Analytical Study of Common Web Application Attacks

Junaid Latief Shah
Dept of Computer Science, University of Kashmir

Abstract— ***In the recent times of technological development, web applications have become a common platform for a company's e-governance and administration software, a public or social forum, web portal, an e-commerce application or any other applications running on the web. The world community is quickly embracing an era of social networking with number of social networking sites coming into existence like Orkut, Facebook, MySpace, and Twitter etc. These social networking sites are used by millions of users all around the world. Technology has also made banking facilities much easier by providing secure e-banking solutions and online money transfers. The web is witnessed by thousands of bank transactions each day. Everything in the present world has migrated to the web ranging from online reservations and bookings to online shopping portals. Thus web applications are a buzz among the users today. Since number and impact of these applications running on the web have increased rapidly over the past years, at the same time, it gave birth to various web security vulnerabilities resulting in an undesirable side to web usage. The web has thus become a preferred platform for malicious users like hackers and spammers to expose these vulnerabilities and gain access or tamper with these applications. Attacks like injection vulnerabilities such as XSS, CSRF and SQL injections are becoming common. Sometimes intent of attackers is to impersonate a real user by stealing his cookies and hijacking his sessions over the web. Attackers may also make fraud banking transactions and make money transfers illegally. Phishing is one of the most common attacks in this category. Sometimes an attacker uses DOS and DDOS attacks to slow down a web server and thus make application to slow down and unavailable to user. To overcome this, a number of methods and techniques have been proposed by researchers over the period of time but still fail to completely secure a web application. Web application developers today find it difficult to completely secure their applications from malicious content received over the web. The focus of this research paper is to compare and analyze the common web attacks, study how they impact applications over the web and how their effects can be possibly mitigated.***

Index Terms— **XSS, DOS, DDOS, Phishing, SQL injection, CSRF, Web Server, Session, Cookies.**

I. Introduction

Security has been the critically important part of majority of web applications. The web applications access the web server which in turn accesses the database servers. Thus proper security has to be implemented at every step during the access mechanism. Analysis carried out by Common Vulnerabilities and Exposures (CVE) [1] reports that majority of today's security loop holes are found in web applications. Commonly known attacks include Cross Site Scripting attack, SQL

injection attack, path traversals etc, whose main aim is to tamper or deface web applications or impersonate as a real legitimate user. Web applications provide users with client server functionality by accessing a series of web pages. These web pages often contain dynamic interactive web content and script code which gets executed in the user browser. Thus web applications are continuously subjected to attacks [2][3][4] such as cross-site scripting, cookie stealing, session hijacking, browser hijacking, and the most recent being self-propagating worms in Web-email and social networking sites. In fact most of the research conducted shows that web application attacks are the most common problems on the internet today.[5]

In many scenarios security remains the major drawback for the global acceptance of web for every day to day activity and transactions. According to Symantec [6], only in 2002, there was about 85% increase in web attacks with majority of them being very severe. Also the reported attacks in 2002 was about 150% more than that was reported in 2001.The major reason behind this is popularity of web and migration of almost every application to web platform. Web applications today are the most important communications mediums between service providers and clients.

II. Popular attacks

A. Cross Site Scripting Attack

Cross Site Scripting (XSS) vulnerabilities have been the nightmare for Web applications for years now. Recent studies have shown that XSS has become the most common security problem. An analysis of the WASC [7] reveals that 100,059 XSS vulnerabilities have been detected by analyzing 31,373 Web sites. *Cross Site Scripting* (XSS) vulnerabilities penetrate web applications by injecting client side script into web pages viewed by other users. Majority of the websites including Face book, Twitter, McAfee, MySpace, eBay and Google have been the targets of XSS exploits. XSS occurs because of various limitations of security existing in many Web applications .i.e. when user inputs are not properly sanitized. The code to execute XSS is written in popular languages like PHP, Java,.NET. Attackers inject malicious code through these inputs, thereby causing unintended script executions through clients‟ browsers. Although a number of solutions have been proposed by researchers over time ranging from static analysis to complex runtime protection

mechanisms, the data collected by semantic as of 2007 reveal that 80.5% of all security vulnerabilities are XSS.
Let's demonstrate XSS with a simple example. Assume there's a public forum where people can ask Questions regarding computer science. Each question is stored in a database and rendered as a list, if someone requests the relevant section of the forum. Such a list might look like this

(No XSS embedded here):

Sample forum listing:

```
<html>
<head>
<title>The Question and Answer example forum – Computer
Science section</title>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
List of questions:
<p>Q: "Which is the best <i> OOP language </i> in current
times
</p>
<p>Q: "What are the attributes of RDMS?"</p>
</body>
</html>
```

When a hacker visits this page he will immediately notice that the text OOP language is rendered italic in his browser and conclude that the user that posted the question added the corresponding tags himself. Now the hacker might post a "question" in a different way like this:

<Script>alert ('you have been XSSed') ;< /script>

Forum listing with embedded XSS attack:

```
<html>
<head>
<title>The Question and Answer example forum – Computer
Science section</title>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
List of questions:
<p>Q: "Which is the best <i> OOP language </i> in current
times
</p>
<p>Q: "What are the attributes of RDMS?"</p>
<p>Q: "<Script>alert ('you have been XSSed') ;
< /script>"</p>
</body>
</html>
```

Now, every time a user requests this list, a pop-up will be generated and appear in that user's browser that displays the words "you have been XSSed". While only some clever users will actually consider this an attack, other will surely not pay any heed and consider it as a normal pop up. By this way of injecting malicious scripts into web pages, an attacker can gain high access-privileges to sensitive page content, cookies, and a variety of other information maintained by the browser on behalf for user, making cross-site scripting attacks a unique case of code injection [8].

Types of XSS Attacks:

XSS attacks are mainly categorized into three types:

1. Persistent or Stored XSS
2. Non Persistent or Reflected XSS
3. DOM based XSS

Persistent or Stored Attack:

Stored XSS works if an HTML page includes data stored on the Web server (e.g. from a database) that originally comes from user supplied data. All an attacker has to do is to find a vulnerable server and post an attack. From that moment on, the server will distribute the exploit automatically to all users requesting the vulnerable page. Persistent or stored XSS attack is called persistent because it gets stored somewhere on the server and the effect of the attack is not immediate.
An example of this type of attack is when someone writes a HTML formatted review or comments on a review board like social networking websites or forum for other users to read. When some user reads the review the code gets executed on the user's browser and does some unwanted stuff like stealing cookies, redirect to some other page including website defacement etc.
For example the code in the comment or review can be like this

```
<b>Thank for your review <script>
window.location.href="http://www.abc.com"</script></b>
```

The above message will be stored in the database as it is and when some future user visits the page, the comment will be displayed but immediately the code in the script tag will be executed and the victim will be redirected to "*abc.com*".

Non-Persistent or Reflected XSS:

The second type (*reflected* XSS) works because some part of an HTTP request (usually a URL
Parameter, cookie or the referrer location) is reflected *by the Web server* into the HTML content that is returned to the requesting browser. The word "Reflected" here means that input is written back unaltered. In this case, a hacker would

have to craft a malicious URL and make someone else follow/open that link:

http://www.example.com/mypage.asp?id=<script>doBadThings () ;< /script>

This can be done by sending someone a manipulated e-mail (with the link) and use Phishing techniques to make the receiver believe that clicking on the link is a good thing. A second Approach would be to post such a link somewhere on the Internet, e.g. in a blog, forum, and wait for someone to follow it.

DOM based XSS:

The third type (*DOM-based* XSS) is very similar to the reflected attack. The difference is that the attack code isn't embedded into the HTML content back sent by the server. Therefore all server-side XSS detection techniques fail. Instead, it is embedded in the URL of the requested page and executed in the user's browser by faulty script code, contained in the HTML content returned by the server. Faulty means that the script reads a URL parameter and dynamically adds it to the *document object model* without any validation: document. Write (document.location.href);
This way, unwanted tags are added to the DOM *locally* at runtime and are subsequently executed.

B. *Cross Site Request Forgery*

CSRF is a type of injection attack which somehow makes an end user to execute unwanted/undesired actions on a web application in which he/she is having a session and currently authenticated. With some techniques of social engineering like sending a link via an email or online chat, an attacker may force the users of a web application to execute actions of the attacker's choosing [9][10]. A successful CSRF attack can compromise end user data and operation in case of normal user. If the end user has administrative privileges, this can compromise the entire web application.

Cross-Site Request Forgery (CSRF) fools the victim into loading a page that contains a malicious request. The term “malicious” is used in the sense that it uses the identity and authenticated privileges of the victim to perform an undesired functionality on the victim's behalf, like change the victim's home address, or password, e-mail address, making online purchases. In other words, an attacker can easily use victim’s session identity. CSRF attacks generally cause a state change on the server but can also be used to access sensitive data.

For most of the sites, browsers automatically store credentials associated with the site, such as the user's cookie information, basic authentication credentials, an IP address, Windows credentials, etc. Therefore, if the user is currently authenticated to that site, the website will have no way to distinguish this from a legitimate user request.

Using this way, the attacker can make his victim perform actions that he didn't intend to do, such as logout, purchase item, change account information, retrieve account information, or any other function provided by the vulnerable website.

Sometimes, it is feasible to store the CSRF attack (like a social forum website) on the vulnerable site. Such vulnerabilities are sometimes called Stored CSRF flaws (like stored XSS). This may be accomplished by simply storing an IMG or IFRAME tag in a field that accepts HTML, or by a more complicated cross-site scripting attack. Suppose if the attack can store a CSRF attack in the website, the severity of the attack is amplified. Specifically, the likelihood is increased; as a result the victim is more likely to view the page containing the attack than some random page on the net. The likelihood is also increased because the victim is sure to be authenticated to the site already.Known by the number of other names including Session Riding, "Sea Surf", XSRF, Cross-Site Reference Forgery, and Hostile Linking, CSRF attacks are one of the dangerous web attacks today. Microsoft quotes this type of attack as a One-Click attack in their threat modeling process and many places in their online documentation.

There are a number of ways by which a naive-user can be tricked into loading information from or submitting information to a web application. In order to carry out an attack, we must first learn how to generate a malicious request for our victim to execute. Let us consider the following example: Robert wishes to transfer $1000 to Stevens using xyz.com. The request generated by Robert will look similar to the following:

```
POST-http://xyz.com/transfer.doHTTP/1.1
...
...
..
Content-Length: 19;
acct=Stevens&amount=1000
```

However, Sandra notices that the same web application will execute the same transfer using URL parameters as follows:

*GET h*ttp://xyz.com/transfer.do?acct=Stevens&amount=1000 *HTTP/1.1*

Now that her malicious request is generated, Sandra must trick Robert into submitting the request. The most basic method is to send Robert an HTML email containing the following:

<a href="http://xyz.com/transfer.do?acct=Sandra&amount=100 0000">View my Pictures!</a>

Assuming Robert is authenticated with the application when he clicks the link, the transfer of $100,0000 to Sandra's account will occur. However, Sandra realizes that if Robert clicks the link, then Robert will notice that a transfer has occurred. Therefore, Sandra decides to hide the attack in a zero-byte image:

<img src="http://xyz.com/transfer.do?acct=Sandra&amount =1000000" width="1" height="1" border="0">

If this image tag were included in the email, Robert would only see a little box indicating that the browser could not display the image. However, the browser *will still* submit the request to xyz.com without any visual indication that the transfer has taken place.

C. SQL Injection Attacks

SQL Injection is one of the deadly web attacks used by hackers to steal vital data from big organizations. It is perhaps one of the most common application layer attack techniques that target databases. This type of attack that takes advantage of improper coding of your web applications that allows hacker to inject malicious SQL commands into a web form to allow them to gain access to the data contained within your database. The "injection" term commonly refers to the insertion of a SQL query via the input data from the client to the application. If the injection is successful, an attacker can read sensitive data from the database, modify database data (DML commands, execute administration operations on the database (such as shutdown the DBMS), recovery of the contents of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks have been around for years now. According to the Web Application Security Consortium (WASC) 9% of the total hacking incidents reported in the media until 27th July 2006 were due to SQL Injection [11]. Recent data from acunetix research shows that about 50% of the websites scanned every year are susceptible to SQL Injection vulnerabilities. In 2005, Socrates [12], the math-addicted hacker used a SQL hack to break into a Michigan uniform company's website to steal 3,000 customer profiles. In 2008[12], a hacker used an efficient mechanism of executing SQL attack. Instead of trying to steal data from one website at a time, he used botnets to probe the Internet for WebPages whose databases could be easily injected with a small bit of code. Acunetix [13] reports that in 2009, SQL injection was used by three men and stole around 130 million credit and debit card numbers from five leading companies, thus being the largest data security breach in U.S till date. The main reason for the occurrence of SQL Injection attack is when the programmers who write the code behind the page neglect to properly escape strings that are used in SQL queries. Consider the following example. In SQL if we write the following query

*Select * from users where userid = ' ' and password = ' ';*

If one provides userid as juna'id and password as research, the string query becomes

*Select * from users where userid = 'juna'id' and password = 'research';*

Which is executed by database as

Incorrect syntax near id' as database tries to execute juna

The reason why the error occurred is that escape character i.e. "single quote" was not filtered out by the application developer and thus making it vulnerable to SQL injection. Consider the following query

*Select * from users where userid = ' ' and password = ' ';*

If an attacker with the user id 'junaid' enters the string "research' OR '1'='1" for password, then the query becomes the following:

*Select * from users where userid = ' ' and password = ' ' OR '1'='1';*

The addition of the OR '1'='1' condition causes the where clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

*Select * from users;*

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner. The query in the above example can also be manipulated by appending another query to it. For Example, if the attacker with the user id junaid enters the string "junaid'); DELETE FROM users; --" for userid, then the query becomes the following two queries:

*Select * from users where userid = ' ';*
Delete from users; -- '

Many database servers, including Microsoft® SQL Server 2000 above, allow multiple SQL statements separated by semicolons to be executed at once. This type of attack allows the attacker to execute arbitrary commands against the database. The trailing pair of hyphens (--), signifies to most database servers that the remainder of the statement is to be treated as a comment and not executed. The main consequences of SQL injection attack [9] is loss of confidentiality, tampering of Authentication and Authorization and compromise of integrity of database.

D. Blind SQL Injection

Blind SQL injection is one of the variants of SQL injection [9] that aims to ask database true or false questions and determines the answer based on what application responses. The technique used by Blind SQL injection is to insert several SQL sentences in one place. The information thus obtained is analyzed and then attacker modifies the clauses in the SQL to keep getting additional information. This attack is usually used once the web application is organized to point out generic error messages, however has not mitigated the code that's at risk of SQL injection.

When a hacker exploits SQL injection, often the web application displays error messages from the database reporting that the SQL Query's syntax is incorrect. Blind SQL injection is sort of a twin of traditional SQL Injection, the sole distinction being the manner by which the information is retrieved from the database. When the database doesn’t output data to the web page, an attacker is forced to steal data by asking the database a series of true or false queries. This makes exploiting the SQL Injection vulnerability harder, but not impossible.

Using a simple web page, which displays an article with given ID as the parameter, the attacker may perform a couple of simple tests to determine if the page is vulnerable to SQL Injection attacks.

Example URL:

http://newspaper.com/items.php?id=2

Sends the below query to the database:

SELECT title, description, body FROM items WHERE ID = 2

The attacker may then try to execute a query that returns 'false':

http://newspaper.com/items.php?id=2 and 1=2

The following SQL query is generated:

SELECT title, description, body FROM items WHERE ID = 2 and 1=2

If the web application is vulnerable to SQL Injection, then it will not return anything. To confirm, the attacker will execute a query that will return 'true':

http://newspaper.com/items.php?id=2 and 1=1

If the content of the page that returns 'true' is totally different than that of the page that returns 'false', then the attacker is able to distinguish when the executed query returns true or false.

Once this has been verified, the sole limitations are privileges set up by the database admin, different SQL syntax, and the attacker's imagination.

E. Phishing

Phishing is the act of attempting to acquire information such as usernames, passwords, and credit card details (and sometimes, indirectly, money) by impersonating as a trustworthy entity in an electronic communication [14][15].The purpose of phishing emails, websites and phone calls is to steal money and sensitive information that requires user credentials. Messages and communications that are faked to come from popular social websites, banks, and online payment portals are the commonly used method used by attacker to lure its victim. The phishing emails mostly contain links to websites that are infected with spam and malware. E-mail spoofing and instant messaging are the common methods of carrying out phishing and it often directs users to enter personal details at a fake website by making the victim click over a link. The fake website has the look and feel almost identical to the legitimate one. Phishing is an example of what is known as social engineering technique to deceive the users.

III. MITIGATION SCENARIO AND RELATED WORK

Analysis of majority of web application attacks reveal that they are caused due to improper coding of web applications and inability to filter or sanitize input coming into web. The attacks such as XSS, CSRF and injection attacks occur due to non sanitization of user input. The majority of these web application attacks are mitigated either on the client side or server side. The client side mitigation normally involves input validation techniques. These techniques normally restrict a user from supplying malicious data to the web page. On the other hand, server side mitigation involves filtering the user input or output sanitation. Analyzing xss attacks, there
are number of client side solutions implemented by the developers all over the world.

Hallaraker et al. [21] proposed a client-side mechanism for detecting malicious JavaScript's in Mozilla Firefox Web Browser. It uses an auditing system that can perform vulnerability or misuse detection. The system checks the execution of JavaScript and compares it to high level policies to detect malicious or abnormal activity. For each specific scenario, rules have to be implemented to enable detection. These rules allow specifying sequences of JavaScript methods, together with their corresponding, that are considered malicious parameters. With this information, state driven rules can be implemented. The system performs most of the auditing in XP Connect, a layer that connects the JavaScript engine with the other components of Mozilla Firefox. The various experiments show that this solution is insufficient because if new vulnerabilities should be detected, new rules have to be implemented and the browser has to be rebuilt.

Some researchers [16-20] have proposed the use of static analysis techniques to detect input validation flaws in a web application; however, this approach requires access to the source code of the application [16, 17]. Moreover, those static analysis schemes are usually complemented by the use of dynamic analysis techniques. Huanget al [18], Balzarotti et al [22] used these techniques to confirm potential vulnerabilities detected during the static analysis by watching the behavior of the various running applications at runtime.

Kirda et al [23] developed Noxes which is a client-side Web-proxy that relays all Web traffic and serves as an application-level firewall. The main problem of Noxes it's requires user-specific configuration (firewall rules), as well as user interaction when a suspicious event occurs.

A client-side approach suggested by Vogt et al [24] , which monitors information leakage using tainting of input data in the browser. The solution presented in this paper stops XSS attacks on the client side by tracking the flow of sensitive information inside the web browser. If sensitive information is about to be transferred to a third party, the user can decide if this should be permitted or not. As a result, the user has an additional protection layer when surfing the web, without solely depending on the security of the web application.

All client-side solutions share one drawback. All software updates have to be installed separately on each workstation. This pre-condition is viewed as a drawback and does not offer a good viable solution.

Many of the vulnerabilities exploited by the attacks outlined in this paper need to be fixed by the developers of the web applications, which aren't under the direct control of the victims. However browser users can take some mitigation measures. Modern web browsers are starting to include mechanisms for resisting attacks outlined above. For instance, Internet Explorer 8 includes anti-XSS, anti-CSRF and anti-click jacking features. Google Chrome offers capabilities of this manner as well. Users of Firefox can mitigate the risks of CSRF, XSS and other attack on web applications by using the No Script extension [25].

Attacks that target the web browser's human element—the user—by using social engineering can be very effective. The developers of web browsers are placing greater emphasis on this attack vector. They do this by providing better guidance to the user regarding the risky actions taken by the website.NSS Labs recently tested browsers' ability to protect users from socially-engineered malware. According to NSS Labs, Internet Explorer surpassed the effectiveness of other browsers to protect the user from this attack vector.
Internet Explorer achieved such performance as the result of its improved Smart Screen feature, which incorporates application reputation capabilities starting with Internet Explorer 9. This aspect of Smart Screen maintains reputation data about known bad and known good executables. It also warns the user when the executable that he or she is attempting to run doesn't have reputation data.

IV. Conclusion

This paper carried out analysis of various web application attacks and classified those attacks. The information contained in this paper could be very useful for new application/web developers for developing smarter and secure applications running over the web. The paper also lists some of the related work and mitigation scenarios. Although a complete secure application is not guaranteed in the modern world, but still a considerable amount of work and research has been done in this area. Completely securing a web application seems to be a daunting task for developers today.

REFERENCES

[1]Sandeep Bhatkar, Abhishek Chaturvedi, and R. Sekar.Dataflow anomaly detection. In IEEE Symposium on Security and Privacy, May 2006
[2]E. Chien. Malicious Yahooligans. http://www.symantec.com/avcenter/reference/malicious. yahooligans.pdf, 2006.
[3]Open Web Application Security Project. The ten most critical Web application security vulnerabilities http://umn.dl.sourceforge.net /sourceforge/owasp/ OWASPTopTen2004.pdf,2004
[4]The Samy worm. http://namb.la/popular
[5]MITRE. Common vulnerabilities and exposures. http://cve.mitre.org/cve/, 2007
[6]Higgins, M., Ahmad, D., Arnold, C. L., Dunphy, B., Prosser, M.,and Weafer, V., "Symantec Internet Security Threat Report—Attack Trends for Q3 and Q4 2002," Symantec, Feb 2003.
[7]Web Application Security Statistics,06(WASC) http://www.webappsec.org/projects/statistics/
[8]Xie and A. Aiken, "Static Detection of Security Vulner-abilities in ScriptingLanguages,Proc. 15th Use nix Security Symp. (Use nix-SS 06), vol. 15, Use nix, 2006, pp.179-192.
[9]https://www.owasp.org/index.php
[10]https://www.isecpartners.com/media/11961/CSRF_Pap
e
r.pdf
[11]http://www.acunetix.com/websitesecurity/sql-injection/
[12]http://lastwatchdog.com/faq-sql-injection-attacks/
[13]http://www.acunetix.com/blog/news/sql-injection-used-
i
n-largest-data-security-breach-in-u-s-history/
[14]Ramzan, Zulfikar (2010). "Phishing attacks and countermeasures". In Stamp, Mark & Stavroulakis,

Peter.*Handbook of Information and Communication Security*. Springer. ISBN 9783642041174.

[15]Van der Merwe, A J, Loock, M, Dabrowski, M. (2005), Characteristics andResponsibilities involved in a Phishing Attack, Winter International Symposium on Information and CommunicationTechnologies ,CapeTown,January2005

[16]G. Wassermann and Z. Su. “Static detection of cross-site scripting vulnerabilities”In Proceedings of the 30th international conference on Software engineering, pages 171–180. ACM New York, NY, USA, 2008.

[17]N. Jovanovic, C. Kruegel, and E. Kirda. “Pixy: A static analysis tool for detecting web application vulnerabilities”.In IEEE Symposium on Security and Privacy, page 6,2006.

[18]Y. Huang, F. Yu, C. Hang, C. Tsai, D. Lee, and S. Kuo. “Securing web application code by static analysis and runtime protection”. In Proceedings of the 13th international conference on World Wide Web, pages 40–52. ACM New York, NY, USA, 2004

[19]P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. “ Crosssite scripting prevention with dynamic data tainting and static analysis”. In Proceeding of the Network and Distributed System Security Symposium (NDSS07), 2007.

[20]D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G.Vigna. Saner: “ Composing static and dynamic analysis to validate sanitization in web
applications” In IEEE Symposium on Security and Privacy, 2008. SP 2008, pages 387–401, 2008.

[21]O.Hallaraker and G. Vigna. “ Detecting Malicious JavaScript Code in Mozilla”, In proceedings of the IEEE
International Conference on Engineering of Complex Computer Systems (ICECCS), 2005.

[22]D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Saner: “ Composing static and dynamic analysis to validate sanitization in web
applications” In IEEE Symposium on Security and Privacy, 2008. SP 2008, pages 387–401, 2008.

[23]E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, “Noxes: A client-side solution for mitigating cross-site scripting attacks”, In 21st ACM Symposium on Applied Computing (SAC), 2006.

[24]P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda,C.Kruegel,
and G. Vigna. “ Cross site scripting prevention with dynamic data tainting and static analysis”. In Proceeding
Of the Network and Distributed System Security Symposium (NDSS07), 2007.

[25]http://blog.zeltser.com/post/2497630353/targeting-web-browser-user