Detection and Fixing of web application vulnerabilities on field data

Michael Joshuva A.[1], Savaridassan P.[2]
[1] SRM UNIVERSITY / Information Security and Cyber Forensics, Chennai, India
[2] SRM UNIVERSITY / Information Security and Cyber Forensics, Chennai, India

Abstract— **The detection process for security vulnerabilities in ASP.NET websites/applications is a complex process, where most of the code is written by somebody else and there is no documentation to determine the purpose of some code. Other factors due to the fact that ASP.NET which is part of .NET framework that separate the HTML code from the programming code in two files, one for (aspx) file and another for the programming code depending on the compiled language (Visual Basic VB, C sharp C#, Java Script). Since the VB and C# are the most common languages in use around the world with ASP.NET files, we have adopted those compiled languages in the construction of our proposed algorithm in addition to aspx files. Therefore, the scanning process inspects at least those three types of files (aspx, VB and C#).**

Index Terms— **Security, Vulnerabilities, Languages, Web Applications, Second-order injection, Vulnerability Analysis, Penetration Testing, Dynamic Analysis, Taint Analysis**

I. INTRODUCTION

In the field of information security, weakness occurs due to the presence of flaw in its security that allows an attacker to reduce a system's information confidentiality, integrity and availability (CIA). Security and protection are very important in the field of information technology and modern communications through internet. The Web Application Vulnerability Assessment is a method to test that assesses the security of interactive applications using web technologies such as e-banking, news and e-commerce web applications. Web applications are, programs that allows website viewers to submit and access data from/to a database using the Internet by their familiar web browser. Within their browser the data is then provided to the user as information is dynamically generated in a particular format for example HTML using CSS by the web application via web server

For web applications, one of the most common security problem in the web application is SQL command injection attacks. Many people who are using internet nowadays says that they know what SQL injection is, but all they have read, heard about or experienced from their insignificant examples. SQL injection is one of the most Devastating vulnerabilities in the web application security to impact a business, if it is exploited that leads to the exposure of all Sensitive information stored in an application's database, that includes classified information such as names, passwords, usernames, credit card details, addresses and mobile numbers.

Web applications and its supporting technologies and environments use different technologies and that contains a remarkable amount of customized and modified code. Nowadays there is a lot of improvement in the network security technology market and it leads to limit the opportunities to exploit information systems via network based vulnerabilities, although hackers are rapidly developing their skills and switching their focus to find the faults in the web applications and attempting to compromise the security

To prevent these security problems from occurring and also understand the importance of typical software faults. This project contributes to this body of knowledge by presenting a field study on one of the most widely spread and critical web application vulnerabilities like SQL Injection. It compares and analyzes the source code of widely used web applications that written in .net languages and its security patches

This paper also contributes to fill this gap by presenting a study on characteristics of source code defects generating major web application vulnerabilities. The main research goal is to understand the typical software faults that are behind the majority of web application vulnerabilities, taking into account different programming languages. To understand the relevance of these kinds of vulnerabilities for the attackers, the paper also analyzes the code used to exploit them

II. RELATED WORK

It is not possible to create large applications without vulnerabilities and, even if it occurs, it is not feasible to find it, repeat it and prove it scientifically [Hatton01]. Software programmers cannot promise the security and quality of code scalability and sustainability, even when security is defined

ISSN: 2278 – 1323

from the ground up [Giorgini05]. The main reason that leads to safety problems seems to be connected to learn how unusual programming languages are different in terms of natural inclination for mistakes. Clowes analysed common security related problems to the easiness in programming with ASP.NET and its features, but this property affects few other programming languages. The options to choose the strong type system or weak type system and the checking type i.e. dynamic or static of the programming language also affects the robustness of the software. For example, a static type checking for a strong typed language that can help to produce a secure application without affecting its performance. Scholte discussed an detailed study in three programming languages on a variety of input validation vulnerabilities developed [Scholte12]. This work focused mainly on the relationship between the particular programming languages used and their vulnerabilities that are most commonly reported are not elaborate into details in what concerns the root cause of typical software faults that generate vulnerabilities, it seems like we do in the present work. One of the best practices to find software faults is by perform a static analysis to the program code [Livshit05]. This is a labor-intensive job, normally perform with automated tools, although they lack the precision of the manual counterpart. To improve and help the predict software failures in strong typed languages a new defect classification algorithm was proposed [Nagappan04].

Another research work proposed a security resources indicator that seems to be strongly correlated with change in vulnerability density over time [Walden09]. Web application vulnerabilities have been addressed by recent studies from several points of view, but without any code analysis. To overcome the low level of detail of existing vulnerability databases, some researchers proposed approaches based on the market, instead of on software engineering [Ozment07]. The attacker's perspective has also been of some focus in the literature, but mainly through empirical data gathered by the authors highlighting social networking and what could be obtained from attacking specific vulnerabilities. Some studies analyzed the attacks from the victim's perspective, including the proposal of a taxonomy to classify attacks based on their similarities and the analysis of attack traces from Honey Pots to separate the attack types [Cukier06]. There is, however, a lack of knowledge about existing exploits and their correlation with the vulnerabilities

III VULNERABILITIES AND PROGRAMMING LANGUAGES

The Open Web Application Security Project Report populated the top most critical web application security risks, with SQLi at the top, followed by XSS [OWASP10]. Several studies also found SQLi and XSS as the most widespread vulnerabilities. Fig 1 depicts the annual percentage of disclosed SQLi and XSS among all the causes of web application vulnerabilities depicting that they are increasing over time [Neuhaus10]. SQLi attacks takes advantage of the unconstrained input fields in the web application interface to maliciously pinch the SQL query sent to the back-end database.

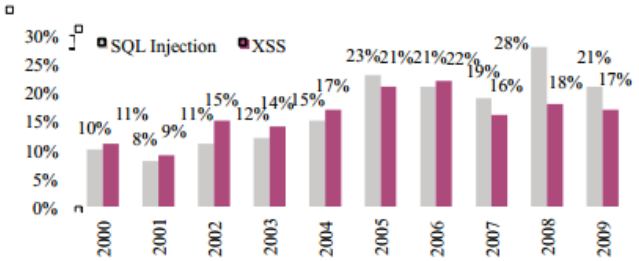

Fig 1. Analyzed reports of attack evolution of XSS and SQLi causes of vulnerabilities [Neuhaus10].

By using XSS vulnerability, the attacker will be able to insert into web pages some unintentional/unwanted client-side script code, usually in JavaScript and HTML. XSS and SQLi provides a way for the attackers to access unauthorized data (read, insert, change or delete), gain their way in to confidential database accounts, masquerade as other users (such as the administrator), impersonate web applications, deface web pages, view and control remote files on the server, inject and carry out server side programs that allow the construction of botnets controlled by the attacker, etc. Information on the most common vulnerabilities, including SQLi and XSS, along with the detail of their survival, attacks, best practices to avoid, detect and diminish them can be found in many referenced works. Many different programming languages are currently being used to develop web applications. Ranging from proprietary languages (e.g., C#, VB) to open source languages (e.g., PHP, CGI, Perl, Java), the range of languages available for web development is immense. Programming languages can be classified based on taxonomies, like the programming paradigm, the type system, the execution mode, etc. The type system, is particularly important in the context of the present work, it specifies how data types and data structures are managed and constructed by the programming language, namely how the language maps values and expressions into types, how it manipulates these types, and how these types correlate. Regarding the sort system they can be typed vs. untyped, static vs. dynamic typed, weak strong typed. In particular strong typed languages provides the means to produce more healthy software, since a value of one type cannot be treated

ISSN: 2278 – 1323

as another type (e.g., a string cannot be treated as a number), as in weak typed languages.

One of the main contributions of this work is to help understanding the collision of the type system in the security of web applications. This is of particular significance, as critical security vulnerabilities like SQLi and XSS are strongly related to the way the programming language manages data types [OWASP10]. For example, it is very common to find attacks that inject SQL code by taking advantage of variables that supposedly should not be strings (e.g., numbers, dates) as the type of the variable is determined based on the assigned value. On the other hand, in strong typed languages this is impossible because the type of variables is determined before runtime and any attempt to store a string in a variable of any other type raises an error. However, this does not prevent the occurrence of vulnerabilities in strong typed languages, but only by taking advantage of string variables. In fact, although Java is a safe programming language [CERT09] and it is a strong typed language, vulnerabilities can be found in Java programs due to implementation faults.

IV CLASSIFICATION OF SOFTWARE FAULTS – SECURITY VULNERABILITIES

After choosing a vulnerable web application we find the web for all analyzed XSS and SQLi patches that were differentiated based on the work presented. This type of classification is originate from the code imperfection types (task, interface , checking and algorithm) of the software fault types. As ODC fault types are silent too broad [Durães06] we explained them according to the nature of the fault: missing construct, irrelevant construct and wrong construct. All the security vulnerabilities reported could be differentiated using only 15 of the defect types that already identified in [Durães06] and one extra defect type. The Missing Function Call Extended (MFCE), however not everything was found in both strong typed and weak typed web applications. The Missing Function Call Extended is a new count and it is based on a lost function in situations where the return value is used in the code (as contrasting to the MFC where the return value is not used).

A) SQL injection

SQL Injection is a type of attack that can happen when an application uses user input that has not been validated to see whether it is valid and the malicious user uses this malicious input to exploit classified information from the database.

For example, the user can use the following malicious input:' OR 1=1 – to penetrate in to the database.
This would roll the database query into:

SELECT auth_lastname, auth_firstname FROM authors WHERE auth_id = '' OR 0=0 --

Since 0=0 always evaluates to true, this query will returns more than 0 rows.

B) XSS (Cross Site Scripting)

Cross Site Scripting mainly occurs in dynamic web pages that are integration of browser data (HTML) with the code (<script> tag) which is rooted in the data. The script can be (VBScript, JavaScript, ActiveX, Flash or HTML) .The main intention of 'XSS' is to control client-side scripts of a web application to accomplish in the manner desired by the malicious user.

There are two main types of Cross Site Scripting:
1) Stored Cross Site Scripting
2) Reflected Cross Site Scripting

1) Stored cross site scripting:

The stored (or persistent) Cross Site Scripting occurs when the data given by the hacker is saved by the server and then displayed lastingly on "normal" pages returned to other users. Stored XSS needs particular kind of vulnerability in the web application where the information is placed in somewhere (eg. Data base) and soon after feedback is mail to the user, this can be done through Blog, Forum, etc. The hacker can send <JavaScript> or <HTML> to the application as a replacement for the normal input to be stored in the data base, soon after when the victim comes to their application web site he/she will access and download the <JavaScript> or <HTML> located there. The application here works for the hacker but acts as an attack site.

2) *Reflected Cross Site Scripting*:

Reflected (or non-persistent) Cross Site Scripting can occur when the input data given by a web client, most probably in HTTP query parameters or in HTML form submissions. It is used instantly by server-side scripts to produce a page of results and reflected back for the user, without sanitizing the request. For example, if we have a user Log-In process (User-Id, Password) and the user has provided his Log-In information. Incase the user enter his Password incorrectly, he may get a message like ("Sorry, unauthorized user Log In"), but sometimes we get a message like ("Sorry john, , unauthorized Log-In") and here's the difficulty, where the (user name) sends and reflected back to the output. If there is no input check for Log-In text boxes, the malicious user can exploit this vulnerability to insert his malicious input 'XSS' as a substitute of User-Id. The hacker can craft an email contains a link request from the user to make him to click on the link to update classified data.

ISSN: 2278 – 1323

C) Hijack Session:

The attacker needs the cookie form the victim to hijack the session. This is can be implemented by creating one form and make it submit to the attacker site.
Example:
</form><form name = 'a' action = 'attackersiteaddress'
method = 'post'>
<input type = hidden value = '<script> + document.cookie +
</script>'>
</form>
<script> a.submit() </script>

D) Cookie Poisoning:

The attacker can damage the value of the cookie if he identify that an application is relying on the cookie value to exhibit specific exploit done by the user with "response.write". Presume the application accumulate the value of the last search done by the user along with the date-time in cookies.
Example:
The hacker here can keep informed the value of the last search with a herf pointing to his site as following:
<script> document.cookie.userlastsearch = '<A herf =
"attackersiteAddress"> You have picked up a random prize
please click here to continue </A>'
</script>
There he may inquire the user to log-in again to fool him with a offer. The attacker may attract him with 5$ and latter ask him to give 50$ for some magnificent product. The amount not the main target but the details of credit card.

E) IFrame:

The <iframe> tag specifies an inline frame. The main use of inline frame is to embed another document within the current HTML document. The attacker can simply trick the user by showing the UI that has size 100% in height and width to look the same as an application site through writing the following malicious code:
<iframe SRC="attacker site" height = "100%" width ="100%">
From the previous examples, we reach to the main reasons that make an application susceptible to Cross Site Scripting attacks:
1. There is no input validation control for the inputs coming to an application.
2. There is no sanitization control for the output coming from the application.

V FAULT DETECTION

The Fault detection mechanism uses the various below methods to detect the vulnerabilities. The recommended algorithm performs a scanning process for all application files /website. This scanner tool relies on analyzing the source code of the application depends on ASP.NET files and the code following files (C sharp C# and Visual Basic VB).To detect the security vulnerabilities and leaks. Therefore, the scanner tool tries to detect the vulnerabilities that can help hackers from the reflected output or messages, check most of the ASP.NET server controls and the commands in the code behind that interact with the database. The below table represent the output of the detection process and it shows the vulnerabilities presence in the web application by showing the fault line, name of the fault and the fault location.

Table 1: Output of the fault detection

FileName	Fault Line	Fault	Fault Locatioin
FileName :F\201\web	271	Input Box should ..**	The TextBox ID ="txtuname" is not Validate
FileName :F\201\web	321	In web config Ac..	The TextBox ID ="txtuname" is not Validate
FileName :F\201\web	46	Input Box should ..**	The TexBox for=<"txtWebsite>...*
FileName :F\201\web	51	Input Box should..**	The TexBox for=<"txtWebsite>...*
FileName :F\201\web	55	Input Box should ..**	The TexBox for=<"txtWebsite>...*.
FileName :F\201\web	60	Input Box should ..**	The TexBox for=<"txtWebsite>...*
FileName :F\201\web	77	Input Box should ..**	The TexBox for=<"txtWebsite>...*.
FileName :F\201\web	84	Input Box should ..**	The TexBox for=<"txtWebsite>...*.
FileName :F\201\web	112	Input Box should ..**	The TexBox for=<"txtWebsite>...*.
FileName :F\201\web	117	Input Box should ..**	The TexBox for=<"txtWebsite>...*...
FileName :F\201\web	121	Input Box should ..**	The TexBox for=<"txtWebsite>...*...
FileName :F\201\web	126	Input Box should ..**	The TexBox for=<"txtWebsite>...*..
FileName :F\201\web	38	No parameters d..	Nil

***** The TexBox for=<"txtWebsite>Website</label> is not validate
****** Input Box should be validated using either Regular Expression Validator or Range Validator

VI FAULT RECOVERY

After scanning process, it will generate a report list all the discovered leaks and vulnerabilities by displaying the

ISSN: 2278 – 1323

name of the infected file, the description and its location. The recommended algorithm will help association to fix the vulnerabilities and recover the overall security**.** This report requires a response from the organization to take the necessary corrections steps.

A. Parameterized queries in .NET (C#)

Microsoft .NET provides a number of various ways to use the parameterize statements in ADO.NET Framework. ADO.NET also opted for some additional functionality, allows you to advance check the parameters supplied; such as by using type-checking the data that you are passing in.ADO.NET provides four variety of data providers, depending on the nature of database that is being accessed: System.Data.OracleClient is for Oracle databases, *System.Data.SqlClient* is for Microsoft SQL Server and System.Data.Odbc and System.Data.OleDb for ODBC and OLE DB data sources, respectively. It depends on which provider you use on the database drivers and server being used to access the database. Regrettably, the syntax for using parameterized statements varies among the providers, especially in how the parameters and statement are specified. shows how parameters are precise in each provider.

Table 2: ADO.NET Data Providers, and Parameter Naming Syntax

Data Provider	Parameter Syntax
System.Data.SqlClient	@parameter
System.Data.OracleClient	:parameter(only in parameterized SQL command text)
System.Data.OleDb	Positional parameters with a question mark placeholder (?)
System.Data.Odbc	Positional parameters with a question mark placeholder (?

The following example shows the vulnerable example query rewritten as a parameterized statement in .NET using the SqlClientprovider:

```
SqlConnection con = new SqlConnection(ConnectionString);
String Sql = "SELECT * FROM users WHERE username=@username" + "AND passwd=@passwd";
cmd = new SqlCommand(Sql, con);

// Add parameters to SQL query
cmd.Parameters.Add("@username",   // name
SqlDbType.NVarChar,   // data type 16);   // length
 cmd.Parameters.Add("@passwd",SqlDbType.NVarChar, 16);
cmd.Parameters.Value["@username"] = username;   // set parameters
cmd.Parameters.Value["@passwd"] = passwd;  // to supplied values
reader = cmd.ExecuteReader();
```

B. Validating Input in .NET(C#)

ASP.NET features a number of built-in controls that you can use for input validation, the most useful of which are the CustomValidatorcontrol and RegularExpressionValidator control. Using these controls with an ASP.NET application provides the added advantage that client-side validation will also be performed, which will develop the user experience in case the user genuinely enters erroneous input. The following code is an example of the use of RegularExpressionValidator to validate that a username contains only letters (uppercase and lowercase) and is between seven and 11 characters long:

```
<asp:textbox id="userNme" runat="server"/>
<asp:RegularExpressionValidator id="usernmeRegEx" runat="server"
ControlToValidate="userNme"
ErrorMessage="Username must contain 7–11 letters only."
ValidationExpression="^[a-zA-Z]{8,12}$" />
```

C. Using Stored Procedures

One design practice that can mitigate or prevent the impact of SQL injection is to design the application to exclusively use stored procedures for accessing the database. Stored procedures are programs saved within the database and you can rewrite them in a various languages and variants depending on the database, such as SQL (Transact-SQL for SQL Server, PL/SQL for Oracle, SQL:2003 standard for MySQL), Java (Oracle), or others.

Stored procedures can be very useful for mitigating the significance of a possible SQL injection vulnerability, as it is possible to construct access controls at the database level when using stored procedures on most databases. This is important, because it means that if an exploitable SQL injection issue is found, the attacker should not be able to access classified information within the database if the permissions are correctly configured. This happens because dynamic SQL, due to its dynamic nature, needs more permissions on the database than the application harshly needs. As dynamic SQL is assembled at the application, or elsewhere in the database, and is then sent to the database for execution, all data within the database that needs to be writable, readable, or updateable by the application needs to be handy to the database user account that is used to contact the database. Therefore, when an SQL injection problem occurs, the attacker can potentially access all of the sensitive

ISSN: 2278 – 1323

data within the database that is accessible to the application, as the attacker will have the database permissions of the application.

With the use of stored procedures, you can change this circumstance. In this case, you would create stored procedures to carry out all of the database access the application needs.

The database user that the application uses to access the database is given access to execute the stored procedures that the application requirements, but does not have any other data access within the database (i.e., the user account does not have SELECT, DELETE, INSERT, or UPDATE rights to any of the application's data, but it have EXECUTE rights on the stored procedures). The stored procedures then access the data with contradictory permissions—for example, the permissions of the user who created the procedure rather than the user invoking the procedure—and can act together with the application data as necessary. This can help you to mitigate the collision of an SQL injection issue, as the attacker will be restricted to calling the stored procedures, therefore limiti**)** ng the data the attacker can access or modify, and in many cases preventing the attacker from accessing classified information in the database.

CONCLUSIONS

We have analyzed for verifying a group of secure code properties for web applications. we have offered techniques for the detection of Cross Site Scripting and SQL command injection vulnerabilities in these applications. We also adopted detection techniques to find the vulnerable code in .net web application. The main feature of this project is replacement of the vulnerable code by the patch code. The various vulnerable error codes substitution are done by various techniques of code level defenses like Parameterized statements, Validating input , and Encoding the input. These techniques used to detect the vulnerability in the .Net web application by validating the faults in the source code.

FUTURE ENHANCEMENT

In the current system this techniques are implemented in a one strong typed language i.e C# .NET. In future we plan to demonstrate this technique to find the fault and replace the vulnerabilities in other strong typed languages (Java, VB) and weak typed languages (PHP).

This work can be extended by comparing more vulnerability of web applications written in different languages and developed by independent programmers. Another follow-up work may focus on the importance of the attack surface in the distribution of vulnerabilities and exploits. This may compare different results of vulnerabilities and exploits of both web applications and their add-ons, regarding their size

ACKNOWLEDGMENT

This is to acknowledge our sincere thanks to my advisor **Mr. P. SAVARIDASSAN, M.E,** Assistant Professor, SRM UNIVERSITY and all others who assisted me in bringing out this work successfully.

REFERENCES

[1]wikipedia.org,"ASP.NET"[Online]
http://en.wikipedia.org/wiki/ASP.NET.
May 2 , 2012.
[2] D. White, N. Foster, "What is ASP programming?" [Online]http://www.wisegeek.com/what-is-asp-programming.htm. June
2012.
[3] C. Mackay, "SQL Injection attacks and some tips on how to prevent them" [Online]
http://www.codeproject.com/Articles/9378/SQL-Injection-Attacks-and-Some-Tips-on-How-to-Prev. Jan 2005.
[4][CERT09] US-CERT Vulnerability Notes Database, January, 2009,
http://www.kb.cert.org/vuls/, accessed 1 May 2013
[5] [Cukier06] Cukier, M., Berthier, R., Panjwani, S., Tan, S., "A statistical analysis of attack data to separate attacks". International Conference on Dependable Systems and Networks, pp. 383-392, 2006
[6][Durães06] Durães, J., Madeira, H., "Emulation of Software Faults: a
Field Data Study and a Practical Approach", Transactions on Software Engineering, pp. 849-867, 2006
[7][Giorgini05] Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N., "Modeling Security Requirements through Ownership, Permission and Delegation" IEEE International Conference on Requirements Engineering, pp. 167-176, 2005
[8] [Hatton01] Hatton, L., "The Chimera of Software Quality", IEEE Software, 40(8), 104-103, 2007
[9] J.D. Meier, A. Mackman, B. Wastell, P. Bansod, and A. Wigley "How to: Protect from SQL Injection in ASP.NET." [Online] http://msdn.microsoft.com/enus/
library/ms998271.aspx. May 2005.
[10] B. Jovicic, and D. Simic, "Common web application attack types and security using ASP.NET". Belgrade, Serbia and Montenegro : ComSIS, Vol. 3, No. 2. December 2006.
[11] J. Guillaumier, "Cross Site Scripting - XSS - The Underestimated exploit" [Online]
http://www.acunetix.com/websitesecurity/xss.htm.
[12] IMPREVA Protecting the Data That Drives Business, "Cross-Site Scripting" [Online]
http://www.imperva.com/resources/glossary/cross_site_scripting.html.
[13] I. Poison, "Cross site scripting: Common threats in web Applications" [Online]

ISSN: 2278 – 1323

http://www.codeproject.com/Articles/10732/Cross-sitescripting-Common-threats-in-web-applica. June 2005.
[14] J. Shanmugam1, Dr. M. Ponnavaikko2, " Cross Site Scripting-Latest developments and solutions: A survey". Pilani, India :s.n., Vol. 1, No. 2. September 2008.
[15] Livshits, B., Lam, S., "Finding security vulnerabilities in java applications with static analysis", USENIX Security Symposium, pp. 18-18, 2005
[16] Nagappan, N., Hudepohl, J., Snipes, W., Vouk, M., "Preliminary Results On Using Static Analysis Tools For Software Inspection." International Symposium on Software Reliability Engineering, pp. 429-439, 2004
[17] Neuhaus, S., Zimmermann, T., "Security Trend Analysis with CVE Topic Models" International Symposium on Software Reliability Engineering, pp. 111-120, 2010
[18] Ozment, A., "Vulnerability Discovery & Software Security". PhD thesis, Computer Laboratory Computer Security Group: University of Cambridge, 2007
[19] Safelight of security advisors, "Cross Site Scripting (StoredXSS)demo."[Online]
http://www.youtube.com/watch?v=7MR6U2i5iI.
Jan 2009.
[20] Scholte, T., et al. "An Empirical Analysis of Input Validation Mechanisms", ACM Symposium On Applied Computing, pp. 1419-1426, 2012
[21] Walden, J., Doyle, M., Welch, G., Whelan, M., "Security of Open Source Web Applications" International Symposium on Empirical Software Engineering and Measurement, 2009

ISSN: 2278 – 1323