Web Applications Vulnerabilities Analysis

A Master Thesis project

By
George Iakovakis

Under the supervision of
Dimitris Gritzalis,
Professor in Department of Informatics

January 2016

Department of Informatics
Athens University of Economics & Business
Athens, Greece

Declaration of Originality
and Compliance of Academic Ethics

I hereby declare that this thesis titled "Web Applications Vulnerabilities Analysis" contains literature survey and original research work by me, as part of my degree of Master in Information Systems in the Department of Informatics, Athens University of Economics and Business. All information have been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Acknowledgements

This research would not have been possible without the sincere help and contributions of the many individuals involved directly or indirectly at different levels of my work. I would like to use this opportunity for expressing my sincere gratitude to them.

I would like to thank my supervisor, Professor Dimitris Gritzalis for his kind attention, patience and motivation during this research and the preparation of this thesis. His guidance encouraged me to work hard and inspired me for finding a new way of thinking about security in modern world.

I would also like to thank my thesis advisor, PhD candidate Nikos Tsalis for his purposeful comments and encouragement, but also for his continuous advising for this thesis writing. Finally, I feel obliged to express my deepest gratitude and love to my parents, for their unconditional, constant love and psychological support.

Abstract

From the last twenty five years, the web has evolved into an important part of our lives. There are web applications for every kind of job and activity. In parallel, all this situation is very attractive for attackers who want to manipulate users' data and other sensitive information. For this reason web applications are needed to be secure for every user. In this thesis we discuss many types of web attacks and vulnerabilities, giving examples of the way the attackers exploit these assailable parts. We also discuss about the defense ways and the use of detection and prevention tools. Finally we have to be referred that this analysis is guided by the OWASP Top Ten 2013.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Most of us use Web applications on a daily basis, either as part of our vocation or in order to access our e-mail, book a holiday, purchase a product from an online store, view a news item of interest and so forth. Web applications come in all shapes and sizes. Web applications have become common way for companies to conduct business with the outside world. The static web pages are gone and now, almost every company has its own dynamic, interactive Web application that communicates with their clients. Web application is an application that is accessed by using a web browser to communicate with a web server. They are written in many languages and run on every kind of operating system. One thing that Web applications have in common, regardless of the language in which they were written, is that they are interactive and more often than not, are database driven. Often, these applications are written by programmers not acquainted with security problems behind their Web application front-end or their work is guided by hard to get deadlines so application security is not very high on their priority list. Thus these Web applications become main threat to secrecy and integrity of company's sensitive data.

There is Figure 1 below which shows the high percentage of vulnerability distribution by product type.

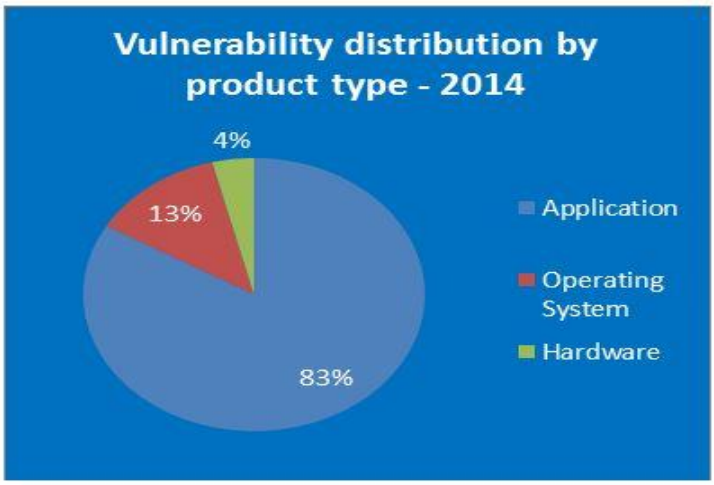
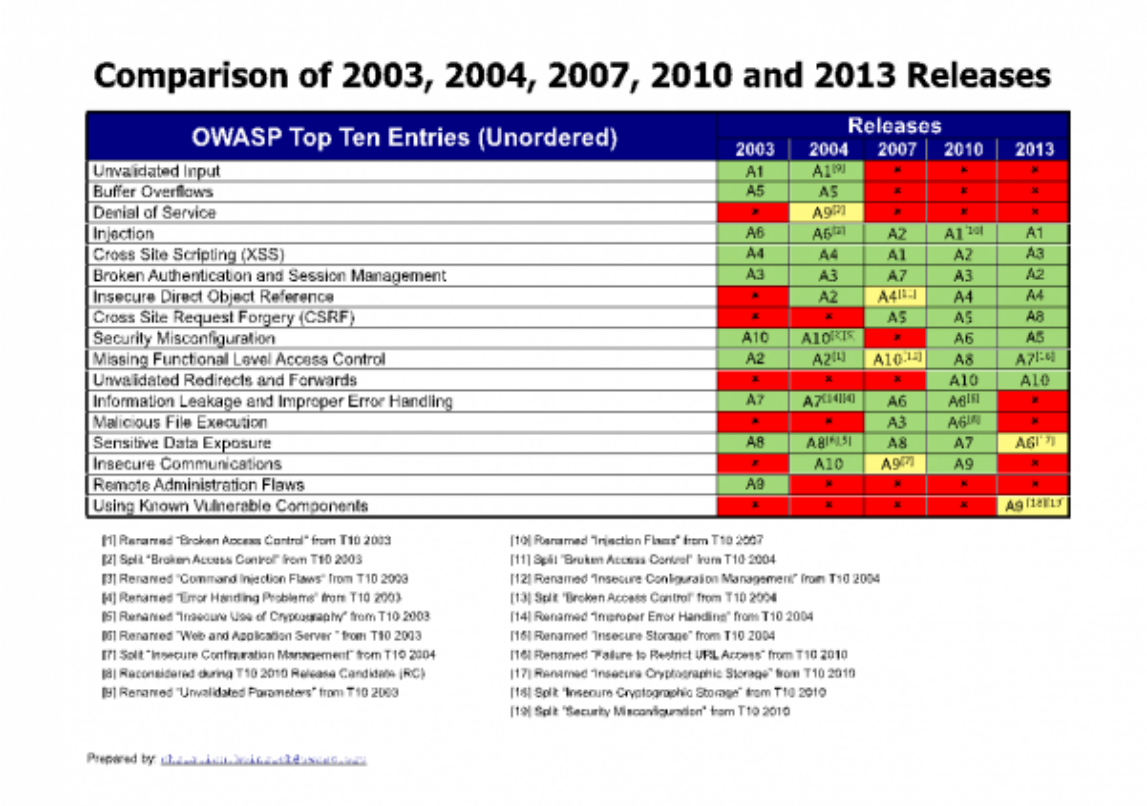


Figure 1: Vulnerability distribution by product type – 2014

The OWASP Foundation came online on December 1st 2001 it was established as a not-for-profit charitable organization in the United States on April 21, 2004 to ensure the ongoing availability and support for this work at OWASP. OWASP is an international organization and the OWASP Foundation supports OWASP efforts

around the world. OWASP is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted. All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security. OWASP advocates approaching application security as a people, process and technology problem because the most effective approaches to application security includes improvements in all of these areas.

The OWASP Top Ten is a powerful awareness document for web application security. It represents a majority opinion about what the most critical web application security flaws are. Project members include a variety of security experts from around the world who have shared their expertise to produce this list. The figure 2 shows the Top Ten lists from 2003 to 2013 and how the list has changed over the years (Owasp.org, 2016).



Comparison of 2003, 2004, 2007, 2010 and 2013 Releases

OWASP Top Ten Entries (Unordered)	Releases				
	2003	2004	2007	2010	2013
Unvalidated Input	A1	A1[9]	*	*	*
Buffer Overflows	A5	A5	*	*	*
Denial of Service	*	A9[7]	*	*	*
Injection	A6	A6[3]	A2	A1[10]	A1
Cross Site Scripting (XSS)	A4	A4	A1	A2	A3
Broken Authentication and Session Management	A3	A3	A7	A3	A2
Insecure Direct Object Reference	*	A2	A4[11]	A4	A4
Cross Site Request Forgery (CSRF)	*	*	A5	A5	A8
Security Misconfiguration	A10	A10[12][13]	*	A6	A5
Missing Functional Level Access Control	A2	A2[11]	A10[12]	A8	A7[16]
Unvalidated Redirects and Forwards	*	*	*	A10	A10
Information Leakage and Improper Error Handling	A7	A7[14][16]	A6	A6[18]	*
Malicious File Execution	*	*	A3	A6[18]	*
Sensitive Data Exposure	A8	A8[14][15]	A8	A7	A6[17]
Insecure Communications	*	A10	A9[7]	A9	*
Remote Administration Flaws	A9	*	*	*	*
Using Known Vulnerable Components	*	*	*	*	A9[18][19]

[1] Renamed "Broken Access Control" from T10 2003
[2] Split "Broken Access Control" from T10 2003
[3] Renamed "Command Injection Flaws" from T10 2003
[4] Renamed "Error Handling Problems" from T10 2003
[5] Renamed "Insecure Use of Cryptography" from T10 2003
[6] Renamed "Web and Application Server " from T10 2003
[7] Split "Insecure Configuration Management" from T10 2004
[8] Reconsidered during T10 2010 Release Candidate (RC)
[9] Renamed "Unvalidated Parameters" from T10 2003
[10] Renamed "Injection Flaws" from T10 2007
[11] Split "Broken Access Control" from T10 2004
[12] Renamed "Insecure Configuration Management" from T10 2004
[13] Split "Broken Access Control" from T10 2004
[14] Renamed "Improper Error Handling" from T10 2004
[15] Renamed "Insecure Storage" from T10 2004
[16] Renamed "Failure to Restrict URL Access" from T10 2010
[17] Renamed "Insecure Cryptographic Storage" from T10 2010
[18] Split "Insecure Cryptographic Storage" from T10 2010
[19] Split "Security Misconfiguration" from T10 2010

Prepared by

Figure 2: Comparison of OWASP Top Ten Entries

This thesis contains 14 chapters. In Chapter 1, there is a reference about the meaning of web application, the role of OWASP and some statistics very useful for understanding the present situation. Chapter 2, includes a description about the way we work and analyze the types of each attack and the protection for each one of them. That is our methodology and how we use for the best results. From Chapters 3 to 12, there are all the OWASP Top Ten. Each chapter is structured in four sections. At first an introduction about the vulnerability, then the ways an attacker can exploit the web application, consequently the detection methods for a user to detect the threat and finally the prevention techniques and tools. Chapter 13 concludes this thesis's contributions in two different areas. The first area is that of detecting and preventing web threats, using appropriate tools and techniques. The second one is about presenting threats and trends with the most dangerous threats and vulnerabilities that we found the time period 2013-2015. The thesis finishes with conclusion in Chapter 14 which contains an overall discussion.

Chapter 2: Methodology

In this section we discuss the methodology for analyzing OWASP Top Ten 2013. In order to understand better each threat and vulnerability we used a specific project. This project has been divided into four areas:

1. *Introduction*: In this part we give a description of each threat, a definition and some statistics about it. We take data from different websites and data sources like National Vulnerability Database (NVD), Aspect Security, Trustwave, Veracode, WhiteHat Security etc.
2. *Types of Attacks*: In this part we introduce the main attack methods and their subcategories when it is necessary that we have found after an exhaustive research. We used to mapping OWASP Top Ten to Web Hacking Incident Database (WHID) which has real life examples and new entries for many attack methods.
3. *Detection*: In this part we try to show as more detection methods and tools as we can, emphasizing to the most important ones. We want to introduce more academic tools so we give more attention to them and less to commercial. This part contains scanning tools, firewalls, Intrusion Detection Systems (IDS), penetration testing etc.
4. *Prevention*: In similar manner, in this fourth part we represent prevention techniques and tools. We have examined separately for every attack the more suitable way to prevent the attacker from being harmful. There is an age-old advisory that says, “It’s too late to sharpen your sword when the drum beats for battle”. Make no mistake, we are in a war and we must prepare for the cyber battles by sharpening our skills. Information security professionals must continuously mature their capabilities by working smarter not harder. It is always better to prevent, then to pursue and prosecute.

Chapter 3: Injection Flaws

3.1 Introduction

Injection flaws were rated the number one attack in web applications on the OWASP top ten in 2013 (Owasp.org, 2015). The most publicly recognized is SQL injection, also known as an attack vector for websites but can be used to attack any type of SQL database. According to Acunetix OVS (Acunetix, 2015), only SQL injections cover 27% of all vulnerabilities reported in year 2015 (see Fig 3) and every forth site is vulnerable to SQL injection. For those reasons there is a very detailed analysis of SQL injection in this chapter, cause is the most injurious threat of the other types of injections.

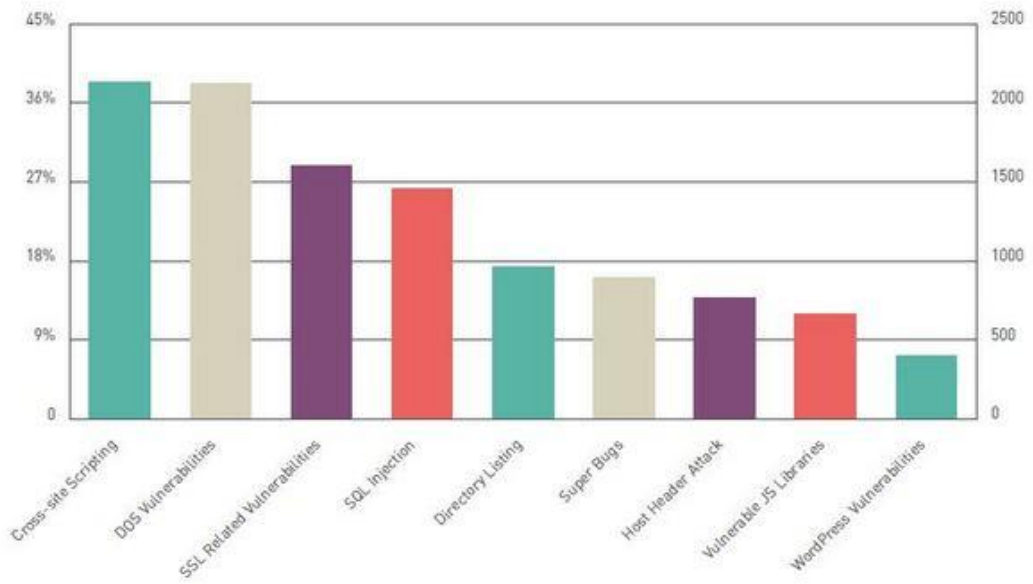


Figure 3: Statistical Vulnerabilities report 2015

Other injection flaws such as LDAP, XPath, etc. adds to this numbers which makes this kind of threat very widespread and exploitable. Injection flaws happen when user's data is used in constructing queries or statements to the underlying system without filtering. Malicious input can change content and logic of query or statement and interpreter can execute attacker's commands. In this section, it will be given the description of the following injection flaws and the defense methods against them:

1. SQL Injection
2. LDAP Injection
3. XPath Injection

4. NoSQL Injection
5. MX Injection
6. OS Command Injection
7. XML Injection

3.2 Types of attacks

3.2.1 *SQL Injection*

SQL injection is an attack in which SQL (Structured Query Language) code is inserted or appended into application and malicious user inputs parameters that are later passed to a back-end SQL server for parsing and execution. It is one of the most devastating vulnerabilities to impact a business, as it can lead to exposure of all of the sensitive information stored in an application's database, including handy information such as usernames, passwords, names, addresses, phone numbers and credit card details. It is categorized as one of the top-10 2013 Web application vulnerabilities experienced by Web applications according to OWASP (Open Web Application Security Project) (Owasp.org, 2015). In this vulnerability an attacker has the ability to influence what is passed to the database, and besides, he can leverage the syntax and capabilities of SQL itself, as well as the power and flexibility of supporting database functionality and operating system functionality available to the database (Clarke, 2009). According to NVD (National Vulnerability Database), multiple SQL Injection vulnerabilities have been discovered for various applications every year. The following table shows the SQL Injection related vulnerabilities found in different years. The SQL Injection vulnerability counts topped at 2008 and declined in the following years (Web.nvd.nist.gov, 2015):

Table 1: SQL Injection Statistical Report

Year	Matches	Total	Percentage
2002	7	2,156	0.32%
2003	11	1,527	0.72%
2004	9	2,451	0.37%
2005	50	4,931	1.01%
2006	91	6,608	1.38%

2007	252	6,514	3.87%
2008	1,088	5,632	19.32%
2009	948	5,732	16.54%
2010	514	4,639	11.08%
2011	288	4,150	6.94%
2012	233	5,288	4.41%
2013	145	5,186	2.80%
2014	296	7,937	3.73%
2015	165	4,258	3,88%

Although the SQL Injection attacks recently are much less than the peak time in 2010, it is still active and the following figure shows the geography graph of the attacks attempts happened in year 2014.

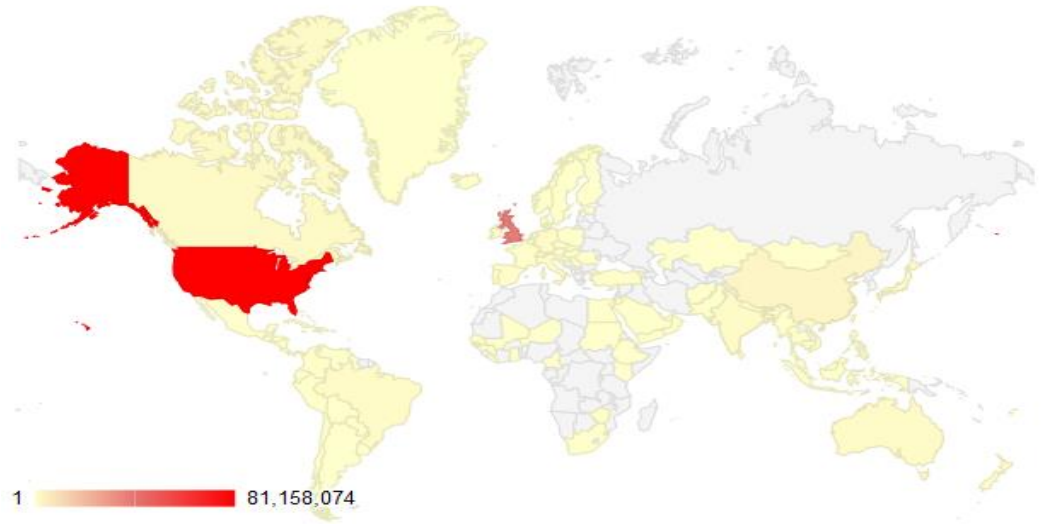


Figure 4: Geography graph of the attacks attempts – 2014

In this chapter, an extensive review is presented for the different types of SQL injection attacks known to date. For each type of attack, it is provided description and example of how attacks of that type could be performed. It is also presented and analyzed existing detection and prevention techniques against SQL injection attacks. There is a detailed presentation of each one and a comparative analysis of all those SQL injection attack types and prevention techniques. The SQL injection field contains numerous types of attacks and defense techniques, which conceal a global view of the SQL injection attack problem. As a consequence, the aim of this chapter

is to answer in many questions about what SQL Injection is, how an attacker can exploit a vulnerability and which defense methods must be used to solve this security threat.

3.2.1.1 Tautology

In this technique attackers try to bypass authentication, identify injectable parameters and extract data from a given database by simply using regular expressions as WHERE and the conditional OR operators such that the query always evaluates to be true. To do this, the Web application might construct an SQL command as follows:

```
SELECT “accounts” FROM “users” WHERE username = “ ‘’ OR 1=1 “
AND password = ” ” AND pin = ” ”
```

When the SQL command is executed then the database server only checks the username field and successfully bypass the authentication mechanism. So as the WHERE clause is always evaluated to true, the database displays the entire rows from the “account” column of the “users” table on the hacker’s screen (Halfond et al., 2006)

3.2.1.2 Union Query

In this type of attack the hackers insert a UNION SELECT statement between two SQL SELECT queries which have nothing in common. As a result, the query returns a dataset that is the union of the result of the original first query and the results of the injected query. For example, consider the following SQL query:

```
SELECT “accounts” FROM “users” WHERE username = “”
“UNION SELECT “cardNumber” from “CreditCard”
WHERE accountNumber = “12345”--“ AND password =” ” AND pin =” ”
```

Observing the above example, the first “SELECT” query returns the null set, while the second “SELECT” query returns “cardNumber” for account “12345”. Thus, after executing the above statement, the “cardNumber” for account “12345” will be displayed on the attacker’s screen (Halfond et al., 2006)

3.2.1.3 Stored Procedures

In most of databases there are standard procedures for allowing the interaction with the operation system and to extend the database's functionality. Attacker's aim is to execute store procedures using malicious SQL injection codes, performing privilege escalation, denial of service and remote commands related to a given type of database. The example below shows how a hacker can send a stored procedure SQLi attack through user interface to database (Sheykhkanloo, 2015).

```
SELECT "accounts" FROM "users" WHERE username = "LIKE '1' or '1'='1'"
AND password = " ; exec master.dbo.xp_cmdshell 'dir c:\temp\*.sql '
SHUTDOWN; --" AND pin = " "
```

The statement "exec master.dbo.xp_cmdshell 'dir c:\temp*.sql '" displays all the file in "c:\temp" directory that have a ".sql" extension to the hacker. By executing the "SHUTDOWN" command the back-end database will be shutdown (Halfond et al., 2006)

3.2.1.4 Piggy Backed Queries

In this technique malicious users supply several different queries within a single string of code. The attacker tries to inject additional malicious queries along with the original query resulting the database receives multiple SQL queries for execution.
Thus the intruder has the ability to extract data, add or modify them, perform denial of service attack or execute remote commands on a back-end database. It has to be mentioned that vulnerability of this kind of attack is dependent of the kind of database. The example below shows how the piggy-backed query SQLi can be sent by hackers:

```
SELECT name from student WHERE pass='abc' ; drop table users;
OR update employee set name='john' WHERE id ='123';
 delete from employee WHERE id ='456';
```

Firstly occurring the execution of first query and after the recognition of the delimiter ";", the second query will run right after the first query. As a result of the second execution the database will drop the table 'users' and in this way will erase all information (Halfond et al., 2006)

3.2.1.5 Logically Incorrect Queries – Error Based SQLi Attacks

Logically incorrect queries SQLi attack is a type of attack in which the attacker tries to gather information from the rejected error messages about the type and structure of the back end database of a web application to find useful data facilitating injection to the database. The following example helps for understanding better this type of attack, assuming that the back end database is a Microsoft SQL server:

```
SELECT “accounts” FROM “users” WHERE username = “”
 AND password = “”
 AND pin = “convert (int, (select top 1 name from sysobjects WHERE xtype = ‘u’))”
```

After executing the query above, the computer screen will show the following error message: “Microsoft OLE DB Provider for SQL Server (0x80040E07) Error converting nvarchar value ’CreditCards’ to a column of data type int.” This message reveals a lot of information for attackers, as that the back-end database is a Microsoft SQL, the type of the first defined table in “sysobjects” metadata database, which is ‘nvarchar’ and third the name of the first defined table in “sysobjects” metadata database, which is “CreditCards”. All this information will be extremely useful for the aspiring hackers (Halfond et al., 2006)

3.2.1.6 Alternate encodings

In this type of attack the injected text is modified so as to avoid detection by defensive coding practices. The attackers obscure their injected commands by using encodings techniques such as ASCII, hexadecimal, Unicode character encoding, etc. Looking at the below example, hackers try to conceal their attacks by using ASCII hexadecimal alternate of “char (Ox73687574646j776e)”, which is equal to “SHUTDOWN’’. The following example sets the aforementioned conclusion: (Halfond et al., 2006)

```
SELECT “accounts” FROM “users”
WHERE username = “johnpap; exec (char (Ox73687574646j776e)) --"
AND password =”” AND pin = “”
```

3.2.1.7 Inference attack

SQL Injection vulnerabilities are often detected by analyzing the error messages received from the database. When no data returns to an end-user from a slightly secured website, it is called an Inference SQLi attack–Blind attack (Symantec, 2008). There are two popular inference SQLi attacks as follows:

- Standard Blind SQL Injection.
- Double Blind SQL Injection/Time-based.

3.2.1.8 Standard Blind SQL Injection

Standard Blind SQLi is applied on well secured databases which do not return any usable feedback or descriptive error messages. The attack is created in the style of true/false statement. It could be understood better this type of SQL attack mentioning the following example (Clarke, 2009):

```
SELECT “accounts” FROM “users” WHERE username =
“LIKE ‘1’ or ‘1’=’1’; IF SYSTEM_USER = ‘sa’ SELECT 1/0
ELSE 5; --“ AND password =”” AND pin =””
```

Looking carefully above, the first interesting point is that the attacker tries to obtain the back-end database behavior by sending an “IF ELSE” statement in which a division by zero will be executed on the database if the current user is a system administrator, “sa”. It is an accepted fact that the specific division is undefined, so the database is obligated to throw an error if the current user is a system administrator “sa”, or else a valid instruction would be executed, “ELSE 5” (Saini, 2015).

3.2.1.9 Double Blind SQL Injection/Time-based

The attacker designs a conditional statement and injects through the vulnerable parameter and gather information based on time delays in the response of the database. For instance, a true response received from a given database means that the time delay was executed successfully while a false response means hackers weren’t successful to execute the time delay. The example below represents, how a hacker can use the time-based injection:

```
SELECT “accounts” FROM “users” WHERE username = “johnpap and 1>0
WAITFOR 5 --“ AND password = “” AND pin = “”
```

There is an always true statement, “1>0”, along with “WAITFOR” keyword, which is a popular keyword for issuing the inference timing SQLi attack. According the results of the above example attackers will try to work out the behavior of the back-end database (Chema et al., 2007).

3.2.2 *LDAP Injection*

The Lightweight Directory Access Protocol (LDAP) is a protocol for querying and modifying directory services over TCP/IP. These services are software applications that store and organize information sharing certain common attributes. LDAP is also based on the client-server model and is object oriented (Alonso et al., 2008). In the following figure there is a LDAP scenario (Alonso et al., 2008):

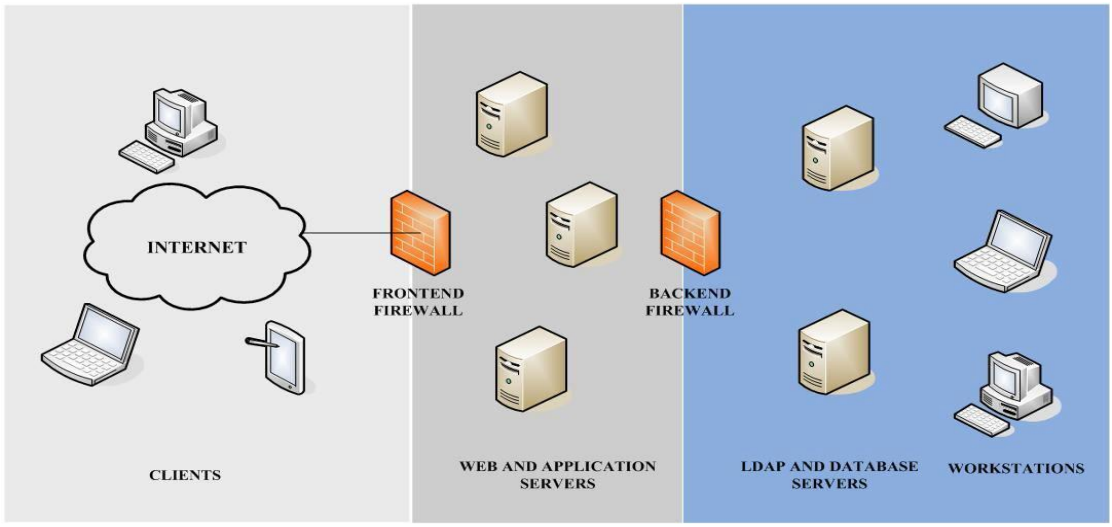


Figure 5: Typical LDAP scenario

LDAP injection attacks are based on similar techniques that have been described in previous section to SQL injections attacks. Due to the importance of the LDAP services for the corporate networks, the LDAP servers are usually placed in the backend with the rest of the database servers. For every LDAP query, a secure web application should sanitize the parameters that sent by the user before its arrival to the server. If the environment is vulnerable these parameters are not properly filtered and the attacker can inject malicious code. For example, web application parameter can be transferred directly to LDAP server without filtering. Query in web browser can look like this (Faust, 2002):

```
http://www.someapplication.com/ldap-queryuser.asp?name=xxx
```

If the value of the name parameter is directly transferred to LDAP query, it is possible to inject more code, change the meaning of the query and access information that the user shouldn't be able to access. For example, the following two queries may return the user's password and enumerate all the users:

```
http://www.someapplication.com/ldap-queryuser.asp?name=xxx)(|password=*)
http://www.someapplication.com/ldap-queryuser.asp?name=*
```

There is also Blind LDAP injection slower than the classic ones based on binary logic. The hacker asks the server true or false questions waiting for LDAP filter generates a valid response and extract information from the LDAP directory (Alonso et al., 2008).

3.2.3 *XPath Injection*

The usage of XML documents instead of relational databases makes web applications vulnerable to XPath Injection Attacks (XPIAs). This is because of the loose typing nature of the XPath language. XPath is language designed specifically to query nodes in an XML document. XPath is a query language for XML document like SQL is for relational databases. Using XPath querying, a malicious user may extract a complete XML document, expose sensitive information, and compromise the integrity of the entire database (Gupta et al., 2013). For achieving a XPIA, a attacker needs to insert arbitrary XPath code into the form fields and URL query parameters in order to inject this code directly into the XPath query parser engine. Doing so would allow him to bypass authentication (if an XML-based authentication system is used) or to access restricted data from the XML data source. If the user searches an XML document for user named xxx whose password is yyy the query may look something like this (Ravichandran et al., 2011):

```
/users/user[username = 'xxx' and password = 'yyy']
```

Altering the value of the password to some value that adds always true condition to the query, it is possible to bypass authentication. The query may now look like this:

```
/users/user[username = 'xxx' and password = 'yyy' or '1'='1']
```

In some cases, an attacker can further manipulate the XPath query to force the server to return various parts of the document. Hence, this XPath injection also leads to extracting document structure and modify the document information in addition to escalate privileges (Mitropoulos et al., 2009).

Another form of XPath injection is blind XPath which employs hit and trial method guess the root node of the XML document and then the attacker is aware of the structure of the XML document. As a consequence, Blind XPath injection vulnerability can be used to retrieve the whole backend XML document and simple XPath can lead to bypassing the authentication protocol. It must also be mentioned that the principle between SQL injection attacks and XPIAs is the same (Klein, 2005).

3.2.4 *No SQL*

There are two major types of database injection attacks:

- SQL Injection that targets traditional database systems
- NoSQL Injection that targets Big Data platforms (Akanji and Elusoji, 2014).

NoSQL (Not Only SQL) is a trending term in modern data stores. These are non-relational databases that rely on different storage mechanisms such as document store, key-value store, graph and more. The wide adoption of these databases is facilitated by the new requirements of modern large scale applications like Amazon (Dynamo), Google (BigTable), LinkedIn (Voldemort), Facebook, Twitter (Cassandra), MondoHQ (MongoDB) which need to distribute the data across a huge number of servers (Ron et al., 2015) (Patel et al., 2015). Just like their traditional RDMBS counterparts, NoSQL databases are susceptible for injection attacks, especially SQL ones and those heavily use server--side JavaScript and PHP to enhance database performance (Patel et al., 2015).Here is an example of malicious input of NOSQL injection via HTTP POST:

username=*tolkien', $or***: [{}, {** *'a':'a*&password=**'** *}], $comment:'successful MongoDB injection'*

This query will succeed as long as the username is correct and it can be turned to ignore the password and login into a user account without the password (Akanji and Elusoji, 2014) (Kadebu and Mapanga, 2014).

3.2.5 *MX Injection*

MX Injection is a vulnerability against Web applications that communicate with mail servers, generally webmail applications. It is possible that such applications are vulnerable to injection of commands from mail protocols (IMAP and SMTP). So, when command injection is done over the IMAP/SMTP server the injection is called as IMAP injection and SMTP injection, respectively. If the attacker can alter IMAP and SMTP commands being transmitted to the mail server, the MX injection can occurs. The added value of this technique is the full injection of commands in the affected mail servers, without any type of restriction. That is to say, this exploit allows not only the possibility of injecting email headers ("From: ", "Subject: ", "To: ", etc.), but also arbitrary command injection to the mail servers (IMAP/SMTP) communicating with the webmail application. This will depend on the validation filters that were implemented. Attackers can use this vulnerability to perform actions on the mail server not normally permitted through the web interface (Diaz, 2006). This can lead to:

- Information leakage

Applied technique is IMAP injection without needing to be authenticated, so in the case of being detected in a parameter vulnerable to IMAP Injection, could always be executed (Ietf.org, 2016)

- Avoidance of CAPTCHA

Applied technique is IMAP injection without needing to be authenticated. If the authentication mechanism of the IMAP server is vulnerable to IMAP Injection, a malicious user could circumvent the restrictions imposed by the CAPTCHA (Captcha.net, 2016)

- Evasion of Restrictions

Applied technique is SMTP injection and is required user authentication .The scenario is the same as both precedent cases of Relay and Spam. The SMTP command injection permits, in this case, evasion of restrictions imposed at the application level.

- Relay

Applied technique is SMTP injection and is required user authentication. In this situation it would be possible to conduct a mail server relay attack.

- Sending SPAM

Applied technique is SMTP injection and is required user authentication. With the aim of evading possible restrictions (e.g., maximum e-mails sent by a user), the attacker injects in the vulnerable parameter as many commands as the attacker wants to send (the quantity of e-mails it desires). By sending the one POST request to the web server below an attacker is able to perform multiple actions.

- Exploitation of protocol vulnerabilities

The fact of being able of execute arbitrary commands in the mail servers could allow an attacker to exploit existing vulnerabilities by sending a certain command to the server. A known example of above exploitation is DoS Attacks to the mail server.

3.2.6 *Shell Injection – OS Commanding*

OS commanding is a method of attacking a Web server by remotely gaining access to the operating system (OS) and then executing system commands through a browser (Webappsec.org, 2016) (Jourdan, 2009) OS Commanding is the direct result of mixing trusted code and untrusted data. This attack is possible when an application accepts untrusted input to build operating system commands in an insecure manner involving improper data sanitization, and improper calling of external programs. In OS Commanding, executed commands by an attacker will run with the same privileges of the component that executed the command, (e.g. database server, web application server, web server, wrapper, application). Since the commands are executed under the privileges of the executing component an attacker can leverage this to gain access or damage parts that are otherwise unreachable (e.g. the operating system directories and files) (SearchSoftwareQuality, 2016).

This is due to improper input validation. For example, if the string variable filename is insufficiently sanitized, the PHP code fragment:

```
exec("open(".$filename.")");
```

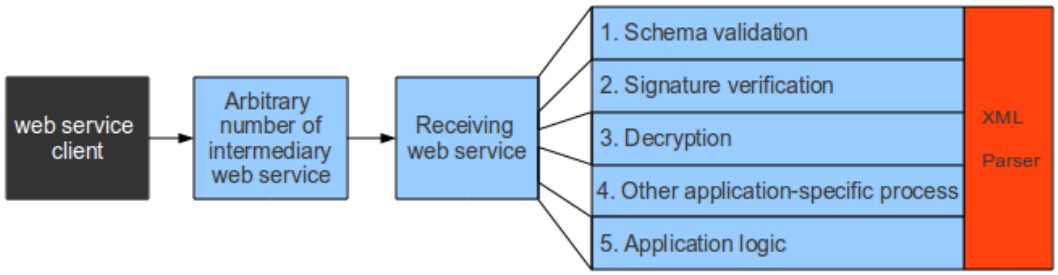
will allow an attacker to be able to execute arbitrary shell commands if filename is not a valid syntactic form in the shell's grammar. This vulnerability is not confined to web applications. A *setuid* program with this vulnerability allows a user with restricted privileges to execute arbitrary shell commands as *root* (Su and Wassermann, 2006) (Capec.mitre.org, 2016). *This attack method also appears in Chapter 7.*

3.2.7 *XML Injection*

An XML (eXtensible Markup Language) Injection attack tries to modify the XML structure of a SOAP (Simple Object Access Protocol) message (or any other XML document) by inserting content e.g. operation parameters containing XML tags. Such attacks are possible if the special characters "<" and ">" are not escaped appropriately (Cwe.mitre.org, 2016). At the Web Service server side, this content is regarded as part of the SOAP message structure and can lead to undesired effects. Some of the requirements that an attacker need to do are mentioned below (Ws-attacks.org, 2016):

- Using XML queries to process user input and retrieve information stored in XML documents
- User-controllable input not correctly sanitized
- To know endpoint of web service**,** otherwise he is not able to reach the web service.
- To know metadata such as the WSDL file of web service.
- To know how to reach endpoint from its location**.** Access to the attacked web service server is possible for the attacker. This prerequisite is important if the web service is only available to users within a certain network.

Here is a graphic representation of the XML attack:



- *Red = attacked web service component*
- *Black = location of attacker*
- *Blue = web service component not directly involved in attack.*

Figure 6: XML attack

3.3 Detection

The injection field contains many defense methods and techniques, which obscure the whole view of the injection attack problem. As a matter of fact, a complete and final presentation cannot be built in a few pages, because this research area is very large in dimension. However, in this section, there is a systematic survey of published work about different detection ways to protect web applications. It is given a mention to some main methods-techniques which are used widely for protection of the injection attacks that are referred to previous section.

3.3.1 *SQL Injection*

SQL injection detection techniques use several different defense methods to detect the existence of vulnerabilities giving protection to the web applications. There are a reasonably small number of things that someone can do to reduce or eliminate the threat of SQL injection. It is likely that the defender need to implement more than one of these methods that are included in this chapter to fully secure an application against SQL injection.

3.3.1.1 Using vulnerability scanning tools.

Web Application Vulnerability Scanners are the automated tools that scan web applications to look for known security vulnerabilities and are very useful for detecting SQL injection attacks. In order to exhaustively scan a web application for security problems, a web application scanner must first map out the web application's structure and functionality. The mapping process is done by the web crawler component, which makes use of different types of content parsers to extract information from web content. This information may include URLs, HTML forms, HTML form parameters, HTML comments, and so forth. A large number of both commercial and open source tools are available and all these tools have their own strengths and weaknesses. OWASP site provides a listing of vulnerability scanning tools currently available in the market. Some of them are: w3af, AppScan, Retina etc. (Owasp.org, 2016).

3.3.1.2 Web Application Firewalls

The most well-known runtime solution in Web application security is the use of a Web application firewall (WAF). The WAF examines every request submitted by the user to the application to decide if the request should be accepted (when it is a legal request) or rejected (if the request is malicious). There is a set of rules that is created and maintained by the application owner. The WAF accepts or rejects by examining each input value in the request and checking if the value matches an attack pattern typically using a set of rules. For example, using a WAF is necessary to be compliant with the Payment Card Industry Data Security Standard (Pcisecuritystandards.org, 2015) for systems that use/process credit cards. As a consequence, testing an application for SQL injection vulnerabilities through a WAF can be used to identify and prioritize vulnerabilities that can be detected through the WAF (Appelt et al., 2013) (Clarke, 2009).

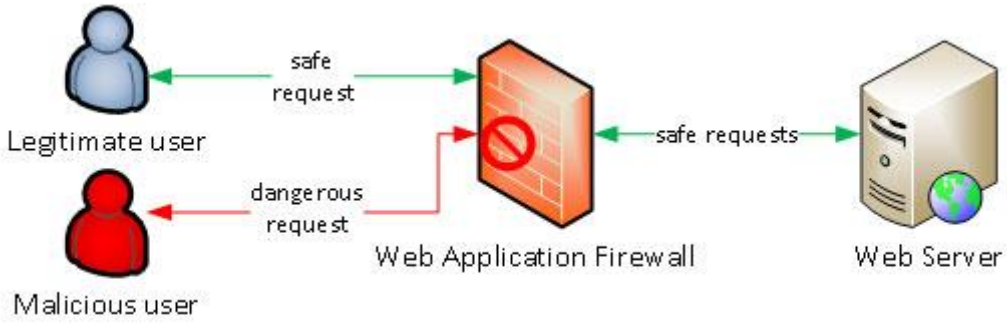


Figure 7: Web Application Firewall

3.3.1.3 Database Firewall

Database firewall is essentially a proxy server that sits between the application and the database. The application connects to the database firewall and sends the query as though it were normally connecting to the database. The proxy server monitors and analyzes each SQL statement for malicious commands. It can also serve as an application - level IDS for malicious database activity by monitoring connections in passive mode and altering administrators of suspicious behavior (Appelt et al., 2013) (Clarke, 2009).

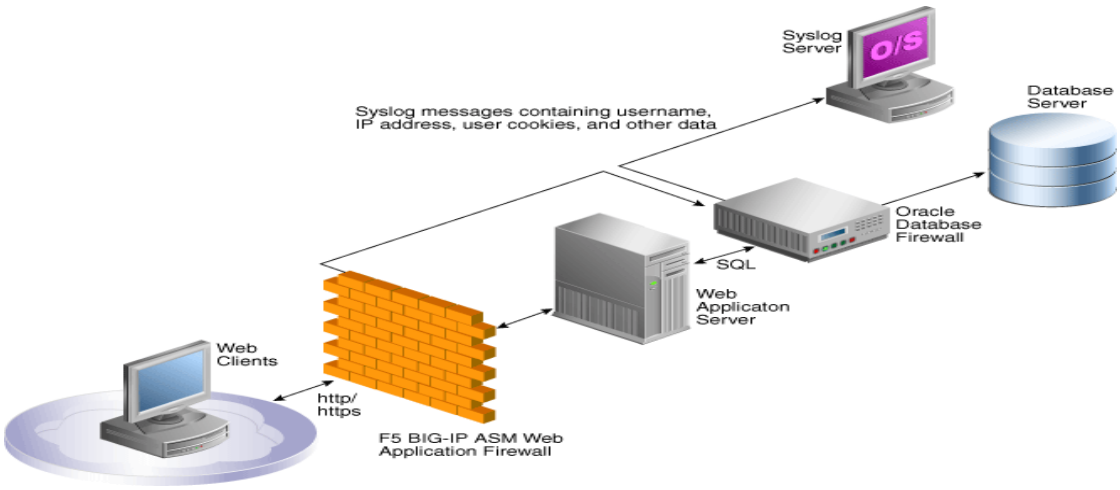


Figure 8: Database Firewall

3.3.1.4 AMNESIA

AMNESIA (Analysis and Monitoring for NEutralizing SQL Injection Attacks) is the prototype tool that implements the technique to detect and prevent SQLIAs for Java-based web applications. AMNESIA uses the combination of static and dynamic analysis to counter against SQL injection attack. In static analysis part, it analyzes the application code and automatically generates the model of legitimate queries. In dynamic analysis part, it monitors all the dynamically generated queries at run time and compares them with the legitimate queries built with static model Queries that violate the model represent potential SQLIAs and are thus prevented from executing on the database and reported. The technique consists of four main steps.

- **Identify hotspots**: Scan the application code to find hotspots points which issues SQL queries.

.

- **Build SQL-query model**: For each hotspot identified, build a SQL query model which represents all the possible SQL queries that may be built at that hotspot. It uses Java String Analysis to construct character-level automata, parse automata to group characters into SQL tokens. A *SQL-query model* is a non-deterministic finite-state automaton in which the transition labels consist of SQL tokens (SQL keywords and operators), delimiters, and place holders for string values.
- **Instrument Application**: At each hotspot, add function calls to the runtime monitor.

- **Runtime monitoring**: Check the dynamically generated queries against the static SQL query model and block queries that violate the model.

AMNESIA has three modules for the implementation of the above four steps.

- **Analysis module.** This module implements Steps 1 and 2 of the technique. Its input is a Java web application and it outputs a list of hotspots and a SQL-query models for each hotspot.
- **Instrumentation module.** This module implements Step 3. It inputs a Java web application and a list of hotspots and instruments each hotspot with a call to the runtime monitor.
- **Runtime-monitoring module.** This module implements Step 4. It takes as input a query string and the ID of the hotspot that created the query, retrieves the SQLquery model for that hotspot, and verifies the query against the model.(Halfond and Orso, 2005)

3.3.1.5 SQLrand

SQLrand is a concept of instruction set randomization to the SQL Language. The SQL keywords are manipulated by appending a random integer to them, that makes an attacker cannot easily guess. Hence, attacker who attempts an SQL injection attack would be discouraged. The design of SQLRand consists of a proxy which is placed between the client and database server. The primary obligation of proxy is to decipher the random SQL query received and then conveys then forward the SQL command with the standard set of keywords to the database for computation.. If an intruder attempts SQL injection attack, the proxy's parser will fail to identify the randomized query and thus reject it. To implement this approach, developer needs to modify client library (Boyd and Keromytis, 2004).

3.3.1.6 SQL DOM

SQLDOM (SQL Domain Object Model) is a method to encourage the developer to use a set of classes which is strongly typed to database schema. Using these classes, the application developer is able to construct dynamic SQL statements without manipulating any strings. It escapes malicious characters to prevent SQL injection attacks. SQLDOM consists of two parts:

- **Abstract object model**: Construct an object model that could be used to build every possible legal SQL statement which would execute at runtime.
- **Sqldomgen**: It is an API generation tool. It is developed using C# and .NET framework and follows three main steps. First step is to obtain the database schema. The second step is to iterate through a table and columns of database schema and generate a number of files containing a strongly-typed instance of the abstract object model. The third step is to compile the generated output into Dynamic Link Library (DLL).

SQL DOM allows detecting errors in code that accesses the database during compile-time instead of at runtime. This has a direct impact on the reliability of the running system: runtime error messages and exceptions pertaining to the errors addressed above can be avoided (Kruger and McClure, 2005).

3.3.1.7 WebSSARI

WebSSARI (Web application Security by Static analysis and Runtime Inspection) is a tool that implements the technique of detection input-validation related errors by performing information flow analysis. It protects from attacks web applications without user intervention by instrumenting vulnerable sections of code with runtime guards. Static analysis is performed to find out taint flows against preconditions for susceptible functions. If WebSSARI detects the use of untrusted data following correct treatment, the code is left as-is. This approach verifies the most commonly used language for Web application programming PHP and incorporate support for extending to other languages. The primary drawback of this approach is that it assumes plentiful preconditions for vulnerable functions can be accurately articulated using their typing system (Hang et al., 2004).

3.3.1.8 SQLGuard

SQLGuard protects web applications by comparing parse tree structure of SQL query. A parse tree is a data structure for the parsed representation of a statement. Parsing a statement requires the grammar of the statement's language. By parsing two statements and comparing their parse trees, it can be determined if the two queries are equal. When a malicious user successfully injects SQL into a database query, the parse tree of the intended SQL query and the resulting SQL query do not match. The

model is articulated as a grammar that only accepts legal queries and is generated at runtime by examining the structure of the queries before and after the inclusion of user-input. A secret key is used to delimit user input by runtime checker during parsing, so security of this technique is fully dependent on the safety of secret key. It has implemented in J2EE platform. In order to deploy this technique, developer has to either manually insert special indicators in the code where user input is inserted to a dynamically created query or rewrite code to employ a special intermediate library (Buehrer et al., 2005).

3.3.1.9 CANDID

Candid (CANdidate evaluation for Discovering Intent Dynamically) is a tool that is based on modifying web applications written in Java through a program transformation. This technique works by comparing dynamically infer programmer intended SQL query structure against the issued actual SQL query structure. For every user input, this technique creates benign sample inputs known as candidate inputs and the program is executed over actual inputs and sample inputs (candidate inputs) simultaneously. Then a candidate SQL query is created along with the actual SQL query where the candidate SQL query is always benign and actual SQL query is possibly malevolent. The actual query is rejected if parse structures of both queries do not match (Bisht et al., 2010).

3.3.1.10 SQL- IDS

SQL-IDS (SQL Injection Detection System) is a tool which monitors Java based applications. There is no need of source code modification in existing applications to use it. This technique follows a specification based methodology to detect SQL injection vulnerabilities (Fig.9) (Kemalis and Tzouramanis, 2008). It utilizes specifications that characterize the programmer intended syntactic structure of queries which are generated and executed by the application. It also monitors the web application for executing SQL queries that are in breach of the specification. It controls and filters the traffic between application server and the back end database server where each SQL query passes through the validation process in order to specific target system, DBMS, or application environment. According to evaluation of this technique, the effectiveness is its advantage for detecting all the attempts of attack

that were injected through the application in addition to the non-existence of false negatives (Kemalis and Tzouramanis, 2008)

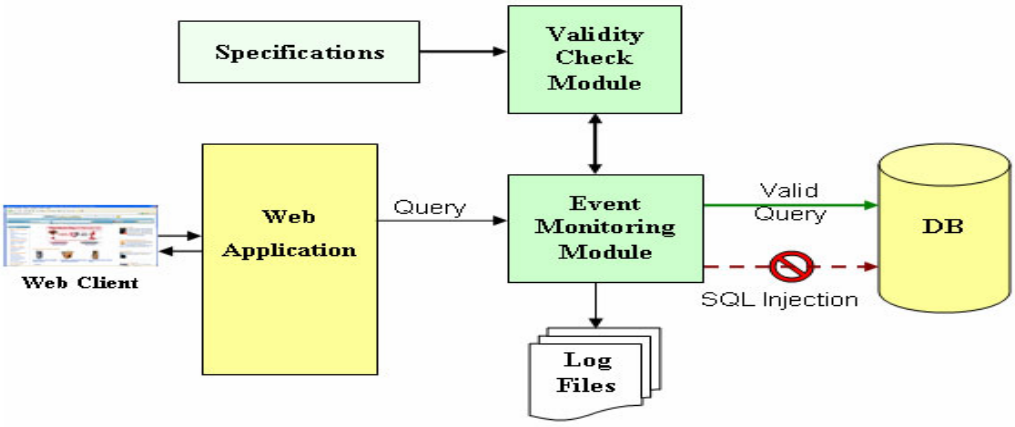


Figure 9: Architecture of SQL- IDS

3.3.1.11 SecurePHP

SecurePHP is a tool that automatically detects and suggests fixes to SQL queries that are found to contain SQL Injection Vulnerabilities. It was written in C# and utilized the .NET framework. The tool provides the solution using prepared statements. It is important to note that this solution specifically targets PHP and MYSQL based web application and for this reason it is called SecurePHP. It has three major steps, vulnerability detection, creation of prepared statements and report generation. GUI of SecurePHP is used to look for vulnerabilities. It uses grammar based violation principle (by parsing SQL statement) to detect the vulnerabilities in the query. The purpose of this technique is to notify the user of any possible weaknesses proposing solutions. When a query is used more than once, prepared statements can actually lead to an increase in performance because the procedure is temporarily stored on the MySQL server. Lastly, user can generate reports for files containing vulnerabilities (Dysart and Sherriff, 2008).

3.3.1.12 DIWeDa

DIWeDa (Detecting Intrusions in Web Databases) is a practical solution to the web database intrusion detection problem. It is a model which acts at the session level rather than the SQL statement or transaction stage, detecting the intrusions in web based applications. The proposed approach is shown to be efficient and could identify SQL injections and business logic violations. There is a need to be tested against new types of SQL injection attacks and requires a great need of accuracy improvement.

The software architecture for the proposed IDS design is shown in Figure 10 (Gudes and Roichman, 2008):

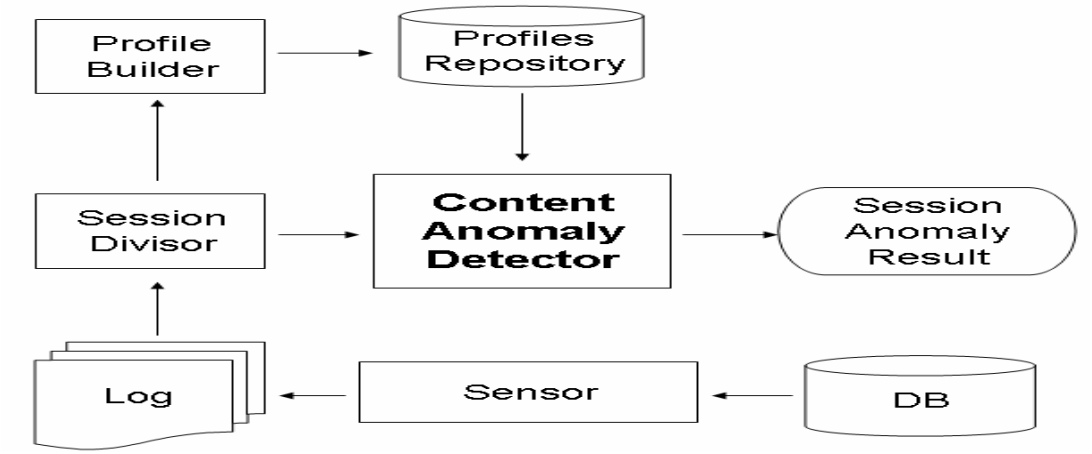


Figure 10: Architecture of DIWeDa

3.3.1.13 SDriver

Smart Driver (SDriver) is a secure database driver and a prototype application that prevents SQLIAs against web applications. If an SQL injection happens, the structure of the query, and therefore its signature will be altered, and SDriver will be able to detect it. SDriver is located between the application and its underlying relational database management system. To detect an attack, the driver uses stripped down SQL queries and stack traces to create SQL statement signatures that are then used to distinguish between injected and legitimate queries (Mitropoulos and Spinellis, 2009).

3.3.1.14 SQLmap

Sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections (Sqlmap.org, 2016).

3.3.2 *NoSQL Injection*

A very effective detection method for NoSQL injection is scanning with vulnerabilities tools and auditing. It is the same technique that it is presented in section 3.3.1.1.

3.3.3 *Shell Injection – OS Commanding*

The vulnerability of a server or other network-connected computer to OS commanding attacks can be minimized by (SearchSoftwareQuality, 2016) (Cwe.mitre.org, 2016):

- Blacklisting of forbidden character sequences.
- Whitelisting of allowed character sequences.
- Filtering out command directory names.
- Using vulnerabilities scanning tools, as Acunetix Web Vulnerability Scanner (Acunetix, 2016) and IBM Rational AppScan (Www-01.ibm.com, 2016)

According to security experts, the main reason that OS commanding and similar exploits are on the rise is that security is not sufficiently emphasized in the development of operating systems and applications. To protect the integrity of network servers, experts recommend the implementation of simple precautions during development, such as controlling the types and numbers of characters that are accepted by servers from users (SearchSoftwareQuality, 2016) (Antunes et al., 2009).

3.3.4 *XML Injection*

There are new security detection mechanisms like state-based XML firewall (S-Wall) (Fitch et al., 2011) which can be used to protect a service provider from various XML-based attacks. The S-Wall is capable of real-time detection and verification of XML-based attacks by checking the SOAP message as well as the parameters passed to a web service operation.

3.4 Prevention

Prevention technique averts or severely handicaps the possibility of success of injection attacks by statically identifying vulnerabilities in the code, proposing a different development paradigm for generating SQL queries, or inspecting the

application to enforces best defensive coding practices during development. In this section, it is presented and reviewed some important prevention methods - tools and there is also a comparative analysis of all included methods in tables, accordingly to specific criteria.

3.4.1 *SQL Injection*

3.4.1.1 Using Parameterized Statements - Bind Variables

The most powerful protection against SQL injection attacks is the use of bind variables. Using bind variables will improve application performance. Application coding standards should require the use of bind variables in all SQL statements. Not all dynamic SQL statements can be parameterized. Parameterized statements are a method of supplying potentially insecure parameters to the database, usually as a query or stored procedure call. In particular, only data values can be parameterized, and not SQL identifiers or keywords. For example the following statement can't be parameterized:

```
SELECT * FROM users WHERE username LIKE 'k&' ORDER BY ?
```

Using parameterized statements one of the root causes of SQL injection is avoided, the creation of SQL queries as strings that are then sent to the database for execution. To defense SQL injection, user input is absolutely cannot directly to be embedded SQL statements. On the contrary, the user input must be filtered, or use of parameterized statement (Kost, 2004) (Clarke, 2009) (Ping-Chen, 2011).

3.4.1.2 Use Strong Authentication – Access Control

Authentication is a mechanism employed by web application to assert whether a user is who he/she claimed to be. Matching a user name and a password stored in the database is the most common authentication mechanism for web applications. Using a strong authentication, it is an effective countermeasure to evade malicious users entering in your personal/business web environment. The use of input bypassing filters is especially known and an example is Oracle Application Server that provides useful custom filters (Docs.oracle.com, 2016).

Giving also different privileges to the different users then the database is more secure. For example, if a user only requires read-access to the database, does not permit the user to execute UPDATE or INSERT queries (Lau et al., 2007).

3.4.1.3 Error Messages

Another way to prevent SQL injection is to avoid some detailed error messages, because the hackers can use the information (Ping-Chen, 2011). If an attacker cannot obtain the source code for an application, error messages become critically important for a successful attack. Most Java applications do not return detailed error messages. Testing and analysis should be performed to determine if the application returns detailed error messages. Handling exceptions and suppression of error messages is most effective when done with application-level error handlers. Therefore, it is a good practice to configure the application framework and Web server to return a custom response when unexpected application errors result. The configured response could be a custom error page that displays a generic message or a redirection to the default Web page. The important point is that the page should not reveal any of the technical details related to why the exception occurred. It is recommended to use a standard input mechanism to verify all confirmed the input data of length, type, a statement, enterprise rules, etc (Kost, 2004).

3.4.1.4 Input data Validation

Every passed string parameter should be validated. Many web applications use hidden fields and other techniques, which also must be validated. If a bind variable is not being used, special database characters must be removed or escaped (Kost, 2004). There are two different types of input validation approaches: whitelist validation (sometimes referred to as inclusion or positive validation) and blacklist validation (sometimes known as exclusion or negative validation) (Clarke, 2009). The best way to filter data is with a default-deny regular expression that includes only the type of characters that you want. For instance, the following regular expression will return only letters and numbers*: s/[^0-9a-zA-Z]//g* (Faust, 2002).

3.4.1.5 Using encryption

Another way to prevent SQL injection is to use encryption. Especially for storing sensitive data the use strong cryptography is considered appropriate. If someone must store sensitive data, he must protect it with a strong symmetric encryption algorithm such as Advanced Encryption Standard (AES) or Triple DES (Data Encryption Standard) or hashing algorithms as Message Digest (MD5) and Secure Hash Algorithm (SHA-1) For example, if a user encrypts his username and password field

then tautology and piggy-back query does not occur. If an attacker is able to view the table storing this data, only password hashes will be returned. (Clarke, 2009) (Ying et al., 2012)

3.4.1.6 Using page overriding - URL Rewriting

If a page is vulnerable and needs replacing, a replacement page or class needs to be created that is substituted at runtime. The substitution is accomplished with configuration in the Web application's configuration file. In ASP.NET applications, defenders users can use HTTP handlers to accomplish this task. An almost same technique to page overriding is URL rewriting. In this case, there is a configuration of Web server or application framework to take requests that are made to a vulnerable page or URL and redirect them to an alternative version of the page. The new version of the page would implement the logic of the original page in a secure manner. The redirection should be performed server-side so that it remains seamless to the client. The Apache module (Httpd.apache.org, 2016) *mod_rewrite* and the .NET Framework (Msdn.microsoft.com, 2016) *urlMappings* element are two examples to accomplish this defense method, depending on the Web server and application platform.

3.4.1.7 SQLCheck

SQLCHECK is a prevention technique and an implementation for the setting of SQL command injection attacks. This technique is based on context free grammars and comparing the parse tree constructed with user input at run time. They have evaluated SQLCHECK on real world application written in PHP and JSP. SQLCHECK is built around the idea of blocking the queries in which user input can change the syntactic structure of query. To track the user input, meta-data have been used to mark the beginning and end of each input strings. SQLCHECK is developed using grammar of output language and policy defining valid syntactic form of query. It is also placed on the web server. The web application then generates augmented queries through assignments, concatenations etc., which is parsed by SQLCHECK. If the query parses successfully, query is legitimate and SQLCHECK sends it without the meta-data to the backend database. Otherwise, SQLCHECK will reject the query (Su and Wassermann, 2006).

There are also prevention tools and techniques the following ones, which have been analyzed in section 3.3: AMNESIA, SQL DOM, SQL Rand, WebSSARI, SDriver, CANDID, Web Application Firewalls.

3.4.2 *LDAP and Blind LDAP Injection*

The work on mitigating LDAP injections must use techniques as dynamic checks, static code analysis, complex input validation and defensive programming. Users must also pay attention in general security recommendations, as applying minimum exposure point and privileges principles. All data returned to the user should be validated and the amount of data returned by the queries should be restricted as an added layer of security (Faust, 2002). However, the most important is to check and sanitize the variables used to construct LDAP filters before sending the queries to the server. In order to sanitize correctly web application inputs which are going to be used in LDAP search filters, developers must only pay attention to ten special characters: |, &, (,), *, <, >, =, ~, !. If the developer sanitizes in a secure way the input to forbid those characters LDAP Injection attacks won't work. In addition, the LDAP server should not be directly accessible on the Internet, thereby eliminating direct attacks to the server itself. The access level used by the Web application to connect to the LDAP server should be restricted to the absolute minimum required. That way, even if an attacker manages to find a way to break the application, the damage would be limited (Faust, 2002) (Alonso et al., 2008) (Alonso et al., 2008).

There are also three precautions that can be taken for the second flaw in LDAP:

- Not allowing **plain-text** passwords to be used for authentication; hash them with at least SHA-256
- Using the **TLS service** LDAP supports
- Having all authentication connections connect to server through a **virtual private network (VPN)** (Obimbo and Ferriman, 2011)

3.4.3 *XPath Injection*

Defending against XPath Injection is basically similar to defending against SQL injection (Klein, 2005). The common ways to prevent XPath Injections are *Strong input validation*, *Use of parameterized XPath queries and Use of custom error*. So,

the developer has to ensure that the application does accept only legitimate input and another way is use parameterized queries to prevent XPATH injection. Specifically, the single and double quote characters should be disallowed. This can be done either in the application itself, or in a third party product (e.g. application firewall.) (Mitropoulos et al., 2009) (Ravichandran et al., 2011).

3.4.4 *NoSQL Injection*

To eliminate the security threats every organization must define a security policy also that should be strictly enforced. A strong security policy must contain well defined security features. Mitigating security risks in NoSQL deployments and, by extension, NoSQL Injection attacks there some critical areas that need to be considered. The countermeasures are similar to those that are used for SQL injection attacks and have been mentioned in previous section. Meaningfully, the defender is based on the following dimensions: (Kriha et al., 2011) (Patel et al., 2015) (Altrafi et al., 2014) (Nematbakhsh and Sahafizadeh, 2015)

- Strong User Identification - Authentication
- Access Control and Prevention of Privilege Escalation
- Data encryption

3.4.5 *MX Injection*

For the purpose of reduction the ability by an attacker to use MX injection, mail server administrators must adopt proper security measures to achieve a defense in-depth solution. Here are some measures that would be valid countermeasures to these type of attacks:

- Application Firewalls

The deployment of an application firewall with other protection systems, is an additional solution for preventing the injection of IMAP/SMTP commands to the mail servers from the vulnerable MX Injection parameters.

- IMAP/SMTP servers configuration

If the access to the mail servers is only accomplished through the webmail application, those servers must be hidden from Internet. In addition to this, these servers should be also appropriate hardened (by disabling commands that are not absolutely necessary, etc.) with the aim of minimizing the impact of MX Injection attacks. (Diaz, 2006)

- Input Data Validation

All input data used by the application (even if not supplied directly by the user and is used internally by the application) must be sanitized. Validation must be performed before any type of operation is executed using the data (Diaz, 2006).

3.4.6 *Shell Injection – OS Commanding*

Expect from the detection methods that are introduced in section 3.3, there are also two major prevention procedures for minimizing OS commanding :

- Blacklisting of forbidden character sequences
- Restricting permissions on OS commands

3.4.7 *XML Injection*

An important step in preventing such attacks is a strict schema validation on the SOAP message, including data type validation as possible. All user-controllable input must be validated and filtered for illegal characters as well as content that can be interpreted in the context of an XML data or a query. This defense method can reject the message and protect the user from the attack. It is understood that strict schema validation is resource intensive, however if well written it guarantees maximum security against the attack (Ws-attacks.org, 2016). Additionally, the use of custom error pages (Capec.mitre.org, 2016) so that attackers cannot glean information about the nature of queries from descriptive error messages. Input validation must be coupled with customized error pages that inform about an error without disclosing information about the database or application.

The following Table 2 (Kindy and Pathan, 2011), shows the techniques and their defense capabilities against SQLIAs are analyzed in previous section. It is a comparative analysis of the SQL Injections detection - prevention tools and the attack types. Furthermore, they have been added four extra techniques in this table: SDriver, SecurePHP, SQL-IDS, SQLmap.

Table 2: Techniques and Types of SQLi Attacks

Technique	TYPES OF SQL INJECTIONS ATTACKS						
	Tautol	Union Query	Stored Procedure	Piggy Backed Queries	Logically Incorrect Queries	Alternate Encodings	Inference
AMNESIA	Y	Y	N	Y	Y	Y	Y
SQLRand	Y	Y	N	Y	N	N	Y
SQL DOM	Y	Y	N	Y	Y	Y	Y
WebSSARI	Y	Y	Y	Y	Y	Y	Y
SQLGuard	Y	Y	N	Y	Y	Y	Y
CANDID	Y	N	N	N	N	N	N
SecurePHP	Y	Y	N	Y	Y	Y	Y
SQLCHECK	Y	Y	N	Y	Y	Y	Y
DIWeDa	N	N	N	N	N	N	Y
SQL-IDS	P	P	P	P	P	P	P
SDriver	Y	Y	N	Y	N	N	Y
SQLmap	Y	Y	Y	Y	Y	Y	Y

Y = Yes, N = No, P = Partly Effective

In the Table 3 (Shakya and Aryal, 2011), there are the defense techniques - tools with some of their characteristics according to the analysis that has been implemented above.

Table 3: Defense techniques characteristics

Technique	Detection time	Detection location	Analysis Method
AMNESIA	Run time	Server side application	Hybrid
SQLrand	Run time	Server side proxy	Dynamic
SQL DOM	Compile time	Server side application	Secure Programming
WebSSARI	Run time	Server side application	Hybrid
SQLGuard	Run time	Server side application	Dynamic
CANDID	Run time	Server side application	Dynamic
SecurePHP	Run time	Server side application	Secure Programming
SQLCHECK	Run time	Server side proxy	Dynamic
SQL IDS	Run time	Server side proxy	Dynamic

SDriver	Run time	Server side proxy	Dynamic
DIWeDa	Run time	Server side proxy	Dynamic
SQLmap	Run time	Server side application	Dynamic

Chapter 4: Broken Authentication and Session Management

Broken authentication and session management were rated the number two attack in web applications on the OWASP top ten in 2013 (Owasp.org, 2015). Broken authentication deals with improper implementation of authentication mechanism and broken session management deals with belonging functions such as logout, timeout, secret question, password reset etc. If the authentication mechanism is not well implemented it is possible to use that weakness to exploit the application.

According to NVD (National Vulnerability Database), authentication issues and credential management vulnerabilities have been discovered for various applications every year. The following tables show the related vulnerabilities found in different years. The improper authentication vulnerability counts topped at 2009 and declined in 2015 and the credentials management counts topped at 2014 (Web.nvd.nist.gov, 2015):

Table 4: Authentication Issues Statistical Report

Year	Matches	Total	Percentage
2002	4	2,156	0.19%
2003	7	1,527	0.46%
2004	6	2,451	0.24%
2005	5	4,931	0.10%
2006	13	6,608	0.20%
2007	68	6,514	1.04%
2008	146	5,632	2.59%
2009	210	5,732	3.66%
2010	75	4,639	1.62%
2011	55	4,150	1.33%
2012	99	5,288	1.87%
2013	107	5,186	2.06%
2014	143	7,937	1.80%
2015	25	4,258	0.50%

Table 5: Credentials Management Statistical Report

Year	Matches	Total	Percentage
2002	8	2,156	0.37%
2003	8	1,527	0.52%
2004	6	2,451	0.24%
2005	2	4,931	0.04%
2006	5	6,608	0.08%
2007	24	6,514	0.37%
2008	52	5,632	0.92%
2009	67	5,732	1.17%
2010	53	4,639	1.14%
2011	38	4,150	0.92%
2012	52	5,288	0.98%
2013	89	5,186	1.72%
2014	115	7,937	1.45%
2015	54	4,967	1.09%

According to the annual Verizon 2015 Data Breach Investigations Report (Verizon, 2015), most of the attacks make use of stolen credentials. The following figure 11 (Verizon, 2015) shows how serious problem the broken authentication is.

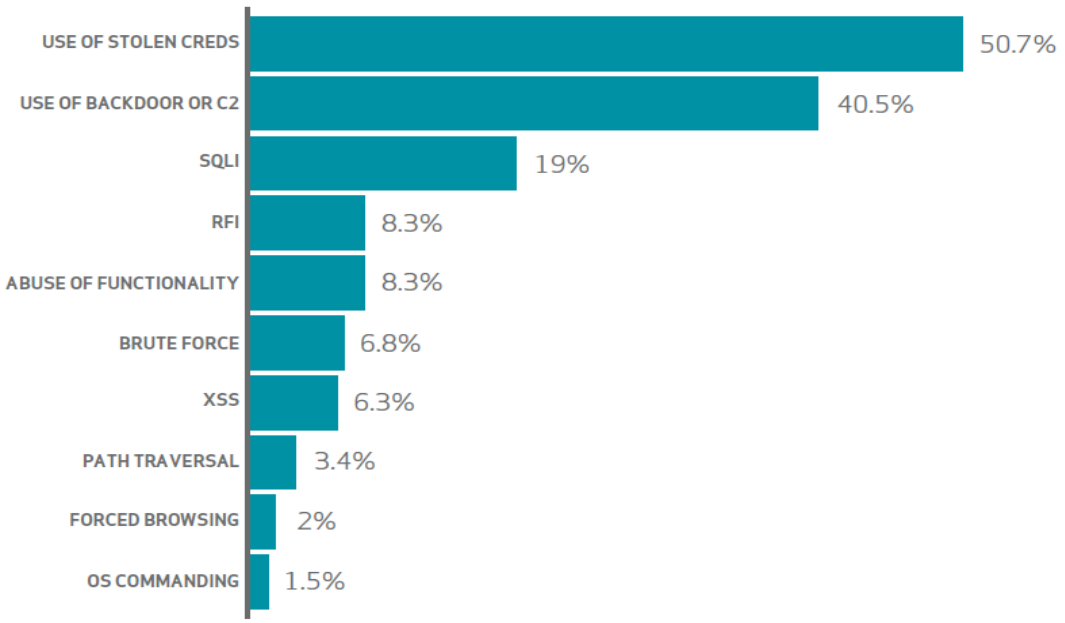


Figure 11: Variety of hacking actions within Web

In this chapter, there is a systematic survey of published work about different attack ways to exploit web applications with this vulnerability. It is given a mention to some main types which are used widely and especially it is given the description of the following attacks and the defense methods against them:

1. Phishing attack
2. Brute force attack
3. Replay attack
4. Session Hijacking attack
5. Session Fixation attack
6. Dictionary/Rainbow table attack

4.1 Types of attacks

4.1.1 *Phishing Attack*

Phishing attacks use social engineering techniques mixed with technical tricks to fool the user and steal sensitive information and banking account credentials. Social engineering schemes are typically based on spoofed emails to lead users to visit infected websites designed to appear as legitimate ones. The websites are designed to lead customers to divulge financial data, such as account usernames, credit card numbers, passwords and social security numbers. The frequently used attack method is to send e-mails to potential victims, which seemed to be sent by banks, online organizations, or ISPs. In these e-mails, they will make up some causes, e.g. the password of a credit card had been mis-entered for many times, or they are providing upgrading services, to allure victim visit their Web site to conform or modify his account number and password through the hyperlink provided in the e-mail. He will then be linked to a counterfeited Web site after clicking those links. The style, the functions performed, sometimes even the URL of these faked Web sites are similar to the real Web site. It's very difficult for user to know that he is actually visiting a malicious site. If he inputs the account number and password, the attackers then successfully collect the information at the server side, and is able to perform their next step actions with that information (e.g., withdraw money out from his account) (Chen and Guo, 2006). All phishing attacks fit into the same general information flow. Fig 12 shows steps in a typical scenario of deceptive phishing flow, and each step is depicted in the following (Huang et al., 2009):

1. Phishers set up a counterfeited Web site which looks exactly like the legitimate Web site, including setting up the web server, applying the DNS server name and creating the web pages similar to the destination Web site, etc.
2. Send large amount of spoofed e-mails to target users in the name of those legitimate companies and organizations, trying to convince the potential victims to visit their Web sites.
3. The victim is deceived by the spoofed email into clicking the link to visit the phishing website and takes an action that makes her vulnerable to confidential information compromise.
4. The phisher uses the identity information of the victim to the target website
5. Impersonates the victim's identity to gain the illegal financial benefits.

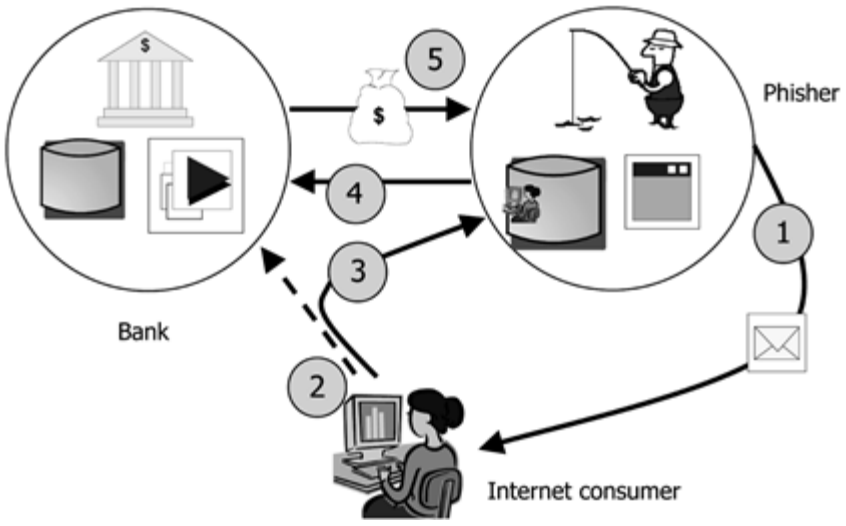


Figure 12: Typical Phishing Attack Schema

Financial phishing attacks, including phishing against banks, payment systems and e-shops accounted for 28.73% of all phishing attacks detected in 2014 by the Heuristic anti-phishing component of Kaspersky Lab products. Each attack was an attempt to download a phishing page into the browser of the user. The source carrying the link could be an email message, or a message from an instant message services or a social network etc. (Kaur and Singh, 2014).

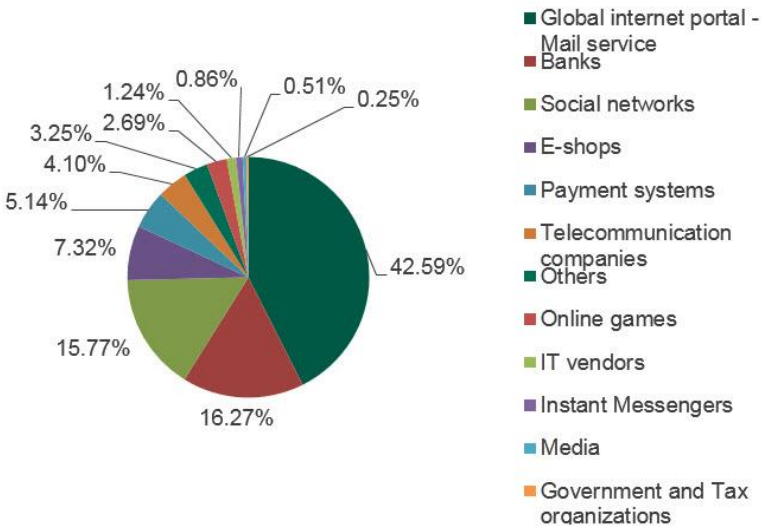


Figure 13: Distribution of phishing attacks in 2014

4.1.2 *Brute force Attack*

Brute force attacks are one of the most famous attacks to obtain pairs of user names and passwords to login some network services illegally (Honda et al., 2014). A brute-force attack, also known as exhaustive key search, is a cryptanalytic attack that can, in theory, be used against any encrypted data. It consists of systematically checking all possible keys or passwords until the correct one is found. In the worst case, this would involve traversing the entire search space. Brute force attacks work by calculating every possible combination that could make up a password and testing it to see if it is the correct password. As the password's length increases, the amount of time, on average, to find the correct password increases exponentially. This means short passwords can usually be discovered quite quickly, but longer passwords may take decades. When key guessing, the key length used in the cipher determines the practical feasibility of performing a brute-force attack, with longer keys exponentially more difficult to crack than shorter ones. A cipher with a key length of N bits can be broken in a worst-case time proportional to 2^N and an average time of half that. *This attack is also mentioned in Chapter 7.*

4.1.3 *Replay Attack*

A replay attack is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. This is carried out either by the originator or by an adversary who intercepts the data and retransmits it, possibly as part of a masquerade attack by IP packet substitution (such as stream cipher attack).

For example, suppose Alice wants to prove her identity to Bob. Bob requests her password as proof of identity, which Alice dutifully provides (possibly after some transformation like a hash function); meanwhile, Eve is eavesdropping on the conversation and keeps the password (or the hash). After the interchange is over, Eve (posing as Alice) connects to Bob; when asked for a proof of identity, Eve sends Alice's password (or hash) read from the last session, which Bob accepts thus granting access to Eve (Stallings, 1999).

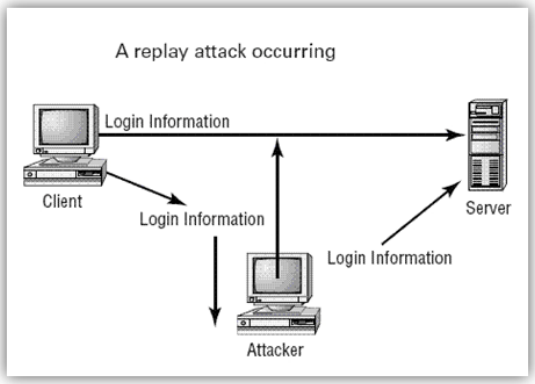


Figure 14: Replay attack

4.1.4 *Session Hijacking Attack*

The Session Hijacking attack (Owasp.org, 2016) consists of the exploitation of the web session control mechanism, which is normally managed for a session token. Because http communication uses many different TCP connections, the web server needs a method to recognize every user's connections. The most useful method depends on a token that the Web Server sends to the client browser after a successful client authentication. A session token is normally composed of a string of variable width and it could be used in different ways, like in the URL, in the header of the http requisition as a cookie, in other parts of the header of the http request, or yet in the body of the http requisition. The Session Hijacking attack compromises the session token by stealing or predicting a valid session token to gain unauthorized access to the Web Server. The session token could be compromised in different ways, the most common are:

- Session sniffing attack
- Man in the middle attack

4.1.5 *Session sniffing*

Sniffing (Daud et al., 2011) uses the weakness found in design of application to reveal more information to the intruder than what it intends to show by using sniffer to monitor and eavesdropping the input or output data. In textual password, the user starts to send "7958F" as his password which transferred in sequence of packets. The packets go up and down through network along with the packet's destination address to show which computer is permitted to accept it. At the same time the attacker uses a sniffer software to change the configuration of his Network Interface Card to a promiscuous mode to collect all these packets. In Recognition based Graphical Password, the attacker can sniff the ID of image password. This ID is usable only if user can attack the image gallery at the same time. No research has found the impact of sniffing attacks on click based and draw based algorithms. In the figure 15 first the attacker uses a sniffer to capture a valid token session called "Session ID", then he uses the valid token session to gain unauthorized access to the Web Server (Owasp.org, 2016).

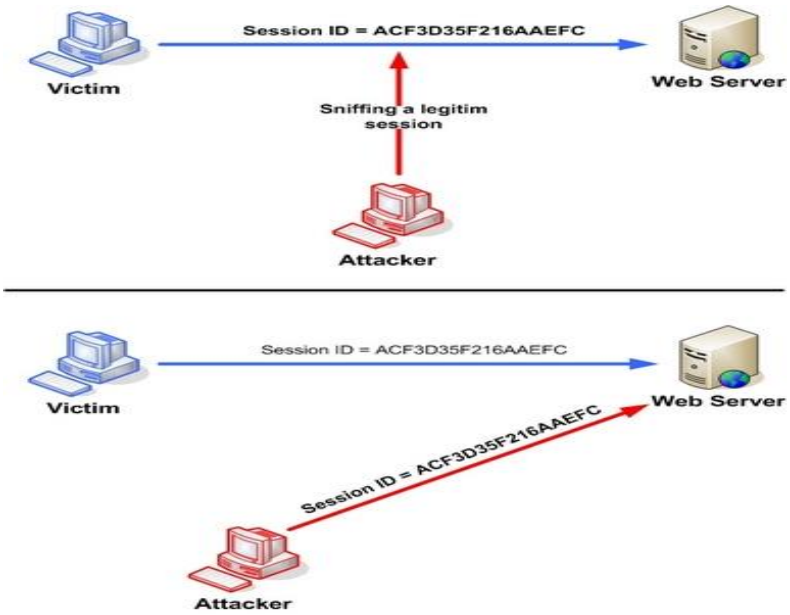


Figure 15: Manipulating the token session executing the session hijacking attack

4.1.6 *Man in the middle attack*

The man-in-the middle attack (Owasp.org, 2016) intercepts a communication between two systems. For example, in an http transaction the target is the TCP connection between client and server. Using different techniques, the attacker splits the original TCP connection into two new connections, one between the client and the attacker and the other between the attacker and the server, as shown in figure 16. Once the TCP connection is intercepted, the attacker acts as a proxy, being able to read, insert and modify the data in the intercepted communication.

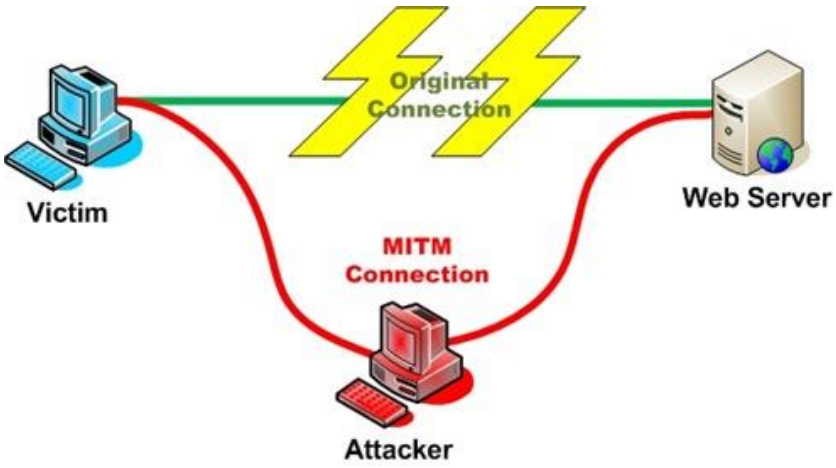


Figure 16: Man in the middle attack

In case of successful MITM, the attacker can have several consequences like DNS poisoning, denial of service attack or even Https sniffing. In case of sniffing, any textual or graphical password can be observed by the intruder, especially when data transfer in TCP protocol as data are transferred without any encryption (Daud et al., 2011). The MITM attack could also be done over an https connection by using the same technique; the only difference consists in the establishment of two independent SSL sessions, one over each TCP connection. The browser sets a SSL connection with the attacker, and the attacker establishes another SSL connection with the web server. In general the browser warns the user that the digital certificate used is not valid, but the user may ignore the warning because he doesn't understand the threat. In some specific contexts it's possible that the warning doesn't appear, as for example, when the Server certificate is compromised by the attacker or when the attacker certificate is signed by a trusted CA and the CN is the same of the original web site.

The following table presents the differences between session fixation and session hijacking vulnerabilities in terms of attack timing, impact duration, session maintenance, attack vectors and attack target area (Kolšek, 2002).

Table 6: Differences between session fixation and session hijacking

Timing	
Session fixation	Attacker attacks the user's browser *before* he logs in to the target server.
Session hijacking	Attacker attacks the user's browser *after* he logs in to the target server.
Impact Duration	
Session fixation	Attacker gains *one-time*, *temporary* or *long-term* access to the user's session(s).
Session hijacking	Attacker usually gains *one-time* access to the user's session and has to repeat the attack in order to gain access to another one.
Session Maintenance	
Session fixation	Can require the attacker to maintain the trap session until the user logs into it.
Session hijacking	Requires no session maintenance.
Attack Vectors	
Session fixation	1. Tricking the user to log in through a malicious hyperlink or a malicious login form 2. Exploiting a cross-site scripting vulnerability on any web server in the target server's domain 3. Exploiting a meta tag injection vulnerability on any web server in the target server's domain 4. Exploiting the "session adoption" feature of some web servers 5. Breaking into any host in the target server's domain 6. Adding a domain cookie-issuing server to the target server's domain in the user's DNS server 7. Network traffic modification
Session	1. Exploiting a cross-site scripting vulnerability on the target

hijacking	server 2. Obtaining the session ID in the HTTP Referer header sent to another web server 3. Network traffic sniffing (only works with an unencrypted link to the target server)
Attack Target Area	
Session fixation	Communication link, target web server, all hosts in target server's domain, user's DNS server
Session hijacking	Communication link, target web server

4.1.7 *Session Fixation Attack*

Session Fixation is an attack that permits an attacker to hijack a valid user session. The attack explores a limitation in the way the web application manages the session ID, more specifically the vulnerable web application. When authenticating a user, it doesn't assign a new session ID, making it possible to use an existent session ID. The attack consists of obtaining a valid session ID (e.g. by connecting to the application), inducing a user to authenticate himself with that session ID and then hijacking the user-validated session by the knowledge of the used session ID. The attacker has to provide a legitimate Web application session ID and try to make the victim's browser use it (Owasp.org, 2016). There are several techniques to execute the attack it depends on how the Web application deals with session tokens. Below, there are some of the most common techniques:

1. *Session ID in an URL argument*

The Session ID is sent to the victim in a hyperlink and the victim accesses the site through the malicious URL. The attacker needs to trick the user into logging in to the target web server through the hyperlink she provides, for example:

```
http://online.worldbank.dom/login.jsp?session=1234.
```

This method, while feasible, is relatively impractical and comes with quite a risk of detection (Kolšek, 2002). The figure 17 below explains a simple form, the process of the attack, and the expected results.

(1)The attacker has to establish a legitimate connection with the web server which issues a session ID

(2) or the attacker can create a new session with the proposed session ID, then,

(3) the attacker has to send a link with the established session ID to the victim, she has to click on the link sent from the attacker accessing the site,

(4) the Web Server saw that session was already established and a new one need not to be created,

(5) the victim provides his credentials to the Web Server,

(6) knowing the session ID, the attacker can access the user's account.

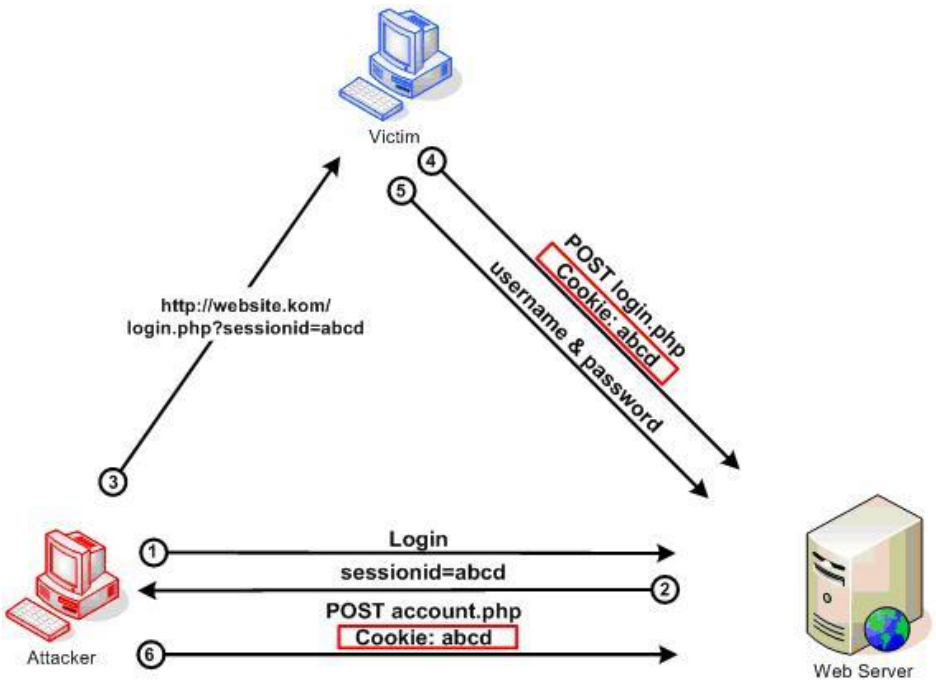


Figure 17: Simple example of Session Fixation attack.

2. *Session ID in a hidden form field*

The attacker needs to trick the user into logging in to the target web server through a look-alike login form that in reality probably comes from another web server. This method is at least as impractical and detection-prone as the former one and is included here only for the sake of completeness (Kolšek, 2002). In the best case, the attacker could exploit a cross-site scripting vulnerability on the target web server in order to

construct a login form (coming from the target server) containing a chosen session ID. However, the attacker managing to trick the user into logging in through a malicious login form could just as well direct the user's credentials to her own web server, which is generally a greater threat than that of fixing his session (Desmet et al., 2012).

3. *Session ID in a cookie*

- Client-side script

Most browsers support the execution of client-side scripting. In this case, the aggressor could use attacks of code injection as the XSS (Cross-site scripting) attack to insert a malicious code in the hyperlink sent to the victim and fix a Session ID in its cookie. Using the function document.cookie, the browser which executes the command becomes capable of fixing values inside of the cookie that it will use to keep a session between the client and the Web Application. For example in JavaScript:

```
document.cookie=”sessionid=1234”;
```

What the attacker wants to achieve is for the target web server to provide a client side script like the one above that will issue the desired trap session ID cookie to the user's browser. (Kolšek, 2002)

- <META> tag

<META> tag also is considered a code injection attack, however, different from the XSS attack where undesirable scripts can be disabled, or the execution can be denied. The attack using this method becomes much more efficient because it's impossible to disable the processing of these tags in the browsers (Desmet et al., 2012). For example:

```
<meta http-equiv=Set-Cookie content="sessionid=1234">
```

Now, the attacker wants the target server - or any other web server in the target server's domain - to return a specific, trap session ID cookie-issuing <META> tag to the user's browser. This can be achieved by using a "meta tag injection" method (Kolšek, 2002).

- HTTP header response

This method explores the server response to fix the Session ID in the victim's browser. Including the parameter Set-Cookie in the HTTP header response, the attacker is able to insert the value of Session ID in the cookie and sends it to the victim's browser (Owasp.org, 2016). For example:

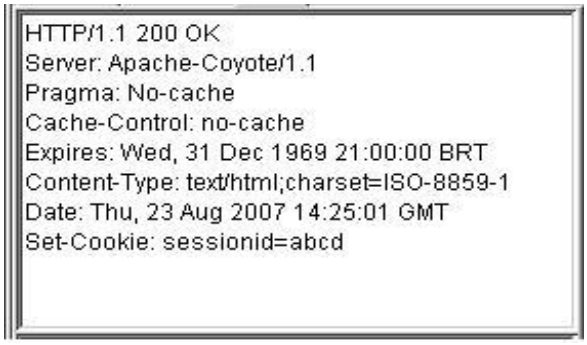


Figure 18: Set-Cookie in the HTTP header response

4.1.8 *Dictionary – Rainbow table Attack*

A dictionary attack is a technique for defeating a cipher or authentication mechanism by trying to determine its decryption key or passphrase by trying hundreds or sometimes millions of likely possibilities, such as words in a dictionary (Daud et al., 2011). It is based on trying all the strings in a pre-arranged listing, typically derived from a list of words such as in a dictionary. In contrast to a brute force attack, where a large proportion of the key space is searched systematically, a dictionary attack tries only those possibilities which are deemed most likely to succeed. Dictionary attacks often succeed because many people have a tendency to choose short passwords that are ordinary words or common passwords, or simple variants obtained, for example, by appending a digit or punctuation character. These attacks are relatively easy to defeat, e.g. by choosing a password that is not a simple variant of a word found in any dictionary or listing of commonly used passwords.

Nowadays administrators try to save the password of users in hash form. If the attackers want to find the plain text of hash value, he has two choices. One is to calculate the hash value of many plain texts to find the same hash which might take a long time. Or the other choice is pre computing the hash of billions of passwords and store them in a Rainbow table in order to find the correct password (Thorpe, 2008). These tables take a very long time and uses large space to generate, but once the attacker has the tables, it facilitates attacks by cracking a large number of passwords in a second. The main idea of the rainbow table is using chain of hashing and

reduction function. A hash function maps the plaintexts to hashes, and the reduction function reduces the length of hash function to a fix value. The chain of rainbow table starts with a plaintext and finishes with a hash value (Daud et al., 2011).

4.2 Detection

4.2.1 *Phishing Attack*

Detect the phishing Web sites in time, the user can block the sites and prevent phishing attacks. When a user visits a Web site, the anti-phishing tool searches the address of that site in a blacklist stored in the database. If the visited site is on the list, the anti-phishing tool then warns the users. Main reason of all these tools is to notify their users whether the Web site is verified and trusted. Below there are some techniques in phishing detection.

4.2.1.1 Using add-in toolbars for the browser

- Black-list/White-list based tools

Using "List Based" technique is recommended for mitigating this attack which divides to black list and white list. The black list contains the list of all phishing website which gathered by web crawlers or list maintainers but these list are helpful only if their data is accurate and fresh. The white list on the other hand includes the list of all trusted domains. So by visiting any website that does not record in white list, an alert message will be shown to the user (Huang et al., 2010). When a user visits a Web site, the anti-phishing tool searches the address of that site in a blacklist stored in the database. If the visited site is on the list, the anti-phishing tool then warns the users. The most known tools in this category is: ScamBlocker (Earthlink.net, 2016), PhishGuard (PhishGuard, 2016), Netcraft (Toolbar.netcraft.com, 2016), Google Safe Browsing (Wiki.mozilla.org, 2016), McAfee SiteAdvisor (Siteadvisor.com, 2016). The Netcraft tool bar assesses the phishing probability of a visited site by trying to determine how old the registered domain is. Google Safe Browsing uses blacklists of phishing URLs to identify phishing sites. SiteAdvisor is a database-backed solution. It includes automated crawlers that browse web sites, perform tests, and create threat ratings for each visited site. The PhishGuard system utilizes the collective observations of Internet users plus a rapid server-based submission and distribution

system. This unique architecture dramatically reduces the chance that any phishing scam can "slip through the cracks" and blindside an unsuspecting Internet user. ScamBlocker toolbar provides one-click access to user's WebMail & Google search capability, along with real-time protection from risky web sites.

- Rule-based tools

This category of tools uses certain rules in their software, and checks the security of a Web site according to these rules. Examples of this type of tools include SpoofGuard (Chou et al., 2004), TrustWatch (Geotrust.com, 2016). A well-known academic solution in literature is SpoofGuard. SpoofGuard uses domain name, url, link, and image checks to evaluate the likelihood that a given page is part of a spoof attack. It uses history, such as whether the user has visited this domain before and whether the referring page was from an email site such as Gmail or Yahoo!Mail. TrustWatch is GeoTrust's website verification service that helps the user, identify trusted websites and alerts him to potentially unsafe, or phishing web sites, which can steal private information and lead to identity theft. It also informs that this site has been verified and is not on a list of disreputable sites.

Additionally reminds users when the site was previously approved for visiting and was put on white list or if the site is block on user's personal black list. Passpet (Yee and Sitaker, 2006) is a tool that improves both the convenience and security of website logins through a combination of techniques. It uses password hashing and user assigned site lebels (petnames) that help users securely identify sites in the face of determined attempts at impersonation.

4.2.1.2 Using password hashing

Among the many implementations of password hashing are the Lucent Personalized Web Assistant (LPWA) (Gabber et al., 1999), HP Site Password (Huang et al., 2010), and PwdHash (Boneh et al., 2005). HP Site Password is a standalone tool that accepts a master password and a site name and puts the computed site password on the system clipboard so the user can paste it into a password field. PwdHash is an implementation of password hashing in Firefox. Whenever the user enters a password beginning with "@@", PwdHash hashes the password together with the domain name of the website before submitting it. LPWA is an HTTP proxy that manages usernames, passwords, and e-mail addresses to anonymize and protect the user's

website accounts. At the beginning of a session the user enters a master secret; in login forms, the user enters "\U" as the username and
"\P" as the password, which the LPWA proxy replaces with computed values (Yee and Sitaker, 2006).

4.2.1.3 Using authentication mechanisms

These mechanisms broadly fall into user authentication, server authentication and email authentication. AOL Passcode (Booth, 2004) is one such user authentication system designed to protect against password phishing. It uses a device that generates a unique six digit numeric code every 60 seconds. Microsoft, on the other hand implemented SenderID (Microsoft, 2016) to encounter the problem of domain spoofing. The Sender ID Framework is an e-mail authentication technology protocol that helps address the problem of spoofing and phishing by verifying the domain name from which e-mail messages are sent. Sender ID validates the origin of e-mail messages by verifying the IP address of the sender against the alleged owner of the sending domain.

4.2.1.4 Using visual customization

Some anti-phishing schemes employ visual customization of the login form, making the form look different for each user so that it will be hard to attackers to imitate accurately. Dynamic Security Skins (Dhamija and Tygar, 2005) transparently overlays a user-selected image over the entire login form and relies on a new login protocol, SRP, for securing the authentication process. This technique, uses a shared secret image that allows a remote server to prove its identity to a user, in a way that supports easy verification by humans and is hard to spoof by attackers. This protocol, however does not provide security for situations where the user login is from a public terminal.

4.2.1.5 Using a trusted third party

In this technique a credential is issued by a trusted third party and provides assurance on its bearer's identity. Typically, it is displayed as logos, icons, seals of the brand in the browser window to attract user's attention. TrustBar (Herzberg and Gbara, 2004), a third party certification solution against phishing. The authors propose creating a Trusted Credentials Area (TCA). The TCA controls a significant

area, located at the top of every browser window and large enough to contain highly visible logos and other graphical icons for credentials identifying a legitimate page. While their solution does not rely on complex security factors, it does not prevent against spoofing attacks. Specifically, since the logos of websites do not change, they can be used by an attacker to create a look alike TCA in an untrusted web page.

4.2.1.6 Using machine learning algorithm

On this approach (Basnet et al., 2008), the authors use learning machines in detecting and classifying phishing emails. They applied different methods for detecting phishing emails using known as well as new features. They also employ a few novel input features that can assist in discovering phishing attacks with very limited a-prior knowledge about the adversary or the method used to launch a phishing attack. The scope of this approach is to classify phishing emails by incorporating key structural features in phishing emails and employing different machine learning algorithms to our dataset for the classification process. The use of machine learning from a given training set is to learn labels of instances (phishing or legitimate emails). This method provides insights into the effectiveness of using different machine learning algorithms for the purpose of classification of phishing emails. After the performance of six different machine learning methods that they used, it is found that Support Vector Machine (LIBSVM) achieved consistently the best results.

4.2.2 *Brute force Attack*

4.2.2.1 Using flow-based detection

Brute-force attacks are most frequently detected at the host level by inspecting access logs. If the predefined number of unsuccessful login attempts is reached, an alert is fired, the attacker blocked or other attempts significantly delayed. This approach is effective, even for distributed attacks. The main drawback is that it does not scale well (Drasar et al., 2013). Below there are some detection approaches that profit from the scalability of network flows:

1. Signature-based Approach

The flow-based signatures describe network traffic by specific values, or ranges of values, of flow features and computed statistics. The signatures are then searched

in acquired flows. This is done in separate time windows, typically when exported flows are sent from the collecting process to the metering process. So this simple approach does not consider changes of the monitored traffic in time. Concerning brute-force attacks, the relevant signature can be comprised of features and statistics describing both requests and replies thanks to the interactive nature of the attacks. The requests carry attempted credentials and the replies information about whether the login was successful or not.

2. Similarity-based Approach

Deriving signatures as described above is a time-consuming process. Existing signatures need maintenance as tools and systems generating monitored traffic are evolving and traffic patterns are changing. Additionally, "0-day" attacks are not recognized. Trying to address these issues by searching for similar flows instead of matching specific signatures. It is believed that the similarity of traffic can point to machine-generated traffic, for instance brute-force attacks.

3. Automated Actions Approach

This approach uses time window heuristic and the Fourier transform. Both of them allow the discovery of almost constant periodic traffic that may be a symptom of an ongoing attack. Each method fills a specific niche; the heuristic is more useful for evenly distributed attacks, the Fourier transform for attacks with more complicated patterns.

4.2.3 *Session Hijacking Attack*

4.2.3.1 AntiSniff

AntiSniff (Spangler, 2003) is a tool made by L0pht Heavy Industries designed to detect hosts on an Ethernet/IP network segment that are promiscuously gathering data. The most current version (1.02.1) is designed to work on a non-switched network. AntiSniff performs different types of tests to determine whether a host is in promiscuous mode. The tests are broken down into the following three classes: DNS tests, operating system specific tests, and network and machine latency tests.

4.2.3.2 SessionShield

SessionShield (Johns et al., 2011) is a lightweight client-side protection mechanism against session hijacking. It is based on the idea that session identifiers are not used

by legitimate client-side scripts and thus should not be available to the scripting engines running in the browser. SessionShield detects session identifiers in incoming HTTP traffic and isolates them from the browser and thus from all the scripting engines running in it. The evaluation of SessionShield showed that it imposes negligible overhead to a user's system while detecting and protecting almost all the session identifiers in real HTTP traffic, allowing its widespread adoption in both desktop and mobile systems.

4.2.4 *Session Fixation Attack*

4.2.4.1 Serene

Serene (Desmet et al., 2012) is a self-reliant client-side countermeasure that protects the user from session fixation attacks, regardless of the security provisions of a web application. By specifically protecting session identifiers from fixation and not interfering with other cookies or parameters, Serene is able to autonomously protect a large majority of web applications, without being disruptive towards legitimate functionality. The main idea behind Serene is to prevent the browser from sending fixated session identifiers through cookies and to prevent the use of fixated session identifiers through parameters embedded in the pages' contents. To distinguish session identifiers from other cookies or parameters, Serene uses an elementary algorithm that supports a large majority of sites, but still maintains a very low false positive rate. The prototype of Serene is implemented as a Firefox add-on, available for any modern version of Firefox. Once Serene is installed, it protects against session fixation attacks without requiring any configuration.

4.2.5 *Dictionary-Rainbow Table Attack*

There are two basic approaches how to detect network anomalies. Considering host-based dictionary attack detection there are many tools that parse Unix/Linux logs such as logwatch (Logwatch, 2016), or specialized ones such as DenyHosts (Denyhosts.sourceforge.net, 2016). The former is a pluggable application that goes through system logs for a given period of time and make a report. The latter analyzes the sshd server log messages. It keeps tracks of the frequency of attempts from each host and eventually blocks hosts that try to login unsuccessfully many times. This approach is satisfactory at a host level. In short, if one host detects a dictionary attack it informs the others about the attacker’s IP address and they block the hostile address

too. Except for host-based approach, there are also network-based methods of the dictionary attack detection as Snort rules (Snort.org, 2016). These rules actually describe network traffic at a flow level although Snort was originally developed as a signature-based IDS/IPS operating at layer 7 of the ISO/OSI model. Therefore, it suffers from performance degradation on a fully saturated gigabit line. On the contrary, another approach relies entirely on traffic statistics at layers 3 and 4 of the ISO/OSI model. For example, CAMNEP (Rehak et al., 2008) is an agent-based network IDS. It comprises a few general methods of network behavior analysis and profits from the synergy effect. That means the attacks consisting of approximately up to 300 flows. General statistical-based methods are also not capable to distinguish between unsuccessful and successful attack.

4.3 Prevention

4.3.1 *Phishing Attack*

4.3.1.1 Awareness of Users

Firstly, users must educate to understand how phishing attacks work and be alert when phishing-alike e-mails are received. Here are a few basic rules to protect user's digital Identity from this type of attack (InfoSec Resources, 2012):

- Be aware of spam and adopt special cautions for email that:
  - ✓ requests confirmation for personal or financial information with high urgency.
  - ✓ requests quick action by threatening user with frightening information.
  - ✓ is sent by unknown senders.
- Verify online accounts regularly to ensure that no unauthorized transactions have been made.
- Never divulge personal information via phone or on unsecure websites.
- Do not click on links, download files, or open email attachments from unknown senders.
- Be sure to make on-line transaction only on websites that use the https protocol. Look for a sign that indicate that the site is secure (e.g., a padlock on the address bar)
- Beware of phone phishing; never provide personal information over the phone if you receive a call.

- Beware of emails that ask the user to contact a specific phone number to update user's information as well.
- Never divulge personal or financial information via email.
- Beware of links to web forms that request personal information, even if the email appears to come from a legitimate source. Phishing websites are exact replicas of legitimate websites.
- Beware of pop-ups; never enter personal information in a pop-up screen or click on it.
- Adopt proper defense systems such as spam filters, anti-virus software, and a firewall, and keep all systems updated.
- For a social network user, it's fundamental to trust no one and reveal only a limited amount of information. Never post personal information, such as a vacation schedule and home photos. Never click on links and videos from unknown origin and never download uncertificated applications.

4.3.1.2 Block the phishing e-mails by various spam filters

The spoofed e-mails used by phishers are one type of spam e-mails. From this point of view, the spam filters can also be used to filter those phishing e-mails. For example, blacklist, whitelist, keyword filters, Bayesian filters with self-learning abilities and E-Mail Stamp, etc., can all be used at the e-mail server or client systems. Most of these anti-spam techniques perform filtering at the receiving side by scanning the contents and the address of the received e-mails. And they all have pros and cons as discussed below. Blacklist and whitelist cannot work if the names of the spammers are not known in advance. Keyword filter and Bayesian filters can detect spam based on content, hence can detect unknown spams. But they can also result in false positives and false negatives. Furthermore, spam filters are designed for general spam e-mails and may not very suitable for filtering phishing e-mails since they generally do not consider the specific characteristics of phishing attacks (Chen and Guo, 2006)

4.3.2 *Brute force Attack*

Following preventive measures can be applied so as to secure web applications from brute force attacks (Daud et al., 2011):

1. *Password strength*

Password should be a combination of characters, numbers and also special characters. If the password is complex and hard to detect then it is difficult for an attacker to guess the password.

2. *Password use*

The number of attempts to login in a particular time should be recorded. If the user does not login into the system after 3 attempts then the user should be blocked for some time. Increasing the answer's complexity (e.g. requiring a CAPTCHA answer or verification code sent via cell phone), and/or locking accounts out after unsuccessful logon attempts.

3. *Use Strong Encryption*

For transfer data there must be sufficiently strong cryptography. Key lengths less than 64 bits are too weak and keys of at least 128 bits in length are recommended since they are generally considered strong enough to be secure for some time into the future.

Weak password recovery can be implemented in such faulty way that allows the attacker to easily obtains, changes or recovers another user's password. Some password recovery systems are guided by answering the user's secret question and there's no lockout policy defined so the attacker can brute force the answer and obtain the user's password. The hints that are sometimes displayed to the user can also sometimes reveal too much information and help in guessing or brute forcing the password.

4.3.3 *Replay Attack*

The most known ways of prevention a replay attack are the following:

1. *Session tokens*: Session ID or session token is a piece of data that is used in network communications (often over HTTP) to identify a session, a series of related message exchanges. Session identifiers become necessary in cases where the communications infrastructure uses a stateless protocol such as HTTP. Session tokens should be chosen by a random process (usually, pseudorandom processes are used). There are many drawbacks of session ID and it may not be enough to fulfill some security requirements. Examples of the names that some programming languages use when naming their cookie

include JSESSIONID (Java EE), PHPSESSID (PHP) and ASPSESSIONID (Microsoft ASP) (Desmet et al., 2013).

2. *One time passwords*: A one-time password (OTP) is a password that is valid for only one login session or transaction, on a computer system or other digital device. OTPs avoid a number of shortcomings that are associated with traditional (static) password-based authentication; a number of implementations also incorporate two factor authentication by ensuring that the one-time password requires access to *something a person has* (such as a small keyring for device with the OTP calculator built into it, or a smartcard or specific cellphone) as well as *something a person knows* (such as a PIN). They are similar to session tokens in that the password expires after it has been used or after a very short amount of time. They can be used to authenticate individual transactions in addition to sessions. The technique has been widely implemented in personal online banking systems (Ramasamy and Muniyandi, 2012).
3. *Nonce:* A nonce is an arbitrary number that may only be used once. It is similar in spirit to a nonce word, hence the name. It is often a random or pseudo-random number issued in an authentication protocol to ensure that old communications cannot be reused in replay attacks. They can also be useful as initialization vectors and in cryptographic hash function (Stallings, 1999).
4. *Timestamp*: A timestamp is a sequence of characters or encoded information identifying when a certain event occurred, usually giving date and time of day, sometimes accurate to a small fraction of a second. The term derives from rubber stamps used in offices to stamp the current date and sometimes time, in ink on paper documents, to record when the document was received. Synchronization should be achieved using a secure protocol (Stallings, 1999).

4.3.4 *Session Hijacking Attack*

4.3.4.1 Using IPsec

To protect from password sniffing attacks, a very good choice is using “IP Security Protocol (IPsec)” (Static.usenix.org, 2016) that secure data in the network layer by authenticating and encrypting each packet in communication. Internet Protocol

security (IPsec) uses cryptographic security services to protect communications over Internet Protocol (IP) networks. It supports network-level peer authentication, data origin authentication, data integrity, data confidentiality (encryption) and replay protection. It is also an end-to-end security scheme operating in the Internet Layer of the Internet Protocol Suite, while some other Internet security systems in widespread use, such as Transport Layer Security (TLS) and Secure Shell (SSH), operate in the upper layers at the Application layer. Hence, only IPsec protects all application traffic over an IP network. Applications can be automatically secured by IPsec at the IP layer. Cryptographic algorithms defined for use with IPsec include:

- HMAC-SHA1/SHA2 for integrity protection and authenticity.
- 3DES-CBC for confidentiality
- AES-CBC for confidentiality.
- AES-GCM providing confidentiality and authentication together efficiently.

4.3.4.2 Using multi factor authentication

Multi-factor authentication (MFA) is a method of computer access control which a user can pass by successfully presenting several separate authentication stages. Knowledge factors are the most commonly used form of authentication. In this form, the user is required to prove knowledge of a secret in order to authenticate, for example a password, PIN, secret question. Except from knowledge factor there are the possession and inherence ones. An example of possession factor is connected tokens. These are devices that are physically connected to the computer to be used and transmit data automatically. There are a number of different types, including card readers, wireless tags and USB tokens. Additionally, an example of inherence are usually biometric methods, including fingerprint readers, retina scanners or voice recognition (Du et al., 2013).

4.3.4.3 Using digital certificate

A public key certificate (also known as a digital certificate or identity certificate) is an electronic document used to prove ownership of a public key. The certificate includes information about the key, information about its owner's identity and the digital signature of an entity that has verified the certificate's contents are correct. If the signature is valid and the person examining the certificate trusts the signer, then

they know they can use that key to communicate with its owner. Certificates are an important component of Transport Layer Security (TLS, sometimes called by its older name SSL, Secure Sockets Layer), where they prevent an attacker from impersonating a secure website or other server. A certificate authority or certification authority (CA) is an entity that issues digital certificates. It is a trusted third party – trusted both by the subject (owner) of the certificate and by the party relying upon the certificate. If the CA can be subverted, then the security of the entire system is lost, potentially subverting all the entities that trust the compromised CA (Stallings, 1999).

4.3.4.4 Using One Time Cookies

One-Time Cookies (OTC) (Dacosta et al., 2012) is a robust alternative for session authentication. OTC prevents attacks such as session hijacking by signing each user request with a session secret safely stored in the browser. Unlike other proposed solutions, OTC does not require expensive state synchronization in the web application, making it easily deployable in highly distributed systems. It is implemented as a plug-in for the popular WordPress platform and as an extension for Firefox browsers. Instead of using a single, static token to authenticate each request, OTC generates a unique token per request based on a session key. Each OTC token is tied to a particular request by using a Hash-based Message Authentication Code (HMAC); hence, an adversary cannot reuse OTC tokens to illicitly redirect a session. To avoid state in the web application, OTC borrows the concept of Kerberos service tickets (Bella and Paulson, 1998). Unlike cookies, OTC credentials are also securely put in storage and isolated from other browser components. OTC preserves the performance and scalability benefits of cookies while providing stronger security guarantees.

4.3.5 *Session Fixation Attack*

First of all, it must be clear that preventing session fixation attacks is mainly the responsibility of the web application and not the underlying web server. The web server – which usually provides the session management API to applications – should make sure that session IDs can't be intercepted, predicted or brute-forced. But as far as session fixation is concerned, only the web application can implement effective protection.

4.3.5.1 Anti-Fixation in ASP

Some platforms make it easy to protect against Session Fixation, while others make it a lot more difficult. In most cases, simply discarding any existing session is sufficient to force the framework to issue a new session id cookie, with a new value. Unfortunately, some platforms, notably Microsoft ASP, do not generate new values for session id cookies, but rather just associate the existing value with a new session. This guarantees that almost all ASP apps will be vulnerable to session fixation, unless they have taken specific measures to protect against it. The idea is that, since ASP prohibits write access to the ASPSESSIONIDxxxxx cookie and will not allow users to change it in any way, they have to use an additional cookie that they do have control over to detect any tampering. So, they set a cookie in the user's browser to a random value and set a session variable to the same value. If the session variable and the cookie value ever don't match, then they have a potential fixation attack and should invalidate the session and force the user to log on again (Owasp.org, 2016).

4.3.5.2 Preventing logins to a chosen session

Session fixation can easily be addressed during the development phase of a web application, by generating a new session identifier whenever the privilege level of a user changes, for example from an unauthenticated state to an authenticated state. This approach foils any session fixation attacks aimed at obtaining an authenticated session. Even if an attacker forces a session identifier on a user, the session identifier will be overwritten by a newly generated one after authentication. Since the new SID from the fixated one, the authenticated user is never linked to the fixated session. This approach works both for cookie-based and parameter-based session management. Most web frameworks explicitly support the regeneration of SIDs, but require the developer to enable it or explicitly trigger it by calling a function (Desmet et al., 2012).

4.3.5.3 Preventing the attacker from obtaining a valid session ID

If possible, a web application on a *strict* system should only issue session IDs of newly generated sessions to users *after* they have successfully authenticated (as opposed to issuing them along with the login form). This means that an attacker who isn’t a legitimate user of the system will not be able to get a valid session ID and will therefore be unable to perform a session fixation attack (Kolšek, 2002).

4.3.5.4 Restricting the session ID usage

Most methods for mitigating the threat of stolen session IDs are also applicable to session fixation. Some of them are listed below (Kolšek, 2002):

- Binding the session ID to the user's SSL client certificate - very important and often overlooked issue in highly critical applications: *each* server-side script must first check whether the proposed session was actually established using the supplied certificate.
- Session destruction, either due to logging out or timeout, must take place on the server (deleting session), not just on the browser (deleting the session cookie).
- The user must have an option to log out – thereby destroying not just his current session, but also any previous sessions that may still exist (in order to prevent the attacker from using an old session the user forgot to log out from).
- Absolute session timeouts prevent attackers from both maintaining a trap session as well as maintaining an already entered user's session for a long period of time.

Furthermore, (Braun et al., 2011) have proposed two more server-side solutions for combating session fixation attacks against cookie-based session management. One consists of instrumenting the underlying web framework, to automatically regenerate the SID when an authentication process is detected. In a second approach, they propose a server-side proxy that maintains its own SID and couples it to the target site's SID. Renewing the proxy SID after a detected authentication process prevents session fixation attacks. Both approaches do not require modifications to either the web framework or the protected site, but depend on initial training or configuration to identify the authentication process.

4.3.6 *Dictionary Attack*

4.3.6.1 Appropriate error messages

It is important to create appropriate error messages in response to failed login attempts. Many Web sites inadvertently aid hackers by providing overly helpful error messages. Consider the difference between the messages "User ID not found" and "Incorrect password." These messages give a lot of information to a potential attacker.

"User ID not found" tells the hacker that the user he is trying to determine via brute force attack does not exist in the system. There is no point in continuing to try different passwords for this username. He can continue on to the next username in the list, saving himself thousands of useless requests and hours of time. On the other hand, "Incorrect password" tells him that the username he has tried with his dictionary attack does exist, but that the password is wrong. Now he knows that he has a potential victim and can focus his efforts on breaking that user's password. It is much safer for the application to respond with an ambiguous message like "Incorrect username or password" when a login attempt fails. There is no way to tell from this error which part of the credential was invalid. Therefore, there are no clues that a hacker can obtain from this error that can help him reduce his workload and break the system faster (Sullivan, 2007).

4.3.6.2 Delayed response

In this scheme, the server provides a delayed response to the user request, say for example, not faster than one answer per second. This may prevent an attacker from checking enough number of passwords in a reasonable time. There is one serious shortcoming to the incrementing delay approach: state must be kept in order to record the number of failed login attempts by the current user. The dictionary attack tool can be set up to begin a new session on every request by never sending a session identification token to the server. In this situation, the server will not be able to track the number of failed logins and the delay will not be properly applied. It is possible to track a user from his IP address instead of his session token, but this technique has problems as well. Sometimes multiple users share a single IP address and sometimes a single user can change IP addresses between requests. While the incrementing delay technique is not perfect, in many cases it is a better solution to fighting a dictionary attack than the widely used practice of locking out accounts after failed login attempts (Goyal et al., 2006)

4.3.6.3 Use of CAPTCHA

CAPTCHA (Captcha.net, 2016) stands for Completely Automated Public Turing Test to Tell Computers and Humans Apart. In this scheme, some challenge is put forward to the user while attempting to login. It has been established that these challenges, for example a distorted and cluttered image of a word with textured

background, are easy for humans to respond but rather difficult for computers (an online attacker is essentially a programmed computer) to answer (Goyal et al., 2006).

4.3.6.4 Account Locking

Accounts are locked a few unsuccessful login attempts (for example, an account is locked for an hour after five unsuccessful attempts.) Like the previous measure, this measure is designed to prevent attackers from checking sufficiently many passwords in a reasonable time. These countermeasures can be quite useful in a single computer environment, where users login to a local machine using, say, a keyboard that is physically attached to it (Pinkas and Sander, 2002).

4.3.6.5 Use of salt technique

To prevent the risk of Rainbow table, the administrators adds a random character named salt before hashing. The salt value is stored in the database for each user. During every authentication, a new challenge is generated by the server, the Rainbow tables need to either include all the salt combinations which would make them unmanageably large, or recalculate the table every time which makes them similar in terms of efficiency to brute force attacks. In this situation, the attacker needs to find the correct salt for each of his hashing which makes the process much too long. If the selected salt key is long enough, compromising the password would be much harder for the attacker (Thorpe, 2008).

Chapter 5: Cross-Site Scripting (XSS)

5.1 Introduction

On the OWASP top ten in 2013 (Owasp.org, 2015) cross site scripting was rated the number three attack in web applications. In this attack, the attacker injects the code into the output application of web page which will be sent to a visitor's web browser and then, the code which was injected will execute automatically or steal the sensitive information (cookies) from the user input. The malicious data can be embedded either on the server side when Web application constructs web page to respond to the request or on the client side when Document Object Model (DOM) is being built and client side scripts are being executed. In both cases, the problem is that data that is used to create the page and that can be controlled by an attacker is not being properly encoded or filtered. Therefore, an attacker can inject his own code and execute it via supported scripting language, most often Javascript. XSS allows the attacker to inject malicious code because developer trusts user inputs and does not filter the input data. XSS attacks are easy to execute, but the detection and prevention of them is very difficult. One serious reason for that is the high flexibility of HTML encoding schemes, giving the attacker more benefits (Owasp.org, 2016).

Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it. According to NVD (National Vulnerability Database), XSS vulnerability has been discovered for various applications every year. The following table shows the related vulnerability found in different years from 2002 to 2015 (Web.nvd.nist.gov, 2015). This kind of threat counts topped at 2009 and maintains high percentages until 2015.

Table 7: Cross site scripting Statistical Report

Year	Matches	Total	Percentage
2002	34	2,156	1.58%
2003	33	1,527	2.16%
2004	21	2,451	0.86%
2005	31	4,931	0.63%
2006	102	6,608	1.54%

2007	344	6,514	5.28%
2008	790	5,632	14.03%
2009	821	5,732	14.32%
2010	594	4,639	12.80%
2011	454	4,150	10.94%
2012	721	5,288	13.63%
2013	616	5,186	11.88%
2014	1,030	7,937	12.98%
2015	648	5,023	12.90%

According to Acunetix Web Application Vulnerability Report 2015 (Acunetix, 2015), cross site scripting is the top vulnerability. For this report, Acunetix have aggregated the findings of over ***15,000 scans*** performed on ***1.9 million files*** (March 2014-March 2015).

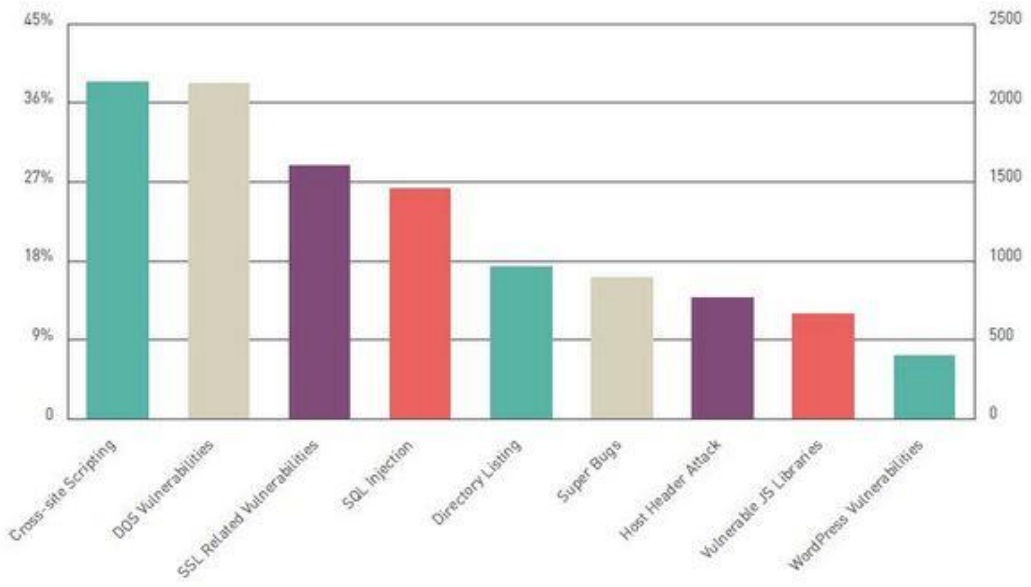


Figure 19: Statistical Vulnerabilities report 2015

There are three types of XSS vulnerabilities and they differ on when and how the malicious data is injected into web page:

1. Persistent or Stored XSS
2. Non-persistent or Reflected XSS
3. DOM Based or Local XSS

For years, most people thought of these (Stored, Reflected, DOM) as three different types of XSS, but in reality, they overlap. You can have both Stored and Reflected DOM Based XSS. You can also have Stored and Reflected Non-DOM Based XSS too, but that's confusing, so to help clarify things, starting about mid-2012, the research community proposed and started using two new terms to help organize the types of XSS that can occur:

- *Server XSS* : Server XSS occurs when untrusted user supplied data is included in an HTML response generated by the server. The source of this data could be from the request, or from a stored location. As such, you can have both Reflected Server XSS and Stored Server XSS. In this case, the entire vulnerability is in server-side code and the browser is simply rendering the response and executing any valid script embedded in it.
- *Client XSS* : Client XSS occurs when untrusted user supplied data is used to update the DOM with an unsafe JavaScript call. A JavaScript call is considered unsafe if it can be used to introduce valid JavaScript into the DOM. This source of this data could be from the DOM, or it could have been sent by the server (via an AJAX call, or a page load). The ultimate source of the data could have been from a request, or from a stored location on the client or the server. As such, you can have both Reflected Client XSS and Stored Client XSS (Owasp.org, 2016).

In this chapter, it will be given the description of the above attacks with annotative examples and the defense methods against them.

5.2 Types of XSS

Cross-site Scripting can be classified into three major categories - Stored XSS, Reflected XSS and DOM-based XSS. In the figure 20 (Acunetix, 2015) it is clear that the most common type of XSS is Reflected one.

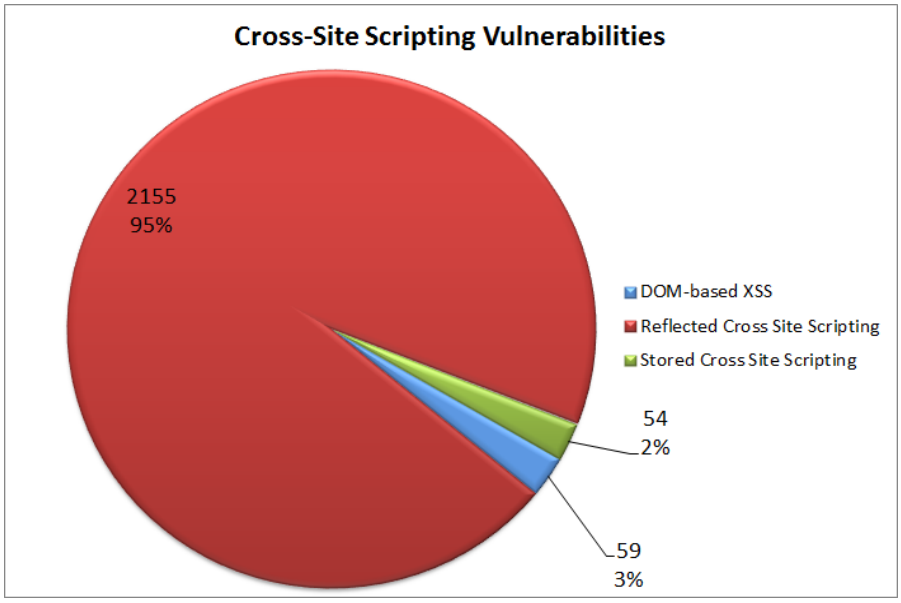


Figure 20: Identified Cross-site Scripting (XSS) vulnerabilities.

5.2.1 *Stored XSS*

The most damaging of the three XSS types is Stored (Persistent) XSS (Johns, 2011). In this attack the injected script is permanently stored on the target servers, such as in a database, blog entries, web forum posts or other sections on the website which required user input at one time. Later, when the user enters in the malicious page then the malicious code is executed in the user's web browser. The browser executes the code or script because the vulnerable server is usually a known or trusted site. The malicious payload does not have to be echoed back immediately. It is possible that it's echoed back immediately, but the main point is that the server embeds payload to the page every time it is requested and it is no more necessary to trick the user to follow a malicious link. The following figure 21 (Johns, 2011) shows how an adversary is able to persistently inject the malicious script in the vulnerable application's storage.

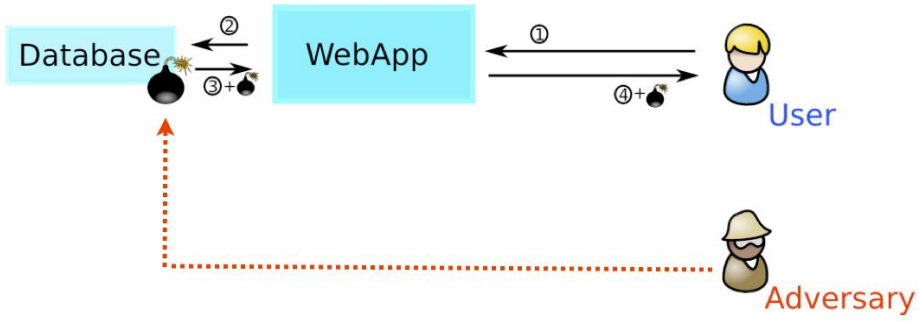


Figure 21: Stored XSS

For example, if an attacker injects malicious payload through some vulnerable page that can look like this (Garcia and Navarro, 2009):

```
<HTML>
<title>Welcome!</title>
Hi everybody! See that picture below, that's my city, well where I come from ...<BR>
<img src="city.jpg">
<script>
document.images[0].src="http://www.malicious.domain/city.jpg?stolencookies=" +document.cookie;
</script>
</HTML>
```

can use this vulnerability for stealing user's personal data, session hijacking, web defacements or performing phishing scams.

Stored XSS issues are the foundations of self-replicating XSS worms. Such worms replicate within the pages of a given web application, spreading the script on multiple pages. Table 8 (Cao et al., 2012) shows the categories of real world XSS worms with specified time period. Most of these XSS worms exploit the vulnerabilities of Stored XSS attack on real world web applications (like Facebook, Twitter etc.)

Table 8 : List of real-world XSS worms

XSS worms	Discovered in
Myspace Samy Worm	October, 2005
Xanga	December, 2005
JavaScript Yamanner Worm	June, 2006
SpaceFlash Worm	July, 2006
My Year Book	July, 2006
Gaia	January, 2007
U-Dominion	January, 2007
Orkut 'Bom Sabado' Worm	December, 2007
Hi5	December, 2007
Justin.tv	June, 2008
Twitter	April, 2009
Renren	2009

Apache Tomcat	February, 2010
Boonana Java Worm	2010
Facebook XSS Worm	March, 2011

The first major and most famous worm of this kind was The Samy Worm which struck MySpace.com in October 2005 (Grossman, 2006). It used a hole in the filtering mechanism that should have blocked all the malicious content if a user tried to enter one. The author of the worm, Samy, succeeded in avoiding filter rules and managed to upload the malicious code to his own profile. The malicious code was embedded in his profile and served to every user that visited it, thus making it a typical example of exploiting stored XSS vulnerability. The malicious code did not only infect the users that opened his profile, but also used the hole in MySpace to copy the source code to the visitor's profile, so it began self-propagating. Over the period of 24 hours, it infected over 1.000.000 users before MySpace had to shutdown his servers. It set a record in fastest propagation leaving far behind worms such as Blaster, Slammer or Code Red. The good thing in this story is that "malicious" code had not done anything malicious. It added Samy to the victim's friends list and added to the profile a message: "but most of all, Samy is my hero."

5.2.2 *Reflected XSS*

This second category, defined in this section as non-persistent attack (and also referred in the literature as reflected attack), exploits the vulnerability that appears in a web application when it utilizes information provided by the user in order to generate an outgoing page for that user (Garcia and Navarro, 2009). This is most common and simple XSS scenario. An attacker crafts link with malicious data that points to the vulnerable web site and tricks the victim to visit it. After victim opens link, vulnerable application takes malicious data from the parameters, embeds it in HTML page and echoes it back to the user (Owasp.org, 2016). Since Reflected XSS isn't a persistent attack, the attacker needs to deliver the payload to each victim - social networks are often conveniently used for the dissemination of Reflected XSS attacks. The following figure 22 (Johns, 2011) shows how a reflected attack is executed.

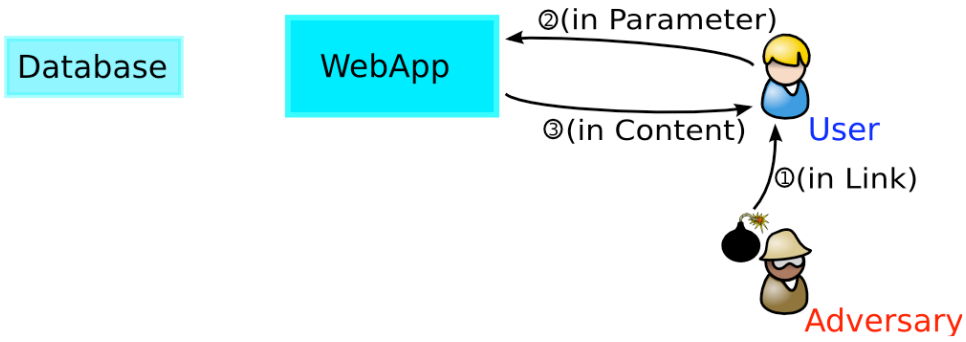


Figure 22: Reflected XSS

For example, if the attacker finds XSS hole in some bank site that the victim uses, he can trick the victim to follow the link to the bank's web site with the malicious payload in the parameter. The malicious payload can be used for stealing the cookie (Garcia and Navarro, 2009):

```
<HTML>
<title>Welcome!</title>
Click into the following <a href='http://www.trusted.domain/VWA/
<script>\
document.location="http://www.malicious.domain/city.jpg?stolencookies="
+document.cookie;\
</script>'>link</a>.
</HTML>
```

If the victim is logged in to the bank site or the credentials are sent automatically when accessing the web site, the victim's credentials stored in the cookie, will be sent via a parameter to the attacker's web site. The attacker can then use this cookie to impersonate victim and access his private data.

5.2.3 *DOM-based XSS*

DOM-based XSS (Fogie et al., 2007) is an advanced type of XSS attack which is made possible when the web application's client side scripts write user provided data to the Document Object Model (DOM). The data is subsequently read from the DOM by the web application and outputted to the browser. If the data is incorrectly handled, an attacker can inject a payload, which will be stored as part of the DOM and executed when the data is read back from the DOM. Unlike the first two attacks, this type of XSS vulnerability doesn't rely on embedding the data on the server side it is often a client-side attack, and the attacker's payload is never sent to the server. This

makes it even more difficult to detect for Web Application Firewalls (WAFs) and security engineers analyzing the server's logs since they will never even see the attack. Among various objects that make up the DOM, there are some objects in particular which an attacker can manipulate in order to generate the XSS condition. Such objects include the URL (document.URL), the part of the URL behind the hash (location.hash) and the Referrer (document.referrer). The web browser on the client side is responsible for connecting Javascript functions with malicious data injected in page URL, HTTP headers or somewhere else(Cwe.mitre.org, 2016).For example, the following HTML page (supposedly this is the content of http://www.vulnerable.site/welcome.html) (Webappsec.org, 2016):

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi
<SCRIPT>
var pos=document.URL.indexOf("name=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
<BR>
Welcome to our system
...
</HTML>
```

Normally, this HTML page would be used for welcoming the user, e.g.:

```
http://www.vulnerable.site/welcome.html?name=Joe
```

However, a request such as:

```
http://www.vulnerable.site/welcome.html?name=
<script>alert(document.cookie)</script>
```

would result in an XSS condition. The explanation why is that: the victim's browser receives this link, sends an HTTP request to www.vulnerable.site and receives the above (static!) HTML page. The victim's browser then starts parsing this HTML into DOM. The DOM contains an object called document, which contains a property called URL and this property is populated with the URL of the current page, as part of DOM creation. When the parser arrives to the Javascript code, it executes it and it modifies the raw HTML of the page. In this case, the code references document.URL and so, a part of this string is embedded at parsing time in the HTML, which is then

immediately parsed and the Javascript code found (alert(...)) is executed in the context of the same page, hence the XSS condition.

As a conclusion for the three types of XSS types that are analyzed in this section, there is Table 9 (Webappsec.org, 2016) below shows a comparison between standard XSS (non-persistent and persistent) and DOM Based XSS:

Table 9: Comparison between standard XSS(non-persistent and persistent) and DOM Based XSS

	Standard XSS	DOM Based XSS
Root cause	Insecure embedding of client input in HTML outbound page	Insecure reference and use (in a client side code) of DOM objects that are not fully controlled by the server provided page
Owner	Web developer (CGI-Common Gateway Interface)	Web developer (HTML)
Page nature	Dynamic only (CGI script)	Typically static (HTML), but not necessarily.
Vulnerability Detection	• Manual Fault injection • Automatic Fault Injection • Code Review (need access to the page source)	• Manual Fault Injection • Code Review (can be done remotely!)
Attack detection	• Web server logs • Online attack detection tools (IDS, IPS, web application firewalls)	If evasion techniques are applicable and used - no server side detection is possible
Effective defense	• Data validation at the server side	• Data validation at the client side (in Javascript)

	• Attack prevention utilities/tools (IPS, application firewalls)	• Alternative server side logic

5.3 Detection

5.3.1 *Using scanning tools*

There are many tools that scan for XSS vulnerabilities but most of them search for only Reflected XSS, and only few search for DOM based XSS. In this section, the following tools will be examined: Gamja, Wapiti, w3af, Javascript XSS scanner, SecuBat, XSSDS.

5.3.1.1 Gamja

Gamja is a command-line tool written in Perl that scans for XSS vulnerabilities and SQL injection flaws (SourceForge, 2013). It is still in early beta phase. Gamja searches only for Reflected XSS. The XSS vector that Gamja provides through request is simple: ">XSS_Check. If that vector occurs in response, it concludes there is a XSS problem. The problem with this technique is that this payload doesn't cover all cases when injection can occur and execute. It is more a sign that the payload was echoed back and that XSS might occur.

5.3.1.2 Wapiti

Wapiti is a Web application vulnerability scanner / security auditor written in Python. It performs black-box scans and acts like a fuzz, injecting payloads to see if application is vulnerable (Wapiti.sourceforge.net, 2016). It searches for file handling errors, XSS injections and various injection flaws (SQL, LDAP, CRLF,...).Wapiti searches for both Reflected and Stored XSS. It constructs XSS vector in more complicated way that can later be used to find out from what page and parameter the vector originated.

```
<script>var wapiti_[0-9a-h]+_[0-9a-h]+=new Boolean\(\);</script>
```

The first group of characters behind 'wapiti_' is encoded string of page that XSS vector originated from and second group is encoded name of parameter that XSS vector originated. The tool claims there is a XSS vulnerability if the XSS vector is found in response.

5.3.1.3 W3af

W3af stands for Web Application Attack and Audit Framework. It is the most detailed tool so far and it searches for many vulnerabilities (W3af.sourceforge.net, 2016). It also searches for all three types of XSS. In creating XSS vectors, it uses multiple variants of injections with single quotes, double quotes or no quotes at all:

```
<SCRIPT>alert2('RANDOMIZE')</SCRIPT>
javascript:alert('RANDOMIZE');
JaVaScRiPt:alert('RANDOMIZE');
javas\tcript:alert('RANDOMIZE');
<SCRIPT>a=/XSS/\nalert(a.source)</SCRIPT>RANDOMIZE
javascript:alert("RANDOMIZE");
JaVaScRiPt:alert("RANDOMIZE");
<SCRIPT>alert("RANDOMIZE")</SCRIPT>
javas\tcript:alert("RANDOMIZE");
```

The term RANDOMIZE is then transferred into some random number value. The scanner doesn't only search for XSS vector in response, but it also analyses the response and tries to remove false positives. If XSS is not found for some parameter, the scanner even reports which filters were used to prevent it. W3af also has a plugin that search for DOM-based XSS. It searches through HTML code for script tags that use one of predefined suspicious Javascript functions with DOM variables that can be controlled by users. The list of functions is:

```
document.write
document.writeln
document.execCommand
document.open
window.open
eval
```

The list of variables that might be controlled by user is:

```
document.URL
document.URLUnencoded
document.location
document.referrer
window.location
```

If there is a connection found between those two, w3af reports a DOM based XSS bug.

5.3.1.4 Javascript XSS Scanner:

Javascript XSS Scanner is GNUCITIZEN's project that demonstrates it is possible to write XSS scanner in Javascript and execute it in every browser (Gnucitizen.org, 2016). It also uses few simple XSS vectors:

```
/XSS_SCAN"><script>
/XSS_SCAN'><script>
```

and search for them in the response.

5.3.1.5 SecuBat

A web vulnerability scanner SecuBat (Jovanovic et al., 2006) is an open-source web vulnerability scanner that relies on black-box technique to crawl and scan the web applications for the presence of exploitable XSS vulnerabilities. This vulnerability scanner incorporates three major components. The initial one is crawling component, which collects a set of target web sites. Secondly, the attack component, which initiate the configured attacks against these web sites. Lastly, the analysis component scans the results returned by the web sites to find out whether an attack was successful or not.

5.3.1.6 XSSDS :

XSSDS (Johns et al., 2008) is a passive and server side XSS detection technique named XSS-Dec, which discovers the XSS attack by measuring the deviation between

the HTTP web request and its associated HTTP response. Firstly, XSSDS discovers the non-persistent XSS attack by analyzing the HTTP request parameters with HTTP response data produced by the web application and diminished by filtering the HTTP response. Secondly, it detects the persistent XSS attack that maintains record of all used scripts in web applications and discovers the deviation in the number of scripts. So the authors had developed a training-based persistent XSS attack discoverer, which is trained by recording all the java scripts used in the web applications. If any script is recognized which is not found in the recorded list then it is reported as a XSS attack.

5.3.2 *S^2XS^2*

S^2XS^2 (Shahriar and Zulkernine, 2011) is a server side approach to automatically detect XSS attacks. It is an automated server side XSS detection approach that is developed on the concept of ‘‘boundary injection’’ to encapsulate dynamic-generated content and ‘‘policy generation’’ to validate the data. The idea of boundary injection identifies probable HTML tags, java script content and other various expected features. These probable legitimate features are analyzed throughout the generation of HTTP response web page to discover the XSS attacks. This technique is applicable for the programs implemented in JSP. The authors have also designed a prototype tool, which repeatedly introduces the comments and produce the policies for JSP programs and evaluated their approach on four real world JSP programs.

5.3.3 *Using XSS Filters*

This feature provides protection only from reflected XSS attacks. A filter can identify which portions of JavaScript code are generated from input parameters (such as the URL) and refuse to execute scripts containing such portions. Unlike its competitors, this filter attempts to account for arbitrary input transformation (using an approximate substring matching algorithm) and injection of malicious code into preexisting scripts (partial injection). The figure 23 shows how the filter interacts with the rest of the browser: it is tightly integrated into the browser framework and it is able to interpose on calls to the JavaScript engine, which happens either when (a) a <script> node or some other HTML construct is parsed by the HTML engine, (b) JavaScript evaluates strings as code (e.g. using eval or setTimeout) or (c) JavaScript uses the DOM API to generate new HTML content that is fed into the parser (Wiki.mozilla.org, 2016).

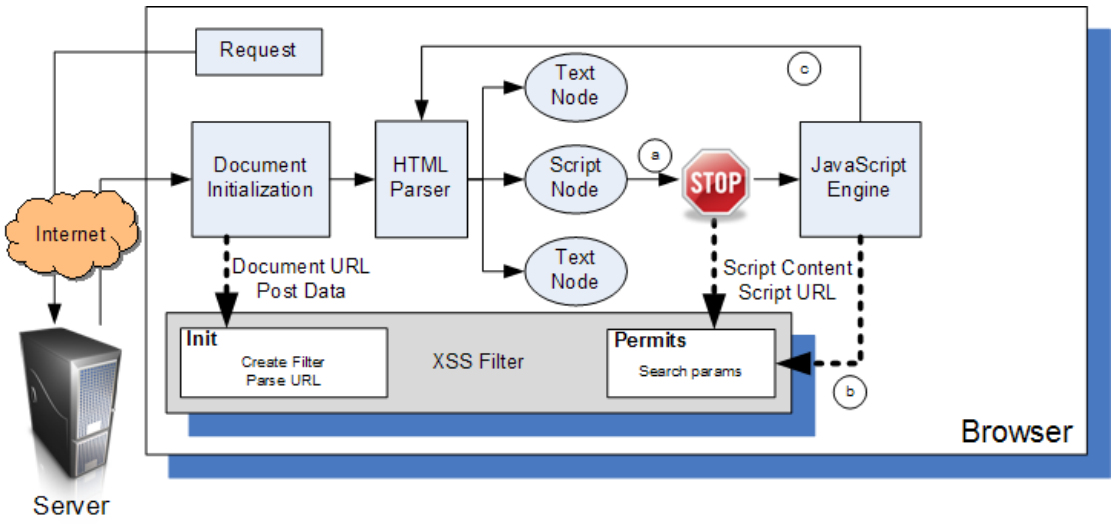


Figure 23: XSS Filter Architecture

The most known client-side XSS filters are: the IE8 filter (Blogs.technet.com, 2016), the noXSS filter (Reith, 2008), the NoScript filter (Noscript.net., 2016), XSSAuditor (Bates et al., 2010), Mozilla XSS Filter (Wiki.mozilla.org, 2016) etc.

5.3.4 *DetectCollectXSS*

DetectCollectXSS (Etoh et al., 2004) is a client-side solution that automatically detects XSS vulnerability by manipulating either client request or server response using user side local proxy servers. It provides both input and output signature detection modes. It copies the input included in the user request at a client-side web proxy before a request is sent to a web server. If the input includes an executable script and the response includes the same script copied at the proxy, then the vulnerability is detected.

5.4 Prevention

5.4.1 *Noxes*

A client-side solution for mitigating XSS attacks Noxes (Jovanovic et al., 2006) is the foremost client side solution that influences the proposal of personal firewalls for preventing the users against XSS attacks. Noxes is a Microsoft-Windows-based personal web firewall application that runs as a background service on the desktop of a user. It generally accepts the HTTP web request connections and can either be blocked or allowed based on the specified firewall rules. These rules are generally created in three ways: Manual Creation, Firewall Prompts and Snapshot Mode.

Manual Creation allows the user to open the database manually and enter a set of rules. When entering a rule, the user has the possibility of using wild cards and can choose to permit or deny requests matching the rule. In Firewall prompts, the user can interactively create a rule whenever a connection request is made that does not match any existing rule, in a way similar to what is provided by most personal firewalls. Finally in snapshot mode, the user can automatically generate permanent filter rules based on the list of domains collected during a specific session.

5.4.2 *Noncespaces*

Noncespaces (Van Gundy and Chen, 2012) is an end-to-end mechanism using randomization to defeat Cross-Site Scripting attacks and facilitates web browsers to differentiate between benign and malicious content to apply the techniques from Instruction Set Randomization (ISR) for thwarting the exploitation of XSS vulnerabilities. By means of Noncespaces, a web application randomizes the (X)HTML tags and attributes in each document before delivering it to the client and its features to discover and mitigate injected malicious script in all documents prior to transferring it to the web browser. As long as the attacker is unable to guess the random mapping, the client can distinguish between trusted content created by the web application and untrusted content provided by an attacker. The purpose of introducing the randomization is two-fold: Firstly, it discovers malicious content so that the web browser can use a policy to restrict the abilities of malicious content. Secondly, it thwarts the malicious content from altering the DOM tree. Because the randomized tags cannot be predictable by an attacker, he cannot inject the accurate delimiters in the malicious content to break the containing node without producing parsing faults.

5.4.3 *SWAP*

SWAP (Secure Web Application Proxy) (Kirda et al., 2009), is a server-side solution for detecting and preventing cross-site scripting attacks. It comprises a reverse proxy that intercepts all HTML responses, as well as a modified Web browser which is utilized to detect script content. SWAP can be deployed transparently for the client and requires only a simple automated transformation of the original Web application. Using SWAP, the user is able to correctly detect exploits on several authentic vulnerabilities in popular Web applications. SWAP is generally operated

based on the notion of discovering all static script calls in the web application and encoded them into syntactically invalid identifiers (script IDs) and therefore unknown to the java script detection component.

The proxy prevents each malicious response from being delivered to the client, and thus effectively inhibits the attack to be carried out on the client's browser. If no scripts are detected, then the reverse proxy decodes all script IDs, capably reinstating all genuine scripts and sends the HTML response to the client. On the other side, if the malicious script is detected by the java script component, then instead of delivering the HTML response to the client, SWAP alerts the client for the XSS attack.

5.4.4 *XSS-GUARD*

XSS-Guard (Bisht and Venkatakrishnan, 2008) is a server-side solution for defending against the XSS attacks by discovering the collection of scripts that a web application intends to create for any HTML web request. It is also incorporates a method for discovering the set of scripts and eliminates any type of script in the HTTP response web page that is not intended by the web application. The key benefits of the XSS-GUARD approach are:

– *Deployment friendly.* This technique does not require any significant level of human involvement in terms of code changes to be applied for XSS defense. It is based on a fully automated program transformation technique that removes the injected scripts.

– *Strong resilience.* It is highly resilient to some very subtle scenarios that occur in XSS inputs, as illustrated by our comprehensive evaluation.

– *Acceptable overheads.* It also does not impose an undue burden on web application performance.

XSS-Guard creates a shadow web page to learn the web application's intent for every HTTP web response, including simply the legitimate and expected scripts. Any divergence between the real generated web page and the shadow web page points towards the possible script inclusions.

5.4.5 *SessionSafe*

SessionSafe tool (Johns, 2006) is a combination of methods that successfully prohibits all currently known XSS session hijacking attacks. This tool utilizes deferred loading, one-time URLs and sub-domain switching to secure a web

application. It has included three server-side methods in, which attempts for XSS-immune sessions to evade XSS session hijacking. To undermine the SID accessibility, the SID is kept in a cookie which belongs to a different sub-domain than the main web application. To undermine the pre-knowledge of the application's URLs, valid One-Time URLs are hidden inside private members of the URL Randomizer JavaScript object. Finally, additional sub-domains are introduced by Sub-domain Switching, in order to create a separate security domain for every single webpage. This technique is not proposed to substitute input and output validation, a key aspect in several Web application security procedures. It must be mentioned that this server-side tool does not require any modifications in the source code of web applications and shield against XSS Attacks.

5.4.6 *BLUEPRINT*

BLUEPRINT (Louw and Venkatakrishnan, 2009) is a server-side solution to prevent XSS attacks where the web application transfers two replicas of output HTML document to a web browser for detecting any deviation, one with user inputs and other with legitimate values. The authors develop an approach that provides facilities for a web application to automatically create a structural representation — a "blueprint" — of untrusted web content that is free of XSS attacks. This approach reduces the dependency on web browsers in recognizing unsafe content and untrusted HTML over the network. BLUEPRINT offers strong protection against script injections and enables support for complex script-free HTML user input. This technique generates the parse trees for unsafe HTML content at the server side with safety measures utilized to make certain the nonexistence of script content in the tree. This parse tree generated at the web server side is transmitted to the document generator of web browser where client-side JavaScript guarantees that it cannot invoke the JavaScript interpreter. Extensive experiments with BLUEPRINT demonstrate its resilience against subtle XSS attacks, reasonable performance overheads, compatibility and effectiveness on over 96% of existing browser market share.

5.4.7 *BEEP*

BEEP (*Browser-Enforced Embedded Policies*)(Jim et al., 2007) proposes to use a modified browser that hooks all script execution attempts, and checks them against a policy, which must be provided by the server. It enhances the client's browser with the capability to detect malicious scripts based on the security policy provided by the client and prevent any malicious scripts from being executed. Two kinds of policies are suggested. Firstly, using a white list of the hashes of all allowed scripts, which the browser can check against. Secondly, labeling those nodes in the HTML source, which are supposed to contain user-provided content, so the browser can determine whether a script's position in the DOM tree is within user-provided content. The modified browser verifies each script with respect to the policy and prohibits scripts from execution that do not comply. Indeed, BEEP does not suggest whether a user should trust a particular site or the scripts it provides. Rather, BEEP ensures that a browser only runs those scripts actually endorsed by a given trusted site.

5.4.8 *WebSecurityAbstraction*

This is a method for abstracting security-critical code from large web applications systems lies in the use of a firewall to enforce restrictions at the HTTP-level (Scott and Sharp, 2002).

This system consists of a number of components:

1. A *security policy description language* (SPDL) is used to specify a set of validation constraints and transformation rules. By abstracting the security policy from the outset programmers have the advantage of a well-defined, centralized set of assertions laid out in the SPDL security specification.

2. A *policy compiler* automatically translates the SPDL into code for checking validation constraints.

3. An application-level *security gateway* is positioned between the web-server and client machines. It works at the HTTP-level and secures web-applications written in all languages. Using a security gateway to abstract security-related code is a useful technique. The security gateway rewrites the HTML in HTTP responses, annotating it with Message Authentication Codes (MACs) to protect state which may have been maliciously modified by clients. Instead of generating code for the security gateway via the SPDL compiler, the authors observe that the security policy can be encoded in a programming language of choice and compiled directly for execution on the security

gateway. Although one loses the specialized features of the SPDL, using a general purpose programming language provides designers with a greater degree of freedom. It is demonstrated that the security gateway can protect against such attacks by searching for sequences of characters which delimit embedded code in HTTP responses (e.g. <% , %> for ASP or <? php,?> for PHP). Detection of such delimiters implies (with reasonable probability) that server side code is about to leaked. Hence, if any delimiters are found the security gateway filters the HTTP response and returns a suitable error message to the client informing them that the page they requested is unavailable.

5.4.9 *WebStaticApproximation*

WebStaticApproximation (Minamide, 2005) uses a static string analysis technique to approximate possible string output for variables in a web application and checks if the approximated string output is disjoint with unsafe strings defined in a specification file. A program analyzer approximates the output of a program with a context-free grammar. If the approximate string output is disjoint with the unsafe strings, WebStaticApproximation reports that the application is not vulnerable. The analyzer is successfully applied to publicly available PHP programs to detect cross-site scripting vulnerabilities and to validate pages dynamically generated by them.

5.4.10 *DSI*

Document Structure Integrity (DSI) (Nadji et al., 2009) is a client-server architecture that enforces document structure integrity in a way that can be implemented in current browsers with a minimal impact to compatibility and that requires minimal effort from the web developer. DSI uses randomized delimiters to allow a web browser to distinguish between trusted and untrusted content. In DSI, the server identifies untrusted content using a prototype taint tracking implementation for PHP. The browser enforces a simple policy that limits untrusted content to terminals in (X)HTML and JavaScript and to tags and attributes whitelisted on a per-page basis. DSI also augments the browser with information flow tracking in order to defeat DOM-based XSS attacks.

5.4.11 *Pixy*

Pixy (Jovanovic, 2006) is an open source prototype tool that is targeted at detecting cross-site scripting vulnerabilities in PHP scripts. It statically checks if user input can reach a target statement without being processed by an input validation routine by performing flow-sensitive, interprocedural, context-sensitive data flow analysis.. Pixy does not support object-oriented features and assumes that data from object member variables and methods are malicious. The final analysis of Pixy showed that it provides a high degree of precision, keeping the number of false positives low. That fact makes this tool very useful for web applications.

5.4.12 *PQLMatcher*

Cross-site scripting can be generalized as taint-based problem. Users can specify taint-based vulnerabilities in a language called PQL (Program Query Language). In fact, PQL extends beyond even taint-based analysis as it includes execution patterns involving any sequence of methods on a set of objects that is describable via a context-free language. PQLMatcher (Martin, Livshits and Lam, 2005) identifies an object flow that matches the specification written in PQL by developers and performs a user-defined action when the match was found at runtime. An example of user-defined action is to escape special characters in a string object. PQLMatcher detects the flow of objects that matches the patterns described by a developer in PQL using both static and dynamic techniques. The synergistic combination of static and dynamic checking is a powerful one. Static analysis is performed to limit code instrumentation only to the relevant code and improve the performance in PQLMatcher and then dynamic analysis is performed to find the flow of objects that matches a specified pattern at runtime. For example, developers can write an input-escaping method that is executed when PQLMatcher detects a user input flow that matches a predefined object flow pattern at runtime.

In the Table 10 (Gupta S. and Gupta B., 2015), there are the defense techniques - tools with some of their characteristics according to the analysis that has been implemented above.

Table 10: Defense XSS techniques characteristics

Technique	Analysis method	Persistent XSS attack detection	Non-persistent XSS attack detection	DOM-based XSS attack detection	Client-side	Server-side
Noxes	Dynamic analysis	N	Y	N	Y	N
Noncespaces	String analysis	Y	Y	N	N	Y
SWAP	String analysis	Y	Y	N	Y	N
XSSDS	String analysis	Y	Y	N	N	N
XSS-GUARD	Rewriting	Y	Y	Y	N	Y
S^2XS^2	String analysis	Y	Y	N	N	Y
SessionSafe	Static analysis	Y	Y	N	Y	N
Blueprint	Parse tree	Y	Y	N	N	N
Secubat	Black box testing	Y	Y	N	Y	N
BEEP	Dynamic analysis	Y	Y	N	Y	N
DSI	Dynamic analysis	Y	Y	Y	Y	Y
WebSecurityAbstraction	Dynamic analysis	Y	Y	N	N	Y
WebStaticApproximati	Static	Y	Y	N	N	Y

on	Analysis					
DetectCollectXSS	Dynamic analysis	Y	Y	N	Y	N
Wapiti	Black box testing	Y	Y	N	N	N
Gamja	Black box testing	N	Y	N	N	N
W3af	Black box testing	Y	Y	Y	N	N
JavascriptXSS	Black box testing	N	Y	N	N	N
PQLMatcher	Dynamic analysis	Y	Y	N	N	Y
Pixy	Static Analysis	Y	Y	N	N	Y
XSSAuditor	Taint-Static Analysis	N	Y	N	Y	N

Chapter 6: Insecure Direct Object References

6.1 Introduction

Insecure Direct Object References has been included in OWASP Top 10 since 2007 (Owasp.org, 2015). In 2013, it was listed as the number fourth vulnerability with a prevalence of common. Although this vulnerability is easy to exploit and easy to detect, it is still usually ignored by developers when they are designing and implementing applications. As OWASP's description (Owasp.org, 2016), "A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. An attacker can manipulate direct object references to access other objects without authorization, unless an access control check is in place".

Below there are two typical stories which show how prevalently this vulnerability exists in web applications. Firstly, Insecure Direct Object References vulnerability was found in Yahoo!, the 4th most visited website on the Internet in February 2014. A hacker spotted it existed in the Yahoo! sub-domain 'suggestions.yahoo.com', which could allow an attacker to delete all the posted thread and comments on Yahoo's Suggestion Board website, which are totally more than 1 million and half records (Pwnrules.com, 2016). There was also an attack in year 2000 to the Australian Taxation Office's GST Start Up Assistance. The legitimate user found out that he can access other company's details by tampering with the parameters in the URL. He managed to gather information on 17.000 companies and e-mailed them to report the vulnerability in the Web application (Abc.net.au, 2016).

According to NVD (National Vulnerability Database), permissions, privileges and access control flaws have been appeared for various applications every year. The following tables show the related vulnerabilities found in different years. The improper access control presents an increase between 2014 to 2015 and the other flaws count topped at 2012 and declined in 2015 (Web.nvd.nist.gov, 2015):

Table 11: Permissions, Privileges, and Access Control Flaws Statistical Report

Year	Matches	Total	Percentage
2002	32	2,156	1.48%
2003	17	1,527	1.11%
2004	19	2,451	0.78%
2005	34	4,931	0.69%
2006	51	6,608	0.77%
2007	221	6,514	3.39%
2008	449	5,632	7.79%
2009	436	5,732	7.61%
2010	356	4,639	7.67%
2011	283	4,150	6.82%
2012	604	5,288	11.42%
2013	576	5,186	11.11%
2014	727	7,937	9.16%
2015	448	5,207	8.60%

Table 12: Improper Access Control Statistical Report

Year	Matches	Total	Percentage
2014	18	7,937	0.23%
2015	126	5,207	2.42%

One of the main reasons for the appearance of Insecure Direct Object References, is Insufficient Authorization weakness. The following figure 24 (WhiteHat, 2015) shows a percentage of 11% of this weakness.

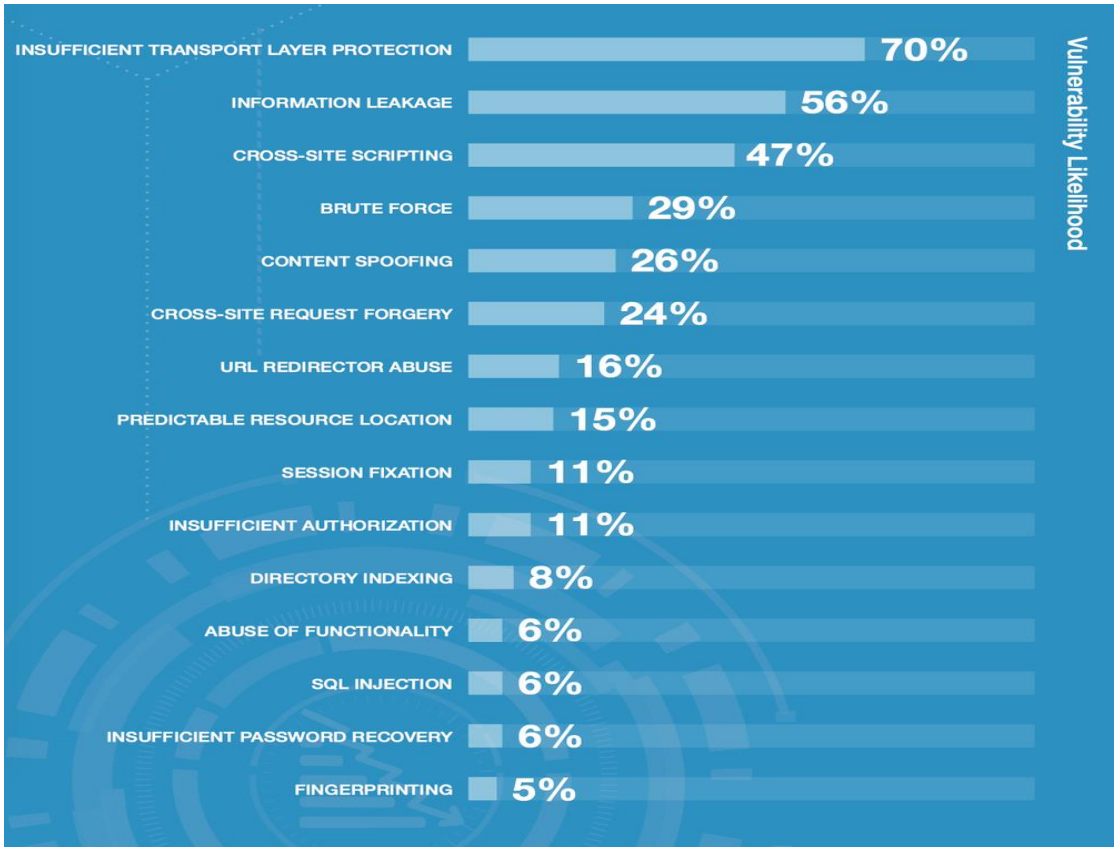


Figure 24: Application vulnerability likelihood – WhiteHat Report 2015

6.2 Types of attack

Insecure Direct Object References occur when an application provides direct access to objects based on user-supplied input. As a result of this vulnerability attackers can bypass authorization and access resources in the system directly, for example database records or files. Insecure Direct Object References allow attackers to bypass authorization and access resources directly by modifying the value of a parameter used to directly point to an object. Such resources can be database entries belonging to other users, files in the system, and more. This is caused by the fact that the application takes user supplied input and uses it to retrieve an object without performing sufficient authorization checks. In this section, there is an analysis of the most known attacks for this vulnerability with demonstrative examples of them.

6.2.1 *By modifying values of parameters in URL string*

6.2.1.1 Path/Directory traversal

The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal. Path traversal is a type of HTTP exploit that is used by attackers to gain unauthorized access to restricted directories and files and it is also known as Directory traversal (DuPaul, 2014). An attacker that exploits a directory traversal vulnerability is capable of compromising the entire web server. A directory traversal vulnerability is the result of insufficient filtering/validation of browser input from users. It can be located in web server software/files or in application code that is executed on the server and it can also exist in a variety of programming languages, including Python, PHP, Apache, ColdFusion, Perl and more (Projects.webappsec.org, 2016). The first dangerous thing is that the attacker knows user's directory structure and if there is any sensitive information in files in user's application directory, the attacker can simply download these files. A worse case is when the application enables the attacker to read files. The attacker can then read the configuration files (e.g., the connection string in web.config file) or the configuration files of the whole system (if the application is used by a highly privileged user) (Docs.kentico.com, 2016). According to the annual Verizon 2015 Data Breach Investigations Report (Verizon, 2015), 3.4% of the attacks make use of path traversal. The following figure 25 (Verizon, 2015) shows what rank the path traversal has.

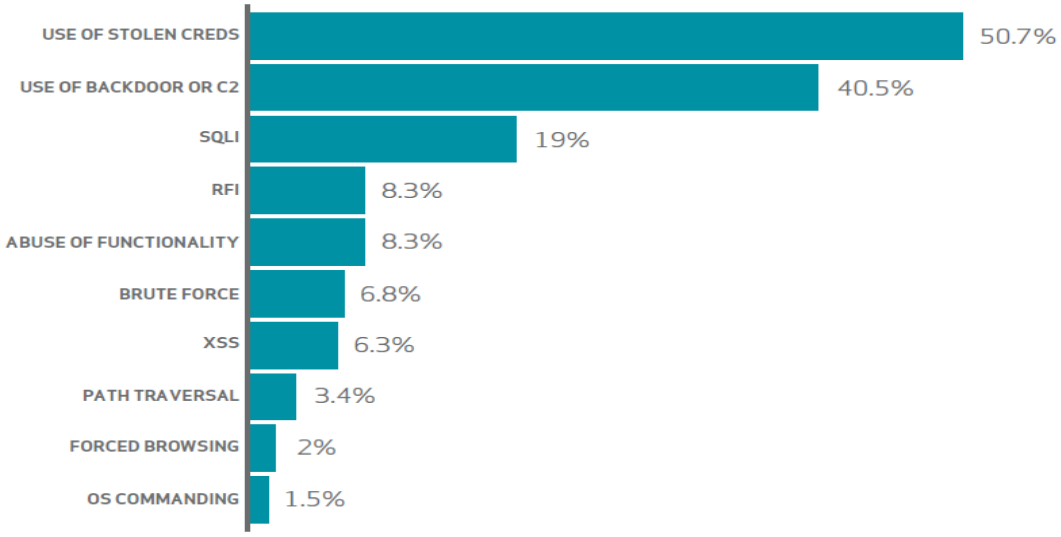


Figure 25: Variety of hacking actions within Web

According to NVD (National Vulnerability Database), the following table shows the path traversal vulnerability statistical report. There was an increase of incidents in 2008-09 and a more stable course in the last five years (Web.nvd.nist.gov, 2015).

Table 13: Path traversal Statistical Report

Year	Matches	Total	Percentage
2001	3	1,667	0.18%
2002	14	2,156	0.65%
2003	17	1,527	1.11%
2004	11	2,451	0.45%
2005	11	4,931	0.22%
2006	21	6,608	0.32%
2007	159	6,514	2.44%
2008	352	5,632	6.25%
2009	319	5,732	5.57%
2010	273	4,639	5.88%
2011	105	4,150	2.53%
2012	118	5,288	2.23%
2013	103	5,186	1.99%
2014	197	7,937	2.48%
2015	120	5,207	2.30%

Directory traversal attacks can be viewed in two basic groups: attacks that target directory traversal vulnerabilities in the web server and attacks that target vulnerabilities in application code. Attackers are able to exploit vulnerabilities in application code by sending URLs to the web server that instruct the server to return specific files to the application. For this method to work, the attacker must find a URL in which an application retrieves a file from the web server. Once attackers discover such a URL, they can simply modify the URL string with commands for the server and the name of the file they seek to access. The "../" directive is commonly used, as it instructs the web server to retrieve a file from one directory up. An attacker that is attempting to access a specific file will simply use trial-and-error to determine how many "…/" commands it takes to locate the correct directory and retrieve the file via the application. Directory traversal vulnerabilities that exist on web servers are typically exploited to execute files. The method for this type of directory traversal attack involves sending URLs to the web server that contain the name of the targeted file and have been modified with commands and web server escape codes. Escape codes are used as workarounds when certain commands are being filtered for; for example, an attacker might use the "%2e%2e/" escape code if the "../" command is blocked. Again, this directory traversal example requires trial-and-error from the attacker, but it is still fairly easy to access and execute files when adequate preventative procedures are not in place. Although most popular web servers will prevent this technique from escaping the web document root, alternate encodings of the "../" sequence may help bypass the security filters. These method variations include valid and invalid Unicode-encoding ("..%u2216" or "..%c0%af") of the forward slash character, backslash characters ("..\") on Windows-based servers, URL encoded characters "%2e%2e%2f"), and double URL encoding ("..%255c") of the backslash character (DuPaul, 2014). The following examples show some examples of the pre-mentioned attacks (Projects.webappsec.org, 2016):

- Path Traversal attacks against a web server

```
http://example/../../../../../etc/passwd
http://example/..%255c..%255c..%255cboot.ini
http://example/..%u2216..%u2216someother/file
```

- Path Traversal attacks against a web application

```
Original: http://example/foo.cgi?home=index.htm
Attack: http://example/foo.cgi?home=foo.cgi
```

In the above example, the web application reveals the source code of the foo.cgi file because the value of the home variable was used as content. Notice that in this case the attacker does not need to submit any invalid characters or any path traversal characters for the attack to succeed. The attacker has targeted another file in the same directory as index.htm.

Below there is also another example in Perl language for understanding much better the path traversal attack in an application code. The following code could be for a social networking application in which each user's profile information is stored in a separate file. All files are stored in a single directory (Cwe.mitre.org, 2016).

```
my $dataPath = "/users/cwe/profiles";
my $username = param("user");
my $profilePath = $dataPath . "/" . $username;

open(my $fh, "<$profilePath") || ExitError("profile read error: $profilePath");
print "<ul>\n";
while (<$fh>) {
print "<li>$_</li>\n";
}
print "</ul>\n";
```

While the programmer intends to access files such as "/users/cwe/profiles/alice" or "/users/cwe/profiles/bob", there is no verification of the incoming user parameter. An attacker could provide a string such as:

```
../../../etc/passwd
```

The program would generate a profile pathname like this:

```
/users/cwe/profiles/../../../etc/passwd
```

When the file is opened, the operating system resolves the "../" during path canonicalization and actually accesses this file:

```
/etc/passwd
```

As a result, the attacker could read the entire text of the password file. Notice how this code also contains an error message information leak if the user parameter does not produce a file that exists: the full pathname is provided. Because of the lack of output encoding of the file that is retrieved, there might also be a cross-site scripting problem if profile contains any HTML, but other code would need to be examined.

6.2.1.2 Null Byte Injection

Null Byte Injection (Projects.webappsec.org, 2016) is an active exploitation technique used to bypass sanity checking filters in web infrastructure by adding URL-encoded null byte characters (i.e. %00) to the user-supplied data. This injection process can alter the intended logic of the application and allow malicious adversary to get unauthorized access to the system files. URLs are limited to a set of US-ASCII characters ranging from 0x20 to 0x7E (hex) or 32 to 126 (decimal). However, the aforementioned range uses several characters that are not permitted because they have special meaning within HTTP protocol context. For this reason, the URL encoding scheme was introduced to include special characters within URL using the extended ASCII character representation. In terms of “null byte”, this is represented as %00 in hexadecimal. The scope of a null byte attack starts where web applications interact with active ‘C’ routines and external APIs from the underlying OS. Thus, allowing an attacker to manipulate web resources by reading or writing files based on the application's user privileges. Below there are some demonstrative examples.

- PHP language

For instance, if a user requests a personal data file from the server, it will be validated by appending ‘.DAT’ extension to the filename. This script itself appears to be secure by enforcing the file extension but the request for the resource can be manipulated by appending a “%00” null byte at the end of URL. Thus, a malicious adversary can take advantage of this vulnerability to read almost any system file through a simple browser.

```
$file = $_GET['file'];
require_once("/var/www/images/$file.dat");
```

Exploitation:

Normal Mode: http://www.example.host/user.php?file=myprofile.dat
Attacking Mode: http://www.example.host/user.php?file=../../../etc/passwd%00

- Java language

The trend of null byte injection attack is also common in Java. For instance, by examining the details of a vulnerable function "File ( )" inside "java.io.File" which passes its argument to the underlying 'C' API to process the user request failed to determine the actual file extension because it treats the occurrence of first null byte as a string terminator. Let us assume the following example in which a user is requesting access to the specific file where the extension is enforced as ".db" by the developer for validation purposes. The same request can be simulated by the attacker but in a different way to access the system resource by embedding a "%00" null byte with a valid filename and extension.

```
String fn = request.getParameter("fn");
if (fn.endsWith(".db"))
{
        File f = new File(fn);
        //read the contents of "f" file
        …
}
```

Exploitation:

Normal Mode: http://www.example.host/mypage.jsp?fn=report.db
Attacking Mode: http://www.example.host/mypage.jsp?fn=serverlogs.txt%00.db

6.3 Detection

The best options and quite effective at finding insecure direct object reference vulnerabilities are:

- Code reviews to identify whether important parameters are susceptible to manipulation with static analysis
- Black box testing
- Penetration testing
- Testing with dynamic analysis
- Firewall
- Sandbox – Jail

6.3.1 *Using static analysis*

Automated techniques can find areas where path traversal weaknesses exist. However, tuning or customization may be required to remove or de-prioritize path-traversal problems that are only exploitable by the software's administrator - or other privileged users - and thus potentially valid behavior or, at worst, a bug instead of a vulnerability (Cwe.mitre.org, 2016).

One important approach is code review of the application and verify whether the following mechanisms are implemented safely (Wang, 2014):

- For direct references to restricted resources, the application needs to verify the user is authorized to access the exact resource they have requested.
- If the reference is an indirect reference, the mapping to the direct reference must be limited to values authorized for the current user.

In Livshits and Lam approach (Usenix.org, 2016), is proposed a static analysis technique for detecting many application vulnerabilities such as Insecure Direct Object References. These vulnerabilities stem from unchecked input, which is widely recognized as the most common source of security vulnerabilities in Web applications. They propose a static analysis approach based on a scalable and precise points-to analysis. In their system, user-provided specifications of vulnerabilities are automatically translated into static analyzers. Their approach also find all vulnerabilities matching a specification in the statically analyzed code. Results of their static analysis are presented to the user for assessment in an auditing interface

integrated within Eclipse, a popular Java development environment. Context sensitivity, combined with improved object naming, proved instrumental in keeping the number of false positives low. In fact, only one of their benchmarks suffered from false alarms. WARlord (Møller and Schwarz, 2014), is another prototype implementation of static analysis for detecting path traversal attacks. The authors have demonstrated that it is possible to provide tool support that can effectively help programmers prevent client-state manipulation vulnerabilities in web application code. The static analysis they have presented is capable of precisely identifying client state, in particular state stored in hidden fields, and help distinguishing between safe and unsafe use of such state. Moreover, they have argued that the information inferred by the analysis can also be used for automatic configuration of a security filter that at runtime protects against client-state manipulation attacks.

6.3.2 *Black box testing*

In Sekar's approach (Sekar, 2009), there is an efficient black-box technique for defeating Web application attacks. This technique is called taint inference. It does not require any source-code or binary instrumentation of the application to be protected; instead, it operates by intercepting requests and responses from this application. For most web applications, this interception may be achieved using network layer interposition or library interposition. The author has developed a class of policies called syntax- and taint-aware policies that can accurately detect and/or block most injection attacks. This policy framework is powerful enough to detect several other types of attacks such as XPath injection and path traversals, As compared to previous works, this approach does not require extensive instrumentation of the applications to be protected. It is robust, and can easily support applications written in many different languages (Java/C/C++/PHP), and on many platforms (Apache/IIS/Tomcat).

6.3.3 *Penetration testing*

Panoptic (Websec.ca, 2016) is a Python tool written by Roberto Salgado with the collaboration and help of Miroslav Stampar, one of the developers of sqlmap. This tool searches and looks for commonly known files in user's web server like configurations, logs, histories, etc. through the path traversal vulnerability. DotDotPwn (Tools.kali.org, 2016) is a very flexible intelligent fuzzer to discover traversal directory vulnerabilities in software such as HTTP/FTP/TFTP servers, Web

platforms such as CMSs, ERPs, Blogs, etc. Also, it has a protocol-independent module to send the desired payload to the host and port specified. On the other hand, it also could be used in a scripting way using the STDOUT module. It's written in Perl programming language and can be run either under *NIX or Windows platforms. It's the first Mexican tool included in BackTrack Linux.

6.3.4 *Using dynamic analysis*

One way to test would be by having multiple users to cover different owned objects and functions. For example, assume two users each having access to different objects and with different privileges (i.e. administrator users or normal users). Log in as one user to see whether there are direct references to objects or functionality that belong to other users. Another advantage of having multiple users is to save testing time in guessing different object names that belong to other users. Typical scenarios for this vulnerability and the methods to test for each include (Owasp.org, 2016) (Owasp.org, 2016):

- The value of a parameter is used directly to retrieve a database record.

Sample request:

```
http://foo.bar/somepage?invoice=12345
```

In this case the value of the "invoice" parameter is the object id used as an index to query the invoice record in the database. By modifying the value of the parameter, it is possible to retrieve any invoice record, regardless of whether the object belongs to the user.

- The value of a parameter is used directly to perform an operation in the system. Sample request:

```
http://foo.bar/changepassword?user=someuser
```

In the example above, the value of the "user" parameter is used to tell the application for which user it should change the password. Typically it is the first step in the password changing wizard and leads to a page that requests new password for the specified user. This URL can be used to test whether a logged in user can open the password modification page for another user.

- The value of a parameter is used directly to retrieve a file system resource.

Sample request:

```
http://foo.bar/file.jsp?file=report.txt
```

In this case, the value of the "file" parameter is used to specify what file the user intent to retrieve. By modifying this value, attackers will be able to retrieve objects belonging to other user. Another sample request can be:

```
http://foo.bar/file.jsp?file=**../../../etc/shadow**
```

In this case that attacker is using directory traversal attack and attempt to retrieve the shadow file for cracking the passwords in the system.

- The value of a parameter is used directly to access application functionality.

Sample request:

```
http://foo.bar/accesPage?menuitem=12
```

In the above example the value of the "menuitem" parameter is used to tell the application which menu item (and therefore which application functionality) the user wants to access. By modifying this value, a user can bypass authorization and access application functionality that are not included in the privileges of this user account.

6.3.5 *Using firewall*

Another detection method for path traversal attack is to use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth. An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization (Cwe.mitre.org, 2016).

6.3.6 *Using Sandbox-Jail*

In this approach user must run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed (Cwe.mitre.org, 2016).

6.4 Prevention

6.4.1 *Access control*

One essential defense is to check access control. On each use of a direct object reference from an untrusted source, the application should perform an access control check to ensure the user is authorized for the requested object or service. One way to implement this is to use role-based authorization (Stuttard and Pinto, 2008). There are named roles, which contain different sets of specific privileges and each user is assigned to one of these roles. This serves as a shortcut for assigning and enforcing different privileges and is necessary to help manage access control in complex applications. Using roles to perform upfront access checks on user requests enables many unauthorized requests to be quickly rejected with a minimum amount of processing being performed. An example of this approach is in protecting the URL paths that specific types of user may access. When designing role-based access control mechanisms, it is necessary to balance the number of roles so that they remain a useful tool to assist in the management of privileges within the application.

6.4.2 *Indirect Reference Map:*

An indirect reference map is a substitution of the internal reference with an alternate ID. It is used for mapping from a set of internal direct object references (i.e. database keys, filenames, etc.) to a set of indirect reference that can be safely exposed externally. An indirect reference map can be implemented as follows (Wang, 2014):

- Create a map on the server between that actual key, user ID in the database (i.e. 1011, 1012, …,) and the substitution, which can be a long hash value or a GUID that is easily generated but difficult to be predicted by users.
- The user ID is translated to its substitution key before being exposed to the UI.
- After the substitution key is returned to the server, it is translated back to the original user ID before the record is retrieved.
- Encoding Data: For example encode the name of your file with common encoding methods (Benoist, 2015):

download.php?getfile=Y29uZmlnLnBocA==

  In above example the file name is base64 encoded.
- Encrypting Data (Benoist, 2015): Where a partial filename was used, prefer a hash of the partial reference. In this method you must use encrypting algorithms:

download.php?getfile=%63%6F%6E%66%69%67%2E%70%68%70%0

According to Enterprise Security API (ESAPI) (Owasp-esapi-java.googlecode.com, 2016), there is a class called RandomAccessReferenceMap that helps developers to prevent the applications from this kind of vulnerability. The AccessReferenceMap interface is used to map from a set of internal direct object references to a set of indirect references that are safe to disclose publicly. This can be used to help protect database keys, filenames, and other types of direct object references. As a rule, developers should not expose their direct object references as it enables attackers to attempt to manipulate them. Indirect references are handled as strings, to facilitate their use in HTML. Implementations can generate simple integers or more complicated random character strings as indirect references. They should probably add a constructor that takes a list of direct references. In addition to defeating all forms of parameter tampering attacks, there is a side benefit of the

AccessReferenceMap. Using random strings as indirect object references, as opposed to simple integers makes it impossible for an attacker to guess valid identifiers. Here's an example of how someone might use it.

```
yObject obj;                    // generate your object
Collection coll;        // holds objects for display in UI

//create ESAPI random access reference map
AccessReferenceMap map = new RandomAccessReferenceMap();

//get indirect reference using direct reference as seed input
String indirectReference = map.addDirectReference(obj.getId());

//set indirect reference for each object - requires your app object to have this method
obj.setIndirectReference(indirectReference);

//add object to display collection
coll.add(obj);

//store collection in request/session and forward to UI
...
```

6.4.3 *Input User Validation*

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules (Stuttard and Pinto, 2008). As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." There must be validation performed in server side, since client-side validation cannot guarantee evil input to be avoided. For instance, attackers may make use of proxy tools to capturing and reissuing HTTP requests to bypass the client-side validation. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks

or determining which inputs are so malformed that they should be rejected outright (Cwe.mitre.org, 2016).

6.4.4 *Use per user or session indirect object references*

This approach can be used to prevent attackers from directly targeting unauthorized resources. Per page authorization is another technique that has a similar goal to the AccessReferenceMap. The idea here is that user already done the appropriate authorization on the master page. For simplicity, let's use the credit card example. Let's say user has two credit cards with a particular bank, and both of those cards show up on his "master" accounts page. In this method would be to take the two credit card records and store them in an array specific to that user. The code has to check what user has access to (authorization) in order to only display the appropriate 2 credit cards to him on that page (Jtmelton.com, 2010). The credit card selection box would be coded like this (Enterprisenetworkingplanet.com, 2016):

```
<select name=" choosecreditcard">
        <option value="1">
           XXXXXXXXXXXX6002
        </option>
        <option value="2">
           XXXXXXXXXXXX1516
        </option>
     </select>
```

This way there is only a direct reference to an array for that user, containing only that user's data. Changing the option value to a value greater than 2 would not result in any other user's credit card details being used. The application would then map the user specific indirect object reference (option value=1 or option value=2) back to the underlying database key (35 or 67 in the example earlier.) What the per-page authorization says is that if and when the attacker tampers with the data sent to the detail page, it still shouldn't be possible for him to see data that is not his. The code that brings up detail about a specific credit card should first include a check that you have access to that card (authorization). Again, if he doesn't have access to that card, either the code is broken, or he is tampering and that is a security incident. Another example is to use a drop down list of resources authorized instead of database key for

the current user to limit the user input. The application has to map the per-user indirect reference back to the actual database key on the server (Owasp.org, 2016).

Chapter 7: Security misconfiguration

7.1 Introduction

Security misconfiguration was rated the number five threat in web applications on the OWASP top ten in 2013 (Owasp.org, 2015). Security misconfiguration vulnerabilities could occur if a component is susceptible to attack due to an insecure configuration option. These vulnerabilities often occur due to insecure default configuration, poorly documented default configuration, or poorly documented side-effects of optional configuration. This could range from failing to set a useful security header on a web server, to forgetting to disable default platform functionality that could grant administrative access to an attacker (Kemptechnologies.com, 2015). When any of the many components that make up a web application are not security hardened, or are configured badly, then there is a target for attackers. In order to provide a secure deployment all the component parts of a web application need to be configured correctly. Not having configuration done correctly might actually be worse than not making any changes to default settings. Given the large number of components that are combined to deploy a web application and the intrinsic complexity of most of these components, it won't be a surprise that there are many ways that misconfiguration can lead to security vulnerabilities. All of which can lead to application hijacking, data loss, reputational damage and expensive downtime. Many users and professionals use hundreds of different software components, including server software, libraries and application frameworks. Each of these components has complex configuration options. It is very important to rely on extensive automation (configuration management, automated deployment, unit testing) to minimize the risk of misconfiguring a security critical component. If an insecure configuration is identified, users can make a centralized change and add a new test case to ensure they do not regress in the future. Common security misconfigurations include (Hurkala A. and Hurkala J., 2013):

1. Default administrator password is not changed – attacker can gain full control over a website i.e. attacker can usually add/modify/delete accounts and access log files.

2. Test accounts are not disabled after they are no longer needed – test accounts are an easy target for brute-force attacks because usernames and passwords are very often easy to guess.
3. Directory listing is not disabled on a web-server – attacker can list directories and possibly find sensitive data such as log files, configuration files or user database files.
4. System logs, configuration files, user database or other sensitive data is located in not protected location inside web-server's root directory – attacker might guess filenames and download sensitive files even if directory listing is disabled.
5. Web-application is configured to output various debug information such as stack- trace or database errors, which can be used to learn important details about website and the underlying application or to perform various attacks such as SQL injection.
6. Applications are not removed from server after there are no longer needed – attacker can exploit those applications even though they should have been removed from the server. This especially applies to old applications that are no longer maintained.

The goal of this chapter is to analyze the main types of attacks of this threat and to recommend the best detection and prevention ways for avoiding to compromise sensitive data to attackers. According to Web Hacking Incident Database (WHID) (Google.com, 2016) the main attack methods for this security weakness are:

1. Remote File Inclusion (RFI)
2. Clickjacking
3. Predictable Resource Location
4. Server Misconfiguration
5. Abuse of Functionality
6. OS Commanding
7. Brute Force
8. Application Misconfiguration
9. Content Spoofing

Reading the annual Verizon 2015 Data Breach Investigations Report, most of the attacks that are mentioned above appear in the following figure 26 (Verizon, 2015), which shows how serious problem the security misconfiguration is.

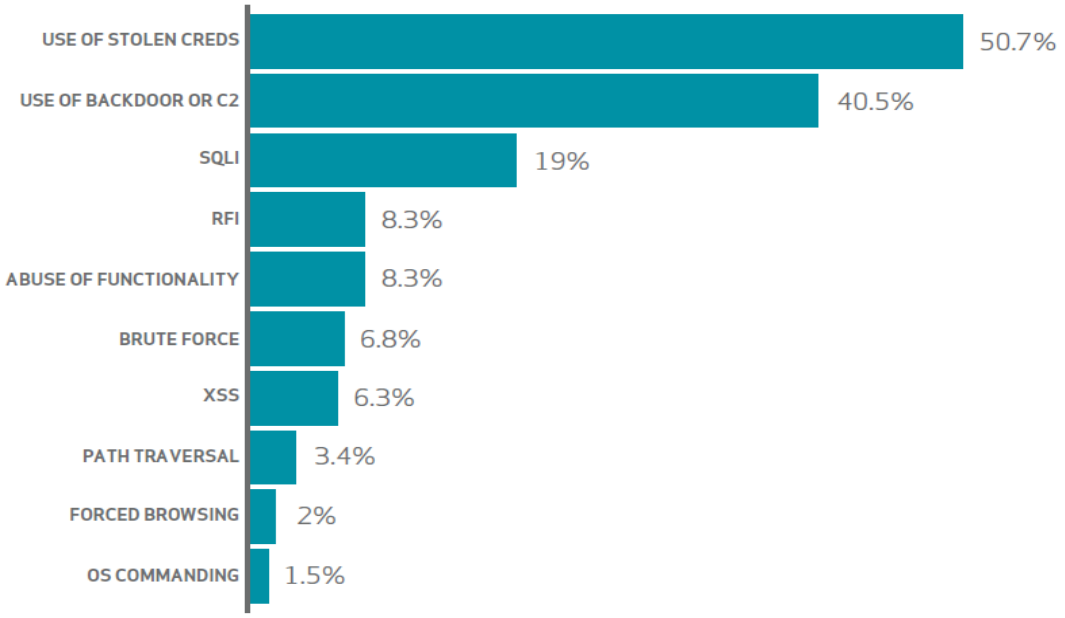


Figure 26: Variety of hacking actions within Web

According to WhiteHat Report 2015 many of the aforementioned attacks appear to the following figure 27 (WhiteHat, 2015).

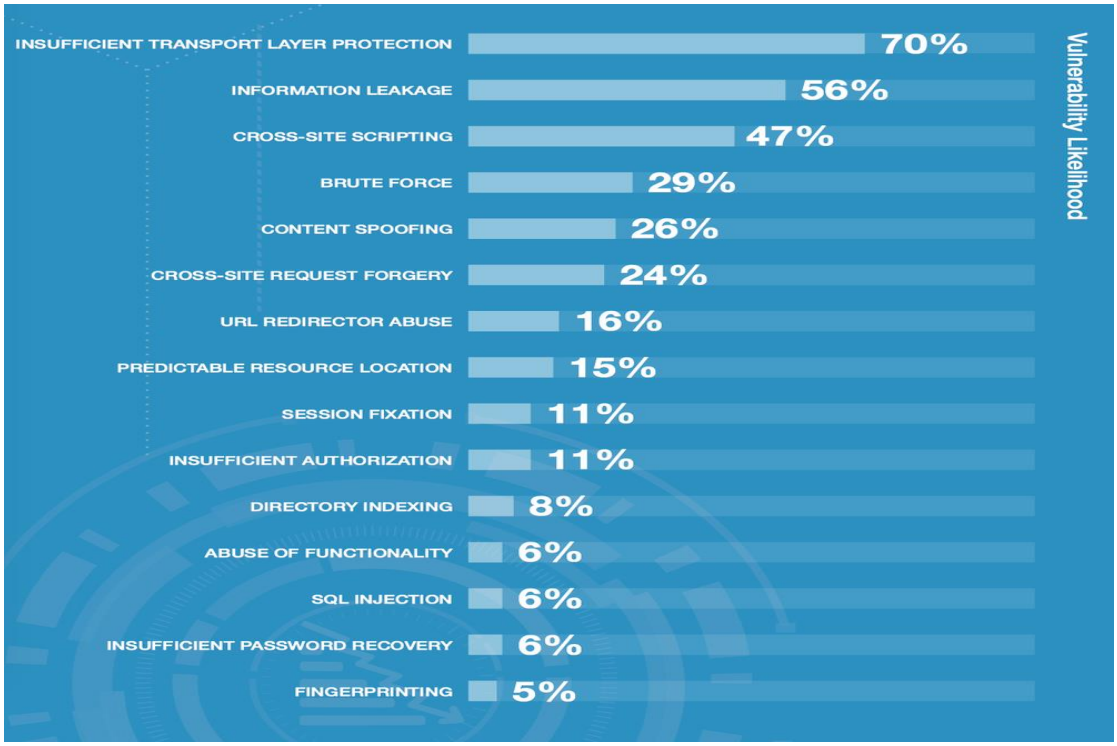


Figure 27: Application vulnerability likelihood – WhiteHat Report 2015

7.2 Types of attacks

7.2.1 *Remote File Inclusion*

Remote File Inclusion (RFI) is an attack technique used to exploit "dynamic file include" mechanisms in web applications. When web applications take user input (URL, parameter value, etc.) and pass them into file include commands, the web application might be tricked into including remote files with malicious code. PHP is particularly vulnerable to RFI attacks due to the extensive use of "file includes" in PHP programming and due to default server configurations that increase susceptibility to an RFI attack (Johnson, 2008). An attacker can use RFI for:

- Running malicious code on the server: any code in the included malicious files will be run by the server. If the file include is not executed using some wrapper, code in include files is executed in the context of the server user. This could lead to a complete system compromise.
- Running malicious code on clients: the attacker's malicious code can manipulate the content of the response sent to the client. The attacker can embed malicious code in the response that will be run by the client (for example, Javascript to steal the client session cookies).

Typically, RFI attacks are performed by setting the value of a request parameter to a URL that refers to a malicious file. Consider the following PHP code:

```
$incfile = $_REQUEST["file"];
include($incfile.".php");
```

The first line of code extracts the value of the file parameter from the HTTP request. The second line of code dynamically sets the file name to be included using the extracted value. If the web application does not properly sanitize the value of the file parameter (for example, by checking against a white list) this code can be exploited (Projects.webappsec.org, 2016).

7.2.2 *Clickjacking*

Clickjacking, also known as a "UI redress attack", is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page. Thus, the attacker is "hijacking" clicks meant for their page and routing them to another page, most likely owned by another application, domain, or both (Huang et al., 2012). The root cause of clickjacking is that an attacker application presents a sensitive UI element of a target application out of context to a user and hence the user gets tricked to act out of context. For example, imagine an attacker who builds a web site that has a button on it that says "click here for a free iPod". However, on top of that web page, the attacker has loaded an iframe with your mail account, and lined up exactly the "delete all messages" button directly on top of the "free iPod" button. The victim tries to click on the "free iPod" button but instead actually clicked on the invisible "delete all messages" button. In essence, the attacker has "hijacked" the user's click, hence the name "Clickjacking" (Owasp.org, 2016).

7.2.3 *Predictable Resource Location*

Over time, many pages on a website become unlinked, orphaned and forgotten. These Web pages often contain payment logs, software backups, future press releases, debug messages, source code – nothing, or everything. Normally, the only mechanism protecting the sensitive information within is the predictability of the URL. Automated scanners have become adept at uncovering these files by generating thousands of guesses. Also, through a process called “Google Hacking,” attackers use search engines to discover sensitive information via forgotten links on a website (Aron et al., 2001).

7.2.4 *Server Misconfiguration*

Server Misconfiguration attacks exploit configuration weaknesses found in web servers and application servers. Many servers come with unnecessary default and sample files, including applications, configuration files, scripts and web pages. They may also have unnecessary services enabled, such as content management and remote administration functionality. Debugging functions may be enabled or administrative functions may be accessible to anonymous users. These features may provide a means for a hacker to bypass authentication methods and gain access to sensitive

information, perhaps with elevated privileges. There are a wide variety of server configuration problems that can plague the security of a site (Owasp.org, 2016). These include:

- Unpatched security flaws in the server software
- Server software flaws or misconfigurations that permit directory listing and directory traversal attacks.
- Default accounts with their default passwords
- Use of default certificates
- Improper authentication with external systems
- Misconfigured SSL certificates and encryption settings

The following default or incorrect configuration in the httpd.conf file on an Apache server does not restrict access to the server-status page:

```
<Location /server-status>
SetHandler server-status
</Location>
```

This configuration allows the server status page to be viewed. This page contains detailed information about the current use of the web server, including information about the current hosts and requests being processed. If exploited, an attacker could view the sensitive system information in the file (Projects.webappsec.org, 2016).

7.2.5 *Abuse of Functionality*

Abuse of Functionality is an attack technique that uses a web site's own features and functionality to attack itself or others. Abuse of Functionality can be described as the abuse of an application's intended functionality to perform an undesirable outcome. These attacks have varied results such as consuming resources, circumventing access controls, or leaking information. The potential and level of abuse will vary from web site to web site and application to application. Abuse of functionality attacks are often a combination of other attack types and/or utilize other attack vectors (Projects.webappsec.org, 2016). The most common attack scenario of abuse of functionality vulnerabilities is denial of service attacks. Once an attacker finds a function that can be abused, they will use it to attempt to slow down or shut down a network server or application (typically not the one that has the abuse of functionality

vulnerability). The result is that the target server or application becomes inaccessible to legitimate users for the duration of the attack. The PERL-based web application "FormMail" was normally used to transmit user-supplied form data to a preprogrammed e-mail address. The script offered an easy to use solution for web site's to gather feedback. For this reason, the FormMail script was one of the most popular CGI (Common Gateway Interface) programs on-line. Unfortunately, this same high degree of utility and ease of use was abused by remote attackers to send e-mail to any remote recipient. In short, this web application was transformed into a spam-relay engine with a single browser web request.

An attacker merely has to craft an URL that supplied the desired e- mail parameters and perform an HTTP GET to the CGI, such as:

```
http://example/cgi-bin/FormMail.pl?
recipient=email@victim.example&message=you%20got%20spam
```

An email would be dutifully generated, with the web server acting as the sender, allowing the attacker to be fully proxied by the web- application. Since no security mechanisms existed for this version of the script, the only viable defensive measure was to rewrite the script with a hard-coded e-mail address. Barring that, site operates were forced to remove or replace the web application entirely (Infosecpro.com, 2016).

7.2.6 *OS Commanding*

An OS command injection vulnerability occurs when a developer uses invalidated user controlled parameters to execute operating system commands. OS command injection vulnerabilities allow attackers to run arbitrary commands on the remote server. *For more details about this attack method see Chapter 3.*

7.2.7 *Brute Force*

Brute-force is an attack in which an attacker uses a set of predefined values to attack a target and analyze the response until he succeeds. Success depends on the set of predefined values. If it is larger, it will take more time, but there is better probability of success. The most common and easiest to understand example of the brute-force attack is the dictionary attack to crack the password. In this, attacker uses a password dictionary that contains millions of words that can be used as a password. Then the attacker tries these passwords one by one for authentication. If this dictionary contains

the correct password, attacker will succeed. *For more details about this attack method, see Chapter 4.*

7.2.8 *Application Misconfiguration*

Application Misconfiguration attacks exploit configuration weaknesses found in web applications. Many applications come with unnecessary and unsafe features, such as debug and QA features, enabled by default. These features may provide a means for a hacker to bypass authentication methods and gain access to sensitive information, perhaps with elevated privileges. Likewise, default installations may include well-known usernames and passwords, hard-coded backdoor accounts, special access mechanisms and incorrect permissions set for files accessible through web servers. Default samples may be accessible in production environments. Application-based configuration files that are not properly locked down may reveal clear text connection strings to the database and default settings in configuration files may not have been set with security in mind. All of these misconfigurations may lead to unauthorized access to sensitive information (Projects.webappsec.org, 2016).

7.2.9 *Content Spoofing*

Content Spoofing is an attack technique that allows an attacker to inject a malicious payload that is sometimes misrepresented as legitimate content of a web application. It is used in phishing scams as a method of forcing a legitimate website to deliver or redirect users to bogus content. For example, users often receive a suspicious link that instructs them to confirm their user name and password information. Typically, phishing websites are hosted on look-alike domain names mimicking the content of the real site. In the case of Content spoofing phishing scams, fake content is injected into the real website, making it very difficult, if not impossible, for users to detect the difference and therefore protect themselves. A common approach to dynamically build pages involves passing the body or portions thereof into the page via a query string value. This approach is common on error pages, or sites providing story or news entries. The content specified in this parameter is later reflected into the page to provide the content for the page. For example:

```
http://foo.example/news?id=123&title=Company+y+stock+goes+up+5+percent+on+news+of+sale
```

The "title" parameter in this example specifies the content that will appear in the HTML body for the news entries. If an attacker where to replace this content with something more sinister they might be able to falsify statements on the destination website. For example:

```
http://foo.example/news?id=123title=Company+y+filing+for+bankrupcy+due+to+
insider+corruption,+investors+urged+to+sell+by+finance+analyists...
```

Upon visiting this link the user would believe the content being displayed as legitimate. In this example the falsified content is directly reflected back on the same page, however it is possible this payload may persist and be displayed on a future page visited by that user (Hayati and Potdar, 2009).

7.3 Detection

Security Misconfiguration is a situation where the configuration of either the application itself or the services it depend upon reveal technical information to a page visitor in the event of an error. This information can come in form of either stack traces or error messages directly from a database management system. This is a weakness for the system and using penetration testing tools can detect the above vulnerabilities and find a solution. The tools that are analyzed below, can detect and report these kind of error messages in order to protect the system. They are very effective especially for abuse of functionality, predictable resource location, server and application misconfiguration.

7.3.1 *PeerPressure*

PeerPressure (Wang et al., 2004) is a novel troubleshooting system that uses statistics from a set of sample machines as the golden state to diagnose the root cause misconfigurations on a sick machine. In PeerPressure, there is a ranking metric based on Bayesian estimation of the probability of a suspect candidate being sick, given the value of that suspect candidate. In addition to achieving the goal of effective troubleshooting, PeerPressure also makes a significant step towards automation in misconfiguration troubleshooting by using a statistical golden state, rather than manually identifying a single healthy state.

7.3.2 *SCAAMP*

SCAAMP (Security Configuration Assistant for Apache, MySQL and PHP) (Eshete et al.,2011) is a tool for auditing, fixing and safety rating security misconfiguration vulnerabilities in web applications and web server environments. The authors conducted an in-depth assessment of recommended security configuration settings for the AMP environment, taking official documentations and security experts' opinions as a basis. It can be customized to other target environments (e.g., Python, Ruby on Rails) with little modifications. The system is instrumental in assisting web application developers and administrators by automatically providing security configuration vulnerability alerts at the early stage of development or right before deploying an application.

7.3.3 *PHPSecInfo*

PHPSecInfo (Phpsec.org, 2016) is a security configuration auditing tool which automatically displays the current values and recommended values of PHP configuration directives from secure configuration perspective. It also summarizes the correctly and incorrectly set percentage of directives against the recommended values along with online links to descriptions of the conditions under which directives are used from security standpoint. Among the limitations of PHPSecInfo are: it is limited to PHP only, it misses some security critical configuration directives in PHP; it does not offer features to automatically fix security configuration values on the spot and re-auditing after fixing; and it maintains recommendations about security configuration directives online.

7.3.4 *PHP Security Audit*

PHP Security Audit (PHP Security Audit., 2016) is another auditing tool for security configuration. This provides an audit script that checks core PHP configuration, available functions and available classes to determine potential vulnerabilities and offer configuration suggestions. One good feature of this tool is that it generates recommendation report indicating what the user has to do (e.g., list of functions to disable) to fix vulnerabilities discovered after the audit. Like PHPSecInfo, this script is also limited only to PHP and has no features to automatically make the suggested changes to configurations.

7.3.5 *Baaz*

Baaz (Das et al.,2010) is a distributed system that monitors updates to access control metadata, analyzes this information to alert administrators about potential security and accessibility issues and recommends suitable changes. It detects misconfigurations that manifest as small inconsistencies in user permissions that are different from what their peers are entitled to and prevents integrity and confidentiality vulnerabilities that could lead to insider attacks a system. Baaz continuously monitors access permissions and group memberships and through the use of two techniques – Group Mapping and Object Clustering – finds candidate misconfigurations in the access permissions. Baaz is very effective at finding real security and accessibility misconfigurations, which are useful to administrators.

7.3.6 *Using Model Driven Security (MDS)*

Model driven security (MDS) is the tool supported process of modelling security requirements at a high level of abstraction and using other information sources available about the system (produced by other stakeholders). Model-driven security is also well-suited for automated auditing, reporting, documenting, and analysis (e.g. for compliance and accreditation), because the relationships between models and technical security implementations are traceably defined through the model-transformations. Model-driven security specifically applies model-driven approaches to automatically generate technical security implementations from security requirements models. There are two approaches appropriate for security misconfiguration (Hochreiner et al.,2014) :

- **Aspect oriented modeling**: The framework is designed to model the potential threats to a system in an aspect-oriented manner. These additions are designed to model an attacker-and-victim relation in the various types of UML diagrams. The premise of the class diagrams is an abstract attacker class that supplies simple attributes and methods. This framework is applicable in the setting of risk-oriented software development. They are only useful for situations involving particular attack scenarios and adding specific countermeasures to a given system. In aspect oriented modeling every possible attack needs to be modeled with respect to its effects on the system, which implies that all possible attacks need to be known beforehand.

- **KAOS**: The name of the methodology KAOS stands for Knowledge Acquisition in autOmated Specification respectively Keep All Objects Satisfied. The methodology describes a framework, to model and refine goals as well as the selection of alternatives. The KAOS model itself starts at a high level, that describes abstract requirements for the system. These abstract requirements are separated in functional and non-functional requirements, while the security requirements lie in the non-functional section. These goal models can be further used to generate object models, operation models or responsibility models to derive concrete software development requirements and restrictions. This model already includes actors and specific requirements related to the rather high-level requirements such as restricted access or authenticity.

7.3.7 *Using CCE configuration scanner*

CCE (Common Configuration Enumeration) (Lin et al., 2008) configuration scanner is a tool for scanning the system and determine the presence of the misconfiguration in it. According to authors the experiments show that this technique can help administrators to understand their own systems and enhance system security. CCE is standardized identifiers for security configuration issues and exposures and maintained by the MITRE Corporation. CCE gives each particular security-related configuration issue a unique identifier. After choose the policy, the configuration scanner will scan the PCs and the servers with CCE definitions. Administrators can determine the presence of misconfigurations and correct it.

7.3.8 *Paros Proxy*

Paros (Paros, 2013) is another Java application that acts as a proxy for ones browser. To perform the scan with this tool, the browser needs to be configured to utilize the proxy. Either one has to browse through the web application or start the scanner within Paros to build the list of URLs it will later scan for vulnerabilities.

7.3.9 *Skipfish*

Skipfish (Code.google.com, 2016) is a no-nonsense tool written in C. It is fully automated in the sense that it crawls the website itself and there is no need for or opportunity for human interaction with the scan flow. Only source code is available at the project website, but the compilation is quick, well documented and only depends on a small set of widely available libraries. The tool itself is started from the command line with a set of switches to set values affecting the scan properties.

7.3.10 *W3af*

W3af (Web Application audit and attack framework) (W3af.sourceforge.net, 2016) is a framework for auditing and exploitation of web applications. It is a cross-platform tool written in the Python. However, it is not that easy to use. It has a lot of profiles that can be chosen to perform the tests, but it turns out that some of them generate a lot of errors.

7.3.11 *ZAProxy*

The OWASP Zed Attack Proxy (ZAP) (Code.google.com, 2016) is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing as well as being a useful addition to an experienced pen tester's toolbox. It is also a Java application. Some improvements were done to Paros, most of them related with menus others related with the application performance, such as the passive scanner that now can look for new vulnerabilities, where between them we can find weak authentication.

7.4 Prevention

The points below outline some of the areas that should be considered when a user analyzes his web application for potential security misconfiguration. It is a summary of basic measures that anyone running a Web server-application should consider essential. They are very efficient especially for abuse of functionality, predictable resource location, server and application misconfiguration (Owasp.org, 2016) (PCMAG, 2016) (Kemptechnologies.com, 2015):

1. Don't run unnecessary servers or interpreters. If you don't need the FTP (File Transfer Protocol) server that's bundled with your Web server, don't give hackers another target: Disable it, or don't install it at all. Similarly, disable scripting languages and sample scripts that you don't absolutely require.
2. Monitor your logs. Your Web server keeps track of every request; review your logs regularly for signs of out-of-the-ordinary behavior.
3. Be careful with your server configuration**.** Limit executable files to specific directories and make sure their source codes can't be downloaded. Turn off features such as automatic directory indexing if you don't need them. Run any security tools your OS or Web-server vendor provides to identify potential weak spots.
4. Install latest updates and security patches. Have an easy to manage updating process with test environments to check updates before deploying to production environments. Security vulnerabilities and bugs are constantly being discovered and fixed.
5. Remove unused features, plugins and web pages. Only include the parts of web applications that you need to provide your service to end users. Remove any plugins or functionality that you are not using. This reduces the target footprint for vulnerabilities.
6. Change usernames, passwords and ports for default accounts. Web application frameworks and libraries often ship with default administration names, passwords and access ports enabled. Everyone knows these. Change all these to non-standard usernames, passwords and ports.
7. Don’t rely on one layer in a web application providing security for layers lower down in the stack. Secure all layers individually. Security should be applied at all levels within an application. Don’t just assume that an authenticated user has the rights to access any data available to the web application. Always check access permissions before any data access.
8. Disable directory browsing. Some versions of popular web server software have Enable Directory Browsing on by default. This allows URLs to be manually changed and the directory structure of the web server to be viewed. This should not be allowed.
9. Make sure folder permissions are set correctly on the server that host the components of the application. Related to the point above. Make sure that file

and directory permissions are set so that users can't access the files if they do gain access to the file system

10. Principle of least privilege should be used whenever possible. The starting position for security should be that everything is off by default. Only turn on the features that are needed for the application to work. And only give access to those accounts that need it.
11. Use SSL, SSH and other forms of encryption (such as encrypted database connections) to prevent data from being intercepted or interfered with over the wire.
12. Encrypted data should not have the key on the database server. Passwords should only be stored in a non-reversible format, such as SHA-256 or similar.
13. The application should connect to the database with different credentials for every trust distinction (e.g., user, read-only user, guest, administrators) and permissions applied to those tables and databases to prevent unauthorized access and modification.
14. The database should be on another host, which should be locked down with all current patches deployed and latest database software in use.

7.4.1 *Preventing RFI*

Preventing remote file inclusion (Owasp.org, 2016) include flaws takes some careful planning at the architectural and design phases, through to thorough testing. In general, a well-written application will not use user-supplied input in any filename for any server-based resource (such as images, XML and XSL transform documents, or script inclusions) and will have firewall rules in place preventing new outbound connections to the Internet or internally back to any other server. However, many legacy applications will continue to have a need to accept user supplied input. Among the most important considerations are:

- Use an indirect object reference map (*see Chapter 6 for more details*)
- Add firewall rules to prevent web servers making new connections to external web sites and internal systems. For high value systems, isolate the web server in its own VLAN or private subnet.
- Strongly validate user input using "accept known good" as a strategy.

7.4.2 *Anti-clickjacking defenses*

- User Confirmation: One straightforward mitigation for preventing out-of-context clicks is to present a confirmation prompt to users when the target element has been clicked. Facebook currently deploys this approach for the Like button, asking for confirmation whenever requests come from blacklisted domains.
- UI Randomization: Another technique to protect the target element is to randomize its UI layout. For example, PayPal could randomize the position of the Pay button on its express checkout dialog to make it harder for the attacker to cover it with a decoy button. This is not robust, since the attacker may ask the victim to keep clicking until successfully guessing the Pay button's location.
- Visibility Detection on Click: An alternative is to allow rendering transparent frames, but block mouse clicks if the browser detects that the clicked cross-origin frame is not fully visible. Adobe has added such protection to Flash Player's webcam access dialog in response to webcam clickjacking attacks; however, their defense only protects that dialog and is not available for other web content (Huang et al., 2012)

7.4.3 *Preventing Content Spoofing*

Best practices to prevent such attacks are within the programming/development phase itself. These practices include:

- Validation of user input for type, length, data-range, format, etc.
- Encoding any user input that will be output by the web application.
- Use of POST parameter if possible.
- Before deployment of any web application on the actual server, test it with web vulnerability scanners, *e.g. Acunetix, etc.*

Practices to Be Followed by Users:

- Don't click on links circulated via social networking sites (SNS), IM, e-mail.
- Always verify that the source is legitimate.
- Check the URL, even though the URL seems legitimate; it may have been sent by an attacker. It is advisable to first visit the domain name of the service and then navigate to the desired page (InfoSec Resources, 2013).

Chapter 8: Sensitive Data Exposure

8.1 Introduction

Sensitive data exposure was rated the number six threat in web applications on the OWASP top ten in 2013 (Owasp.org, 2015). In the context of data security, sensitive data is usually classed as information relating to healthcare records, financial information (credit card details, banking details), personal information (address, date of birth, national insurance number, social security number) and user account information for IT systems. Failure to ensure that sensitive data is protected can be catastrophic for both individuals and any organization that has failed to protect the information. Consequences such as identity theft, financial loss and privacy violations can affect people whose information is compromised. And organizations can experience reputational damage, financial penalties, contractual disadvantages and a loss of trust in their brand and messaging (Kemptechnologies.com, 2015). Databases are used to store and execute transactions. In order to execute transaction database reveals confidential data at the front end like credit card number, date of birth, social security number, etc. Many applications do not properly protect sensitive data. Attackers may steal or modify such sensitive data to execute the financial fraud, identity theft or other attacks. Sensitive data need to be protected while storing, retrieving or sharing with the front end. Many techniques are available like data encryption, user authentication, penetration testing etc. According to ITRC (Interstate Technology and Regulatory Council) (Itrcweb.org, 2016), there are statistics about sensitive data breach until 6/10/2015 which showing in the following figure 28 (Garg, 2015)

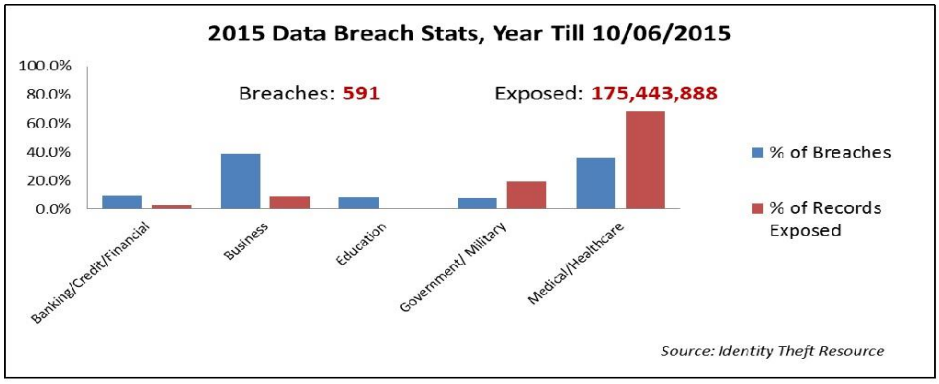


Figure 28: Sensitive data breach statistics, year till 6/10/2015

The aim of this chapter is to analyze the main attacks of this threat and at the same time to recommend detection and prevention approaches for avoiding to compromise sensitive data to attackers. According to Web Hacking Incident Database (WHID) (Google.com, 2016) the main attack methods for this security weakness are:

1. SQL Injection
2. Phishing attack
3. Man in the middle attack
4. Dictionary/Rainbow table attack

SQL Injection and Phishing attack – Man in the middle attack – Dictionary/Rainbow table attack, have especially been reported in previous chapters (*see Chapter 3 and 4 respectively).*

The following figure 29 (Apps.testinsane.com, 2016) shows a general view of this threat:

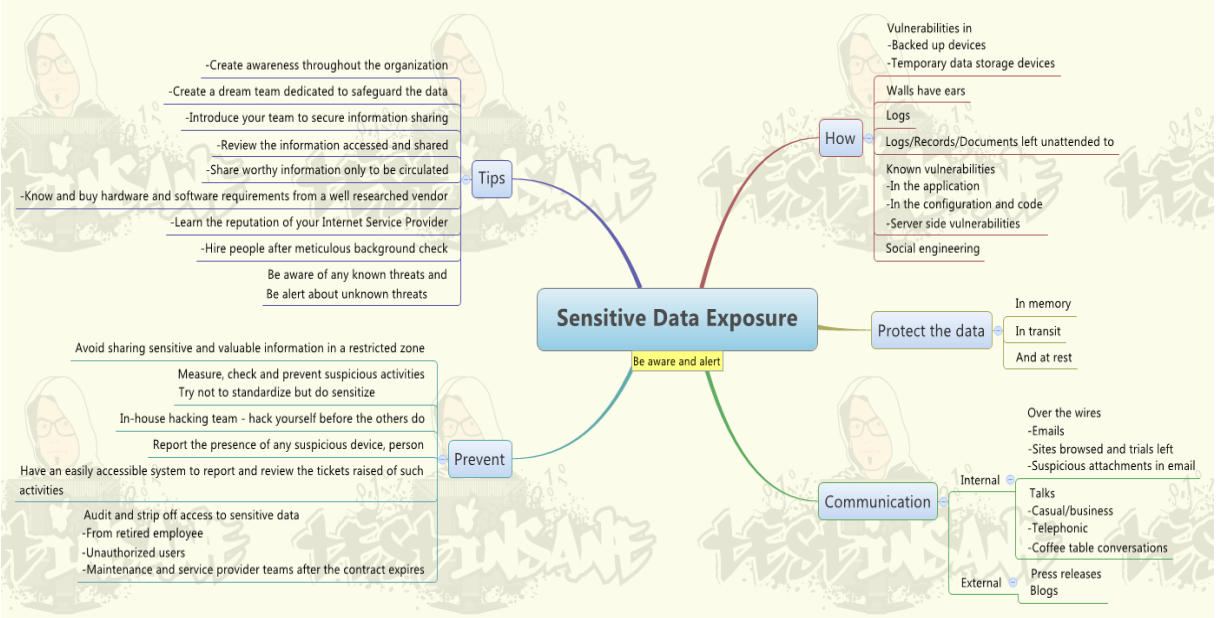


Figure 29: Sensitive Data Exposure

8.2 Types of Attacks

The two most common attack scenarios that take advantage of exposed sensitive information are direct attacks to gain unauthorized access to data and opportunistic use of error messages or status information to disclose paths to sensitive data. The first scenario is when an attacker actively attempts to compromise the information source. In either case, the attacker's goal from the outset is unauthorized access. An attacker could access a network through a weak Wi-Fi access point or take advantage

of a SQL injection or man in the middle attack to access the sensitive information. The second scenario is more opportunistic. Information that leaks through error messages and other status information can often be discovered by an attacker and used as a path for enabling further attacks. The information leaked in messages is usually not confidential, but including internal IP addresses or file system path data might be useful to an attacker to aid further attack.. For example, if an error message had the following text, "*File abc.txt not found in c:\apps\web\files*," then the attacker knows part of the server's directory structure and can use the folder name information to read other parts of the server's hard drive (Microsoft, 2010).

For more details about the four main attack methods that intruders use, see in Chapter 4.

8.3 Detection

The exposure of sensitive data in storage and transmission poses a serious threat to organizational and personal security. Data leak detection aims at scanning content (in storage or transmission) for exposed sensitive data. Below there are some approaches and detection tools from a survey in this research area.

8.3.1 *Using Taint Tracking*

8.3.1.1 TaintEraser

TaintEraser (Zhu et al., 2011) is a tool that tracks the movement of sensitive user data as it flows through off-the-shelf applications. TaintEraser uses application-level dynamic taint analysis to let users run applications in their own environment while preventing unwanted information exposure. It uses dynamic binary translation techniques to implement dynamic taint analysis on unmodified commercial applications running in normal user environments.

8.3.1.2 RIFLE

RIFLE (Bridges et al., 2004) is a runtime information flow mechanism that is as secure as existing static schemes. However, unlike static schemes, security decisions are in the hands of the user since RIFLE works with all programs (not just those written in special languages) and policy decisions are left to the user not to the programmer. The authors implemented RIFLE and demonstrate the performance cost for security is reasonable. Their implementation also demonstrates that RIFLE

successfully tracks information flow and can be effectively used by end-users to manage their confidential data.

8.3.1.3 PRECIP

PRECIP (Wang et al., 2008) is a confidentiality model, as a first step towards practical and retrofit confidential information protection. This model is designed to be used in practical systems, offering an efficient online protection against spyware surveillance without touching the source code of these systems. To this end, PRECIP addresses several important practical issues, including models for human input objects, shared objects and multitasked subjects. The authors applied the PRECIP model to Windows XP to protect the commercial applications for editing or viewing sensitive documents and browsing sensitive websites and evaluated its efficacy using our prototype. They also demonstrate that their implementation works effectively against a wide spectrum of spyware, including key loggers, screen grabbers and file stealers.

8.3.1.4 Asbestos

Asbestos (Efstathopoulos et al., 2005) provides protection through a new labeling scheme that, unlike schemes in previous operating systems, allows data to be declassified by individual users within categories they control. The categories, called tags, use the same namespace as communication endpoints, making them a kind of generalization of capabilities. As in a capability system, processes can dynamically generate new tags and distribute them independently. Processes can specify temporary label restrictions on sent messages to avoid the unintentional use of privilege. Asbestos use labels to indicate the taint level of OS abstractions and restrict information flow from more sensitive object to less sensitive object without the use of a trusted agent.

8.3.1.5 LIFT

LIFT (Kim et al., 2006) is a low-overhead, cheap (no hardware extension), comprehensive (works with libraries) and practical information flow tracking system for detecting a wide range of security attacks that corrupt control data (e.g. return address, function pointer, etc.). It minimizes runtime overhead by exploiting dynamic

binary instrumentation and optimization including the Fast-Path, Merged-Check and Fast-Switch optimizations.

8.3.1.6 DYTAN

DYTAN (Clause et al., 2007) is a prototype tool that is developed and is implemented in framework for x86 binaries. DYTAN leverages additional information possibly provided with the code, such as debugging symbols, but can perform dynamic taint analysis also on stripped binaries alone, which makes the tool widely applicable. Firstly it is used to perform a set of preliminary studies. The first set of studies shows how DYTAN allows for implementing different dynamic tainting approaches with limited effort. The remaining studies illustrate how the different aspects of the taint analysis can affect its results. The studies also show the practical applicability of DYTAN, by running it on the Firefox web browser.

8.3.1.7 Panorama

Panorama (Egele et al., 2007) is a whole-system fine-grained taint analysis to discern fine-grained information access and processing behavior of a piece of unknown code. This behavior captures the intrinsic characteristics of a wide-spectrum of malware, including keyloggers, password sniffers, packet sniffers, stealth back-doors, BHO-based spyware, and rootkits. Thus, the detection and analysis relying on it cannot be easily evaded. To evaluate the effectiveness of this approach, the authors have designed and developed this system. Using Google Desktop as a case study it is demonstrated that Panorama can accurately capture its information access and processing behavior and it was confirmed that it does send back sensitive information to remote servers.

8.3.1.8 HiStar

HiStar (Zeldovich et al., 2006) is an operating system designed to minimize the amount of code that must be trusted. HiStar provides strict information flow control, which allows users to specify precise data security policies without unduly limiting the structure of applications. HiStar's security features make it possible to implement a Unix-like environment with acceptable performance almost entirely in an untrusted user-level library. The system has no notion of super user and no fully trusted code

other than the kernel. This system has been built to integrate the notion of information flow and taint tracking directly into the operating system.

8.3.1.9 TaintTrace

TaintTrace (Cheng et al., 2006) is a high performance flow tracing tool that protects systems against security exploits. It is based on dynamic execution binary rewriting empowering the tool with fine-grained monitoring of system activities such as the tracking of the usage and propagation of data originated from the network. It uses a number of techniques such as direct memory mapping to optimize performance.

8.3.1.10 TaintCheck

TaintCheck (Newsome and Song, 2005) is a mechanism that can perform dynamic taint analysis by performing binary rewriting at run time. It can be used to provide additional information about the attack. It is currently able to identify the input that caused the exploit, show how the malicious input led to the exploit at a processor-instruction level and identify the value used to overwrite the protected data (*e.g.* the return address). TaintCheck is particularly useful in an automatic signature generation system, it can be used to enable semantic analysis based signature generation, enhance content pattern extraction based signature generation and verify the quality of generated signatures.

8.3.2 *Using data tainting*

This approach (Fetterman et al., 2006) demonstrates the use of a well-known technique, data tainting, to track data received from the network as it propagates through a system and to prevent its execution. Using this technique, the authors are able to explore hardware support for taint-based protection that is deployable in real-world situations, as emulation is only used when tainted data is being processed by the CPU. By modifying the CPU, memory and I/O devices to support taint tracking and protection, they guarantee that data received from the network may not be executed, even if it is written to and later read from disk.

8.3.3 *Scanning content with Map-Reduce*

This approach (Butt et al., 2015) presents a MapReduce system for detecting the occurrences of sensitive data patterns in massive-scale content in data storage or network transmission. The specific system provides privacy enhancement to minimize

the exposure of sensitive data during the outsourced detection. It uses the MapReduce framework for detecting exposed sensitive content, because it has the ability to arbitrarily scale and utilize public resources for the task. This transformation enables the privacy-preserving technique to minimize the exposure of sensitive data during the detection. It also supports the secure out-sourcing of the data leak detection to untrusted MapReduce and cloud providers.

8.3.4 *Using network-based and/or host-based approaches*

8.3.4.1 Using black-box differencing

This approach (Croft and Caesar, 2011) proposes a network-wide method of data confinement that detects information leaks by forking copies of processes consuming private data and removing the sensitive data from the input to the copy. The authors introduced the concept of a paired packet to allow both copies of the process to send data onto the network to allow the sharing of sensitive data within the confines of the network.

8.3.4.2 Disclosure

Disclosure (Bilge et al., 2012) is a large-scale, wide-area botnet detection system that incorporates a combination of novel techniques to overcome the challenges imposed by the use of NetFlow data. In particular, the authors identify several groups of features that allow Disclosure to reliably distinguish command and control (C&C) channels from benign traffic using NetFlow records: (i) flow sizes, (ii) client access patterns, and (iii) temporal behavior. They also demonstrate that these features are not only effective in detecting current C&C channels, but that these features are relatively robust against expected countermeasures future botnets might deploy against their system. This approach is capable of preventing personal or organizational sensitive information from being leaked.

8.3.4.3 Aquifer

Aquifer (Nadkarni and Enck, 2013) is a security framework that assigns host export restrictions on all data accessed as part of a UI workflow. Authors' key insight was that when applications in modern operating systems share data, it is part of a larger workflow to perform a user task. Each application on the UI workflow is a potential

data owner and therefore can contribute to the security restrictions. The restrictions are retained with data as it is written to storage and propagated to future UI workflows that read it. In doing so, the authors enable applications to sensibly retain control of their data after it has been shared as part of the user's tasks.

8.3.4.4 Beehive

Beehive (Juels et al., 2013) is a novel system that attacks the problem of automatically mining and extracting knowledge in a large enterprise from dirty logs generated by a variety of network devices. It improves on signature-based approaches to detecting security incidents. Instead, it flags suspected security incidents in hosts based on behavioral analysis. Beehive detected malware infections and policy violations that went otherwise unnoticed by existing, state of the art security tools and personnel.

8.3.4.5 Using measurement algorithms for the HTTP

This approach (Borders and Prakash, 2009) focuses on web traffic, but similar principles can apply to other protocols. Authors' analysis engine processes static fields in HTTP, HTML and Javascript to create a distribution of expected request content. It also executes dynamic scripts in an emulated browser environment to obtain complex request values. Instead of trying to detect the presence of sensitive data—an impossible task in the general case—approach's goal is to measure and constrain its maximum volume. Authors take advantage of the insight that most network traffic is repeated or determined by external information, such as protocol specifications or messages sent by a server. By filtering this data, they can isolate and quantify true information flowing from a computer.

8.4 Prevention

In specific reference to preventing Sensitive Data Exposure, some advice is given below (Kemptechnologies.com, 2015) (Eurovps.com, 2016):

1. Enforce Strict Data Encryption: The first step is to categorize and identify the sensitive data points. Once you have identified the critical data which requires an extra protective cover, the next step would be to implement a proven

encryption technique. Make sure that sensitive data is kept under the wrap of encryption all the time. Such data should neither be stored nor transmitted in clear text format.

2. Use SSL for User Authentication: Protect all authentication gateways on your website with secure HTTPS (SSL/TLS) protocol. SSL authentication uses the concept of public/private key pair. It means, a user must supply the corresponding decryption key for gaining access to sensitive data points.
3. Implement Strong Password Hashing Algorithm: Password hashing is one of those things that's pretty straightforward to implement, yet it is taken lightly by a large number of web developers. Hackers can exploit the weakness in password hashing algorithm to steal sensitive information stored on a web or application server. Only cryptographic hash functions should be used to implement password hashing
4. Have a well-defined password policy and then enforce it. Ensure that passwords are stored using an encryption method such as bcrypt that is specifically designed to protect passwords
5. Make sure user rights and roles are set correctly and that the Principle of least privilege is used. Give access to data if required. Don't try and give general access and then remove access rights from users who shouldn't it.
6. Implement an email content checker that stops sensitive information in emails and email attachments from leaving the organization. Scan for trigger words, Credit card type numbers, bank details, health related numbers etc. Quarantine suspect emails to ensure data isn't being leaked via email.
7. Turn off autocomplete. Having client applications and browsers autocomplete strings in fields runs the risk of presenting information to users that they shouldn't see. It is possible that autocomplete will fill in information that wasn't what the user was going to enter. This can also lead to the wrong information being displayed after queries
8. Don't cache information. Caching information and data in the client browser or on servers provides another location where the data could potentially be compromised.
9. Encrypt backups as well. Data needs to be backed up. Make sure that backups are just as well protected by encryption. Don't allow your backups to be a backdoor to sensitive information.

8.4.1 *Using Database Security Policy*

8.4.1.1 Sensitive DataFilter (SDF)

SDF (Doshi and Trivedi, 2014) provides a highly configurable environment for application development. The authors have proposed highly dynamic and flexible automated approach to prevent sensitive data exposure. This solution is deployed at database level on Oracle database (User can use it to create even for multiple applications. The model is able to restrict sensitive data exposure using dynamic security policy. Filter testing was executed from all three sources (application, Back end and using tools). Whenever any query is executed, the fine grain policy will check whether any security policy is set on objects used in the query.

8.4.1.2 OPES

OPES (Agrawal et. al., 2004), is an encryption scheme that allows queries with comparison operators to be directly applied to encrypted numeric columns. Query results neither contain any false positive nor miss any answer tuple. New values can be added without triggering changes in the encryption of other values. OPES is designed to operate in environments in which the intruder can get access to the encrypted database, but does not have prior information such as the distribution of values and cannot encrypt or decrypt arbitrary values of his choice. In such environments, OPES is robust against an adversary being able to obtain a tight estimate of an encrypted value.

8.4.1.3 Using Mixed Cryptography Database

Mixed Cryptography Database (MCDB) (Amagasa et al.,2009), is a novel framework to encrypt databases over untrusted networks in a mixed form using many keys owned by different parties. The encryption process is based on a new data classification according to the data owner. The proposed framework is very useful in strengthening the protection of sensitive data even if the database server is attacked at multiple points from the inside or outside. First, it introduces new data classification based on who owns the data. Second, it proposes a mixed cryptography database based on data classification methods. Third, it illustrates a query management system over a mixed encryption database. Finally, it analyzes the security of data storage and data transmission in the new cryptography framework.

8.4.1.4 UsingTransparent Data Encryption

Transparent Data Encryption (Deshmukh et al., 2013) plays an especially important role in safeguarding data in transit. Transparent Data Encryption can be used to provide high levels of security to columns, table and tablespace that is database files stored on hard drives or floppy disks or CD's and other information that requires protection. Microsoft SQL Server 2008 Transparent Data Encryption protects sensitive data on disk drives and backup media from unauthorized access, helping reduce the impact of lost or stolen media. Transparent Data Encryption provides a highly configurable environment for application development. User can use Transparent Data Encryption to create both single-machine and networked, environments in which developers can safely try out. The term encryption means the piece of information encoded in such a way that it can only be decoded read and understood by people for whom the information is intended.

8.4.1.5 Using database watermarking

In addition to encryption, watermarking technique is practically proven as another possible solution to enhance databases' content security especially for copyright protection and data tampering detection. In this approach (Pinn & Zung, 2013), authors proposed a watermarking algorithm based on hiding watermark bits in spaces of non-numeric, multi-word, attributes of subsets of tuples. A major advantage of using this approach is the large bit-capacity available to hide large watermarks. The proposed technique must be suitable for different areas like, e-banking, multimedia industries, film industries etc.

8.4.2 *Using steganography-OIBDH*

Steganography is all about concealing and hiding the existence of the information to be hidden and providing a secret communication between sender and receiver. For this reason is very useful for protecting sensitive data. Optimum Intensity Based Distributed Hiding (OIBDH) (Ajmal et al., 2011) is an algorithm proposed for secret data hiding inside a cover image. This algorithm is an improved version of Bit Plane Splicing LSB (*Least Significant Bit*) technique. Secret data are not hidden sequential wise and plane by plane as in Bit Plane Splicing LSB technique, rather data are hidden in a non-sequential manner and not plane by plane. The amount of bits to be hidden in a pixel depends on the gray scale value of the pixel. This makes the

proposed algorithm dynamic in nature and more effective both in terms of data hiding capacity and visual degradation of the cover image than Bit Plane Splicing LSB technique. Results in the form of absolute entropy difference curve and visual degradation of various cover images show the superiority of OIBDH over Bit Plane Splicing LSB technique.

8.4.3 *Preventing Sensitive Leaks in Error Messages*

An effective way to mitigate the risk of leaking sensitive data through error messages is to funnel all errors and warnings through a single checkpoint in the application. This single checkpoint determines who sees what data. For example, a remote, unauthenticated user should see very little warning information. In contrast, a local administrator should see all information. Regardless of the user type, full error details should be written to an administrator-readable log file. An administrator can read the full error information to help determine why an operation failed (Microsoft, 2010).

Chapter 9: Missing Function Level Access Control

9.1 Introduction

Missing Function Level Access Control was rated the number seven threat in web applications on the OWASP top ten in 2013 (Owasp.org, 2015). Web applications do not always protect application functions properly. Function level access control vulnerabilities arise due to incorrect configuration of request handlers within an application. Similar threats may also result from a web server's inability to verify function level access rights before making that functionality accessible to the user. Relying on security by hiding the restricted functions on the user interface is not going to be enough. It will allow an attacker to unlock the restricted functions by simply guessing the probable parameter value. So an administrator must enforce sufficient authorization to restrict general users from accessing privileged functions (Eurovps.com, 2016). Among the most common cases of missing control, there are the following ones (Nbs-system.co.uk, 2015):

- **Functions Exposure**

A user can simply have access to a function he should not have access to. For instance: someone wishing to book a flight ticket usually has to go through the following steps: ticket selection -> paying -> seat selection. If the function "seat selection" is poorly configured and its access poorly controlled, an intruder will potentially be able to bypass the "paying" step, choose his seat and then completely validate his ticket without having spent a single dollar. The access to this function should have been authorized only when the previous step, "paying", is completed; instead, its access was free.

- **Access to a Database**

The access to databases (name/surname/email/credit card number/etc…), such as the CRM or the ERP for instance, can happen in many ways, that are often linked to a missing access control.

- **Authentication Mechanism**

When the authentication mechanism relies on the client, or if the type of data sent to the server is not controlled, the risk is important. Here is the example of an "identifier/password" identification implying a Flash module that makes the dialog box more esthetic. If the code is not well thought, it is possible to find the following pattern:

- The user enters his identifier and password
- This information is sent by the browser to the Flash module, which will send a request to the server
- The server sends back the whole of the "identification/password" database to the Flash module
- The Flash module compares and controls the user's information
- Then validates or invalidates the connection

The Flash module is an unsecure element. When it receives the whole of the database, these elements are "clear" and thus visible by anyone without the need of major hacking competences. If this Flash/server interaction had been well thought, the Flash module would have simply sent the information to the server. The latter would have made the comparison itself and would have sent to the Flash module an agreement or a disagreement for the connection, instead of sending the whole database.

On someone user bank's website, the clerk has a link in his navigation bar to manage the client accounts. The client can't see that link because clients are not allowed to access this section. But the item in the menu exists in the code, it is simply hidden dynamically using a JavaScript function. By analyzing the HTML code, an attacker can easily find the URL to the account management section. And as the access to this section is not controlled, thus the attacker can steal a lot of sensitive data (personal information, account data…). Moreover, functions available in that account management section for clerks are not controlled either. The attacker can then transfer money to his own account. And hopefully, the attacker's actions are not logged (CERY, 2013).

In January 2009, an attacker was able to gain administrator access to a Twitter server because the server did not restrict the number of login attempts. The attacker

targeted a member of Twitter's support team and was able to successfully guess the member's password using a brute force with a large number of common words. Once the attacker gained access as the member of the support staff, he used the administrator panel to gain access to 33 accounts that belonged to celebrities and politicians. Ultimately, fake Twitter messages were sent that appeared to come from the compromised accounts (Cwe.mitre.org, 2016).

According to NVD (National Vulnerability Database), permissions, privileges and access control flaws have been appeared for various applications every year. The following tables show the related vulnerabilities found in different years. The improper access control presents an increase between 2014 to 2015 and the other flaws count topped at 2012 and declined in 2015 (Web.nvd.nist.gov, 2015):

Table 14: Permissions, Privileges, and Access Control Flaws Statistical Report

Year	Matches	Total	Percentage
2002	32	2,156	1.48%
2003	17	1,527	1.11%
2004	19	2,451	0.78%
2005	34	4,931	0.69%
2006	51	6,608	0.77%
2007	221	6,514	3.39%
2008	449	5,632	7.79%
2009	436	5,732	7.61%
2010	356	4,639	7.67%
2011	283	4,150	6.82%
2012	604	5,288	11.42%
2013	576	5,186	11.11%
2014	727	7,937	9.16%
2015	448	5,207	8.60%

Table 15: Improper Access Control Statistical Report

Year	Matches	Total	Percentage
2014	18	7,937	0.23%
2015	126	5,207	2.42%

One of the main reasons for the appearance of Missing Function Level Access Control, is Insufficient Authorization, Predictable Resource Location and URL Redirector Abuse weakness. The following figure (WhiteHat, 2015) shows a percentage of 11%, 15% and 16% respectively.

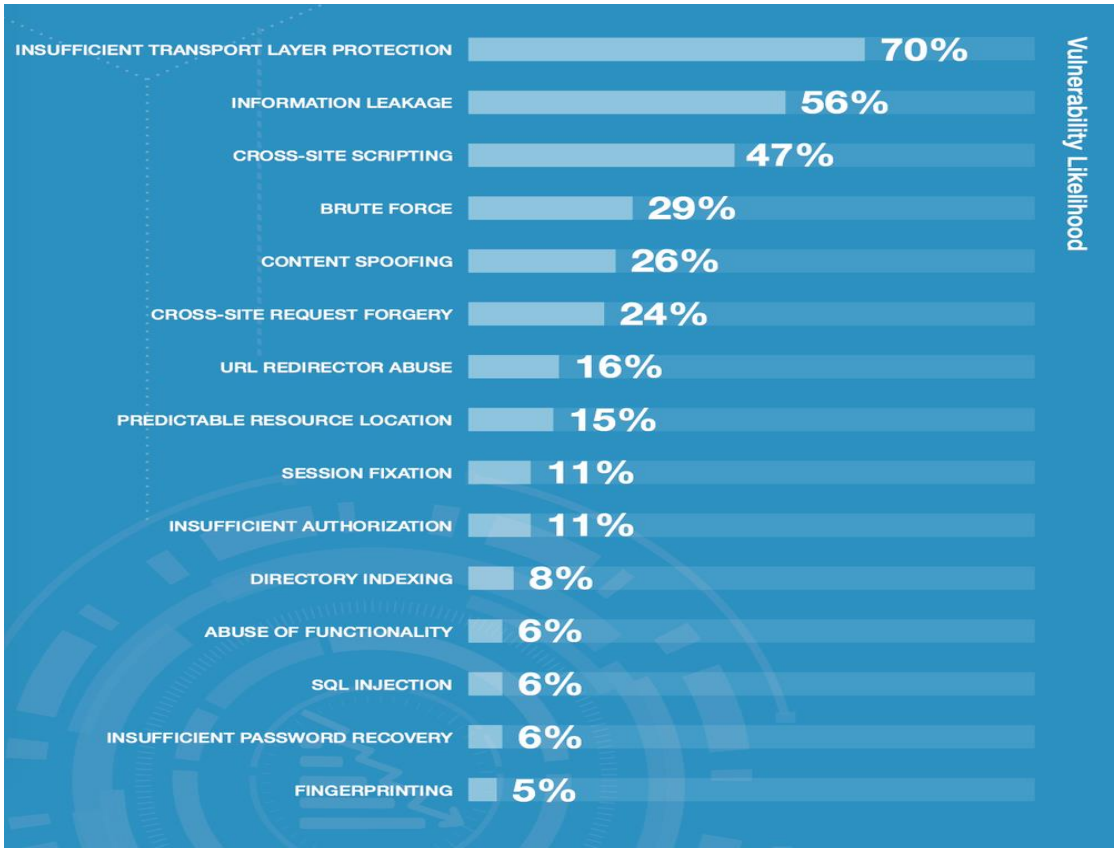


Figure 30: Application vulnerability likelihood – WhiteHat Report 2015

The scope of this chapter is to analyze the main attacks of this threat and at the same time to recommend detection and prevention approaches for avoiding to compromise sensitive data to attackers.

9.2 Types of attacks

According to Web Hacking Incident Database (WHID) (Google.com, 2016) the main attack methods for this security weakness are:

1. Forced Browsing
2. Open Redirect Attack
3. DNS Hijacking
4. Path traversal

The Path traversal and Open Redirect attacks have been analyzed in detail in Chapter 6 and 12 respectively, so there is only a reference to them in this chapter.

9.2.1 *Forced browsing*

Most Web applications use authentication to ensure only users with sufficient rights can access certain pages. Users must provide a username and password before being allowed access. Forced browsing (ComputerWeekly, 2016) is a simple browser attack that attempts to circumvent these controls by requesting authenticated areas of the application directly, without providing valid credentials, or by requesting pages beyond the access level of the logged-in user. If permissions on these pages have not been configured correctly, the pages will be displayed to the unauthorized user. An attacker can use Brute Force techniques to search for unlinked contents in the domain directory, such as temporary directories and files, and old backup and configuration files. These resources may store sensitive information about web applications and operational systems, such as source code, credentials, internal network addressing, and so on, thus being considered a valuable resource for intruders. This attack is also known as Predictable Resource Location, File Enumeration, Directory Enumeration, and Resource Enumeration. Forced browsing is most often a problem when a Web application has more than one user privilege level. For example, a Web application may have admin users, normal users and basic users. Each of these users log in through the same page, but the menus and options they have access to may vary. However, if a user can discover or guess the name of a valid page, he or she can manually request access to that page by typing the URL directly into the address bar. If authorization rights have been set up incorrectly, the user may be granted access. This can be a significant problem with admin pages especially, because the admin of a Web application usually has the ability to create or edit users. Most admin pages use common and easily guessable names, and if user access rights have not been configured correctly, a user may be able to gain access to areas above the user's privilege level.

According to the annual Verizon 2015 Data Breach Investigations Report(Verizon, 2015), 2% of the attacks make use of Forced Browsing. The following figure (Verizon, 2015) shows what rank the Forced Browsing has.

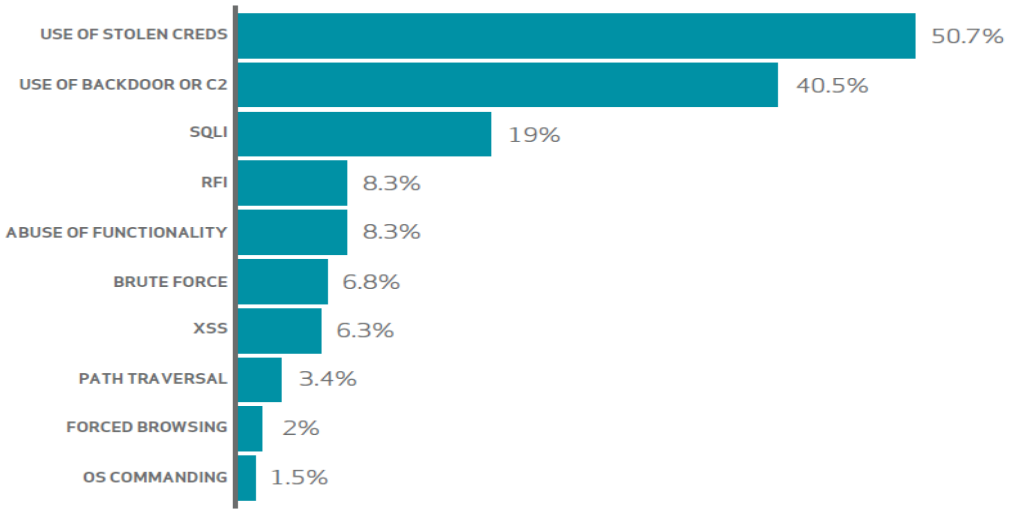


Figure 31: Variety of hacking actions within Web

This example (Owasp.org, 2016) presents a technique of Predictable Resource Location attack, which is based on a manual and oriented identification of resources by modifying URL parameters. The user1 wants to check his on-line agenda through the following URL:

```
www.site-example.com/users/calendar.php/user1/20070715
```

In the URL, it is possible to identify the username (user1) and the date (mm/dd/yyyy). If the user attempts to make a forced browsing attack, he could guess another user's agenda by predicting user identification and date, as follow:

```
www.site-example.com/users/calendar.php/user6/20070716
```

The attack can be considered successful upon accessing other user's agenda. A bad implementation of the authorization mechanism contributed to this attack's success.

9.2.2 *DNS Hijacking*

DNS hijacking (Ramesh, 2013) is a type of malicious attack that overrides a computer's TCP/IP settings to point it at a rogue DNS server, thereby invalidating the default DNS settings. In other words, when an attacker takes control of a computer to alter its DNS settings, so that it now points to a rogue DNS server, the process is referred to as DNS hijacking. As mentioned before, DNS is the one that is responsible for mapping the user friendly domain names to their corresponding IP addresses. This

DNS server is owned and maintained by user's Internet service provider (ISP) and many other private business organizations. By default, user's computer is configured to use the DNS server from the ISP. In some cases, his computer may even be using the DNS services of other reputed organizations such as Google. In this case, he is said to be safe and everything seems to work normally.

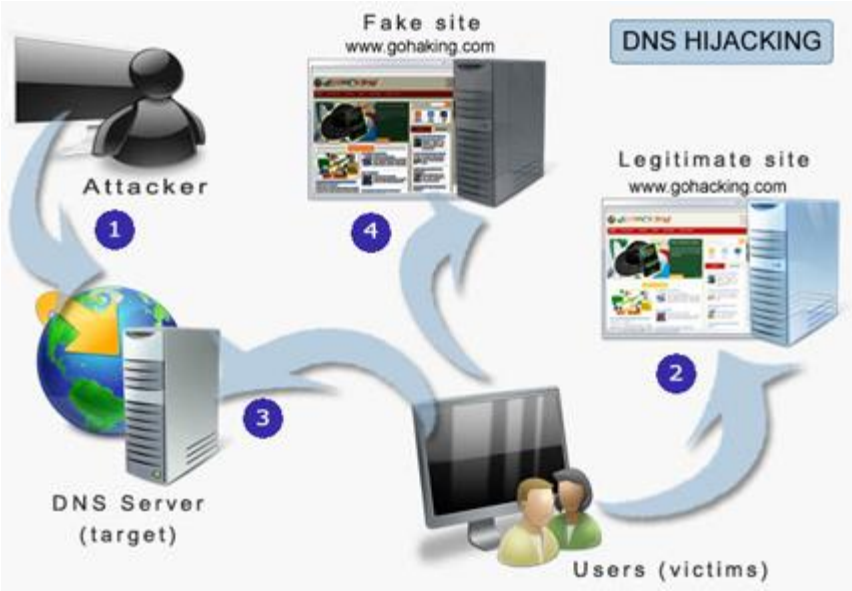


Figure 32: DNS Hijacking

But, imagine a situation where a hacker or a malware program gains unauthorized access to user's computer and changes the DNS settings, so that this computer now uses one of the rogue DNS servers that is owned and maintained by the hacker. When this happens, the rogue DNS server may translate domain names of desirable websites (such as banks, search engines, social networking sites etc.) to IP addresses of malicious websites. As a result, when a user types the URL of a website in the address bar, he may be taken to a fake website instead of the one he is intending for.

9.3 Detection

9.3.1 *Using scanning*

The missing function level access control requires a scanner to identify all features accessible to an authenticated/privileged user. If an admin portal is openly available, automated scanners can notify users of this problem. However, they cannot issue a notification if a logged in user can access records that should be restricted. Unauthenticated scanners cannot perform this task. And although authenticated scanners with detailed, human-designed, configuration can handle this task; every

test-case must be manually defined, including the expected results. Additionally, the missing function level access control is very dangerous to test by automation since the scanner's mapping functions will need to trigger them. This feature includes account creation, deletion, editing and much more. The blind call to these functions may trigger the effect once again and would keep triggering it on rescans if the system is set to work that way (Outpost24, 2014). The table below includes the OWASP Top 10. This table displays the vulnerabilities covered by automated scanning tools, penetration tests and the Secure Web Application Tactics (SWAT) by Outpost24 (Outpost24, 2016).

OWASP TOP 10 2013	Automated tools	Penetration Testing	SWAT
A1-Injection	✓	✓	✓
A2-Broken authentication and session management	Poorly supported	✓	✓
A3-Cross-site scripting (XSS)	✓	✓	✓
A4-Insecure direct object References	Poorly supported	✓	✓
A5-Security misconfiguration	✓	✓	✓
A6-Sensitive data exposure	Only default types	✓	✓
A7-Missing function level access control	Poorly supported	✓	✓
A8-Cross-site request forgery (CSRF)	Low accuracy	✓	✓
A9-Using components with known vulnerabilities	Low accuracy and coverage	✓	✓
A10-Invalidated redirects and forwards	✓	✓	✓

Figure 33: OWASP Top 10 2013 - Detection methods by Outpost

9.3.2 *Detecting DNS Hijacking*

For this attack the detection methods are as same as for path traversal attack and have been described in Chapter 6.

9.4 Prevention

9.4.1 *Preventing Forceful Browsing*

9.4.1.1 Using role-based authentication mechanisms

You need to be sure access to all pages and functions requiring authentication and specific authorization is controlled. To simplify and enforce security, your access

control mechanism should be centralized and role-based. It is also highly recommended to always apply a "deny-by-default" rule, i.e. explicitly define what is allowed and disallow everything else. You can use standard java filters to achieve this goal. Some third-party frameworks also provide API for centralized and role-based authentication and access control. You need to first identify the different roles available on your web site. E.g. you can have the simple USER role for authenticated users with no additional rights, the MANAGER role to manage users and the ADMIN role with administration privileges. Then, you have to properly configure your preferred session management framework. If the requested URL requires authentication, the framework should redirect unauthenticated users to the login page. If the requested URL requires a specific role, the framework should redirect to an "access denied" error page (CERY, 2013).

9.4.1.2 TRBAC

Role-based access control (RBAC) models are receiving increasing attention as a generalized approach to access control. Roles may be available to users at certain time periods, and unavailable at others. Moreover, there can be temporal dependencies among roles. To tackle such dynamic aspects, we introduce Temporal-RBAC (TRBAC) (Bertino et. al., 2001), an extension of the RBAC model. TRBAC supports periodic role enabling and disabling—possibly with individual exceptions for particular users— and temporal dependencies among such actions, expressed by means of role triggers. Role trigger actions may be either immediately executed, or deferred by an explicitly specified amount of time. Enabling and disabling actions may be given a priority, which is used to solve conflicting actions. At any given time, the system must be able to determine the roles that a user can activate, according to the periodic events and triggers contained in the Role Enabling Base (REB) and the run-time requests issued until that point. A request by a user to activate a role is authorized if the user has the authorization to play the role, the role is enabled at the time of the activation request, and no individual exceptions have been specified for the user for that particular role.

9.4.1.3 Akenti

Akenti (Thompson et. al., 2003) is an authorization service (PDP) that uses authenticated X.509 certificates to establish identity and distributed digitally signed

authorization policy certificates to make access decisions about distributed resources. It supports authorization decisions based on policy that it gathers from many sites. It returns an authorization decision as a signed capability certificate that can be used directly by a PEP to grant access or could be used by the subject of the certificate as a rights-granting authorization assertion. It supports Globus proxy identity certificates, and could easily be extended to handle restricted delegation credentials. We have implemented an Apache Web server module that allows the same authorization policy to be used to control access to Web-accessed resources as well as resources accessed by other remote methods. The code is freely available as C++ source code, or Linux and Solaris executables.

9.4.1.4 Hide unauthorized functions

If users don't know a function exists, they won't be tempted to use it. When possible, only show the functions the user can access with its privileges. This way, you won't expose sensitive URL to potential attackers. Of course, that means you should not have anything in the generated HTML/JavaScript code related to these hidden functions. Avoid HTML blocks hidden with CSS or JavaScript. With your preferred web framework, you will find different mechanisms to display or hide an HTML block depending on the user's privileges (CERY, 2013).

9.4.1.5 Log sensitive actions

Last but not least: log in a secure place all sensitive actions done on your website. If your website is attacked, even if the attack failed (because the mitigations I proposed are all applied on your website), it is still interesting to know how and when you were attacked and by who. Thus, you can't take specific actions against the source of the attack (blocking IP, deactivating account, revoking client certificate…) and investigate further. An even better option would be to be alerted when a suspicious action is requested. For example, if an authenticated simple user tries to access a management function, of course, the access will be rejected but it may be interesting to know that there was a potential attack attempt (CERY, 2013).

9.4.2 *Preventing DNS Hijacking*

In most cases, attackers make use of malware programs such as a trojan horse to carry out DNS hijacking. These DNS hijacking trojans are often distributed as video and audio codecs, video downloaders, YouTube downloaders or as other free utilities. So, in order to stay protected, it is recommended to stay away from untrusted websites that offer free downloads. The DNSChanger Trojan (Paul, 2016) is an example of one such malware that hijacked the DNS settings of over 4 million computers to drive a profit of about 14 million USD through fraudulent advertising revenue. Also, it is necessary to change the default password of your router, so that it would not be possible for the attacker to modify your router settings using the default password that came with the factory setting (Ramesh, 2013).

Chapter 10: Cross Site Request Forgery

10.1 Introduction

CSRF stands for cross-site request forgery. It's also known as session riding or XSRF. It was ranked number eight threat in web applications on the OWASP top ten in 2013 (Owasp.org, 2015). CSRF takes advantage of the inherent statelessness of the web to simulate user actions on one website (the target site) from another website (the attacking site). Typically, CSRF will be used to perform actions of the attacker's choosing using the victim's authenticated session. If a victim has logged into the target site, an attacker can force the victim's browser to perform actions on the target website (Blatz, J., 2007). The word CSRF itself means that the attack is done using a cross-site request; it's "forged" because it's invisible to the user. A cross-site request is one where a page loaded from one website makes a request to another site for resources that are part of the page (like images, for example). While it's easy for a malicious site to have such HTML code in its pages, such cross-site requests can also be caused by viewing bulletin boards, forums, or social networking sites (for example) where users are allowed to post images with foreign URL sources — that is, the images are hosted on other sites. CSRF attacks are effective in situations which meet the following criteria (Bajpai, 2010):

- The victim has an active session on the target site.
- The victim is authenticated by implicit authentication mechanisms (like cookies or HTTP authentication) on the target site.

The main aim of a CSRF attack is to perform an action against the target site using the victim's privileges in a properly authenticated session. The key requirement is prior access to and knowledge of the Web application's valid URLs, by the attacker. In a typical CSRF attack, the attacker creates a hidden HTTP request inside the victim's Web browser, which is executed in the victim's authentication context, without his knowledge. In the following table there is the statistic of cross-site request forgery attack from 2002 year to 2015 (Web.nvd.nist.gov, 2015).

Table 16: Cross-Site Request Forgery Statistical Report

Year	Matches	Total	Percentage
2002	1	2,156	0.05%
2003	0	1,527	0.00%
2004	0	2,451	0.00%
2005	1	4,931	0.02%
2006	3	6,608	0.05%
2007	36	6,514	0.55%
2008	78	5,632	1.38%
2009	113	5,732	1.97%
2010	75	4,639	1.62%
2011	55	4,150	1.33%
2012	153	5,288	2.89%
2013	120	5,186	2.31%
2014	245	7,937	3.09%
2015	228	5,955	3,83%

According to TrustWave Global Security Report 2015 (Trustwave, 2015) in the following graph there is the percentage of applications prone to CSRF from 2011 to 2014.

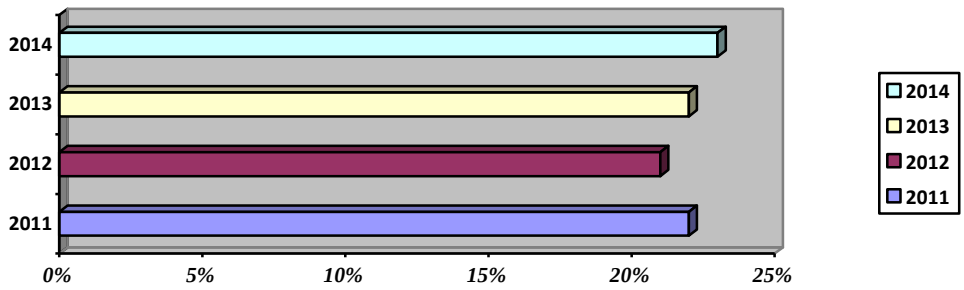


Figure 34: Percentage of applications found by Trustwave to be prone to CSRF

According to WhiteHat Report 2015, Cross-Site Request Forgery has a high percentage of vulnerability likelihood. The following figure (WhiteHat, 2015) shows a percentage of 24%.

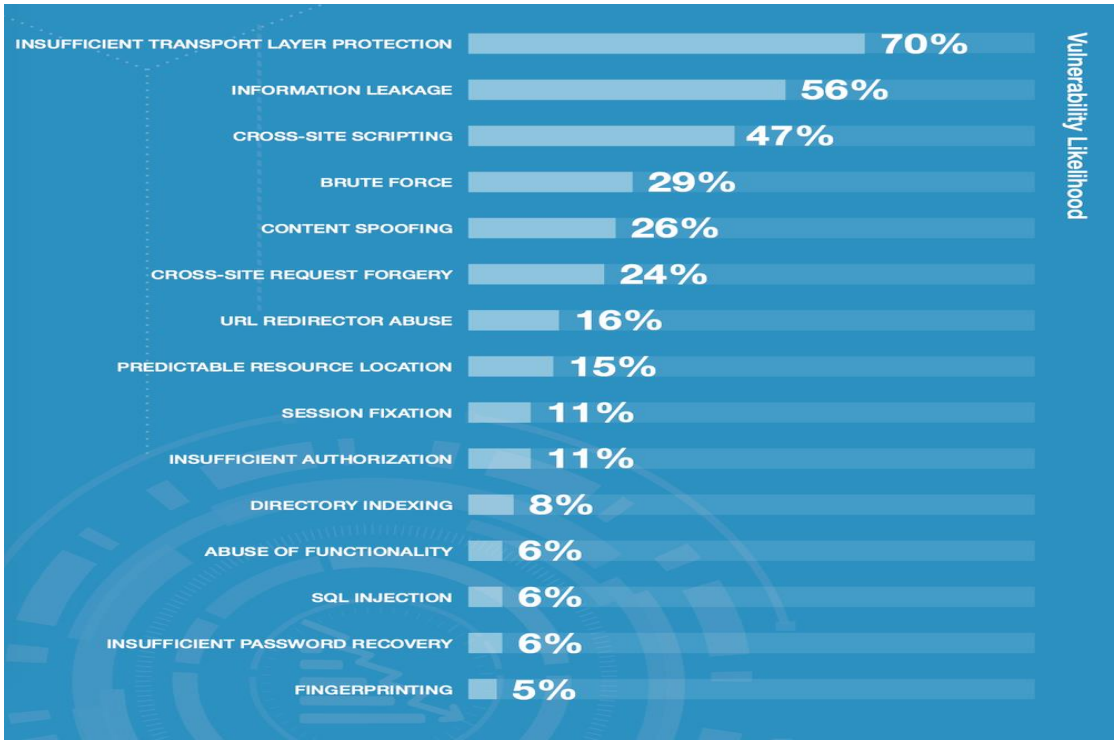


Figure 35: Application vulnerability likelihood – WhiteHat Report 2015

10.1.1 *CSRF vs XSS*

Unlike cross-site scripting (XSS), which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser. CSRF and XSS (especially reflected XSS) are related security risks and the similarity can be confusing. Many people find themselves trying to determine if an attack they have uncovered is exploiting an XSS weakness or an CSRF weakness. The distinction often easier to make by considering the solution to the weakness a user has identified. In the case of an XSS flaw, an attacker exploits a lack of input and / or output filtering. If a change to the application that filters out dangerous characters like <, >, “, ‘, &, ;, or # could resolve the flaw, then it is not an CSRF issue but an XSS issue. CSRF is about the predictability of the structure of the application. XSS is related to the application performing insufficient data validation. XSS flaws may allow bypassing of any CSRF protections by leaking valid values of the tokens, allowing Referrer headers to appear to be the application itself, or by hosting hostile HTML and JavaScript elements right in the target application. Therefore resolving XSS flaws should be given priority over CSRF weaknesses – although CSRF is usually a fatal flaw as well (Burns, J., 2005). CSRF and XSS attacks differ in that XSS attacks require JavaScript, while CSRF attacks do not. XSS attacks require that sites accept

malicious code, while with CSRF attacks malicious code is located on third-party sites. Filtering user input will prevent malicious code from running on a particular site, but it will not prevent malicious code from running on third-party sites. Since malicious code can run on third-party sites, protection from XSS attacks does not protect a site from CSRF attacks. If a site is vulnerable to XSS attacks, then it is vulnerable to CSRF attacks. If a site is completely protected from XSS attacks, it is most likely still vulnerable to CSRF attacks (Zeller, W. et al., 2008).

10.2 Types of attacks

Similarly to Cross-site scripting (XSS) vulnerabilities, CSRF vulnerabilities can be divided into two major categories: *stored* and *reflected* (Burns, J., 2005).

- A ***stored CSRF*** vulnerability is one where the attacker can use the application itself to provide the victim the exploit link or other content which directs the victim's browser back into the application and causes attacker controlled actions to be executed as the victim. Stored CSRF vulnerabilities are more likely to succeed, since the user who receives the exploit content is almost certainly currently authenticated to perform actions. Stored CSRF vulnerabilities also have a more obvious trail, which may lead back to the attacker.
- In a ***reflected CSRF*** vulnerability the attacker uses a system outside the application to expose the victim to the exploit link or content. This can be done using a blog, an email message, an instant message, a message board posting, or even a flyer posted in a public place with an URL that a victim types in. Reflected CSRF attacks will frequently fail, as users may not be currently logged into the target system when the exploits are tried. The trail from a reflected CSRF attack may be under the control of the attacker, however, and could be deleted once the exploit was completed.

Below there is an example for CSRF attacks. In this example, the victim site is *target.example.com*. It has a page on it, form.html, which sends its data to action.html. In legitimate use, the user will load *http://target.example.com/form.html*, fill out the data in the form, then hit the submit button. His or her browser will then send a request to *http://target.example.com/action.html*, which will perform some action

using the data. The goal of the attacker is to cause the victim's browser to submit data of the attacker's choosing to action.html (Blatz, J., 2007):

Creating a CSRF attack can be done in several ways. The simplest is to embed a reference to an external resource (for example, an image) in another page. This attack is very easy to launch, because almost all bulletin board sites allow users to embed images in their posts. Bear in mind that the browser has no way of knowing that the requested "image" is actually a web page that performs some action. As an example, the following post on a message board would cause the browser to send a request to the target web server:

```
I agree with the above poster
<img src="http://target.example.com/action.html?field1=foo&field2=bar"
width="1"
```

If the action requires HTTP POST, the attack is slightly more complex. The attacker can create a form using HTML or JavaScript. These require the attacker to have some degree of control over the CSRF site, as they will have to embed their own HTML in the site. They can gain this control either by being the site owner or by finding some sort of cross-site scripting vulnerability in the site. Unfortunately for us, finding websites with cross-site scripting vulnerabilities is frighteningly easy; the percentage of sites with cross-site scripting vulnerabilities is estimated to be very high—up to 70 to 80 percent. Here is an example (Komlosi, 2015):

1. The attacker uses one of the phishing methods, for example, he sends an email containing a malicious link to the victim.
2. The victim clicks on the malicious link and is redirected to the attacker's website. Now the attacker needs to trick the victim to click a malicious element placed on the page.
3. In the following example, when the victim clicks on the submit button "Donate to charity", a regular POST request will be submitted:

```
<form action="http://yourwebsite.com/transfermoney.aspx"
method="POST">
<input type="hidden" name="account" value="Attacker"/>
<input type="hidden" name="amount" value="9999"/>
<input type="submit" value="Donate to charity"/>
```

4. The POST request is sent to the server with *victim's credentials* (authentication cookie). The web server processes the request and sends $9999 to the attacker's account. The most poignant fact here is that *you* as the victim are not aware of any suspicious behavior.

The CSRF attack using the HTTP POST method is summarized in Figure 36 (Komlosi, 2015).

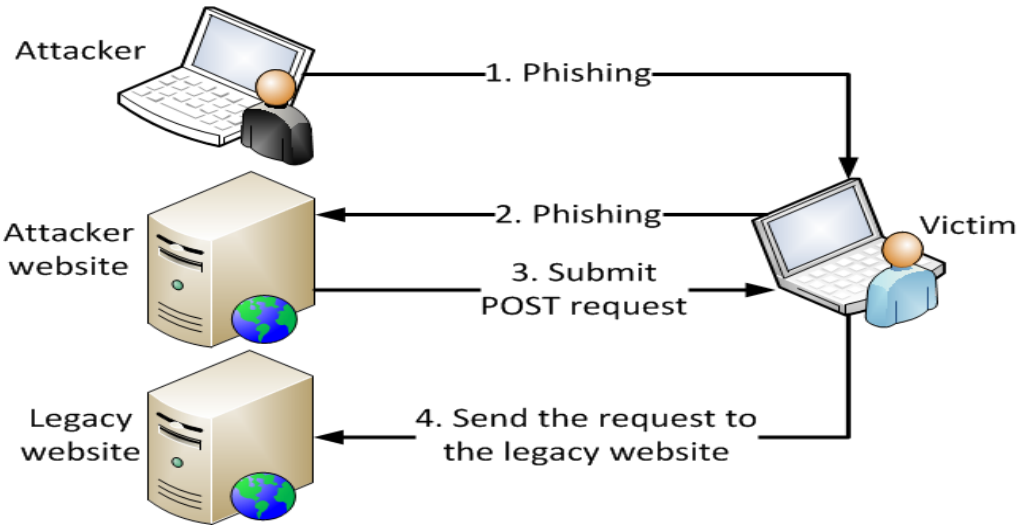


Figure 36: CSRF attack using HTTP POST

10.3 Detection

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. Specifically, manual analysis can be useful for finding this weakness, and for minimizing false positives assuming an understanding of business logic. However, it might not achieve desired code coverage within limited time constraints. For black-box analysis, if credentials are not known for privileged accounts, then the most security-critical portions of the application may not receive sufficient attention.

10.3.1 *Using CSRFTester*

CSRFTester (Owasp.org, 2015) gives verification testers the ability to quickly determine if a site is vulnerable to CSRF. It allows testers to record a web site transaction and then replay that transaction at a later time in order to prove whether an external attacker could cause the same transaction to successfully execute as part of a

CSRF attack. CSRF Tester facilitates detecting CSRF vulnerabilities and proving to interested stake holders that such attacks actually work and the damage they can cause.

10.3.2 *Using automated static analysis*

CSRF is currently difficult to detect reliably using automated techniques. This is because each application has its own implicit security policy that dictates which requests can be influenced by an outsider and automatically performed on behalf of a user, versus which requests require strong confidence that the user intends to make the request. For example, a keyword search of the public portion of a web site is typically expected to be encoded within a link that can be launched automatically when the user clicks on the link (Cwe.mitre.org, 2015).

10.4 Prevention

10.4.1 *Using CSRFGuard*

CSRFGuard (Chen, B., et al., 2011) implements a variant of the synchronizer token pattern to mitigate the risk of CSRF attacks. In order to implement this pattern, CSRFGuard must offer the capability to place the CSRF prevention token within the HTML produced by the protected web application. CSRFGuard provides developers more fine grain control over the injection of the token. Developers can inject the token in their HTML using either dynamic JavaScript DOM manipulation or a JSP tag library. CSRFGuard provides:

- A library that implements a variant of the synchronizer token pattern to mitigate the risk of Cross-Site Request Forgery (CSRF) attacks.
- A JavaEE Filter and exposes various automated and manual ways to integrate per-session or pseudo-per-request tokens into HTML.

10.4.2 *Using double-submit cookie-Server side*

The double-submit cookie method (Zeller, W. et al., 2008) offers a protection against CSRF that can be implemented with the need for server-side state. Thereby, the CSRF token is submitted twice. Once in the cookie and simultaneously within the actual request. As the cookie is protected by the Same-Origin Policy only same-domain scripts can access the cookie and thus write or receive the respective value.

Hence, if an identical value is stored within the cookie and the form, the server side can be sure that the request was conducted by a same-domain resource. As a result, cross-domain resources are not able to create a valid request and therefore CSRF attacks are rendered void.

10.4.3 *Using double-submit cookie-Client side*

In this approach (Lekies S., et al., 2012) authors presented a light-weight transparent CSRF-protection mechanism. This approach can be introduced without requiring any changes of the application code, as the server-side component is implemented as a reverse HTTP proxy and the client-side component consists of a single static JavaScript file, which is added to outgoing HTML content by the proxy. The proposed technique provides full protection for all incoming HTTP requests and is well suited to handle sophisticated client-side functionality, such as AJAX-calls or dynamic DOM manipulation.

10.4.4 *Using Allowed Referrer Lists (ARLs)*

In this approach (Czeskis A., et al., 2013), it is presented a browser/server solution, *Allowed Referrer Lists (ARLs)*, that addresses the root cause of CSRFs and removes ambient authority for participating web sites that want to be resilient to CSRF attacks. This solution is easy for web sites to adopt and does not affect any functionality on non-participating sites. Authors have implemented their design in Firefox and have evaluated it with real-world sites. They found that ARLs successfully block CSRF attacks, are simpler to implement than existing defenses and do not significantly impact browser performance. ARLs are a browser/server solution that removes ambient authority for participating web sites that want to be resilient to CSRF attacks. They are also a robust, efficient, and fundamentally correct way to mitigate CSRF attacks.

10.4.5 *Using Browser-Enforced Authenticity Protection (BEAP)*

Browser-Enforced Authenticity Protection (BEAP) (Mao Z. et al., 2009) is a browser-based mechanism to defend against CSRF attacks. BEAP infers whether a request reflects the user's intention and whether an authentication token is sensitive and strips sensitive authentication tokens from any request that may not reflect the user's intention. The inference is based on the information about the request (e.g., how

the request is triggered and crafted) and heuristics derived from analyzing real-world web applications. Authors have implemented BEAP as a Firefox browser extension and show that BEAP can effectively defend against the CSRF attacks and does not break the existing web applications.

10.4.6 *Using NoForge*

NoForge (Jovanovic, N., et al., 2006) is a server-side proxy that is able to defend PHP applications against XSRF attacks. This proxy is located between the web server and the protected web applications. To realize this in a straightforward fashion, authors decided to implement the proxy as wrapper functions around those PHP applications that they intend to protect. These wrapper functions check the input and output of the application and perform the necessary request and reply processing. Here are the steps that are necessary for protecting a web application with our prototype implementation:

1. Add an appropriate alias to the Apache configuration.
2. Execute the *sed* script on the target application to enable the proxy's wrapper functions.
3. Specify the cookie names that the target application uses to store session IDs (typically, this defaults to "PHPSESSID").
4. Specify the page that the user shall be redirected to in case of an XSRF attack.

10.4.7 *Using RequestRodeo*

RequestRodeo (Johns, M., & Winter, J., 2006) is a client-side proxy which protects the user from CSRF attacks. The proxy processes incoming responses and augments each URL with a unique token. These tokens are stored, along with the URL where they originated. Whenever the proxy receives an outgoing request, the token is stripped off and the origin is retrieved. If the origin does not match the destination of the request, the request is considered suspicious. Suspicious requests will be stripped of authentication credentials in the form of cookies or HTTP authorization headers. RequestRodeo also protects against IP address based attacks, by using an external proxy to check the global accessibility of web servers. If a server is not reachable from the outside world, it is considered to be an intranet server that requires additional protection. The user will have to explicitly confirm the validity of such internal cross-domain requests.

10.4.8 *Using RCSR*

Robust Client Side Request (RCSR) (AlAmeen, 2015) is a client side defensive tool. RCSR is a Firefox extension, which can prevent Reflected CSRF attacks effectively. RCSR is a tool gives computer users with full control on the attack. RCSR tool relies on specifying HTTP request source, whether it comes from different tab or from the same one of a valid user, it observes and intercepts every request that is passed through the user's browser and extracts session information, post the extracted information to the Server, then the server create a token for user's session. In a practical evaluation, the working of this extension was checked against reflected CSRF, the evaluation results show that it is working well. It successfully protects web applications against reflected CSRF.

10.4.9 *Using CsFire*

CsFire (De Ryck et al., 2010) is integrated extension into Mozilla browser to mitigate CSRF attacks. It is the only system that provides formal validation through bounded model checking to defend against CSRF in the formal model of the web. CsFire strips cookies and HTTP authorization headers from a cross-origin request. The advantage of stripping cookies and HTTP authorization headers is that there are no side-effects for cross-origin requests that do not require credentials in the first place. Additionally, CsFire supports users for creating custom policy rules, which use user-supplied whitelist and blacklist to certain traffic patterns. Furthermore, CSFire utilizes a sophisticated heuristic to identify legitimate cross-domain requests which are allowed to carry authentication credentials. The disadvantage of CsFire approach is that without the server supplied or user supplied whitelist, it will not be able to handle complex, genuine cross origin scenarios and the whitelists need to be updated frequently.

10.4.10 *Attack Prevention through Gateway*

This technique presents a gateway that prevents both type of cross site request forgery attacks (Kerschbaum, F., 2007). This can be done in front of every web application and only needs to process the request entirely protected the web site against the method of cross site attacks. This gateway uses three techniques, these are, web page classification, referrer string and cookies. Given a referrer string and cookie it is assumed that the input of a web page originated in the user's browser. By

allowing input only to a limited set of pages, which cannot be the first pages of a visit, cross site attacks are prevented. No response rewriting is done and therefore solution is more scalable than other proposed solutions. The main limitation of this approach is the response time for the attacker.

10.4.11 *Approach of Burns*

Burns (Burns, J., 2005) provide comprehensive introductions to CSRF attacks. To prevent the CSRF attack, they used following methods. Use cryptographic tokens to prove the Action Formulator knows a session specific secret, use secret tokens to prove the Action Formulator knew an Action and user specific secret, use the optional HTTP referrer header to verify Action Formulators, require changes to application state to be done only with HTTP POST operations and use a simplified CSRF Prevention Token. Drawback of their proposed work is that the attackers can adjust their attacks to be form based like CSRF, Submit forms automatically or though tricking users by making huge, mislabeled submit buttons. The header is optional and may not be present, some browsers disable this header and it is not available when interactions occur between HTTPS and HTTP served pages. The risk of header spoofing exists, and tracking the valid sources of invocations may be difficult in some applications.

10.4.12 *Using PCRF*

Prevent Cross-site Request Forgery (PCRF) (Son, S., 2008) is a dynamic token generating defense scheme against CSRF. It approaches this attack in a different way. PCRF uses tokens to ensure that request were issued from the web server previously instead of using session identifiers, PCRF uses clients IP address and local information. These are the ingredients of a cryptographically secure hash function which binds tokens to users. In short, the fact that PCRF tokens do not rely on sessions provides the safety against not only Login CSRF attacks but also normal CSRF attacks. Although PCRF is weak against an eavesdropper in the middle, like other existing token defense systems, PCRF suggests an efficient way to prevent CSRF attacks with a little change to the web server. PCRF filters dynamically add tokens to a HTML code which goes to clients and monitors the request without changing the legacy source codes. Moreover, PCRF doesn't need an additional database table to store the tokens.

Chapter 11: Using Components with Known Vulnerabilities

11.1 Introduction

This type of vulnerability occupies the penultimate spot in OWASP Top 10 list (Owasp.org, 2015). It deals with all the attacks on web applications which originate from using third party components having known weaknesses. This issue was mentioned as part of 2010 -A6 – Security Misconfiguration (Owasp.org, 2015), but now has a category of its own as the growth and depth of component based development has significantly increased the risk of using components with known vulnerabilities. Components (also referred to as binaries, artifacts, jars, or libraries) are software modules designed to perform commonly required functions including support for business logic, data access, resource management, communications and user interface creation. Today's applications commonly use 30 or more components, which in turn might rely on dozens or hundreds of other components. As components run with the full privilege of the application, vulnerability in any given component can completely undermine the security of an entire application. The impact of a component vulnerability often, but not always, depends on how the library is used by an application. Some libraries contain flaws that expose any application that leverages that library to compromise (Sonatype, Aspect Security, 2013). For example, of the 240,000 average component downloads in 2014, the same businesses sourced an average of 15,000 components that included known security vulnerabilities. In many cases, developers were downloading vulnerable component versions, when safer versions of those same components were available from the open source projects. While no one intends to download components with known vulnerabilities, the problem is aggravated due to the lack the visibility into a better recommended version (Blog.sonatype.com, 2015).

Orders	Quality Control		
Average downloads	# with known vulnerabilities	% with known vulnerabilities	% known vulnerabilities (2013 or older)
240,757	15,337	7.5%	66.3%

Figure 37: Average component downloads with known vulnerabilities-2014

Software vendors frequently release software patches in response to security issues in their software. It's imperative that you plan your application maintenance processes to adopt patches and vendor updates as quickly as possible, especially if a patch contains such a security related fixe. Staying on an old software release that has known and documented issues leaves you vulnerable. A good example for such a patching requirement is the recently discovered "heartbleed" OpenSSL bug (http://www.codenomicon.com/, 2015). The vulnerability associated with the bug has been documented in early 2014 along with the release of a patch. Companies that used OpenSSL and didn't install the patch put systems at risk because of the existence of this known vulnerability (Oracle, 2014).

According to a survey from Sonatype (Sonatype, 2013):

- 71% of all applications contain a critical flaw in at least one open source component.
- 80% of an application is assembled from open source components.
- 57% of organizations have policies governing component usage.
- Nearly 2/3 of organizations don't know which components are used in their applications.

11.2 Types of attacks

The specific vulnerability can become exploitable by many types of attacks as injections (*see Chapter 3*) or cross site scripting (*see Chapter 5*) that especially have been reported in previous chapters. Underneath it is analyzed a sort of attack which is not reported as now and is particularly dangerous for all web applications.

11.2.1 *Zero day attack*

A zero day exploit attack occurs on the same day a weakness is discovered in software. At that point, it's exploited before a fix becomes available from its creator. Initially when a user discovers that there is a security risk in a program, they can report it to the software company, which will then develop a security patch to fix the flaw. This same user may also take to the Internet and warn others about the flaw. Usually the program creators are quick to create a fix that improves program protection, however, sometimes hackers hear about the flaw first and are quick to exploit it. When this happens, there is little protection against an attack because the software flaw is so new. There is almost no defense against a zero-day attack: while the vulnerability remains unknown, the software affected cannot be patched and anti-virus products cannot detect the attack through signature-based scanning. For cyber criminals, unpatched vulnerabilities in popular software, such as Microsoft Office or Adobe Flash, represent a free pass to any target they might wish to attack. Zero-day vulnerabilities are believed to be used primarily for carrying out targeted attacks, based on the post-mortem analysis of the vulnerabilities that security analysts have connected to zero-day attacks (Bilge and Tudor, 2012).

Examples of noteworthy zero day attacks are:

- Stuxnet (Falliere et al., 2011) is a threat targeting a specific industrial control system likely in Iran in 2010, such as a gas pipeline or power plant. The ultimate goal of Stuxnet is to sabotage that facility by reprogramming programmable logic controllers (PLCs) to operate as the attackers intend them to, most likely out of their specified boundaries.
- "Aurora Exploit" (Symantec Security Response, 2015) is a widespread attack in 2010 that was carried out using a 0-Day exploit for Internet Explorer as one of the vectors. The code has recently gone public and it was also added to the Metasploit framework. This exploit was used to deliver a malicious payload, known by the name of Trojan.Hydraq the main purpose of which was to steal information from the compromised computer and report it back to the attackers. The exploit code makes use of known techniques to exploit a vulnerability that exists in the way Internet Explorer handles a deleted object. The final purpose of the exploit itself is to access an object that was previously

deleted, causing the code to reference a memory location over which the attacker has control and in which the attacker dropped his malicious code.

- In 2011, security firm RSA suffered a breach via a targeted attack. Analysis revealed that the compromise began with the opening of a spear–phishing email. The attacker in this case sent two different phishing emails over a two-day period. The two emails were sent to two small groups of employees; you wouldn't consider these users particularly high profile or high value targets. The email subject line read "2011 Recruitment Plan." The email was crafted well enough to trick one of the employees to retrieve it from their Junk mail folder, and open the attached excel file. It was a spreadsheet titled "2011 Recruitment plan.xls. The spreadsheet contained a zero-day exploit that installs a backdoor through an Adobe Flash vulnerability. As a side note, by now Adobe has released a patch for the zero-day, so it can no longer be used to inject malware onto patched machines (Micro, Trend, 2012).

11.3 Detection

11.3.1 *Using Dependency–Check*

Dependency-Check (Owasp.org, 2015) is a utility that identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities. Currently Java, .NET, Ruby, Node.js, and Python projects are supported; additionally, limited support for C/C++ projects is available for projects using CMake or autoconf. Dependency-check can currently be used to scan Java applications (and their dependent libraries) to identify any known vulnerable components. Dependency-check automatically updates itself using the NVD Data Feeds hosted by NIST.

11.3.2 *Using Shield*

Shield (Wang et al., 2004) is a prototype framework implementation that filters traffic above the transport layer. Generally, people have been reluctant to patch their systems immediately, because patches are perceived to be unreliable and disruptive to apply. To address this problem, the authors propose a first-line worm defense in the network stack, using *shields,* vulnerability-specific, exploit-generic network filters installed in end systems once a vulnerability is discovered, but before a patch is applied. These filters examine the incoming or outgoing traffic of vulnerable

applications and correct traffic that exploits vulnerabilities. The objective of Shield is to emulate the part of the application level protocol state machine that is relevant to its vulnerabilities with intercepted messages and counter any exploits at runtime. There are three main goals for the Shield design:

1. *Minimize and limit the amount of state maintained by Shield*
2. *Enough flexibility to support any application level protocol*
3. *Design Fidelity*

These network-based vulnerability-specific filters are feasible to implement, with low false positive rates, manageable scalability, and broad applicability across protocols. An examination of a sample set of known vulnerabilities suggests that Shield could be used to prevent exploitation of a substantial fraction of the most dangerous ones.

11.3.3 *Using Vulture*

Vulture (Neuhaus, Stephan, et al.,2007) is a tool that predicts vulnerable components by looking at their features. It is fast and reasonably accurate: it analyzes a project as complex as Mozilla in about half an hour and correctly identifies half of the vulnerable components. Two thirds of its predictions are correct. It uses a technique for mapping past vulnerabilities by mining and combining vulnerability databases with version archives. It also learns from the locations of past vulnerabilities to predict future ones with reasonable accuracy. This is an approach for identifying vulnerabilities that automatically adapts to specific projects and products. A predictor for vulnerabilities that only needs a set of suitable features and thus can be applied before the component is fully implemented.

11.3.4 *Using Vulnerability Alert Service (VAS)*

Vulnerability Alert Service (VAS) (Cadariu et al., 2015) is a tool-based process to track known vulnerabilities in software systems throughout their life cycle. So, when a project includes a library with known vulnerabilities, this input is destined for the vulnerability checker (in authors' study OWASP Dependency Check takes this role) which has two tasks: extract dependency data, recognize them and match them with known vulnerabilities. The software projects are input for the extracting task, which produces a list of recognized dependencies. This list and vulnerability disclosures are input for the matching task. Upon a successful match, the application generates an alert, which is consumed by a human operator. After acknowledging them, the

operator proceeds to filter the alerts based on usefulness and then reports them to the interested party.

11.3.5 *Using Web Application Firewalls (WAF)*

An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization (Cwe.mitre.org, 2015).

11.4 Prevention

11.4.1 *Using good component practices*

There are three gateways at which a vulnerable component may be included within an application: Selection/Consumption, Integration, Deployment. Each of these gateways must be policed and monitored to block, eliminate or manage vulnerable components.

1. Consumption: Selection of the components and where they came from

The first gateway of entrance to monitor for risk management is provenance - where does the component come from. There are three levels of risk to examine at first gateway:

- Licensing
- Security
- Quality

There must be a way to verify that the component downloaded is actually the component selected. Once the component has crossed through the first gate, it becomes part of the development environment with other risk types.

2. Integration: Component management within the development environment

Below there is an outline for helping a user to follow:

- Verify company compliance policies
- Automated inventory of existing components, libraries and frameworks

- Monitor usage of deployed components
- Update risky components
- Move beyond penetration testing to find vulnerable components

3. Deployment: Component maintenance within the production environment

The gateways where risk can get introduced are the main points for automated monitoring and enforcement of policies and procedures. If a user does all that great work up front, (this is validate, confirm integration with his current environment, etc,) then he needs to ensure that the component that was approved doesn't get invalidated by misconfiguration or manual change of version numbers to work around company policies. Validation must be done through ever step in the supply chain to confirm that the component in use is what it says it is and hasn't been changed during the process (Owasp.org, 2015).

- Implement Security Policy

Users must define security policies to govern the usage of third-party components and conduct vulnerability tests to assess their preparedness. Make sure that all the components pass through the defined security tests and guidelines before incorporating them in their project (Eurovps.com, 2015).

- Disable Unused Features and Functionalities

User may not be using all the available features of a third-party component. To restrict the threat surface area, always make it a custom to disable the functionalities that he does not require (Eurovps.com, 2015).

- Run security assessment tests

Even if there are no known CVE identified, the combination of different libraries and how someone uses these APIs could create a vulnerability. Before a user deploys a new major version of his application on production, he should do a full static code analysis and a manual code review. Running a penetration test is also recommended for very sensitive applications (CERY, 2014).

- Strategy: Environment Hardening

It is necessary for a developer to run his code using the lowest privileges that are required to accomplish the necessary tasks. If possible, he needs to create isolated

accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations (Cwe.mitre.org, 2015).

11.4.2 *Using threat patterns*

Threat patterns combine the concept of threat libraries and taxonomies. Threat patterns provide an abstract pattern based threat taxonomy. Each threat is encapsulated in a threat pattern. Additionally, threat patterns provide a common vocabulary for software designers and implementers. These patterns reflect all the steps leading to misuses so several misuse patterns may be derived from one threat pattern. They take advantage of specific vulnerabilities and can be described with respect to the corresponding vulnerability. Designers need to understand first possible threats before designing secure systems. However, identifying threats is not enough; users need to understand how a whole misuse is performed by taking advantage of them. Threat and Misuse patterns appear to be a good tool to understand how misuses are performed. It is possible to build a relatively complete catalog of threats and misuse patterns for application security. Having such a catalog users can analyze a specific application and evaluate its degree of resistance to these misuses. The architecture (existing or under construction) must have a way to prevent or at least mitigate all the threats that apply to it (Fernandez and Sulatycki, 2013).

11.4.3 *Using CLM*

Sonatype Component Lifecycle Management (CLM) (Sonatype, 2013) is the first solution to deliver component information, controls and remediation options directly into the tools that developers use every day. CLM is the practical way to automate governance of open-source component usage throughout the software lifecycle and eradicate flawed components from production applications. It identifies the components and their versions and monitors the security of these components in public databases, project mailing lists and security mailing lists and keep them up-to-date. It also establishes security policies governing component use, such as requiring certain software development practices, passing security tests and meeting license guidelines.

11.4.4 *Prevention against zero-day attack*

Zero-day attacks have been discussed for decades, but no study has yet measured the duration and prevalence of these attacks in the real world, before the disclosure of the corresponding vulnerabilities Zero-day attacks are difficult to prevent because they exploit unknown vulnerabilities, for which there are no patches and no anti-virus or intrusion-detection signatures. It seems that, as long as software will have bugs and the development of exploits for new vulnerabilities will be a profitable activity, users will be exposed to zero-day attacks. The research community has broadly classified the defense techniques against zero-day exploits as statistical-based, behavior-based signature-based and hybrid techniques (Kaur and Singh, 2014):

1. The statistical-based approach to detecting zero-day exploits in real time relies on attack profiles built off of historical data. This approach does not usually adapt well to changes in zero-day exploit data patterns. Any changes in a zero-day exploit's pattern would require a new profile to be learned by the system (Kaur and Singh, 2014).
2. Behavior-based model defense is based on the analysis of the exploit's interaction with the target. While often based on analysis data captured using high interaction honeypots, normal interactions can be learned, future activity predicted, and exploits classified into behavior groups. Interactions outside the normal behavior groups would be suspicious and quarantined. This method then has the potential to detect and analyze potential zero-day exploits in real time (Alosefer Y., Rana O. F., 2011).
3. The signature based detection techniques mainly focus on polymorphic worms. There are three types of signatures: content-based, semantic-based and vulnerability-based. The content-based signatures (Portokalidis and Bos, 2007) capture the features specific to a worm implementation, thus might not be generic enough and can be evaded by other exploits. Furthermore, various attacks can evade the content-based signatures by misleading signature generation processes by using crafted packet injection into normal traffic. Semantic-based signatures (Kirda et al., 2006) are computationally expensive to generate as compared to approaches based on substrings. Moreover, they cannot be implemented in existing IDS like Snort. Vulnerability-driven

signatures (Lanjia Wang et al., 2010) capture the characteristics of the vulnerability the worm exploits and are difficult to generate.

4. The hybrid detection model combines models previously mentioned using a heuristic approach. This method claims to be stronger against polymorphic, metamorphism and other obfuscations (Ting, C. et al., 2009). The hybrid method used will depend on what other methods of detection are combined in the environment.

Chapter 12: Unvalidated Redirects and Forwards

12.1 Introduction

The Open Web Application Security Project (OWASP) has released their top ten web security risks and number 10 is Unvalidated Redirects and Forwards (Owasp.org, 2015). This problem is caused by allowing untrusted data from a URL to decide the user's destination page. This security risk can be used in combination with social engineering for malicious purposes such as tricking the user into downloading malware, redirecting to a phishing site, or using forwards to gain access to unauthorized pages. The malicious URL is able to gain the user's trust because the original site's name appears in the URL.

Consider an example where a user visits a website and browses to the page:

www.victimsite.xyz/myhomepage.

Since, this is a user's home page and it cannot be viewed before signing in so the website redirects the user to the page with URL:

www.victimsite.xyz/signin?from=myhomepage

Once the user enters login credentials the website reads from the URL parameter and redirects to 'myhomepage' which is user's account welcome page. Until here everything seems fine. But suppose of a situation where an attacker sends a mail to this user (this technique in terms of security is referred to as Social Engineering) containing the hyperlink with the URL:

www.victimsite.xyz/signin?from=www.attackersite.abc

The user enters login details from where he/she is redirected to the attacker's website. And the trick is that even when the user is redirected to the attacker's page he/she has no idea about it because the malicious page looks exactly the same as his/her account's homepage because of the phishing technique used by the hacker. The flaw here in terms of logic implementation is that the actual web application did not validate the URL parameter that is used to redirect the user (Sullivan and Liu, 2012). User interaction from the victim is necessary to exploit this vulnerability, meaning that the victim is required to open the URL after receiving it via e.g. e-mail. One of

the dangers behind this vulnerability is the fact that the victim will see a URL pointing to a trusted domain name, thus making it easier for the attacker to entice the victim into opening the URL. Unvalidated redirects are commonly also used by attackers targeting authenticated users for XSS attacks, where a redirection can be used to trigger the vulnerability. Normally, attackers would be depending on finding a user already logged in to an application and hit them with the Cross-site Scripting, but this allows sending of emails to any probable user whom upon authentication will trigger the XSS (Outpost24, 2015).

Unvalidated forwards can be used by hackers to bypass access control. In case the destination URL does not correctly validate the user's authorization, attackers can use the forwarded URL to bypass user authentication checks and gain unauthorized access to privileged functions that they are not supposed to access (Eurovps.com, 2015). For example, the application uses forwards to route requests between different parts of the site. To facilitate this, some pages use a parameter to indicate where the user should be sent if a transaction is successful. In this case, the attacker crafts a URL that will pass the application's access control check and then forwards the attacker to administrative functionality for which the attacker isn't authorized (Owasp.org, 2015).

```
http://www.example.com/boring.jsp?fwd=admin.jsp
```

One of the main reason for the appearance of Unvalidated Redirects and Forwards, is URL Redirector Abuse weakness. The following figure (WhiteHat, 2015) shows a percentage 16% of this weakness in web applications.

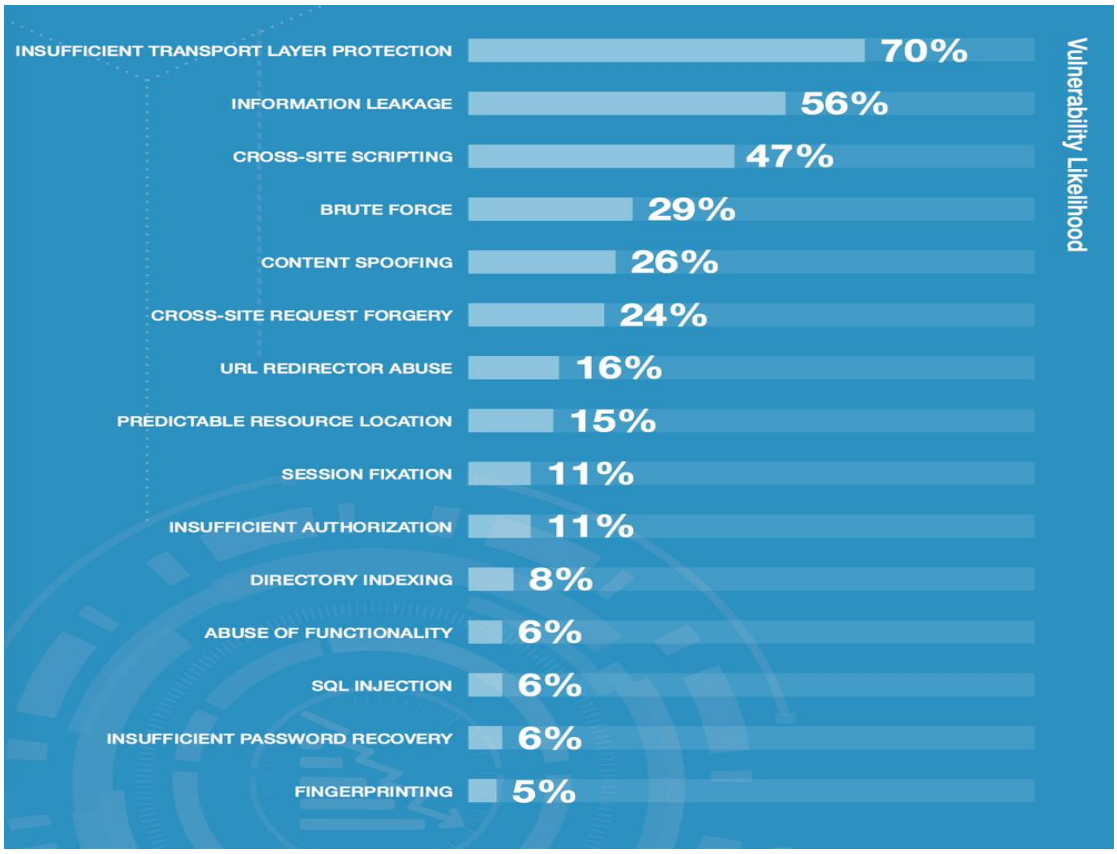


Figure 38: Application vulnerability likelihood – WhiteHat Report 2015

There have been numerous cases of phishing attacks involving unvalidated redirects and several top companies, including Google, AOL and eBay, have been the targets of hackers in the past. Consider what each vulnerability might mean for the business controlled by the affected application. It could be trivial or it could mean complete compromise.

12.2 Types of attacks

According to Web Hacking Incident Database (WHID) (Google.com, 2015) the specific vulnerability can become exploitable by many types of attacks as:

1. Redirection – Open Redirect attack
2. Phishing attack
3. Cross-site scripting (XSS)

Phishing attacks and cross site scripting especially have been reported in previous chapters (*see Chapter 4 and 5 respectively).* Below it is analyzed a sort of attack which is not reported as now and is particularly dangerous for all web applications.

12.2.1 *Open Redirect Attack*

As web-based applications have to direct users to a multitude of sub-application modules, or even other applications; so-called redirects and forwards are frequently used and virtually everyone has experienced a redirect at one time or another. For example, when your session with a web-based application expires owing to inactivity, you are redirected to login again before you can continue to work with the application. Typically, the redirection target is coded as a URL parameter, such as:

```
http://www.myserver.com/myapp?url=login
```

Attackers may manipulate this parameter. They may use social engineering techniques to lure victims into clicking on the link, as the server appears to be well-known and trusted, yet they remain fully unaware they will actually be redirected to a malicious site which, in turn, could infect them with malware, steal credentials etc.

```
http://www.myserver.com/myapp?url=http://malicious-site.com/virus
```

Attackers may obfuscate the redirection target URL by encoding it and/or embedding it within bogus parameters, as shown below:

```
http://www.myserver.com/myapp?appname=jfu77kjjfz?encryption=09883
82773665?url=%68%74%74%70%3A%2F%2F%77%77%77%2E%6D%61%6
C%69%63%69%6F%75%73%2D%73%69%74%65%2E%63%6F%6D%2F%7
6%69%72%75%73?auth=003988
```

The bogus data will be ignored by the application and the parameter "*url*" decodes to the actual malicious URL (Bowbridge, 2014).

Below there are some real-world examples of open redirect attacks:

Bitrix Site Manager 6.5 from(bitrixsoft.com) is an example of an application that includes open redirection by design. A quick Googledork will uncover a number of sites utilizing the script in an unmanaged fashion: *inurl:/bitrix/redirect.php*.

The flaw exists because the application does not validate the "goto" variable upon submission to the *redirect.php script*. Thus, an attacker could utilize the script to

cause redirection to a malicious site in a specially crafted URL sent to intended victims. Even SecurityLab (McRee, 2008) remains at risk at the time of writing, although they and the vendor have both been advised. The vendor was initially responsive, but indicated that no fix was imminent, again providing the classic "by design" response. When prompted to consider implementing a "whitelist" process rather than an open posture the vendor did indicate that they may do so in the future. Google recently fixed an open redirect vulnerability that allowed redirection from Google.com to any other site including those with malicious intent. Again redirect URLs are usually distributed via e-mail and often send people to sites with that are drive-by malware enabled with the intent of compromising the visitor's computer. Also Google worked to fix a redirect vulnerability related to the site of its DoubleClick online advertising unit (Open Redirect Vulnerabilities, 2008).

12.3 Detection

12.3.1 *Testing for unvalidated redirects*

Testing unvalidated redirects and forwards is to detect them in user's web application. There are two basic ways to accomplish this:

1. Automated detection with website spidering or crawling tools

- Netcraft Open Redirect Detection Service: Netcraft can perform an automatic search of a customer's web sites to scan for possible redirection URLs in use, on a daily basis, thereby promptly trapping redirects introduced by inadvertent web design and application development, and giving an excellent cost benefit (Netcraft.com, 2015).
- Using Redirect Remover: Redirect Remover (Redirectremover.mozdev.org, 2015), a Firefox browser extension, analyzes links on the page client is visiting and rewrites them to expose the actual destination. Unfortunately, this breaks some of the legitimate uses of redirects. Further, it does not protect against phishing, where the redirected links come from email messages.

2. Manual code scanning

   Manually reviewing source code generally requires a working knowledge of the language in which the application has been programmed. Most development environments include a search function, which reduces the scope of effort somewhat. Simply searching for "sendRedirect" can help you uncover instances in site source code where redirects are being performed (SearchSoftwareQuality, 2015).

12.4 Prevention

12.4.1 *Using ADF Security*

The most efficient way to protect applications from content-based attacks is through proper programming. All vulnerabilities described in this chapter are the result of sloppy programming, missing so-called "sanitation" of user input. Consequently, fixing all user input processing functions in your applications will solve this problem long-term. The defense in depth design pattern specifies that multiple layers of security to be implemented in an application. This also means that application functionality that executes methods and operations should be guarded by authorization checks even if the underlying data object is protected through entity security. Application Development Framework (ADF) Security provides the following types of options to protect function calls (and user interface components) from unauthorized use:

- **Custom resource permissions** – Function level security relates to high-level actions performed within an application such as placing an order that may go beyond simple entity updates. To encapsulate the protection of these functions, ADF Security allows developers to create custom permissions in addition to the existing permissions provided by framework. The method calls that need to be protected can check against these permissions before proceeding.
- **ADF Security EL Expressions** – ADF Security provides EL expressions that developers can use on the rendered property of user interface components, or on task flow router activities to hide operations and navigations for unauthorized users. It's important to remove the temptation from users to carry out actions which you will know that they will lack the privileges to execute

With these capabilities Oracle ADF provides ample tools to effectively manage your functional security.

Especially for mitigating the specific risk it is necessary to follow the guidelines below:

- Ensure any request that is issued or received through a direct GET request is validated for the identity of the request origin or target.
- Avoid any GET request or redirect for navigation in ADF applications. Use post-back navigation only.
- If data needs to be passed to a separate Java EE application, ensure it is passed such that it cannot be tempered with and so the receiving application knows how to verify the validness of the data. A possible solution for this could be a table in the database that is accessed through a stored procedure. The only parameter that needs to be passed between applications in such a case is a secured token (encrypted with a short lifetime) (Oracle, 2014).

12.4.2 *Using server-side modifications*

1. *White-Listing URLs*: While it may not always be possible to completely avoid user inputs for redirects and forwards, you must ensure that no unvalidated data is being used for this purpose. If user inputs cannot be avoided, then you can sanitize inputs by maintaining a list of trusted URLs. Create a white-list of trustworthy external websites to which redirects and forwards can be allowed. Compare user input values to the white-list entries and ensure that the destination parameter value supplied by the user is requesting redirect to a valid external website. Any attempted redirection to other external pages must be blocked by the server (Eurovps.com, 2015).
2. *Force User Notification on Redirects and Forwards*: Use an intermediate disclaimer page that provides the user with a clear warning that they are leaving the current site. Implement a long timeout before the redirect occurs, or force the user to click on the link. Be careful to avoid XSS problems when generating the disclaimer page (Cwe.mitre.org, 2015).
3. *Parameter Value Validation*: Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups,

query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Many open redirect problems occur because the programmer assumed that certain inputs could not be modified, such as cookies and hidden form fields (Cwe.mitre.org, 2015).

4. *Avoid using URLs as Redirection parameter*: Instead of using the actual URL as the destination parameter, it is highly recommended to employ a mapping value to identify the destination URL. Server side code can be used to translate the mapping value to the destination URL at the time of execution (Eurovps.com, 2015).
5. *Leveraging the referer header supplied by the client browser*: When following a link from one page, which it is called the *source page*, to another page, the *destination page*, the client supplies a referer header to destination page. This header indicates the URL of the source page. In this approach authors propose that the server checks the referer header when processing redirects. If it indicates that the client was at another page on the server, the redirect can be processed normally. Otherwise, it can be halted with an error message because it indicates that the client is coming from a third-party site, which could be potentially malicious (Shue, C. et al., 2008).

12.4.3 *Using heuristics to identify open redirects*

In this approach, the browser would examine the link for any URL patterns in the query string. If a destination is specified, the browser would replace the URL pattern with a test verification Web page. If upon following that link, the client is redirected to the verification page, it has confirmed that the redirect is open. At that point, the browser can either refuse to connect or warn the end-user of the open redirect. This approach has the advantage that the client does not actually follow the redirects, which ensures that the phishing sites cannot confirm that a user attempted to follow a link (Shue, C. et al., 2008).

Chapter 13: Contributions

This thesis makes the following contributions in two different areas that concern the security of web applications. The first area is that of detecting and preventing web threats, using appropriate tools and techniques. The second area is about presenting threats and trends with the most dangerous threats and vulnerabilities that we found the time period 2013-2015.

13.1 Contribution in detection and prevention

Detection methods distinguish attackers' attempts and detect them usually in run time. After detection of attacks, the detection tool alerts the users that something suspicious has happened, so that they can carry out certain actions.

On the other hand, prevention thwarts the possibility of attacks' success, using the best defensive methods to enforce the application, including encryption, firewalls, anti-virus, anti-spyware, passwords, back-ups and biometric security. As detailed below, we introduce three tables for three different kind of threats which exist in OWASP top ten 2013.

The Table 17 shows the techniques and their defense capabilities against Cross Site Request Forgery (CSRF) that have been analyzed in *Chapter 10*. It is a comparative analysis of the CSRF prevention tools using the criteria of prevention location and analysis method. By the meaning of location we consider the various locations an attack method can be detected/prevented. In our table there are four categories of location:

- Server side: Server-side application techniques detect/prevent attack methods by analyzing server side application written in programming and script languages.
- Client side - Browser: Client-side application techniques detect/prevent attack methods by analyzing HTML pages.
- Client side proxy: Client-side proxy is an intermediate server between a user and a web server. It intercepts user's requests and responses from web server in order to detect attacks. After detecting something malicious, either it rejects the request or alters malicious inputs to harmless inputs.
- Server side proxy: Server-side proxy acts as extra server between an application server and a database server. It stops outgoing requests from an

application before reaching to database server. It helps in blocking the malicious query execution in database.

The other criterion, analysis method, has several types. The most known are:

- Static Analysis: Static program analysis is the analysis of computer software that is performed without actually executing programs. In most cases the analysis is performed on some version of the source code and in the other cases, some form of the object code.
- Dynamic Analysis: Dynamic analysis techniques analyze the information obtained during program execution to detect SQL injection vulnerabilities.
- OS Level Analysis: This kind of analysis uses an OS-Level view to get meaningful analysis results. OS level semantics information including process, module, thread, symbolic information and function call context.
- Signature – based: Most intrusion detection systems (IDS) are known as signature-based. This means that they operate in much the same way as a virus scanner, by searching for a known identity - or signature - for each specific intrusion even.

Table 17: Prevention techniques characteristics - CSRF

Technique	Prevention location	Analysis Method
CSRFGuard	Between client and server	Dynamic
Double submit cookie	Server side	Dynamic
Double submit cookie	Client side	Dynamic
Allowed Referrer Lists	Browser/Server	Dynamic
BEAP	Browser	Dynamic
No Forge	Server side proxy	Dynamic
RequestRodeo	Client side proxy	Dynamic
Robust Client Side Request	Client side	Dynamic
CsFire	Browser	Dynamic

PCRF	Server side	Dynamic

Respectively, in Table 18, there are the detection tools for sensitive data exposure with some of their characteristics according to the analysis that has been implemented *in Chapter 8*. The new criterion in this table is the detection time. There are three main types for detection time:

- Run time
- Testing time
- Coding time

Table 18: Detection tools comparison – Sensitive Data Exposure

Detection Tools	Detection time	Analysis Method
RIFLE	Run time	Dynamic Software
Taint Eraser	Run time	Dynamic Taint
PRECIP	Run time	OS Level
Asbestos	Run time	OS Level
LIFT	Run time	Dynamic Taint
DYTAN	Run time	Dynamic Taint
Panorama	Run time	Dynamic Software
Histar	Run time	OS Level
TaintTrace	Run time	Dynamic Taint
TaintCheck	Run time	Dynamic Taint
Beehive	Run time	Signature-Based

Lastly, in the Table 19, there is a classification of detection tools about Security misconfiguration. The criterion we use for doing that is the programming language that has been used for each of these tools. There is a detailed analysis for each tool *in Chapter 7.*

Table 19: Detection tools – Security Misconfiguration

Detection Tools	Written Language
ZAProxy	Java
W3af	Python
Skipfish	C
Paros	Java
MDS	UML
Baaz	C#
PHP Security Audit	PHP
PHP SecInfo	PHP
SCAAMP	MySql-PHP

13.2 Contribution in presenting threats and trends the time period 2013-2015

According to National Vulnerability Database (NVD) (Web.nvd.nist.gov, 2015), multiple vulnerabilities have been discovered for various applications every year. The following tables show the related vulnerabilities found in the period 1/1/2013 – 31/12/2015.

Table 20: Cross Site Scripting (XSS)

Year	Matches	Total	Percentage
2013	616	5,186	11,88%
2014	1,028	7,937	12,95%
2015	724	6,488	11,16%

Table 21: SQL Injection

Year	Matches	Total	Percentage
2013	145	5,186	2,80%
2014	296	7,937	3,73%
2015	212	6,488	3,27%

Table 22: Cross-Site Request Forgery (CSRF)

Year	Matches	Total	Percentage
2013	120	5,186	2,31%
2014	245	7,937	3,09%
2015	235	6,488	3,62%

Table 23: Permissions, Privileges and Access Control

Year	Matches	Total	Percentage
2013	576	5,186	11,11%
2014	729	7,937	9,18%
2015	567	6,488	8,74%

Table 24: Path Traversal

Year	Matches	Total	Percentage
2013	103	5,186	1,99%
2014	197	7,937	2,48%
2015	141	6,488	2,17%

Table 25: Information Leak/Disclosure

Year	Matches	Total	Percentage
2013	252	5,186	4,86%
2014	345	7,937	4,35%
2015	603	6,488	9,29%

We also used the Web Hacking Incident Database (WHID) (Google.com, 2016), which is a unique database in tracking only media reported security incidents that can be associated with web application security vulnerabilities. We try to limit the statistics from the database to targeted attacks only. According to WHID the following pie charts provide information for statistical analysis about attack methods for the years 2013, 2014, 2015.

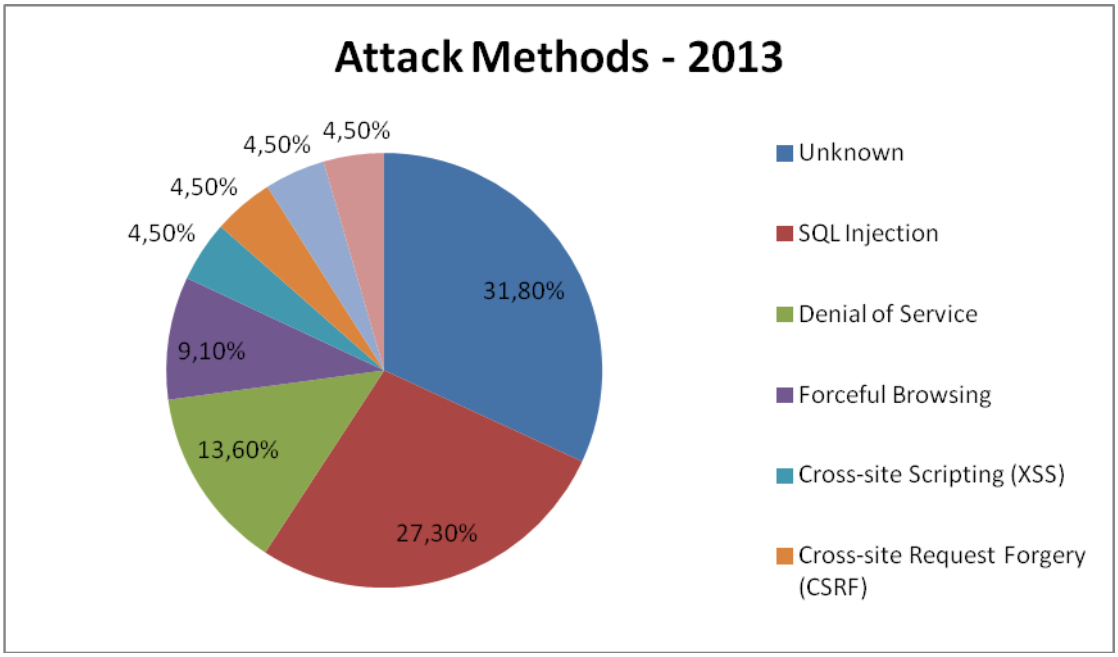


Figure 39: Attack methods - 2013

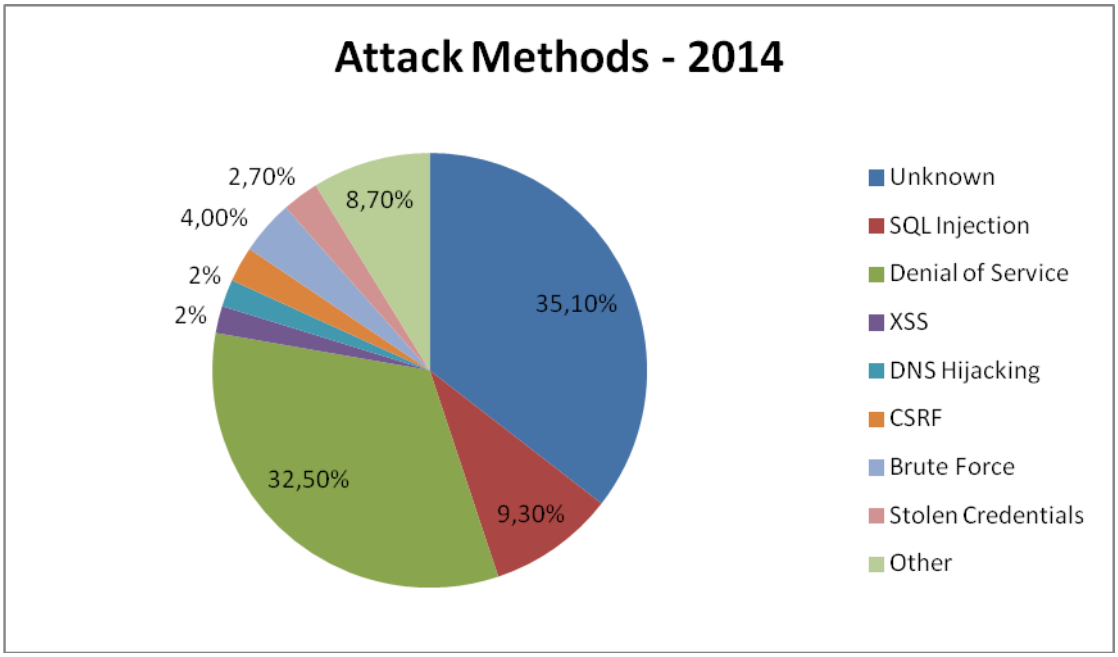


Figure 40: Attack methods - 2014

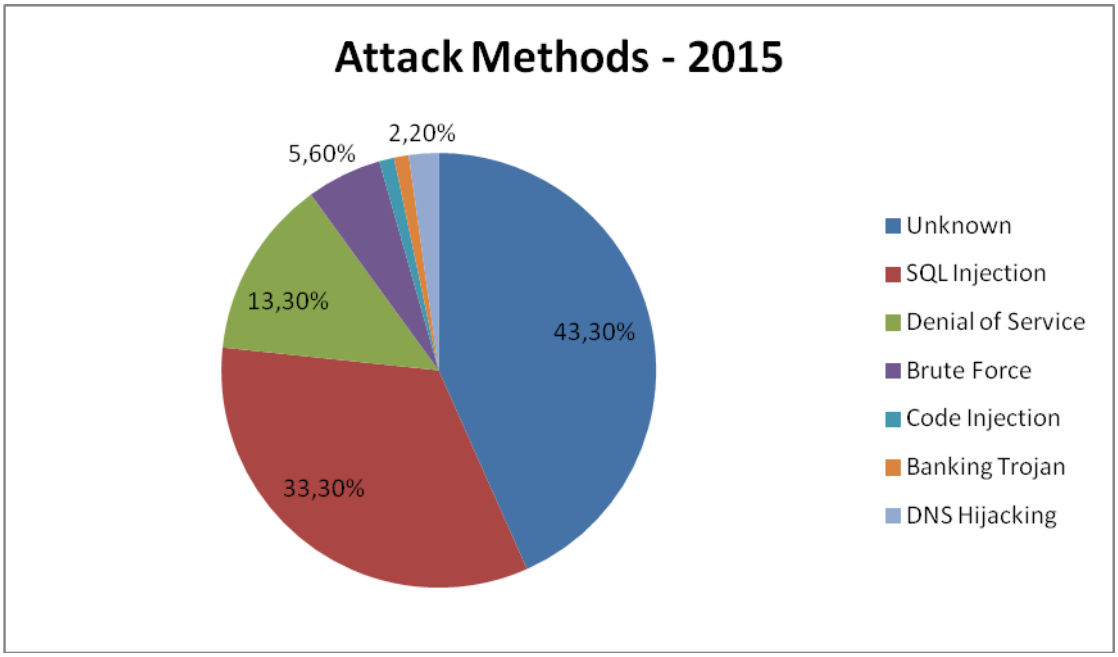


Figure 41: Attack methods - 2015

Verizon, in 2015 Data Breach Investigations Report (Verizon, 2015), mentions that the organized crime became the most frequently seen threat actor for web app attacks. As we can observe in the figure below, CSS and SQLi seem less favored than simply using actual credentials.

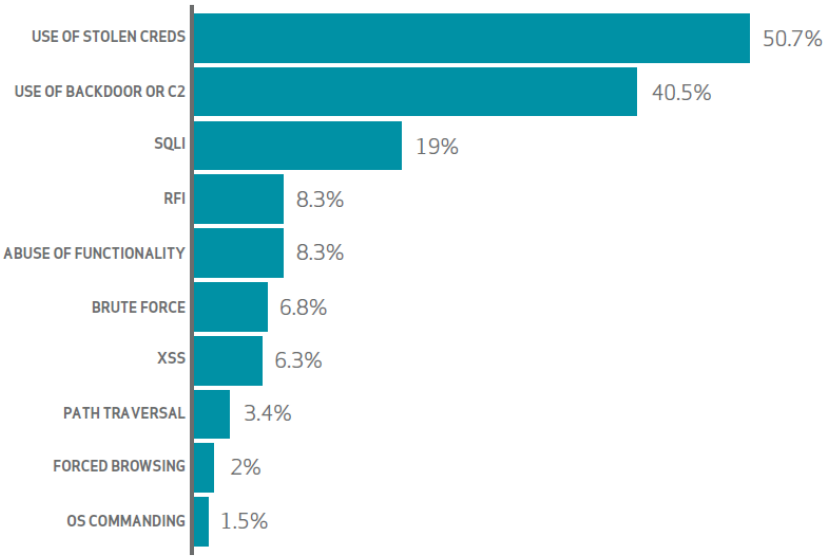


Figure 42: Variety of hacking actions within Web

Aspect Security (Aspect Security, Inc, 2014) observes that:

- Areas where flaws were well understood by developers include:
  - ✓ SQL Injection
  - ✓ Clickjacking
  - ✓ Cross-Site Request Forgery
- Areas where flaws were not well understood by developers include:
  - ✓ Protecting Sensitive Data: It isn't encouraging that developers scored 20% in this area.
  - ✓ Access Control Strategy: Only 34% of developers answered these questions correctly.
  - ✓ Threat Modeling and Architecture Reviews: Developers didn't do very well when asked about security architecture and security models scoring just 26% of questions correctly.
  - ✓ Unsafe Redirects: Only 34% of developers answered these questions correctly.
  - ✓ Secure Session Management: Results show only 52% of participants passed this area. Securing sessions improperly leads to session hijacking and other attacks.

Developers in general have a decent understanding of Injection, Cross-Site Scripting, and Using SSL, which are ranked number 1, 3 and 6 in the OWASP Top Ten 2013, respectively. High scores in these areas might suggest that we will see lower levels of these critical flaws in the future.

The chart below details aggregate scores from 1,425 participants for critical areas:

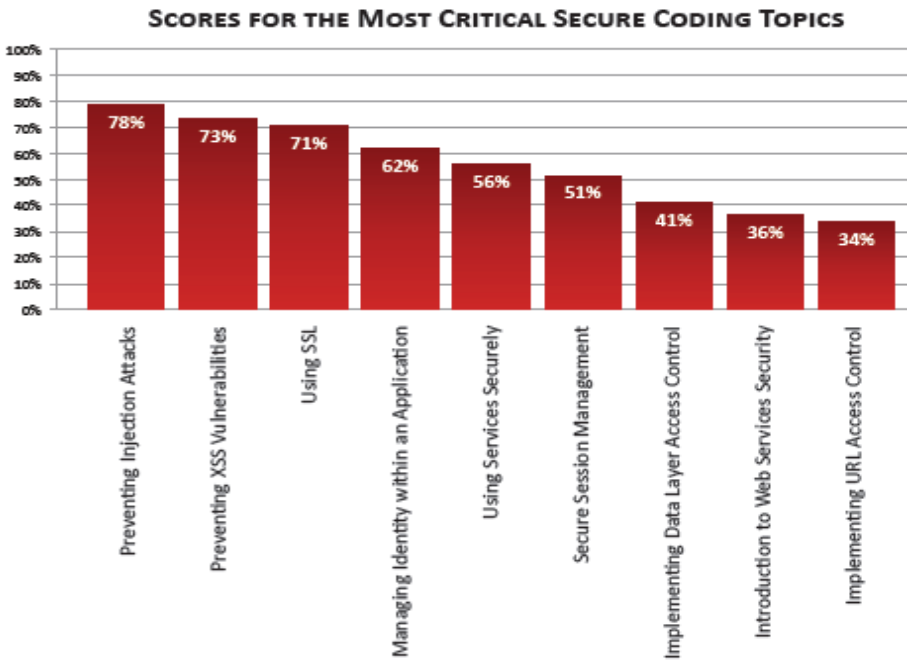


Figure 43: Scores for the most critical secure coding topics

According to Risk Based Security (Risk Based Security, Inc., 2015) analysis for the third quarter 2015, over 91% of all incidents involved electronic data and nearly 100% of the exposed records were in electronic form. This is a constant theme year over year. User names, email addresses and passwords remain a prize target. About a percentage of 66.3% of reported incidents was the result of Hacking, which accounted for 83.2% of the exposed records.

According to IBM (IBM Security, 2015), the infection scenario most commonly encountered by ERS in 2015 was ransomware. As its name suggests, this is a kind of malware that steals something from the user and demands a ransom to give it back. The IBM ERS team estimates that three primary vectors are the source of ransomware infections (see Figure 44). As we know unpatched operating system vulnerabilities can be exploited giving attackers access to the system resources.

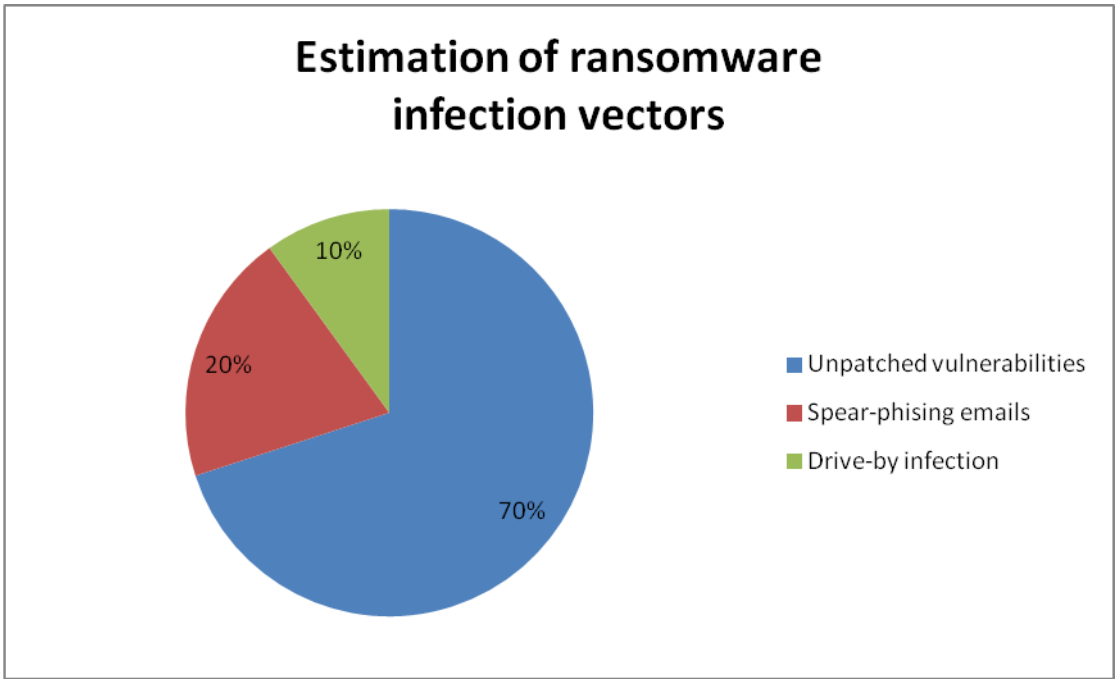


Figure 44: Estimation of ransomware infection vectors

In WhiteHat Website Security Statistics Report 2015 (WhiteHat, 2015), Insufficient Transport Layer Protection, Information Leakage and Cross-Site Scripting are the most likely vulnerabilities in applications (see Figure 45).

- Likelihood of Insufficient Transport Layer Protection: 70%
- Likelihood of Information Leakage: 56%
- Likelihood of Cross-site Scripting: 47%

Likelihood of Content Spoofing, Cross-site Scripting and Fingerprinting has sharply declined in recent years. Content Spoofing was 33% likely in 2012, but only 26% in 2014. Likelihood of Fingerprinting vulnerabilities has dropped from 23% in 2012 to 5% in 2014. Cross-site Scripting has significantly declined as well (from 53% in 2012 to 47% in 2014). The sharp rise in the likelihood of Insufficient Transport Layer Protection can be explained by discovery of zero-day vulnerabilities such as Heartbleed and the new tests added as a result of that.

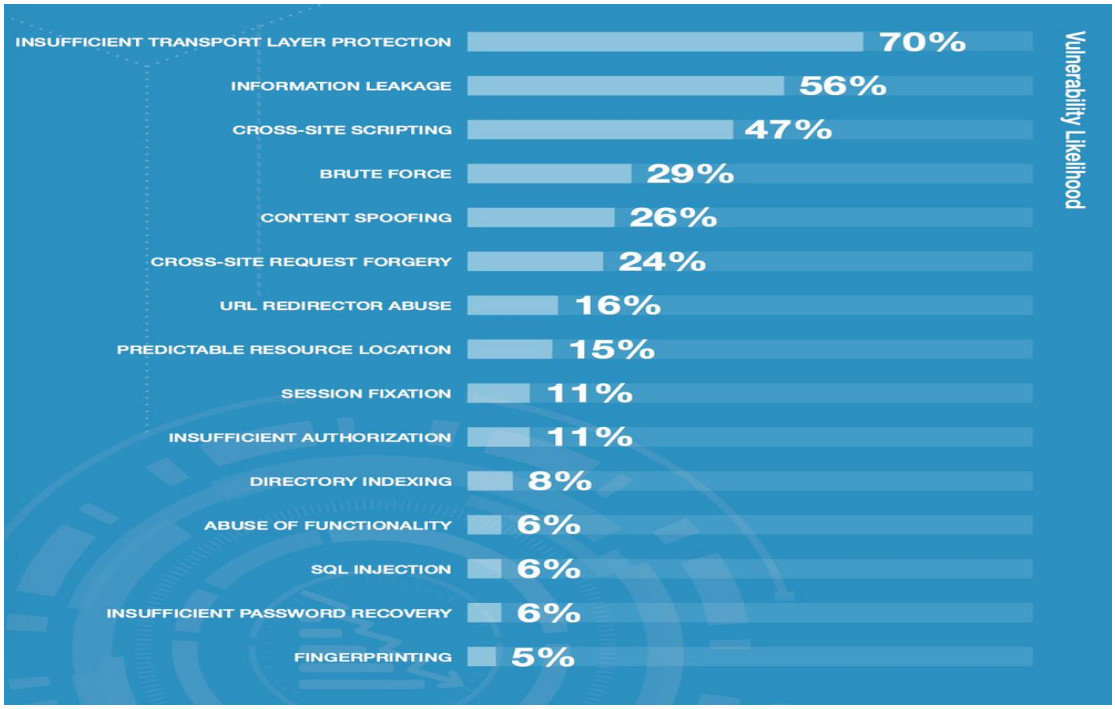


Figure 45: Application vulnerability likelihood – WhiteHat Report 2015

In Trustwave Security Report 2015 (Trustwave, 2015), we see a decline in XSS, CSRF and session management from 2013 to 2014. On the contrary, there is an increase in SQL injection and information leakage (see Figure 46).

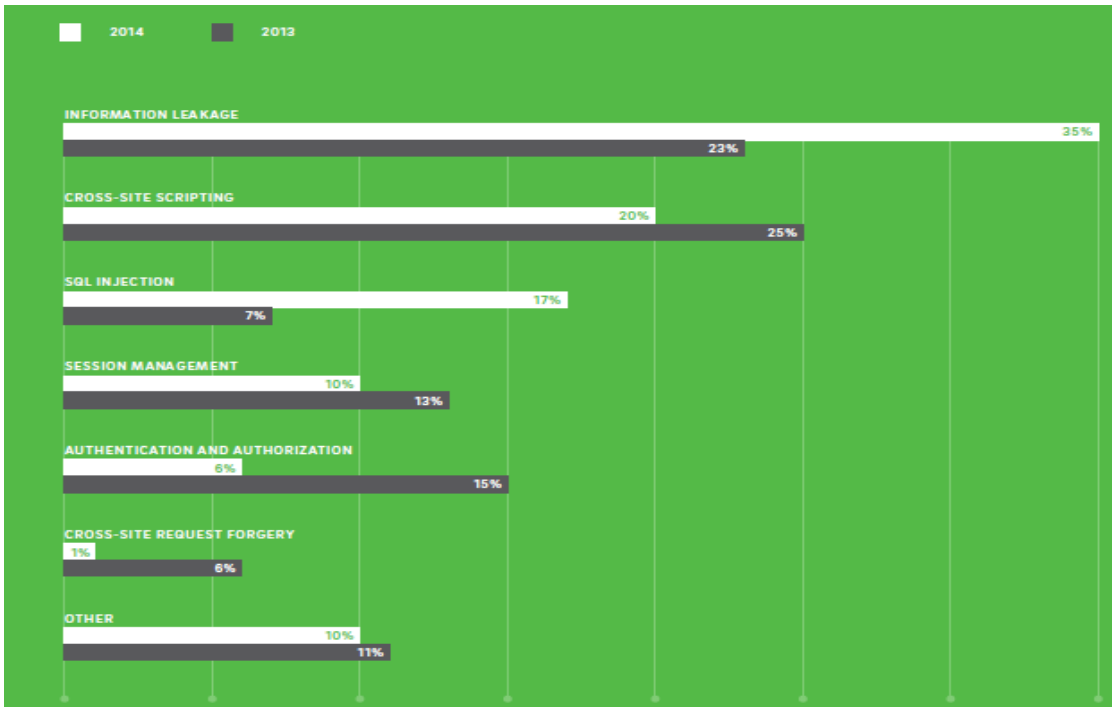


Figure 46: Total vulnerabilities identified by Trustwave in 2013 and 2014

Cross-site Scripting (XSS) and Denial of Service (DoS) vulnerabilities top in Accunetix's Top Vulnerabilities list for 2015 (Acunetix, 2015), with a significant 38% of websites being vulnerable to each of these attacks. Following closely at 28% are SSL related vulnerabilities such as HeartBleed and POODLE and SQL Injection (SQLi) at 27% of the sites scanned by Acunetix. This graph (Figure 47) shows that new security vulnerabilities, such as the super bugs discovered last year and earlier this year are already nearly as common as the older vulnerabilities such as XSS and SQLi, that have been around for decades.

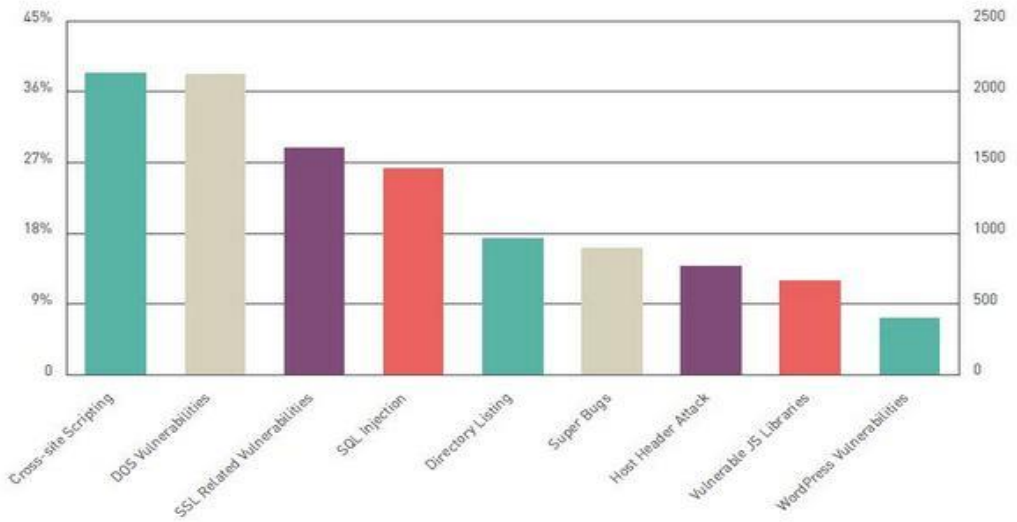


Figure 47: Statistical Vulnerabilities report 2015

According to Imperva's Web Application Attack Report for 2015 (Imperva, 2015), SQLi and XSS are gaining more popularity throughout the years as both distributions show a constant growth. Remote File Inclusion (RFI) shows an obvious decrease, reflecting the humans.txt 2014 mega-trend. HTTP attack remains stable over the years, whereas Directory Traversal attacks show a decrease in popularity (see Figure 48).

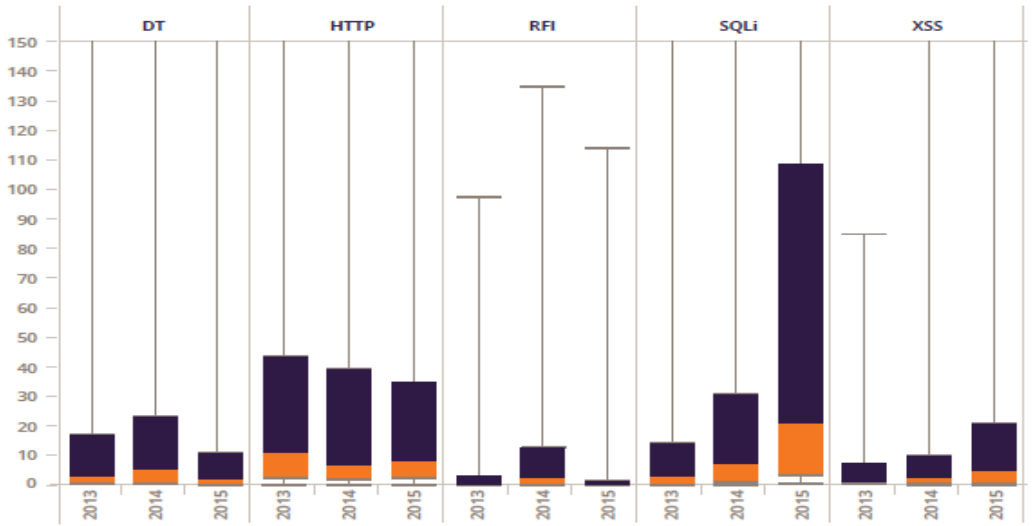


Figure 48: Comparison of Number of Incidents Between Years 2013-2015

Chapter 14: Conclusions

In this thesis, there are described the most common vulnerabilities found in Web applications as we can find them in OWASP Top Ten 2013. Some of them are general and can be found in most of Web applications and some are very specific. All vulnerabilities are considered risk to the company who owns the Web application and exploiting them can lead to harm and financial loss.

We observe that in the past, traditional attack activity primarily used widespread, broadcast attacks aimed at computers deployed on networks. However, as administrators and vendors fortified perimeter defenses with tools such as firewalls and intrusion detection/prevention systems (IDS/IPS), attackers responded by adopting new tactics. Instead of trying to penetrate networks with high-volume broadcast attacks, attackers have adopted stealthier, more focused techniques that target individual computers through the World Wide Web.

According to OWASP Top Ten 2013, these vulnerabilities are a concern because they allow attackers to compromise specific Web sites, which they can then use to launch subsequent attacks against users. This has shown to be an effective strategy for launching multistage attacks and exploiting client-side vulnerabilities. Another important conclusion is that attackers are particularly targeting sites that are likely to be trusted by end users, such as social networking sites. This increases the likelihood that the attacks will be successful because a user is more likely to allow a trusted site to execute code on his or her computer, or to open a file downloaded from a trusted site. Attackers targeting trusted sites can also steal user credentials or launch mass attacks because they may allow attacks to propagate quickly through a victim's social network.

Moreover, this thesis is shown how we can protect ourselves and avoid the impact of being attacked. We observe that user authentication process is not secure if we consider only the session. Protection should be considered for both in the server and client side. User supplied data and components should be checked properly. We must also give different authorization for different types of users. Developers must give more attention to the inside attackers. Encryption does not mean data is secured from the attacker. The above observations are only some of them we analyzed during this thesis writing.

In addition, this thesis makes contributions in two different areas that concern the security of web applications. The first area is that of detecting and preventing web threats, using appropriate tools and techniques. The second area is about presenting threats and trends with the most dangerous threats and vulnerabilities that we found the time period 2013-2015.

Overall, our findings confirm that web application vulnerabilities are increasingly posing serious threats to organizations' overall security posture, such as data loss or alteration, system down-time, loss of reputation and severe fines from the regulator, amongst others. Hackers continue to concentrate their efforts on web-based applications since they often have direct access to back-end data such as customer databases. Thus, website security should be a priority in any organization, but remains the most overlooked aspect of securing the enterprise.

References

1. Sullivan, B. and Liu, V. (2012). *Web application security*. New York: McGraw-Hill.
2. Bowbridge, (2014). *The Blind Spot*. White Paper.
3. Oracle, (2014). *Security in Oracle ADF: Addressing the OWASP Top 10 Security Vulnerabilities*.
4. Google.com, (2015). *Google Fusion Tables*. [online] Available at: https://www.google.com/fusiontables/DataSource?snapid=S2089124qF5 [Accessed 31 Dec. 2015].
5. Netcraft.com, (2015). *Netcraft | Open Redirect Detection Service*. [online] Available at: http://www.netcraft.com/security-testing/open-redirect-detection/ [Accessed 31 Dec. 2015].
6. Cwe.mitre.org, (2015). *CWE - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') (2.9)*. [online] Available at: http://cwe.mitre.org/data/definitions/601.html [Accessed 31 Dec. 2015].
7. Shue, C. A., Kalafut, A. J., & Gupta, M. (2008). Exploitable Redirects on the Web: Identification, Prevalence, and Defense. In *WOOT*.
8. Open Redirect Vulnerabilities. (2008). *(IN)SECURE Magazine*, (17).
9. Redirectremover.mozdev.org, (2015). [online] Available at: http://redirectremover.mozdev.org/ [Accessed 31 Dec. 2015].
10. Imperva, (2015). *Web Application Attack Report (WAAR)*.
11. Owasp.org, (2015). *Top 10 2013-Top 10 - OWASP*. [online] Available at: https://www.owasp.org/index.php/Top_10_2013-Top_10 [Accessed 31 Dec. 2015].
12. WhiteHat, (2015). *Website Security Statistics Report 2015*.
13. Outpost24, (2015). *Unvalidated Redirects 101 - OWASP10 - Outpost24*. [online] Available at: https://www.outpost24.com/unvalidated-redirects-101-owasp10/ [Accessed 31 Dec. 2015].
14. Eurovps.com, (2015). *Unvalidated Redirects and Forwards - OWASP #10*. [online] Available at: https://www.eurovps.com/blog/unvalidated-redirects-and-forwards-owasp-10 [Accessed 31 Dec. 2015].
15. SearchSoftwareQuality, (2015). *Security testing for unvalidated redirects and forwards*. [online] Available at:

http://searchsoftwarequality.techtarget.com/tip/Security-testing-for-unvalidated-redirects-and-forwards [Accessed 31 Dec. 2015].

16. McRee, R. V. (2008). *HolisticInfoSec: The Bitrix open redirect vulnerability: a lesson in the absurd*. [online] Holisticinfosec.blogspot.gr. Available at: http://holisticinfosec.blogspot.gr/2008/07/bitrix-open-redirect-vulnerability.html [Accessed 31 Dec. 2015].
17. Owasp.org, (2015). *OWASP Dependency Check - OWASP*. [online] Available at: https://www.owasp.org/index.php/OWASP_Dependency_Check [Accessed 31 Dec. 2015].
18. Blog.sonatype.com, (2015). *Better and Fewer Suppliers (2015 Software Supply Chain Report) | Sonatype Blog*. [online] Available at: http://blog.sonatype.com/2015/06/better-and-fewer-suppliers-2015-software-supply-chain-report/#.Vm_7GL9Twg_ [Accessed 31 Dec. 2015].
19. Sonatype, Aspect Security, (2013). *Executive Brief: Addressing Security Concerns in Open Source Components*.
20. Wang, H., Guo, C., Simon, D. and Zugenmaier, A. (2004). Shield. *ACM SIGCOMM Computer Communication Review*, 34(4), p.193.
21. Neuhaus, S., Zimmermann, T., Holler, C., & Zeller, A. (2007). Predicting vulnerable software components. In *Proceedings of the 14th ACM conference on Computer and communications security* (pp. 529-540). ACM.
22. Cadariu, M., Bouwers, E., Visser, J., & van Deursen, A. (2015). Tracking known security vulnerabilities in proprietary software systems. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on* (pp. 516-519). IEEE.
23. Owasp.org, (2015). *Top 10 2010-Main - OWASP*. [online] Available at: https://www.owasp.org/index.php/Top_10_2010-Main [Accessed 31 Dec. 2015].
24. Bilge, L., & Dumitras, T. (2012). Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security* (pp. 833-844). ACM.
25. Falliere, N., Murchu, L. O., & Chien, E. (2011). W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, *5*.
26. Symantec Security Response, (2015). *The Trojan.Hydraq Incident: Analysis of the Aurora 0-Day Exploit*. [online] Available at:

http://www.symantec.com/connect/blogs/trojanhydraq-incident-analysis-aurora-0-day-exploit [Accessed 31 Dec. 2015]. (Symantec Security Response, 2015)

27. Micro, T. (2012). *Spear-Phishing Email: Most Favored APT Attack Bait*. Technical report. Trend Micro.
28. Owasp.org, (2015). *OWASP Good Component Practices Project - OWASP*. [online] Available at: https://www.owasp.org/index.php/OWASP_Good_Component_Practices_Project [Accessed 31 Dec. 2015].
29. Eurovps.com, (2015). *Using Components with Known Vulnerabilities - OWASP #9*. [online] Available at: https://www.eurovps.com/blog/using-components-with-known-vulnerabilities-owasp-9 [Accessed 31 Dec. 2015].
30. CERY, P. (2014). *OWASP Top 10 – A9 Using Components with Known Vulnerabilities – Le...*. [online] Le Blog d'Ippon Technologies. Available at: http://blog.ippon.fr/2014/01/28/owasp-top-10-a9/ [Accessed 31 Dec. 2015].
31. Cwe.mitre.org, (2015). *CWE - CWE-829: Inclusion of Functionality from Untrusted Control Sphere (2.9)*. [online] Available at: https://cwe.mitre.org/data/definitions/829.html [Accessed 31 Dec. 2015].
32. http://www.codenomicon.com/, C. (2015). *Heartbleed Bug*. [online] Heartbleed.com. Available at: http://heartbleed.com/ [Accessed 31 Dec. 2015].
33. Sonatype, (2013). *Understanding & Addressing OWASP's Newest Top Ten Threat: Using Components with Known Vulnerabilities*.
34. Eduardo Fernandez and Rohini Sulatycki, (2013). Two threat patterns that exploit "Compromising applications using components with known vulnerabilities" and "Direct access to objects using uncontrolled references" vulnerabilities.
35. Kaur, R. and Singh, M. (2014). A Survey on Zero-Day Polymorphic Worm Detection Techniques. *IEEE Communications Surveys & Tutorials*, 16(3), pp.1520-1549.
36. Ting, C., Xiaosong, Z., & Zhi, L. (2009). A hybrid detection approach for zero-day polymorphic shellcodes. In *E-Business and Information System Security, 2009. EBISS'09. International Conference on* (pp. 1-5). IEEE.
37. Alosefer, Y., Rana, O. F. (2011). Predicting client-side attacks via behaviour analysis using honeypot data. In *Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on* (pp. 31-36). IEEE.
38. Owasp.org, (2016). *Forced browsing - OWASP*. [online] Available at: https://www.owasp.org/index.php/Forced_browsing [Accessed 1 Jan. 2016].

39. ComputerWeekly, (2016). *Forced browsing: Understanding and halting simple browser attacks*. [online] Available at: http://www.computerweekly.com/answer/Forced-browsing-Understanding-and-halting-simple-browser-attacks [Accessed 1 Jan. 2016].
40. CERY, P. (2013). *OWASP Top 10 – A7 Missing Function Level Access Control – Le Blog....* [online] Le Blog d'Ippon Technologies. Available at: http://blog.ippon.fr/2013/12/09/owasp-top-10-a7/ [Accessed 1 Jan. 2016].
41. Outpost24, (2014). *The truth behind automated web application scanners - Outpost24*. [online] Available at: https://www.outpost24.com/the-truth-behind-automated-web-application-scanners/ [Accessed 1 Jan. 2016].
42. IBM Security, (2015). *IBM X-Force Threat Intelligence Quarterly 4Q 2015*.
43. Eurovps.com, (2016). *Missing Function Level Access Control - OWASP #8*. [online] Available at: https://www.eurovps.com/blog/missing-function-level-access-control-owasp-8 [Accessed 1 Jan. 2016].
44. Nbs-system.co.uk, (2015). *CerberHost: missing function level access control*. [online] Available at: https://www.nbs-system.co.uk/blog-2/cerberhost-missing-function-level-access-control.html [Accessed 1 Jan. 2016].
45. Cwe.mitre.org, (2016). *CWE - CWE-287: Improper Authentication (2.9)*. [online] Available at: https://cwe.mitre.org/data/definitions/287.html [Accessed 1 Jan. 2016].
46. Ramesh, S. (2013). *DNS Hijacking: What is it and How it Works | GoHacking*. [online] Gohacking.com. Available at: http://www.gohacking.com/dns-hijacking/ [Accessed 1 Jan. 2016].
47. Paul, I. (2016). *DNSChanger Malware Set to Knock Thousands Off Internet on Monday*. [online] PCWorld. Available at: http://www.pcworld.com/article/258796/dnschanger_malware_set_to_knock_thousands_off_internet_on_monday.html [Accessed 1 Jan. 2016].
48. Google.com, (2016). *Google Fusion Tables*. [online] Available at: https://www.google.com/fusiontables/DataSource?snapid=S208910u7mt [Accessed 1 Jan. 2016].
49. Outpost24, (2016). *SWAT - Secure Web Applications Scanner - Outpost24*. [online] Available at: https://www.outpost24.com/products-vulnerability-management/swat-secure-web-applications-scanner/ [Accessed 1 Jan. 2016].

50. Bertino, E., Bonatti, P. A., & Ferrari, E. (2001). TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security (TISSEC)*, *4*(3), 191-233.
51. Thompson, M. R., Essiari, A., & Mudumbai, S. (2003). Certificate-based authorization policy in a PKI environment. *ACM Transactions on Information and System Security (TISSEC)*, *6*(4), 566-588.
52. Portokalidis, G. and Bos, H. (2007). SweetBait: Zero-hour worm detection and containment using low- and high-interaction honeypots. *Computer Networks*, 51(5), pp.1256-1274.
53. Lanjia Wang, Zhichun Li, Yan Chen, Zhi Fu, and Xing Li, (2010). Thwarting Zero-Day Polymorphic Worms With Network-Level Length-Based Signature Generation. *IEEE/ACM Transactions on Networking*, 18(1), pp.53-66.
54. Trustwave, (2015). *Global Security Report 2015*.
55. Bajpai, A. (2010). *Securing Apache, Part 3: Cross-Site Request Forgery Attacks (XSRF) - Open Source For You*. [online] Open Source For You. Available at: http://opensourceforu.efytimes.com/2010/11/securing-apache-part-3-xsrf-csrf/ [Accessed 31 Dec. 2015].
56. Burns, J. (2005). Cross Site Request Forgery. *An introduction to a common web application weakness, Information Security Partners*.
57. Zeller, W., & Felten, E. W. (2008). Cross-site request forgeries: Exploitation and prevention. *The New York Times*, 1-13.
58. Jovanovic, N., Kirda, E., & Kruegel, C. (2006). Preventing cross site request forgery attacks. In *Securecomm and Workshops, 2006* (pp. 1-10). IEEE.
59. Owasp.org, (2015). *CSRFTester Usage - OWASP*. [online] Available at: https://www.owasp.org/index.php/CSRFTester_Usage [Accessed 31 Dec. 2015].
60. Cwe.mitre.org, (2015). *CWE - CWE-352: Cross-Site Request Forgery (CSRF) (2.9)*. [online] Available at: https://cwe.mitre.org/data/definitions/352.html [Accessed 31 Dec. 2015].
61. Johns, M., & Winter, J. (2006). RequestRodeo: Client side protection against session riding. In *Proceedings of the OWASP Europe 2006 Conference*.
62. Chen, B., Zavarsky, P., Ruhl, R., & Lindskog, D. (2011). A Study of the Effectiveness of CSRF Guard. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on* (pp. 1269-1272). IEEE.

63. Lekies, S., Tighzert, W., & Johns, M. (2012). Towards stateless, client-side driven Cross-Site Request Forgery protection for Web applications. In *Sicherheit* (pp. 111-121).
64. Czeskis, A., Moshchuk, A., Kohno, T., & Wang, H. J. (2013). Lightweight server support for browser-based csrf protection. In *Proceedings of the 22nd international conference on World Wide Web* (pp. 273-284). International World Wide Web Conferences Steering Committee.
65. Mao, Z., Li, N., & Molloy, I. (2009). Defeating cross-site request forgery attacks with browser-enforced authenticity protection. In *Financial Cryptography and Data Security* (pp. 238-255). Springer Berlin Heidelberg.
66. De Ryck, P., Desmet, L., Heyman, T., Piessens, F., & Joosen, W. (2010). CsFire: Transparent client-side mitigation of malicious cross-domain requests. In *Engineering Secure Software and Systems* (pp. 18-34). Springer Berlin Heidelberg.
67. AlAmeen, A. (2015). Building a Robust Client-Side Protection Against Cross Site Request Forgery. *International Journal of Advanced Computer Science and Applications*, 6(6).
68. Kerschbaum, F. (2007). Simple cross-site attack prevention. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on* (pp. 464-472). IEEE.
69. Son, S.(2008) Prevent Cross-site Request Forgery: PCRF. *University of Texas, Austin, Texas USA.(10 pg.).*
70. Blatz, J. (2007). CSRF: Attack and Defense. *McAfee® Foundstone® Professional Services, White Paper*.
71. Komlosi, J. (2015). *Protection against Cross-site request forgery (CSRF, XSRF)*. [online] Kentico DevNet. Available at: https://devnet.kentico.com/articles/protection-against-cross-site-request-forgery-%28csrf-xsrf%29 [Accessed 31 Dec. 2015].
72. Doshi, J., & Trivedi, B. (2014). Sensitive Data Exposure Prevention using Dynamic Database Security Policy. *International Journal of Computer Applications*, *106*(15).
73. Zhu, D. Y., Jung, J., Song, D., Kohno, T., & Wetherall, D. (2011). TaintEraser: protecting sensitive data leaks using application-level taint tracking. *ACM SIGOPS Operating Systems Review*, *45*(1), 142-154.
74. Liu, F., Shu, X., Yao, D., & Butt, A. R. (2015). Privacy-preserving scanning of big content for sensitive data exposure with MapReduce. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy* (pp. 195-206). ACM.

75. Vachharajani, N., Bridges, M. J., Chang, J., Rangan, R., Ottoni, G., Blome, J., August, D. (2004). RIFLE: An architectural framework for user-centric information-flow security. In *Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on* (pp. 243-254). IEEE.
76. Wang, X., Li, Z., Li, N., & Choi, J. Y. (2008). PRECIP: Towards Practical and Retrofittable Confidential Information Protection. In *NDSS*.
77. Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., ... & Morris, R. (2005). Labels and event processes in the Asbestos operating system. In *ACM SIGOPS Operating Systems Review* (Vol. 39, No. 5, pp. 17-30). ACM.
78. Qin, F., Wang, C., Li, Z., Kim, H. S., Zhou, Y., & Wu, Y. (2006). Lift: A low-overhead practical information flow tracking system for detecting security attacks. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on* (pp. 135-148). IEEE.
79. Clause, J., Li, W., & Orso, A. (2007). Dytan: a generic dynamic taint analysis framework. In *Proceedings of the 2007 international symposium on Software testing and analysis* (pp. 196-206). ACM.
80. Yin, H., Song, D., Egele, M., Kruegel, C., & Kirda, E. (2007). Panorama: capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM conference on Computer and communications security* (pp. 116-127). ACM.
81. Zeldovich, N., Boyd-Wickizer, S., Kohler, E., & Mazières, D. (2006). Making information flow explicit in HiStar. In *Proceedings of the 7th symposium on Operating systems design and implementation* (pp. 263-278). USENIX Association.
82. Cheng, W., Zhao, Q., Yu, B., & Hiroshige, S. (2006). Tainttrace: Efficient flow tracing with dynamic binary rewriting. In *Computers and Communications, 2006. ISCC'06. Proceedings. 11th IEEE Symposium on* (pp. 749-754). IEEE.
83. Newsome, J., & Song, D. (2005). Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software.
84. Croft, J., & Caesar, M. (2011). Towards Practical Avoidance of Information Leakage in Enterprise Networks. In *HotSec*.
85. Risk Based Security, Inc., (2015). *Data Breach Intelligence*.
86. Ho, A., Fetterman, M., Clark, C., Warfield, A., & Hand, S. (2006). Practical taint-based protection using demand emulation. In *ACM SIGOPS Operating Systems Review* (Vol. 40, No. 4, pp. 29-41). ACM.

87. Bilge, L., Balzarotti, D., Robertson, W., Kirda, E., & Kruegel, C. (2012). Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference* (pp. 129-138). ACM.
88. Nadkarni, A., & Enck, W. (2013). Preventing accidental data disclosure in modern operating systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 1029-1042). ACM.
89. Yen, T. F., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A., & Kirda, E. (2013). Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *Proceedings of the 29th Annual Computer Security Applications Conference* (pp. 199-208). ACM.
90. Borders, K., & Prakash, A. (2009). Quantifying information leaks in outbound web traffic. In *Security and Privacy, 2009 30th IEEE Symposium on* (pp. 129-140). IEEE.
91. Kadhem, H., Amagasa, T., & Kitagawa, H. (2009). A novel framework for database security based on mixed cryptography. In *Internet and Web Applications and Services, 2009. ICIW'09. Fourth International Conference on* (pp. 163-170). IEEE.
92. Naseem, M., Hussain, I. M., Khan, M. K., & Ajmal, A. (2011). An Optimum Modified Bit Plane Splicing LSB Algorithm for Secret Data Hiding. *International Journal of Computer Applications*, *29*(12).
93. Deshmukh, D., Pasha, A., & Qureshi, D. (2013). Transparent Data Encryption--Solution for Security of Database Contents. *arXiv preprint arXiv:1303.0418*.
94. Pinn, J. Z., & Zung, A. F. (2013). A new Watermarking Technique for Secure Database. *arXiv preprint arXiv:1304.7094*.
95. Kemptechnologies.com, (2015). *OWASP Top Ten Series: Sensitive Data Exposure*. [online] Available at: http://kemptechnologies.com/blog/owasp-top-ten-series-sensitive-data-exposure/ [Accessed 1 Jan. 2016].
96. Apps.testinsane.com, (2016). *Sensitive Data Exposure MindMap*. [online] Available at: http://apps.testinsane.com/mindmaps/Sensitive-Data-Exposure [Accessed 1 Jan. 2016].
97. Eurovps.com, (2016). *Sensitive Data Exposure - OWASP #7*. [online] Available at: https://www.eurovps.com/blog/sensitive-data-exposure-owasp-7 [Accessed 1 Jan. 2016].

98. Agrawal, R., Kiernan, J., Srikant, R., & Xu, Y. (2004). Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (pp. 563-574). ACM.
99. Microsoft, (2010). *Quick Security Reference:Exposure of Sensitive Information*.
100. Google.com, (2016). *Google Fusion Tables*. [online] Available at: https://www.google.com/fusiontables/DataSource?snapid=S2089112yxM [Accessed 1 Jan. 2016].
101. Itrcweb.org, (2016). *Interstate Technology & Regulatory Council (ITRC)*. [online] Available at: http://www.itrcweb.org/ [Accessed 1 Jan. 2016].
102. Garg, R. (2015). *2015: Data Breach Stats*, Year Until 10/06/2015 | Zecurion*. [online] Zecurion.com. Available at: http://zecurion.com/2015/10/12/2015-data-breach-stats-year-until-10062015/ [Accessed 1 Jan. 2016].
103. Kemptechnologies.com, (2015). *OWASP Top Ten Series: Security Misconfiguration*. [online] Available at: http://kemptechnologies.com/blog/owasp-top-ten-series-security-misconfiguration/ [Accessed 1 Jan. 2016].
104. Google.com, (2016). *Google Fusion Tables*. [online] Available at: https://www.google.com/fusiontables/DataSource?snapid=S208909HtmA [Accessed 1 Jan. 2016].
105. Johnson, G. (2008). Remote and Local File Inclusion Explained.
106. Huang, L. S., Moshchuk, A., Wang, H. J., Schecter, S., & Jackson, C. (2012). Clickjacking: Attacks and Defenses. In *USENIX Security Symposium* (pp. 413-428).
107. Projects.webappsec.org, (2016). *The Web Application Security Consortium / Remote File Inclusion*. [online] Available at: http://projects.webappsec.org/w/page/13246955/Remote%20File%20Inclusion [Accessed 1 Jan. 2016].
108. Owasp.org, (2016). *Clickjacking - OWASP*. [online] Available at: https://www.owasp.org/index.php/Clickjacking [Accessed 1 Jan. 2016].
109. Hayati, P., & Potdar, V. (2009). Spammer and hacker, two old friends. In *Digital Ecosystems and Technologies, 2009. DEST'09. 3rd IEEE International Conference on* (pp. 290-294). IEEE.
110. Projects.webappsec.org, (2016). *The Web Application Security Consortium / Abuse of Functionality*. [online] Available at: http://projects.webappsec.org/w/page/13246913/Abuse%20of%20Functionality [Accessed 1 Jan. 2016].

111. Projects.webappsec.org, (2016). *The Web Application Security Consortium / Server Misconfiguration*. [online] Available at: http://projects.webappsec.org/w/page/13246959/Server%20Misconfiguration [Accessed 1 Jan. 2016].
112. Owasp.org, (2016). *Insecure Configuration Management - OWASP*. [online] Available at: https://www.owasp.org/index.php/Insecure_Configuration_Management [Accessed 1 Jan. 2016].
113. Aron, M., Iyer, S., & Druschel, P. (2001). A resource management framework for predictable quality of service in web servers. *Submitted for publication*.
114. Projects.webappsec.org, (2016). *The Web Application Security Consortium / Application Misconfiguration*. [online] Available at: http://projects.webappsec.org/w/page/13246914/Application%20Misconfiguration [Accessed 1 Jan. 2016].
115. Infosecpro.com, (2016). *InfoSecPro.com - Computer, network, application and physical security consultants.*. [online] Available at: http://www.infosecpro.com/applicationsecurity/a61.htm [Accessed 1 Jan. 2016].
116. Wang, H., Platt, J., Chen, Y., Zhang, R. and Wang, Y. (2004). PeerPressure for automatic troubleshooting. *ACM SIGMETRICS Performance Evaluation Review*, 32(1), p.398.
117. Eshete, B., Villafiorita, A., & Weldemariam, K. (2011). Early detection of security misconfiguration vulnerabilities in web applications. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on* (pp. 169-174). IEEE.
118. Lin, C. H., Chen, C. H., & Laih, C. S. (2008). A study and implementation of vulnerability assessment and misconfiguration detection. In *Asia-Pacific Services Computing Conference, 2008. APSCC'08. IEEE* (pp. 1252-1257). IEEE.
119. Das, T., Bhagwan, R., & Naldurg, P. (2010). Baaz: A System for Detecting Access Control Misconfigurations. In *USENIX Security Symposium* (pp. 161-176).
120. Hochreiner, C., Frühwirt, P., Ma, Z., Kieseberg, P., Schrittwieser, S., & Weippl, E. (2014). Genie in a Model? Why Model Driven Security will not secure your Web Application. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, *5*(3), 44-62.
121. Hurkała, A., & Hurkała, J. (2013). Authentication System for Websites with Paid Content: An Overview of Security and Usability Issues. *IJCSNS International Journal of Computer Science and Network Security*, *13*(7), 42-49.

122. PCMAG, (2016). *Web Server Security Best Practices*. [online] Available at: http://www.pcmag.com/article2/0,2817,11525,00.asp [Accessed 1 Jan. 2016].
123. Owasp.org, (2016). *Configuration - OWASP*. [online] Available at: https://www.owasp.org/index.php/Configuration [Accessed 1 Jan. 2016].
124. Code.google.com, (2016). *zaproxy - OWASP ZAP: An easy to use integrated penetration testing tool for finding vulnerabilities in web applications. Now on GitHub: https://github.com/zaproxy/zaproxy - Google Project Hosting*. [online] Available at: http://code.google.com/p/zaproxy/ [Accessed 1 Jan. 2016].
125. Code.google.com, (2016). *skipfish - web application security scanner - Google Project Hosting*. [online] Available at: http://code.google.com/p/skipfish/ [Accessed 1 Jan. 2016].
126. Paros, (2013). *Paros*. [online] SourceForge. Available at: http://www.parosproxy.org/index.shtml [Accessed 1 Jan. 2016].
127. Phpsec.org, (2016). *PHP Security Consortium: PHPSecInfo*. [online] Available at: http://phpsec.org/projects/phpsecinfo/ [Accessed 1 Jan. 2016].
128. PHP Security Audit. (2016). [online] Available at: http://php-security-audit.com [Accessed 1 Jan. 2016].
129. Owasp.org, (2016). *Top 10 2007-Malicious File Execution - OWASP*. [online] Available at: https://www.owasp.org/index.php/Top_10_2007-Malicious_File_Execution [Accessed 1 Jan. 2016].
130. InfoSec Resources, (2013). *Content Spoofing - InfoSec Resources*. [online] Available at: http://resources.infosecinstitute.com/content-spoofing/ [Accessed 1 Jan. 2016].
131. Owasp.org, (2016). *Top 10 2007-Insecure Direct Object Reference - OWASP*. [online] Available at: https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference [Accessed 1 Jan. 2016].
132. Pwnrules.com, (2016). *Vulnerability in Yahoo allowed me to delete more than 1 million and half records from Yahoo database by PWN Rules!*. [online] Available at: http://pwnrules.com/yahoo-suggestions-vulnerability/ [Accessed 1 Jan. 2016].
133. Abc.net.au, (2016). *7.30 Report - 29/6/2000: Computer student hacks into GST web site*. [online] Available at: http://www.abc.net.au/7.30/stories/s146760.htm [Accessed 1 Jan. 2016].
134. Usenix.org, (2016). *14th USENIX Security Symposium — Technical Paper*. [online] Available at:

https://www.usenix.org/legacy/event/sec05/tech/full_papers/livshits/livshits_html/ [Accessed 1 Jan. 2016].

135. Projects.webappsec.org, (2016). *The Web Application Security Consortium / Path Traversal*. [online] Available at: http://projects.webappsec.org/w/page/13246952/Path%20Traversal [Accessed 1 Jan. 2016].
136. DuPaul, N. (2014). *Directory Traversal*. [online] Veracode. Available at: http://www.veracode.com/security/directory-traversal [Accessed 1 Jan. 2016].
137. Docs.kentico.com, (2016). *Directory traversal - Kentico 8 Documentation - Kentico Documentation*. [online] Available at: https://docs.kentico.com/display/K8/Directory+traversal [Accessed 1 Jan. 2016].
138. Cwe.mitre.org, (2016). *CWE - CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (2.9)*. [online] Available at: http://cwe.mitre.org/data/definitions/22.html [Accessed 1 Jan. 2016].
139. Projects.webappsec.org, (2016). *The Web Application Security Consortium / Null Byte Injection*. [online] Available at: http://projects.webappsec.org/w/page/13246949/Null%20Byte%20Injection [Accessed 1 Jan. 2016].
140. Owasp-esapi-java.googlecode.com, (2016). *AccessReferenceMap (ESAPI 2.0.1 API)*. [online] Available at: http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/AccessReferenceMap.html [Accessed 1 Jan. 2016].
141. Jtmelton.com, (2010). *The OWASP Top Ten and ESAPI – Part 4 – Insecure Direct Object Reference – John Melton's Weblog*. [online] Available at: http://www.jtmelton.com/2010/05/10/the-owasp-top-ten-and-esapi-part-5-insecure-direct-object-reference/ [Accessed 1 Jan. 2016].
142. Sekar, R. (2009). An Efficient Black-box Technique for Defeating Web Application Attacks. In *NDSS*.
143. Wang, H.(2014). Preventing Insecure Direct Object References In App Development.
144. Møller, A., & Schwarz, M. (2014). Automated detection of client-state manipulation vulnerabilities. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *23*(4), 29.
145. Enterprisenetworkingplanet.com, (2016). *Protect Your Web Apps From Insecure Direct Object References - Page 2*. [online] Available at:

http://www.enterprisenetworkingplanet.com/netsecur/article.php/10952_3927401_2/Protect-Your-Web-Apps-From-Insecure-Direct-Object-References.htm [Accessed 1 Jan. 2016].

146. Owasp.org, (2016). *Testing Directory traversal/file include (OTG-AUTHZ-001) - OWASP*. [online] Available at: https://www.owasp.org/index.php/Testing_for_Path_Traversal_%28OWASP-AZ-001%29 [Accessed 1 Jan. 2016].
147. Owasp.org, (2016). *Testing for Insecure Direct Object References (OTG-AUTHZ-004) - OWASP*. [online] Available at: https://www.owasp.org/index.php/Testing_for_Insecure_Direct_Object_References_%28OTG-AUTHZ-004%29 [Accessed 1 Jan. 2016].
148. Websec.ca, (2016). *Panoptic - A tool to exploit path traversal vulnerabilities*. [online] Available at: http://websec.ca/blog/view/panoptic [Accessed 1 Jan. 2016].
149. Tools.kali.org, (2016). *DotDotPwn | Penetration Testing Tools*. [online] Available at: http://tools.kali.org/information-gathering/dotdotpwn [Accessed 1 Jan. 2016].
150. Emmanuel Benoist, (2015). Insecure Direct Object Reference.
151. Stuttard, D., & Pinto, M. (2008). *The web application hacker's handbook: discovering and exploiting security flaws*. John Wiley & Sons.
152. Acunetix, (2015). *Web Application Vulnerability Report, 2015*.
153. Owasp.org, (2016). *Cross-site Scripting (XSS) - OWASP*. [online] Available at: https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29 [Accessed 1 Jan. 2016].
154. Owasp.org, (2016). *Types of Cross-Site Scripting - OWASP*. [online] Available at: https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting [Accessed 1 Jan. 2016].
155. Johns, M. (2011). Code-injection Vulnerabilities in Web Applications—Exemplified at Cross-site Scripting. *it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, *53*(5), 256-260.
156. Garcia-Alfaro, J., & Navarro-Arribas, G. (2009). A survey on cross-site scripting attacks. *arXiv preprint arXiv:0905.4850*.
157. Grossman, J. (2006). Cross-site scripting worms and viruses. *Whitehat Security, 2006*.
158. Webappsec.org, (2016). *DOM Based Cross Site Scripting*. [online] Available at: http://www.webappsec.org/projects/articles/071105.html [Accessed 1 Jan. 2016].

159. Fogie, S., Grossman, J., Hansen, R., Rager, A., & Petkov, P. D. (2007). Xss exploits: Cross site scripting attacks and defense. *Syngress*, *2*(3), 5.

160. Cwe.mitre.org, (2016). *CWE - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (2.9)*. [online] Available at: http://cwe.mitre.org/data/definitions/79.html [Accessed 1 Jan. 2016].

161. Gupta, S., & Gupta, B. B. (2015). Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, 1-19.

162. Cao, Y., Yegneswaran, V., Porras, P. A., & Chen, Y. (2012). PathCutter: Severing the Self-Propagation Path of XSS JavaScript Worms in Social Web Networks. In *NDSS*.

163. SourceForge, (2013). *Gamja : Web vulnerability scanner*. [online] Available at: http://sourceforge.net/projects/gamja [Accessed 1 Jan. 2016].

164. Wapiti.sourceforge.net, (2016). *Wapiti : a Free and Open-Source web-application vulnerability scanner in Python for Windows, Linux, BSD, OSX*. [online] Available at: http://wapiti.sourceforge.net/ [Accessed 1 Jan. 2016].

165. W3af.sourceforge.net, (2016). *w3af - Open Source Web Application Security Scanner*. [online] Available at: http://w3af.sourceforge.net/ [Accessed 1 Jan. 2016].

166. Gnucitizen.org, (2016). *Landing Proxify*. [online] Available at: http://www.gnucitizen.org/projects/javascript-xss-scanner/ [Accessed 1 Jan. 2016].

167. Jim, T., Swamy, N., & Hicks, M. (2007). Defeating script injection attacks with browser-enforced embedded policies. In *Proceedings of the 16th international conference on World Wide Web* (pp. 601-610). ACM.

168. Johns, M., Engelmann, B., & Posegga, J. (2008). Xssds: Server-side detection of cross-site scripting attacks. In *Computer Security Applications Conference, 2008. ACSAC 2008. Annual* (pp. 335-344). IEEE

169. Kirda, E., Kruegel, C., Vigna, G., & Jovanovic, N. (2006). Noxes: a client-side solution for mitigating cross-site scripting attacks. In *Proceedings of the 2006 ACM symposium on Applied computing* (pp. 330-337). ACM.

170. Kals, S., Kirda, E., Kruegel, C., & Jovanovic, N. (2006, May). Secubat: a web vulnerability scanner. In *Proceedings of the 15th international conference on World Wide Web* (pp. 247-256). ACM.

171. Van Gundy, M., & Chen, H. (2012). Noncespaces: Using randomization to defeat cross-site scripting attacks. *computers & security*, *31*(4), 612-628.

172. Bisht, P., & Venkatakrishnan, V. N. (2008). XSS-GUARD: precise dynamic prevention of cross-site scripting attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 23-43). Springer Berlin Heidelberg.
173. Nadji, Y., Saxena, P., & Song, D. (2009). Document Structure Integrity: A Robust Basis for Cross-site Scripting Defense. In *NDSS*.
174. Louw, M. T., & Venkatakrishnan, V. N. (2009). Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In *Security and Privacy, 2009 30th IEEE Symposium on* (pp. 331-346). IEEE.
175. Wurzinger, P., Platzer, C., Ludl, C., Kirda, E., & Kruegel, C. (2009). SWAP: Mitigating XSS attacks using a reverse proxy. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems* (pp. 33-39). IEEE Computer Society.
176. Shahriar, H., & Zulkernine, M. (2011). S2XS2: a server side approach to automatically detect XSS attacks. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on* (pp. 7-14). IEEE.
177. Johns, M. (2006). SessionSafe: Implementing XSS immune session handling. In *Computer Security–ESORICS 2006* (pp. 444-460). Springer Berlin Heidelberg.
178. Scott, D., & Sharp, R. (2002). Abstracting application-level web security. In *Proceedings of the 11th international conference on World Wide Web* (pp. 396-407). ACM.
179. Minamide, Y. (2005). Static approximation of dynamically generated web pages. In *Proceedings of the 14th international conference on World Wide Web* (pp. 432-441). ACM.
180. Ismail, O., Etoh, M., Kadobayashi, Y., & Yamaguchi, S. (2004). A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability. In *Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on* (Vol. 1, pp. 145-151). IEEE.
181. Bates, D., Barth, A., & Jackson, C. (2010). Regular expressions considered harmful in client-side XSS filters. In *Proceedings of the 19th international conference on World wide web* (pp. 91-100). ACM.
182. Wiki.mozilla.org, (2016). *Security/Features/XSS Filter - MozillaWiki*. [online] Available at: https://wiki.mozilla.org/Security/Features/XSS_Filter [Accessed 1 Jan. 2016].

183. Noscript.net., (2016). *NoScript - JavaScript/Java/Flash blocker for a safer Firefox experience! - what is it? - InformAction*. [online] Available at: http://www.noscript.net. [Accessed 1 Jan. 2016].
184. Reith, J. (2008). *Internals of noXSS*. [online] noXSS.org. Available at: http://www.noxss.org/wiki/Internals. [Accessed 2 Jan. 2016].
185. Blogs.technet.com, (2016). *IE 8 XSS Filter Architecture / Implementation - Security Research & Defense - Site Home - TechNet Blogs*. [online] Available at: http://blogs.technet.com/srd/archive/2008/08/18/ie-8-xss-filter-architecture-implementation.aspx [Accessed 1 Jan. 2016].
186. Jovanovic, N., Kruegel, C., & Kirda, E. (2006). Pixy: A static analysis tool for detecting web application vulnerabilities. In *Security and Privacy, 2006 IEEE Symposium on* (pp. 6-pp). IEEE.
187. Martin, M., Livshits, B. and Lam, M. (2005). Finding application errors and security flaws using PQL. *ACM SIGPLAN Notices*, 40(10), p.365.
188. Halfond, W. G., Viegas, J., & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering* (Vol. 1, pp. 13-15). IEEE.
189. Saini, P.(2015). Survey and Comparative Analysis of SQL Injection Attacks, Detection and Prevention Techniques for Web Applications Security.
190. Sheykhkanloo, N. M. (2015). A Pattern Recognition Neural Network Model for Detection and Classification of SQL Injection Attacks. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, *9*(6), 1288-1298.
191. Chema Alonso, Daniel Kachakil, Rodolfo Bordσn, Antonio Guzmαn y Marta Beltran. (2007) Time-Based Blind SQL Injection using Heavy Queries.
192. Clarke, J. (2009). *SQL injection attacks and defense*. Elsevier.
193. Symantec, (2008). *Symantec Internet Security Threat Report, 2008*.
194. Kost, S. (2004). An introduction to SQL injection attacks for Oracle developers.
195. Ping-Chen, X. (2011). SQL injection attack and guard technical research. *Procedia Engineering*, *15*, 4131-4135.
196. Ying Jin, Xiaoying Shen, Chunhui Song. (2012). A filter-based approach for sql injection attack detection.

197. Appelt, D., Alshahwan, N., & Briand, L. (2013). Assessing the impact of firewalls and database proxies on sql injection testing. In *Proceedings of the 1st International Workshop on Future Internet Testing*.

198. Pcisecuritystandards.org, (2015). *Official PCI Security Standards Council Site - Verify PCI Compliance, Download Data Security and Credit Card Security Standards*. [online] Available at: https://www.pcisecuritystandards.org [Accessed 2 Jan. 2016].

199. Alonso, J. M., Bordon, R., Beltran, M., & Guzmán, A. (2008). LDAP injection techniques. In *Communication Systems, 2008. ICCS 2008. 11th IEEE Singapore International Conference on* (pp. 980-986). IEEE.

200. Halfond, W. G., & Orso, A. (2005). AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering* (pp. 174-183). ACM.

201. Boyd, S. W., & Keromytis, A. D. (2004). SQLrand: Preventing SQL injection attacks. In *Applied Cryptography and Network Security* (pp. 292-302). Springer Berlin Heidelberg.

202. R. A. McClure and I. H. Kruger. (2005). Sql dom: compile time checking of dynamic sql statements. In *Proceedings of the 27th international conference on Software engineering*, ICSE ’05, pages 88–96, New York.

203. Y. W. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee, and S. Y. Kuo. (2004). Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th international conference on World Wide Web*, WWW ’04, pages 40–52, New York.

204. G. Buehrer, B. W. Weide, and P. A. G. Sivilotti. (2005). Using parse tree validation to prevent sql injection attacks. In *Proceedings of the 5th international workshop on Software engineering and middleware*, SEM ’05, pages 106–113, New York.

205. P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan. (2010). Candid: Dynamic candidate evaluations for automatic prevention of sql injection attacks. *ACM Trans. Inf. Syst. Secur.*

206. Z. Su and G. Wassermann. (2006). The essence of command injection attacks in web applications. *SIGPLAN Not.*, 41:372–382.

207. K. Kemalis and T. Tzouramanis. (2008). Sql-ids: a specification-based approach for sql-injection detection. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC ’08, pages 2153–2158, New York, NY, USA.

208. F. Dysart and M. Sherriff. (2008). Automated fix generator for sql injection attacks. *Software Reliability Engineering, International Symposium*, 0:311–312.
209. Sasha Faust. (2002). LDAP INJECTION.
210. Alonso, J. M., Bordon, R., Beltran, M., & Guzmán, A. (2008). LDAP injection techniques. In *Communication Systems, 2008. ICCS 2008. 11th IEEE Singapore International Conference on* (pp. 980-986). IEEE.
211. A. Roichman, E. Gudes. (2008). "DIWeDa - Detecting Intrusions in Web Databases". In: Atluri, V. (ed.) DAS 2008. LNCS, vol. 5094, pp. 313–329. Springer, Heidelberg.
212. Owasp.org, (2016). *Top 10 2013-Top 10 - OWASP*. [online] Available at: https://www.owasp.org/index.php/Top_10_2013-Top_10 [Accessed 1 Jan. 2016].
213. Klein, A. (2005). Blind XPath Injection. *Whitepaper from Watchfire*.
214. Mitropoulos, D., Karakoidas, V., Spinellis, D. (2009). Fortifying Applications Against Xpath Injection Attacks. In *MCIS* (p. 95).
215. Shanmughaneethi, V., Ravichandran, R., & Swamynathan, S. (2011). PXpathV: Preventing XPath Injection Vulnerabilities in Web Applications. *International Journal on Web Service Computing*, *2*(3), 57.
216. Aspect Security, Inc, (2014). *State of Developer Application Security Knowledge*.
217. Gupta, A. N., Thilagam, D. P. S. (2013). Attacks on Web Services Need to Secure XML on Web. *Computer Science & Engineering*, *3*(5), 1.
218. Ron, A., Shulman-Peleg, A., Bronshtein, E. (2015). No SQL, No Injection? Examining NoSQL Security. *arXiv preprint arXiv:1506.04082*.
219. Akanji, A. W., Elusoji, A. A. (2014). A Comparative Study of Attacks on Databases and Database Security Techniques.
220. Sahafizadeh, E., & Nematbakhsh, M. A. (2015). A Survey on Security Issues in Big Data and NoSQL. *Advances in Computer Science: an International Journal*, *4*(4), 68-72.
221. Mohamed, M. A., Altrafi, O. G., & Ismail, M. O. (2014). Relational vs. NoSQL Databases: A Survey. *International Journal of Computer and Information Technology (ISSN: 2279–0764) Volume*.
222. Patel, K., Sharma, K., Hasan, M. (2015). Encrypting MongoDB Data using Application Level Interface. *History*, *46*(214), 164-169.
223. V. A. Diaz. (2006). MX Injection - Capturing and Exploiting Hidden Mail Servers, Internet Security Auditors.

224. Ietf.org, (2016). [online] Available at: http://www.ietf.org/rfc/rfc3501.txt [Accessed 1 Jan. 2016].

225. Captcha.net, (2016). *The Official CAPTCHA Site*. [online] Available at: http://www.captcha.net/ [Accessed 1 Jan. 2016].

226. Webappsec.org, (2016). *The Web Application Security Consortium / OS Commanding*. [online] Available at: http://www.webappsec.org/projects/threat/classes/os_commanding.shtml [Accessed 1 Jan. 2016].

227. SearchSoftwareQuality, (2016). *What is OS commanding? - Definition from WhatIs.com*. [online] Available at: http://searchsoftwarequality.techtarget.com/definition/OS-commanding [Accessed 1 Jan. 2016].

228. Jourdan, G. V. (2009). Command Injections.

229. Vieira, M., Antunes, N., & Madeira, H. (2009). Using web security scanners to detect vulnerabilities in web services. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on* (pp. 566-571). IEEE.

230. Su, Z., & Wassermann, G. (2006). The essence of command injection attacks in web applications. In *ACM SIGPLAN Notices* (Vol. 41, No. 1, pp. 372-382). ACM.

231. Acunetix, (2016). *Web Application Security with Acunetix Web Vulnerability Scanner*. [online] Available at: http://www.acunetix.com/vulnerability-scanner [Accessed 1 Jan. 2016].

232. Www-01.ibm.com, (2016). *IBM - Software - IBM Security AppScan*. [online] Available at: http://www-01.ibm.com/software/awdtools/appscan/ [Accessed 1 Jan. 2016].

233. Cwe.mitre.org, (2016). *CWE - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (2.9)*. [online] Available at: http://cwe.mitre.org/data/definitions/78.html [Accessed 1 Jan. 2016].

234. Capec.mitre.org, (2016). *CAPEC - CAPEC-88: OS Command Injection (Version 2.8)* . [online] Available at: http://capec.mitre.org/data/definitions/88.html [Accessed 1 Jan. 2016].

235. Kadebu, P., & Mapanga, I. (2014). A Security Requirements Perspective towards a Secured NOSQL Database Environment.

236. Mitropoulos, D., & Spinellis, D. (2009). SDriver: Location-specific signatures prevent SQL injection attacks. *computers & security*, *28*(3), 121-129.

237. Sqlmap.org, (2016). *sqlmap: automatic SQL injection and database takeover tool*. [online] Available at: http://sqlmap.org/ [Accessed 1 Jan. 2016].

238. Strauch, C., Sites, U. L. S., & Kriha, W. (2011). NoSQL databases. *Lecture Notes, Stuttgart Media University*.

239. Httpd.apache.org, (2016). *URL Rewriting Guide - Apache HTTP Server*. [online] Available at: http://httpd.apache.org/docs/2.0/misc/rewriteguide.html [Accessed 1 Jan. 2016].

240. Msdn.microsoft.com, (2016). *urlMappings Element (ASP.NET Settings Schema)*. [online] Available at: https://msdn.microsoft.com/en-us/library/ms228302%28v=vs.85%29.aspx [Accessed 1 Jan. 2016].

241. Docs.oracle.com, (2016). *Servlet Authentication Filters*. [online] Available at: http://docs.oracle.com/cd/E12839_01/web.1111/e13718/servlet.htm#DEVSP510 [Accessed 1 Jan. 2016].

242. Sun, S. T., Wei, T. H., Liu, S., & Lau, S. (2007). Classification of sql injection attacks. *University of British Columbia, Term Project*.

243. Owasp.org, (2016). *Category:Vulnerability Scanning Tools - OWASP*. [online] Available at: https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools [Accessed 1 Jan. 2016].

244. Ws-attacks.org, (2016). *XML Injection - WS-Attacks.org*. [online] Available at: http://www.ws-attacks.org/index.php/XML_Injection [Accessed 1 Jan. 2016].

245. Cwe.mitre.org, (2016). *CWE - CWE-91: XML Injection (aka Blind XPath Injection) (2.9)*. [online] Available at: https://cwe.mitre.org/data/definitions/91.html [Accessed 1 Jan. 2016].

246. Capec.mitre.org, (2016). *CAPEC - CAPEC-250: XML Injection (Version 2.8)* . [online] Available at: https://capec.mitre.org/data/definitions/250.html [Accessed 1 Jan. 2016].

247. Xu, H., Reddyreddy, A., Fitch, D. F. (2011). Defending Against XML-Based Attacks Using State-Based XML Firewall. *Journal of Computers*, *6*(11), 2395-2407.

248. Kindy, D. A., Pathan, A. S. K. (2011). A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques.

249. Shakya, A., Aryal, D. (2011). *A Taxonomy of SQL injection defense techniques* (Doctoral dissertation, MS thesis, School of Computing, Blekinge Institute of Technology, Karlskrona, Sweden).

250. Chen, J., Guo, C. (2006). Online detection and prevention of phishing attacks. In *Communications and Networking in China, 2006. ChinaCom'06. First International Conference on* (pp. 1-7). IEEE.
251. Huang, H., Zhong, S., Tan, J. (2009). Browser-side countermeasures for deceptive phishing attack. In *Information Assurance and Security, 2009. IAS'09. Fifth International Conference on* (Vol. 1, pp. 352-355). IEEE.
252. Kaspersky, (2014). *Financial cyberthreats, 2014*. Lab Report.
253. Honda, S., Unno, Y., Maruhashi, K., Takenaka, M., Torii, S. (2014). Detection of Novel-Type Brute Force Attacks Used Ephemeral Springboard IPs as Camouflage. *Journal of Advances in Computer Networks*, *2*(4).
254. Owasp.org, (2016). *Session hijacking attack - OWASP*. [online] Available at: https://www.owasp.org/index.php/Session_hijacking_attack [Accessed 1 Jan. 2016].
255. Towhidi, F., Manaf, A. A., Daud, S. M., Lashkari, A. H. (2011). The Knowledge Based Authentication Attacks. In *World Congress in Computer Science*.
256. Owasp.org, (2016). *Man-in-the-middle attack - OWASP*. [online] Available at: https://www.owasp.org/index.php/Man-in-the-middle_attack [Accessed 1 Jan. 2016].
257. Spangler, R. (2003). Packet sniffer detection with antisniff. *University of Wisconsin, Whitewater. Department of Computer and Network Administration*.
258. De Cristofaro, E., Du, H., Freudiger, J., Norcie, G. (2013). A comparative usability study of two-factor authentication. *arXiv preprint arXiv:1309.5344*.
259. Static.usenix.org, (2016). *14th USENIX Security Symposium — Technical Paper*. [online] Available at: http://static.usenix.org/event/sec05/tech/full_papers/yin/yin_html/ [Accessed 1 Jan. 2016].
260. InfoSec Resources, (2012). *Phishing: A Very Dangerous Cyber Threat - InfoSec Resources*. [online] Available at: http://resources.infosecinstitute.com/phishing-dangerous-cyber-threat/ [Accessed 1 Jan. 2016].
261. Vykopal, J., Dražar, M., Winter, P. (2013). Flow-based Brute-force Attack Detection.
262. Rehak, M., Pěchouček, M., Bartoš, K., Grill, M., Čeleda, P., Krmíček, V. (2008). CAMNEP: An intrusion detection system for high-speed networks.
263. Denyhosts.sourceforge.net, (2016). *Welcome to DenyHosts*. [online] Available at: http://denyhosts.sourceforge.net/ [Accessed 1 Jan. 2016].

264. Logwatch, (2016). *Logwatch - Browse Files at SourceForge.net*. [online] Sourceforge.net. Available at: http://sourceforge.net/projects/logwatch/files/ [Accessed 1 Jan. 2016].

265. Snort.org, (2016). *Snort.Org*. [online] Available at: https://www.snort.org/ [Accessed 1 Jan. 2016].

266. Owasp.org, (2016). *Session fixation - OWASP*. [online] Available at: https://www.owasp.org/index.php/Session_fixation [Accessed 1 Jan. 2016].

267. Owasp.org, (2016). *Session Fixation Protection - OWASP*. [online] Available at: https://www.owasp.org/index.php/Session_Fixation_Protection [Accessed 1 Jan. 2016].

268. De Ryck, P., Nikiforakis, N., Desmet, L., Piessens, F., Joosen, W. (2012). Serene: Self-reliant client-side protection against session fixation. In *Distributed Applications and Interoperable Systems* (pp. 59-72). Springer Berlin Heidelberg.

269. Kolšek, M. (2002). Session fixation vulnerability in web-based applications. *Acros Security*, 7.

270. Stallings, W. (1999). *Cryptography and network security*. Upper Saddle River, N.J.: Prentice Hall.

271. Johns, M., Braun, B., Schrank, M., Posegga, J. (2011). Reliable protection against session fixation attacks. In *Proceedings of the 2011 ACM Symposium on Applied Computing* (pp. 1531-1537). ACM.

272. Pinkas, B., Sander, T. (2002). Securing passwords against dictionary attacks. In *Proceedings of the 9th ACM conference on Computer and communications security* (pp. 161-170). ACM.

273. Sullivan, B. (2007). Preventing a Brute Force or Dictionary Attack: How to Keep the Brutes Away from your Loot. *AvailableOnline: http://h71028. www7. hp. com/ERC/cache/568358-0-0-0-121. html/(accessedon21February2010)*.

274. Goyal, V., Kumar, V., Singh, M., Abraham, A., Sanyal, S. (2006). A new protocol to counter online dictionary attacks. *computers & security*, *25*(2), 114-120.

275. Thorpe, J. (2008). *On the predictability and security of user choice in passwords* (Doctoral dissertation, CARLETON UNIVERSITY Ottawa).

276. Ramasamy, R., Muniyandi, A. P. (2012). An Efficient Password Authentication Scheme for Smart Card. *IJ Network Security*, *14*(3), 180-186.

277. De Ryck, P., Desmet, L., Piessens, F., Joosen, W. (2013). Improving the security of session management in web applications. *status: published*, 0-0.

278. Earthlink.net, (2016). *EarthLink® Software & Tools - Do More*. [online] Available at: http://www.earthlink.net/software/free/toolbar/ [Accessed 1 Jan. 2016].
279. Toolbar.netcraft.com, (2016). *Netcraft Extension - Phishing Protection and Site Reports*. [online] Available at: http://toolbar.netcraft.com/ [Accessed 1 Jan. 2016].
280. PhishGuard, (2016). *Protect Against Internet Phishing Scams*. [online] Available at: http://www.phishguard.com/ [Accessed 1 Jan. 2016].
281. Huang, C.-Y., S. PinMaa, and K. TaChen. (2010). Using onetime passwords to prevent password phishing attacks, *in Journal of Network and Computer Applications*.
282. Wiki.mozilla.org, (2016). *Phishing Protection: Design Documentation - MozillaWiki*. [online] Available at: https://wiki.mozilla.org/Phishing_Protection:_Design_Documentation [Accessed 1 Jan. 2016].
283. Siteadvisor.com, (2016). [online] Available at: http://www.siteadvisor.com [Accessed 1 Jan. 2016].
284. Chou, N., Ledesma, R., Teraguchi, Y., Mitchell, J. C. (2004). Client-Side Defense Against Web-Based Identity Theft. In *NDSS*.
285. Geotrust.com, (2016). *GeoTrust TrustWatch and Comcast: Anti-phishing Technologies*. [online] Available at: https://www.geotrust.com/comcasttoolbar/ [Accessed 1 Jan. 2016].
286. Yee, K. P., Sitaker, K. (2006). Passpet: convenient password management and phishing protection. In *Proceedings of the second symposium on Usable privacy and security* (pp. 32-43). ACM.
287. Gabber, E., Gibbons, P. B., Kristol, D. M., Matias, Y., Mayer, A. (1999). Consistent, yet anonymous, Web access with LPWA. *Communications of the ACM*, *42*(2), 42-47.
288. Ross, B., Jackson, C., Miyake, N., Boneh, D., Mitchell, J. C. (2005). A browser plug-in solution to the unique password problem. In *Proceedings of the 14th Usenix Security Symposium*.
289. Booth, P. (2004). *America Online, RSA Security launch AOL PassCode premium service*. [online] ITWeb Technology News. Available at: http://www.itweb.co.za/index.php?option=com_content&view=article&id=17713 [Accessed 1 Jan. 2016].
290. Microsoft, (2016). *Sender ID*. [online] Available at: http://www.microsoft.com/mscorp/safety/technologies/senderid/default.mspx [Accessed 1 Jan. 2016].

291. Dhamija, R., Tygar, J. D. (2005). The battle against phishing: Dynamic security skins. In *Proceedings of the 2005 symposium on Usable privacy and security* (pp. 77-88). ACM.

292. Herzberg, A., Gbara, A. (2004). *Trustbar: Protecting (even naive) web users from spoofing and phishing attacks*. Cryptology ePrint Archive, Report 2004/155.

293. Basnet, R., Mukkamala, S., & Sung, A. H. (2008). Detection of phishing attacks: A machine learning approach. In *Soft Computing Applications in Industry* (pp. 373-383). Springer Berlin Heidelberg.

294. Nikiforakis, N., Meert, W., Younan, Y., Johns, M., Joosen, W. (2011). SessionShield: Lightweight protection against session hijacking. In *Engineering Secure Software and Systems* (pp. 87-100). Springer Berlin Heidelberg.

295. Dacosta, I., Chakradeo, S., Ahamad, M., Traynor, P. (2012). One-time cookies: Preventing session hijacking attacks with stateless authentication tokens. *ACM Transactions on Internet Technology (TOIT), 12*(1), 1.

296. Bella, G., Paulson, L. C. (1998). Kerberos version IV: Inductive analysis of the secrecy goals. In *Computer Security—ESORICS 98* (pp. 361-375). Springer Berlin Heidelberg.

297. Obimbo, C., Ferriman, B. (2011). Vulnerabilities of LDAP as an Authentication Service. *Journal of Information Security*, *2*(04), 151.

298. Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G. (2006). Polymorphic worm detection using structural information of executables. In *Recent Advances in Intrusion Detection* (pp. 207-226). Springer Berlin Heidelberg.


ATHENS UNIVERSITY OF ECONOMICS & BUSINESS
ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ