. A whitelisting which is well-implemented should examine each piece of data supplied by user against the expected data format.
The input validation and sanitization functions that are used to filter SQL injection supporting characters can potentially be generalized and used to filter characters that are indicative of cross-site scripting attacks.
The application must perform a precise check when it receives invalid characters from a user. Dissimilar levels of response should be applied based on the conditions in which the unexpected data is used and the effect it could have.
An example, applications should reject users that submit
- Two dash characters (--)
- A semi-colon character (;)
- Apostrophe (‘)

As part of the login name or password, and a high-severity alert should be sent to application administrators. This could be an issue if the input containing the above character is part of an address, it should still be alerted to the administrator.
However looking at it from a security standpoint, it must be assumed that any and all types of data will originate from a user's browser, notwithstanding the safeguards that are placed on the client side. Nonetheless, client-side data validation methods can enhance application usability.

ii. Strong typing:

Usage of strongly typed parameterized query APIs with placeholder replacement markers, even when calling stored procedures. If these techniques perform all required escaping of dangerous characters before the SQL statement is passed to the underlying database system.

The following example represents the use of prepared statements in Java:

```
String sql = "select * from Users where (username = ? and password = ?)";
PreparedStatement preparedStmt = connection.prepareStatement(sql);
preparedStmt.setString(1, Username);
preparedStmt.setString(2, Password);
results = preparedStmt.executeQuery();
```

It should be noted that prepared statement if used incorrectly is considered insecure and could leave the application open to SQL injection attacks.

iii. Use of explanatory error messages:

An application should be able to catch and remove all SQL-generated error messages before they reach a user. Depending on the structure of the error message, each piece of information could assist attackers in creating their application input, and also increases the possibility of the attack being successful.

iv. Stored procedures:

Show care when using stored procedures since they are generally safe from injection. However, careful as they can be injectable (such as via the use of exec() or concatenating arguments within the stored procedure).

v. Use least privileges:

Attention should be taken ensure that users with the permission to access the database have the least privileges. Likewise, you should always make sure that a database user is created only for a specific application and this user is not able to access other applications. Another method for preventing SQL injection attacks is to remove all stored procedures that are not in use [21].

VII. CROSS SITE SCRIPTING EXPLAINED

Cross site scripting can simply be described as an injection of client side scripts into a website. These scripts can be HTML scripts or JavaScript scripts. According to White Hat Security Top Ten which states that more than 50% of the websites are vulnerable to cross site scripting

Cross site scripting (XSS) can be used in a different ways to cause severe problems. The customary use of XSS is that it gives an attacker the ability to steal session cookies thereby allowing an attacker to pose as a victim. The use of XSS is not only about stealing cookies, XSS has been used by attackers to cause destruction on social networks, steal credentials, spread viruses, deface websites.

Cross-site Scripting (XSS) can be classified into three major categories

- Stored XSS,
- Reflected XSS
- DOM-based XSS

A. *Stored XSS*

Stored XSS also known as Persistent XSS is the most destructive type of XSS. This type of attacks involves an attacker injecting a script (also called payload) on the target application. This payload is permanently stored on the target application for example within a database.

B. *Reflected XSS*

This is the second, and it is also the most common type of XSS. In Reflected XSS, the attacker's payload script must be placed in such a way that becomes part of the request which is sent to the web server and echoed back in such a way that the HTTP response includes the payload from the HTTP request.

For this method to be successful the attacker lures the victims to make a request to the server which comprises of the XSS script and thus ends-up executing the script that is reflected and implemented inside the browser using methods such as; Phishing emails and other social engineering techniques. Attackers mostly make use of social networks for the distribution of Reflected XSS attacks.

C. *DOM-based XSS*

DOM-based XSS is a modified version of both persistent and reflected XSS. In a DOM-based XSS attack, the malicious string is not actually parsed by the victim's browser until the website's legitimate JavaScript is executed.

First the attacker crafts a URL containing a malicious string and sends it to the victim; this URL is used to trick the victim into requesting the URL from the website. When the website receives the request, it does not include the malicious string in the response, thus allowing the victim's browser to execute the legitimate script inside the response, causing the malicious script to be inserted into the page, while the victim's browser executes the malicious script inserted into the page, it sends the victim's cookies to the attacker's server.[22]

The difference between DOM based and the other two types of XSS is that in the other two types, the server inserts the malicious script into the page, which is then sent in a reply to the victim. When the victim's browser receives the reply, it assumes the malicious script to be part of the page's valid content and automatically executes it during page load as with any other valid script. While in DOM based there is no malicious script inserted as part of the page; the only script that is automatically executed during page load is a valid part

of the page. The malicious JavaScript is executed at some point after the page has loaded, as a result of the page's legitimate JavaScript treating user input in an unsafe way.

D. PREVENTION OF CROSS SITE SCRIPTING

XSS can only be prevented by carefully sanitizing all input which is not known to be secure. [20] Classes of input which is known NOT to be secure include:

- HTTP referrer object
- The URL
- GET parameters
- POST parameters
- Window.location
- Document.referrer
- document.location
- document.URLUnencoded
- All headers
- Cookie data
- Potentially data from your own database (if not properly validated on input)[20]

i. Encoding:

A common technique for preventing XSS vulnerabilities is "escaping". The purpose of character and string escaping is to make sure that every part of a string is interpreted as a string primitive, not as a control character or code. Encoding is the action of escaping user input so that the browser reads it only as data, not as code. The most recognizable type of encoding in web development is HTML escaping, it converts characters like < and > into < and > respectively. [22]

ii. Validation

Validation is the action of sieving user input so that all malicious parts of it are detached, without necessarily removing all code in it. One of the most noticeable types of validation in web development is allowing some HTML elements (such as and) but disallowing others (such as <script>). [22]

VIII. RESULT

The results of this analysis show that a wide range of things could cause an application to be vulnerable to SQL injection and cross site scripting. These reasons do not necessarily all have to do with errors in coding but could also be as a result of the Human factor, it is usually said that the weakest link in any security situation is the human factor.

The cause of security vulnerabilities in web applications includes but is not limited to the following;

- Buffer overflows
- Unvalidated input
- Race conditions
- Access-control problems
- Weaknesses in authentication and/or authorization

I was also able to run the preventive measures (codes) using different parameters e.g.

- stored procedure,
- input validation
- None explanatory error messages.

There is no perfect system we can only take measures to minimize the effect of attack on applications.

IX. CONCLUSION

This paper analyzes vulnerabilities and exploits of some web applications using field data on past security fixes. Some web applications were written in a weak typed language and others in strong typed languages. Results suggest that applications written with strong typed languages have a smaller number of reported vulnerabilities and exploits. Consideration was made to more strong typed applications to obtain a fair amount of vulnerabilities when compared to the weak typed.

According to findings, weak typed are the preferred targets for the development of exploits. We saw that the fault types responsible for XSS and SQLi belong to a narrow list, which points a path to the improvement of web applications, namely in the context of code inspections and the use of tools for static analysis. This study showed that the way programmers fix vulnerabilities seems to have a degree of dependence with the type of language used. However, the number of vulnerabilities analyzed in this and other studies show that there is no guarantee of a completely safe application even when a strong typed language is used in building that application and does not prevent vulnerabilities. No application is completely secure.

X. FUTURE WORK

The easiest and probably most preferred form of prevention for this type of attack is to restrict the valid input to be free of characters that have special meanings under the HTML specification. For example, if the value of a user input should be a number, and is validated by the web application as such, we are sure that it cannot be used to launch a cross-site scripting attack. A collective problem in software development is that software developers tend to allow too much freedom in terms of what values an input can take. They do not ask themselves relevant questions like, Does an input value have to allow for characters such as greater than (<) and

double quotes(“), single quotes (‘) ? Many developers ignore such issues at design time or choose to allow unnecessary flexibility. Restricting the input set can greatly simplify a program and reduce its vulnerability.

REFERENCES

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

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

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

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

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

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

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

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

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

[10] T. Manjaly, “C# Coding Standards and Best Practices,” http://www.codeproject.com/KB/cs/c__coding_standards.aspx, May 2013.[11] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, second ed., Lawrence Erlbaum, 1988.

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

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

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

[15] Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: A client-side solution for mitigating cross-site scripting attacks. In The 21st ACM Symposium on Applied Computing (SAC 2006), 2006.

[16] Curphey, M., Wiesman, A., Van der Stock, A., Stirbei, R.: ―A Guide to Building Secure Web Applications and Web Services‖. OWASP (2005).

[17] Andrews, M.: ―The State of Web Security‖. IEEE Security & Privacy, vol. 4, no. 4, pp. 14-15 (2006).

[18] sourceforge, December, 2007, http://sourceforge.net/community/index.php/2007108/0l/community-choice-awards-winnersl

[19] Packt Publishing Ltd, December, 2007, http://www.packtpub.com

[20] Golem Technologies http://www.golemtechnologies.com /articles/prevent-xss#how-to-prevent-cross-site-scripting

[21] Veracode http://www.veracode.com/security/sql-in.

[22] Excess XSS a comprehensive tutorial on Cross site scripting http://excess-xss.com/

[23] Cisco online Resource, http://www.cisco.com

[24] Symantec Online Resource, www.symantec.com