Int J Syst Assur Eng Manag
DOI 10.1007/s13198-015-0376-0

REVIEW

# Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art

**Shashank Gupta**[1] · **B. B. Gupta**[1]

Received: 28 November 2014 / Revised: 6 June 2015


**Abstract** Nowadays, web applications are becoming one of the standard platforms for representing data and service releases over the World Wide Web. Since web applications are progressively more utilized for security-critical services, therefore they have turned out to be a well-liked and precious target for the web-related vulnerabilities. Even though several defensive mechanisms have been building up to reinforce the modern web applications and alleviate the attacks instigated against them. We have analyzed the major concerns for web applications and Internet-based services which are persistent in several web applications of diverse organizations like banking, health care, financial service, retail and so on by the referring the Website Security Statistics Report of White Hat Security. In this paper, we highlight some of the serious vulnerabilities found in the modern web applications and revealed various serious vulnerabilities. Cross-Site Scripting (XSS) attack is the top most vulnerability found in the today's web applications which to be a plague for the modern web applications. XSS attacks permit an attacker to execute the malicious scripts on the victim's web browser resulting in various side-effects such as data compromise, stealing of cookies, passwords, credit card numbers etc. We have also discussed a high level of taxonomy of XSS attacks and detailed incidences of these attacks on web applications. A detailed comprehensive analysis of the exploitation, detection and prevention mechanisms of XSS attacks has also been discussed. Based on explored strength and flaws of these mechanisms, we have discussed some further work.

**Keywords** Cross-Site Scripting (XSS) · White Hat Security · Internet · World Wide Web (WWW) · JavaScript code injection attacks · Malicious JavaScript

## 1 Introduction

Now days the World Wide Web (WWW) has been converted into a multifaceted network incorporating wide variety of components and technologies including client-side technologies [JavaScript (Flanagan 2001) etc.], server-side technologies [ASP (MacDonald and Szpuszta 2005) etc.], HTTPs Protocol and wide variety of other technologies. Web applications developed on these platforms accommodate wide range of users, facilitating the Internet users with rich features of these advanced technologies. However the discrepancies among these technologies introduce the challenge of providing the defensive security measures for the safe development of web application. Although, current defensive measures offer restricted support to the platforms of web applications. Hence it is clear that deployment of defensive security measures need a significant effort. As a result, a high fraction of Internet-based web applications are vulnerable to serious vulnerabilities. White Hat Security's Website Security Statistics Report (WhiteHat 2013) offer a kind of perception on the current issues of security of web applications and the concerns that industries should deal for performing the online business in a secure way. This website has been distributing the security statistics report on the WWW since year 2006. They have analyzed the current state of

✉ B. B. Gupta
gupta.brij@gmail.com

[1] National Institute of Technology Kurukshetra, Kurukshetra, India

Published online: 14 September 2015

website security for the following popular domains: Banking, Financial Services, Health Care, Insurance and Retail.

They have analyzed the security of the website of these domains individually. Based on the following parameters they prepare a score card for these domains which are as shown below:

- Always Vulnerable
- Frequently Vulnerable
- Regularly Vulnerable
- Occasionally Vulnerable
- Rarely Vulnerable

Figure 1 shows some of the statistics for the banking websites on the basis of above specified parameters. The results indicate that 33 % of the banking websites are frequently vulnerable to serious cyber attacks. Also 24 % of these websites are always remains vulnerable. We broadly analyzed the current state of website security for the above mentioned domains on the following two constraints:

- Frequency of Modification of Source Code of Web Applications
- Static Analysis of Source Code of Web Application.

### 1.1 Frequency of modification of source code of web applications

Rate at which web application modifies or updates its source code changes the whole posture of the website. Whenever a new source code is incorporated in the web applications, the chances of occurrence of serious vulnerabilities in the web application automatically get increases. Therefore, we have collected the statistics from White Hat Security's Website Security Statistics Report that checks the websites of different domains for this constraint. Figure 2 clearly illustrates the percentage of web application's code change for different domains of websites (i.e. Banking, Financial Services, HealthCare, Insurance and Retail) on daily, weekly, monthly, quarterly and annually basis.

It is clearly depicted from Fig. 2 that organization like retail change their code more frequently even compared to banking. On the other hand, financial services possess some websites that only encounter 9 % change in the source code.

### 1.2 Static analysis of source code of web application

Static analysis of code is the process of analyzing the source code of web applications for discovering various loopholes of security in an automated manner (Wassermann and Su 2008). An organization of a website should perform this activity on a regular basis. Therefore, we again collect the statistics from White Hat Security's Website Security Statistics Report to check the websites of different domains for static analysis of source code of the web applications.

Figure 3 clearly illustrates the rate of performing static code analysis for different domains of websites (i.e. Banking, Financial Services, HealthCare, Insurance and Technology) with respect to prior to development phase, i.e. weekly, monthly, quarterly basis. It is clearly depicted from the figure that overall only 39 % of the websites execute the static code

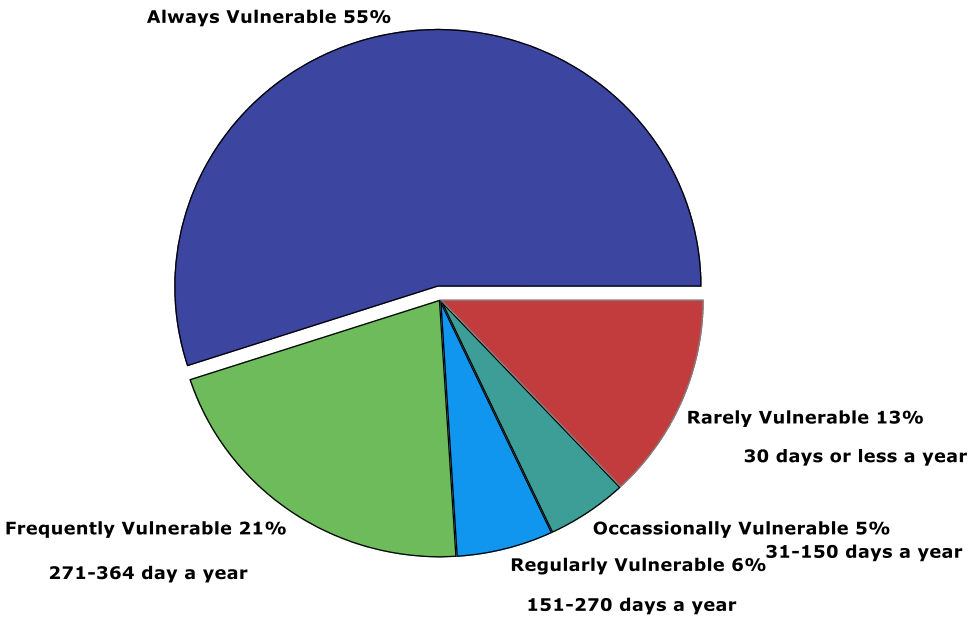


**Fig. 1** Percentage of banking websites exposed to cyber vulnerabilities

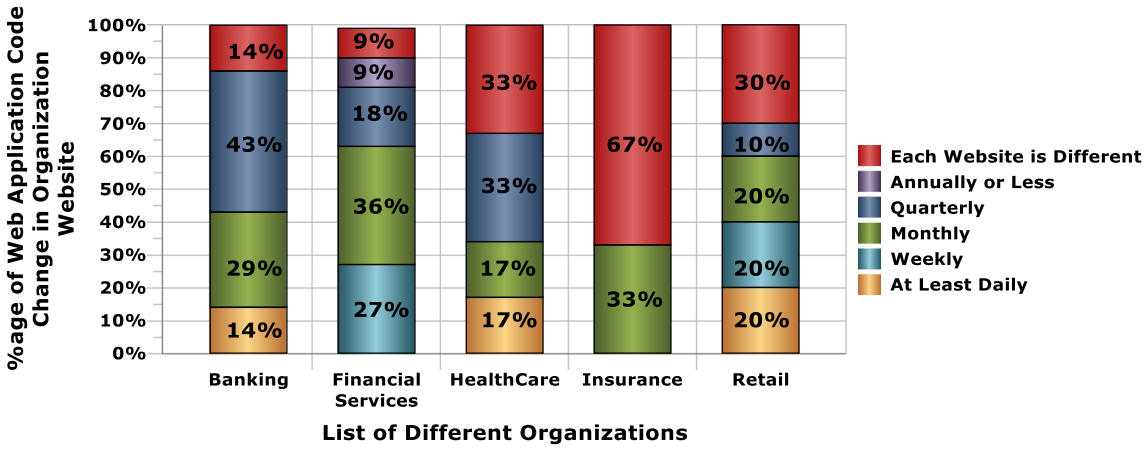


**Fig. 2** Rate of web application code change in the organization's website

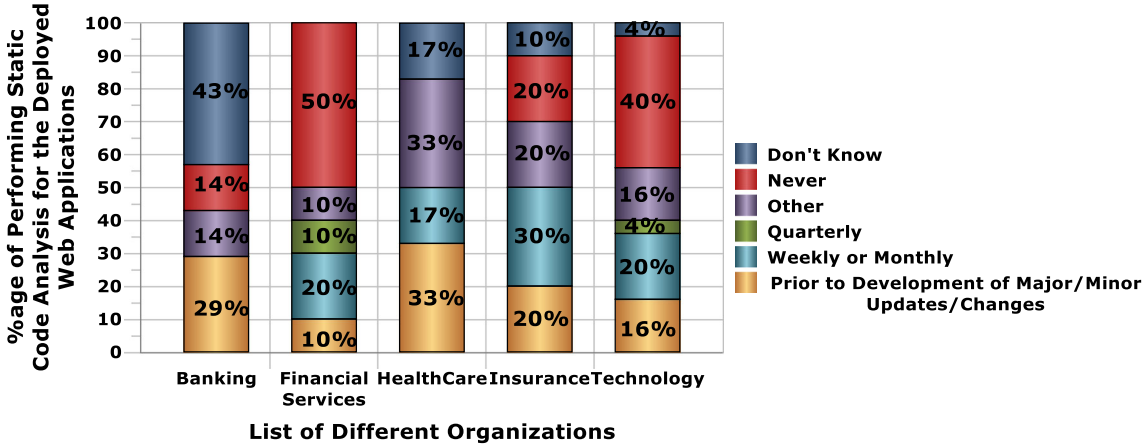


**Fig. 3** Rate of executing static code analysis on the websites

analysis on their web applications. Also financial services appear to perform the Static code analysis in a regular basis. On the other hand, 40 % of the websites of Technology and 50 % of the websites of Healthcare websites does not perform any Static code analysis.

In addition, there has been wide variety of defensive mechanisms employed by different websites of different organizations. The defensive mechanism has been exploited in different websites to control the behavior of each underlying web application. Following are some of the defensive controls employed by the different websites of different organizations:

- Programmers received computer-based software training
- Web Applications contain library that centralizes and enforces security controls
- Perform Static Code Analysis on their web site underlying applications
- Web Application Firewall Deployed
- Transactional/Anti-Fraud Monitoring System deployed

By referring these controls, the White Hat Security's Website Security Statistics Report shows some statistics in which the percentage of these controls exploited by different organizations is shown in Fig. 4. The statistics in Fig. 4 clearly highlights that 92 % of the web applications of different organizations perform static code analysis on their websites. Also 72 % of the web applications contain some sort of library of white-listed security measures which enforces the security controls.

According to the recent survey done by White Hat Security in May 2013, Cross-Site Scripting (XSS) (Louw and Venkatakrishnan 2009; Frenz and Yoon 2012) Vulnerability is the top most vulnerability among other vulnerabilities found in the real world web applications as shown in Fig. 5.

The statistics shown in Fig. 5 illustrates that Cross-Site Scripting, Content Spoofing and Information Leakage seize the top three places since they contribute 43, 13 and 11 % of total population of the vulnerability threats.

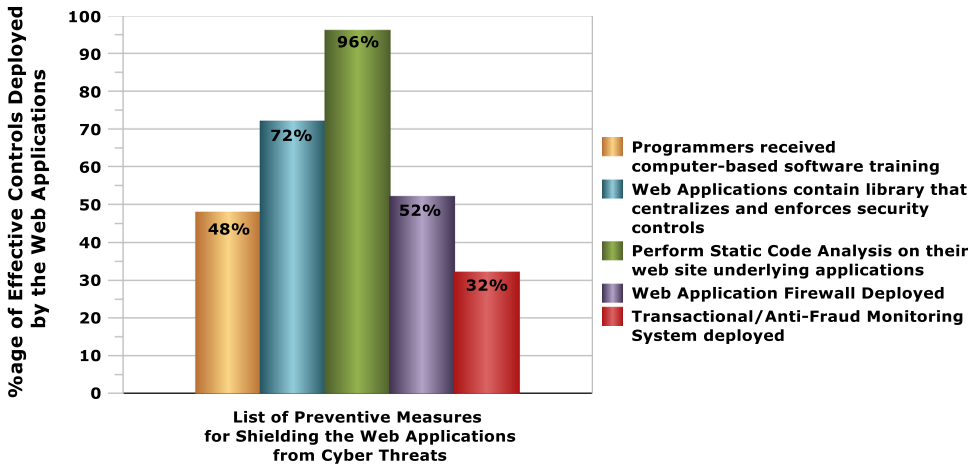


**Fig. 4** Statistics of effective controls employed by different organizations

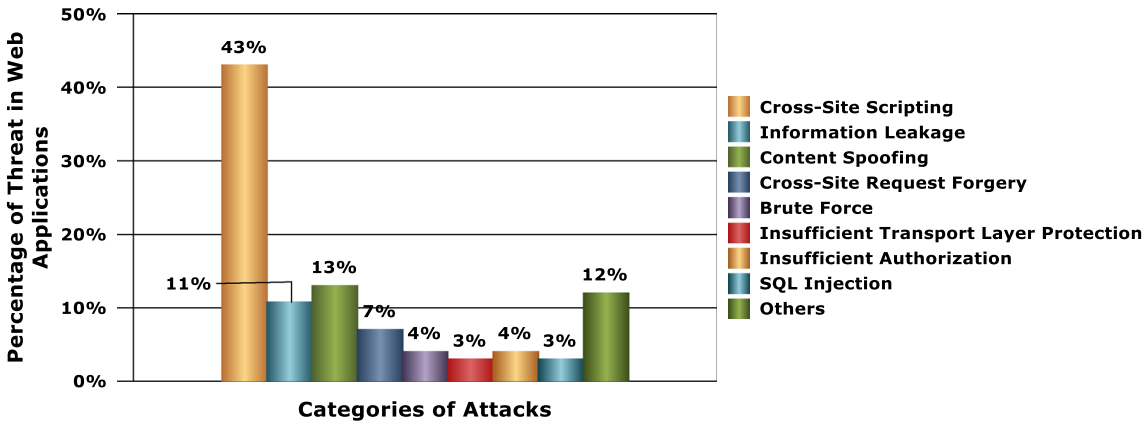


**Fig. 5** Overall vulnerability percentage of attacks in web applications

### 1.3 Incidences of XSS attacks

XSS is a malicious attack vector that is escalating exponentially in prominence. Moreover, the effect of XSS attacks has been seen globally by the entire WWW. XSS attacks occur almost daily. Websites such as Twitter, Facebook, Google have already become victim for XSS attacks. Table 1 illustrates some of the details regarding statistics of incidence of XSS attacks. In addition, numerous mobile websites are also found to be vulnerable to XSS attacks, which is as shown in Table 2.

The key contributions of this paper are as follows:

- Initially, we emphasize on the major security-related issues for web applications and Internet-based services which are persistent in several web applications of wide variety of diverse organizations like banking, health care, financial service, retail, technology and so on by the referring the Website Security Statistics Report (2013) of White Hat Security.
- We present a detailed comprehensive study on the XSS attacks describing its whole nomenclature. We also present taxonomy ofXSS attack detection and mitigation mechanisms along with its exploitation technique on the web applications.
- We highlight the techniques of determining whether a website is exposed to XSS attack or not.
- We also discuss several previous and latest discovered XSS worms along with several vulnerability scanners for the discovery XSS attack vectors in the source code of web applications.

We also discussed several state-of-art techniques based on XSS attacks and discovered their research gaps and discuss future scope.

## 2 Overview of Cross-Site Scripting (XSS) attacks

Cross-Site Scripting[1] (Martin and Lam 2008) is a Java Script code injection attack (Zhang and Wang 2010) that permits an attacker to execute injected JavaScript in

[1] http://excess-xss.com/.

**Table 1** XSS attack incidents (2001–2014)

| S. no. | Target | Incident details | Month, year | Consequence |
|---|---|---|---|---|
| 1. | UK Parliament Web Site | XSS flaws discovered in the popular search engine of UK parliament web site. This search engine permits the user to write a code for retrieving the images, videos and even request for passwords | March, 2014 | Disinformation |
| 2. | Video Web Site | Recently, security investigators revealed an unusual DDoS attack against numerous websites, which relied on a stored XSS vulnerability in an important video Web site and hijacked the browsers of users for flooding the site with intolerable congestion | April, 2014 | DDoS attacks |
| 3. | Yahoo Web Site | The accounts of Yahoo mail users have been hacked by exploiting a Dom-Based XSS Vulnerability by a hacker named Shahin Ramezany. The users of Yahoo mail had been either compromised by clicking on the malicious links or receiving the spams from other yahoo users | January, 2013 | Account hijacking |
| 4. | PayPal Web Site | PayPal has discovered potential XSS flaws in their login web pages. Due to this, cyber criminals inject the malicious scripts via crafted URL, hijacking the login web pages to steal sensitive credentials like user-id and passwords | March, 2012 | Account takeover |
| 5. | Hotmail Web Site | Hotmail email web services was found to contain a XSS vulnerability in which cyber criminals can steal keystrokes, cookies, or other pieces of sensitive information. Instead of clicking on the malicious link, the act of simply previewing this malicious email can simply compromise the account of Hotmail user | June, 2011 | Session hijacking |
| 6. | Twitter Web Site | Stored XSS Vulnerability was discovered on the Twitter website and researchers said that even previewing a link can simply display the pop-up window incorporating the login cookie of user | September, 2010 | Worm |
| 7. | Justin.tv Website | XSS worm was found on Justin.tv website, which permits the users to broadcast their videos online. The worm had injected a self-replicating malicious code on 2525 accounts in less than 24 h | July, 2008 | Worm |
| 8. | Orkut Web Site | The popular social networking site, Orkut was also hit by a XSS worm. Victim users receive a scrap comprising the words "BomSabado".This vulnerability permitted the attackers to insert malicious script into their profiles, set the platform for a stored XSS vulnerability that infected more than 650,000 profiles of Orkut users | December, 2007 | Worm |
| 9. | eBay Web Site | XSS flaw was also discovered in the famous e-commerce website, eBay. This website permits the cyber criminals to inject the malicious script tags in the auction description which produces an XSS vulnerability in the eBay website | April, 2006 | Disclosure |
| 10. | MySpace Web Site | The popular social networking site MySpace was infected with Samy XSS worm, which infected more than one million users of MySpace in just less than 20 h | November, 2005 | Worm |
| 11. | Price Lotto Web Site | Victims, who visited the web site of Price Lotto via Microsoft's Internet Explorer 4.x and 5.x, automatically downloaded malicious scripts that were programmed to modify the software configuration of their desktops | August, 2001 | Defacement |

victim's web browser in order to gain access to the sensitive resources like cookies, password, credit card numbers etc. XSS is an attack on the client-side web browser, but its capabilities are exploited on the web server side. For the exploitation of XSS vulnerabilities on the web applications, an attacker crafts and injects a malicious JavaScript payload on the web application. This script is injected in such a way that it seems to be benign component of the website and finally this script is executed within the domain of the trust of the website.

### 2.1 Malicious JavaScript payload injection process

As there are a variety of techniques to inject the malicious Java Script code into the web application of victim, but the most frequent way that an attacker follows is that an attacker injects the script into one of the pages of web site, so that the victim downloads the script from the web site. This is possible only if the web application accepts an input from the user-side into its web pages because an attacker can inject a malicious JavaScript string that will be reflected as a code on the browser of victim. Figure 6 illustrates a server-side script that is exploited in a web page to display the recent post.

Since the victim assumes that the post will include only text. But the attacker has the right to incorporate the script: <script>alert("XSS Exploited")</script> in the form of an input. Hence any victim visiting the web page will now receive the message in the form of following response as shown in Fig. 7. When the victim's browser clicks on this recent post, the JavaScript code will get executed automatically and as a result, the XSS attack will get exploit on the victim's web page.

**Table 2** Top XSS vulnerable mobile websites

| Target site | Web site URL address | Alexa ranking[a] |
|---|---|---|
| Pinterest | http://m.pinterest.com/ | 00338 |
| The New York Times | http://mobile.nytimes.com/search | 112 |
| Dictionary | http://m.dictionary.com/ | 182 |
| StatCounter | http://m.statcounter.com/feedback/?back=/ | 188 |
| MapQuest | http://m.mapquest.com/ | 525 |
| Nokia | http://m.maps.nokia.com/#action=search&params=%7B%7D&bmk=1 | 568 |
| Intel | http://m.intel.com/content/intel-us/en.touch.html | 1107 |
| MTV | http://m.mtv.com/asearch/index.rbml?search= | 1168 |
| SlashDot | http://m.slashdot.org/ | 2267 |
| HowStuffWorks | http://m.howstuffworks.com/s/4759/Feedback | 2882 |

[a] http://developers.evrsoft.com/find-traffic-rank.shtml

**Fig. 6** Server-side script exploited in a web page

```
<html>
Recent Post:
<script>alert("XSS Exploited")</script>
</html>
```

**Fig. 7** Script in the shape of HTTP response message

```
<html>
Recent Post:
<script>alert("XSS Exploited")</script>
</html>
```

### 2.2 Malicious JavaScript payload

It is not at all necessary that whenever the JavaScript code will get executed on the web browser of any Internet user, it will always perform any malicious activity. Instead JavaScript executes in a very confined area which has very restricted access to user's credentials and several files of operating system (Meyerovich and Livshits 2010). However, there are some specifics in which the probability of becoming JavaScript code malicious gets very high:

- JavaScript payloads can access the sensitive credentials of user like cookies, passwords, credit card numbers etc.
- JavaScript has the capability to transfer the HTTP requests to web servers by utilizing the capabilities of XML-HTTP request and several other in-built methods.
- It can also alter the syntax and semantics of web page by exploiting the capabilities of Document Object Model (DOM) manipulation techniques.

### 2.3 Negative aspects of malicious JavaScript

Figure 8 shows the three main negative aspects of inserting the poorly written JavaScript code on the vulnerable web applications. These issues have proved to be very serious in the modern era of WWW.

- *Cookie stealing* The attacker can theft the cookie by injecting the XSS attack vectors on the vulnerable web application for stealing the session IDs or session hijacking (Putthacharoen and Bunyatnoparat 2011).
- *Phishing attacks* The attacker can trick the victim to provide his/her sensitive credentials by injecting a forge login form by utilizing the capabilities of DOM manipulation to aim the web server.
- *Key logging* The attacker can utilize the capabilities of keyboard event listener to trace all the keystrokes of the victims and transfer this information to the web server for accessing the sensitive information like password, credit card numbers etc.

### 2.4 Process to inject the XSS attack vectors on the vulnerable web applications

Cross-Site Scripting vulnerability is such type of vulnerability on the web applications in which an attacker injected the malicious XSS payload vector for stealing the sensitive credentials (e.g. cookies, credit card numbers etc.) of victim (Bisht and Venkatakrishnan 2008; Frenz and Yoon


**Fig. 8** Drawbacks of XSS attacks

2012; Shahriar and Zulkernine 2009; Zhenyu et al. 2007). Therefore it is reflected that XSS attacks normally took place due to the inappropriate sanitization of user-supplied input. The key objective of this attack is to theft the victim's web browser credentials (e.g. cookies, passwords etc.) by injecting an un-sanitized JavaScript code. In the present era of Internet, web applications are developed by utilizing the advanced HTML and Java Script tags. The old-fashioned technique of bye-passing the XSS attack is to just disable the scripting languages on the web browser. However by doing this, web user has to compromise with the enhancement and readability of the web page.

Figure 9 is illustrates the chain of steps for injecting the XSS attack vector on the vulnerable web application of victim which are as follows:

- Firstly the attacker found a website which is susceptible to XSS attack. Then, the attacker inject an unfiltered malicious script on the VWA in the form of a recent post whose goal is to theft the sensitive credentials (e.g. cookies) of the victim's web browser.
- Afterward, the victim's web browser browses the VWA and malicious JavaScript comes in the form of a current post in HTTP response message. The malicious script then executes surrounded by the field of reliance of the web application site.
- Once the malicious script gets executed, the victim's credentials (e.g. cookies) will be transferred to the web server of the attacker.

Lastly, the cookies will be used by the attacker to exploit session hijacking. 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.

## 3 Taxonomy of XSS attacks

XSS vulnerabilities can be largely categorized into two main classes: Client-side XSS Vulnerabilities [Plugin[2] and DOM-Based XSS vulnerabilities (Klein 2005)] and Server-side XSS Vulnerabilities [Persistent and Non-Persistent XSS Vulnerabilities (Gupta et al. 2012; Gupta and Sharma 2012; Gupta and Gupta 2014, 2015)]. Figure 10 illustrates the taxonomy of XSS Attacks and Worms.

Now a day's malicious JavaScript could also be inserted by utilizing the plug-ins. This has given rise to the propagation of plug-in XSS vulnerabilities like *Flash XSS vulnerability*, which has been exploited to insert scripts into web applications. The Renren XSS worm[3] makes use of a Flash vulnerability to facilitate access to the tainted web page vulnerability, and insert malicious JavaScript code. To repair such type of attacks, internet users should update their Adobe Flash plug-into thwart such malicious activity.

Normally XSS attacks have been broadly classified into three main categories (as shown in Fig. 11): Persistent, Non-Persistent and DOM-Based XSS attacks.

### 3.1 Persistent XSS attack

This type of XSS attack is also known as Stored XSS attack which can be usually found in those areas of web applications where Internet users are permitted to input HTML/JavaScript code (e.g. in search text box) (Wang et al. 2007, 2011; Van-Acker et al. 2012). For this cause, stored XSS attack eternally stores the malicious script on the storage area of web application.

Figure 12 illustrates the pattern example of Stored XSS attack to exploit its vulnerability. Following are some sequence of steps which explains the scenario of exploiting the vulnerability of Stored XSS attack:

- The attacker utilizes one of the web application forms to inject a malicious JavaScript string into the repository of the website.
- The victim browses the VWA and requests a web page from the web application.
- The web application incorporates the malicious JavaScript string from the web application's repository in the HTTP response message and transmits it to the browser of victim.
- The web browser of victim's web application runs the malicious JavaScript code within the HTTP response message, finally transferring the victim's credentials to the web server of the attacker.

[2] http://lwn.net/Articles/216223/.

[3] http://issmall.isgreat.org/blog/archives/2.

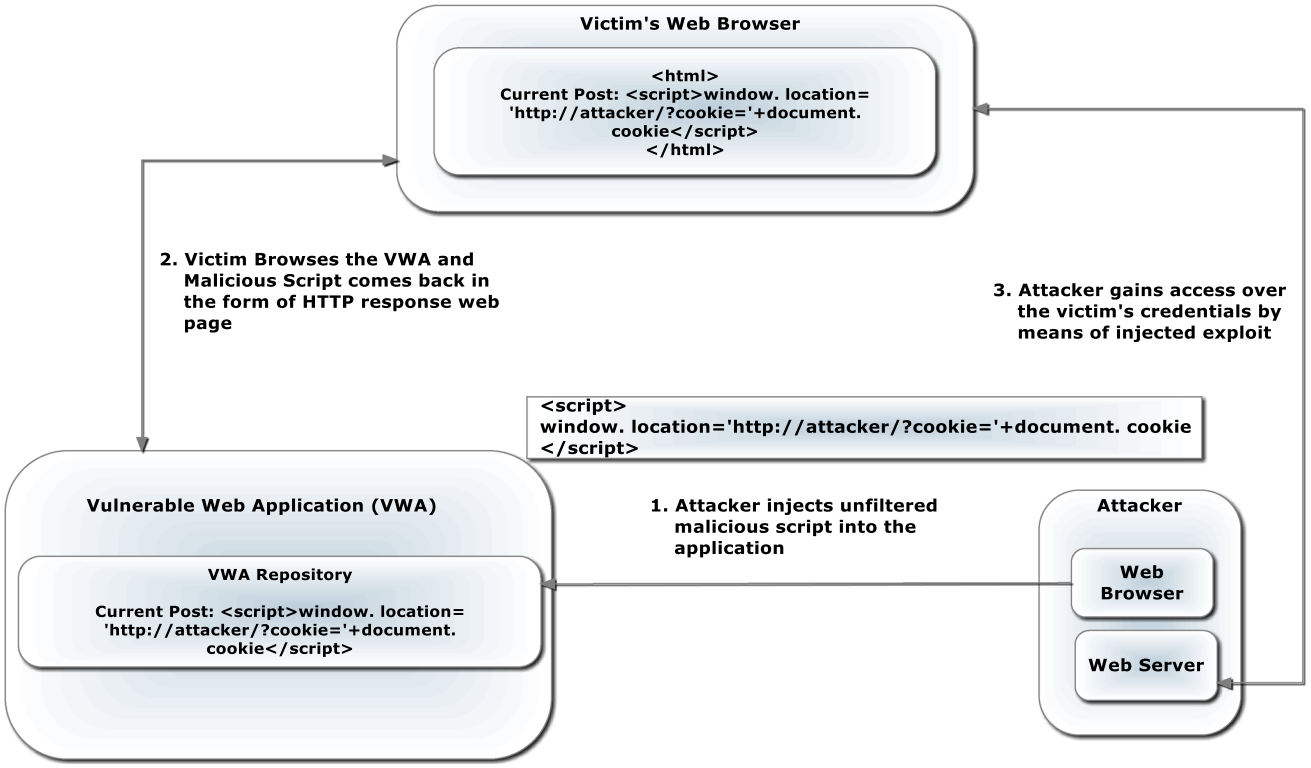


**Fig. 9** Cross-Site Scripting (XSS) exploit

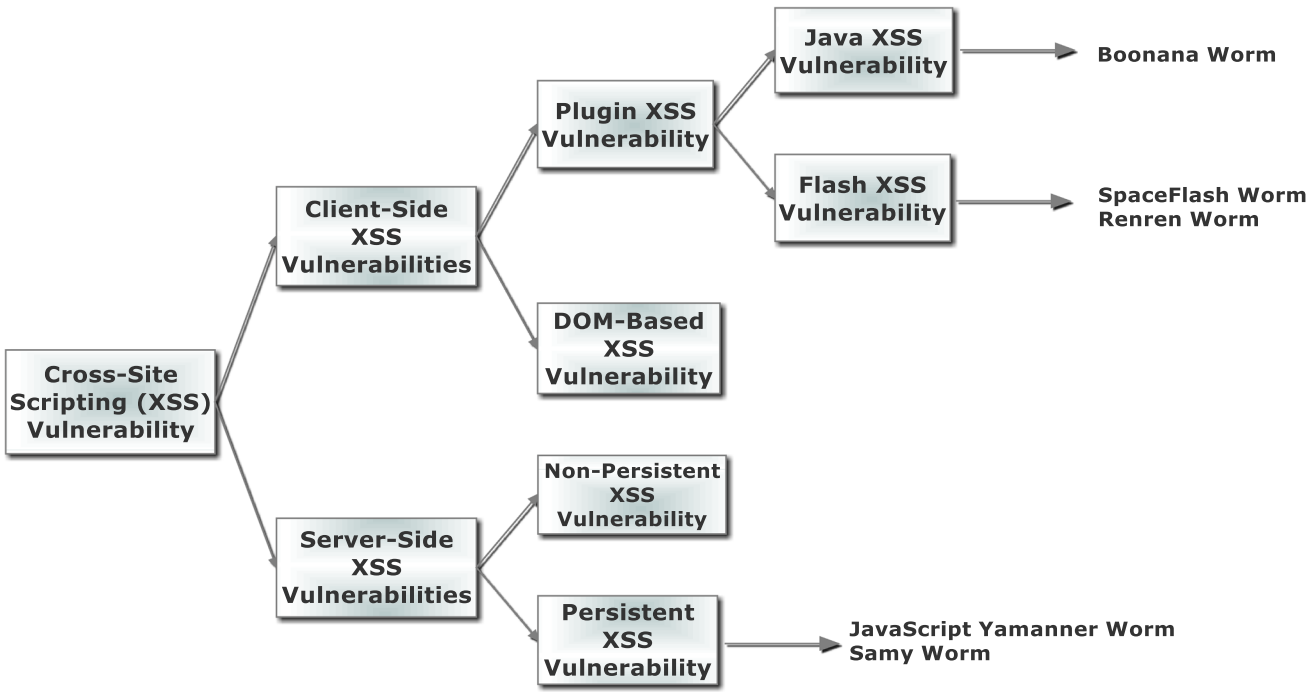


**Fig. 10** Taxonomy of XSS attacks and worms

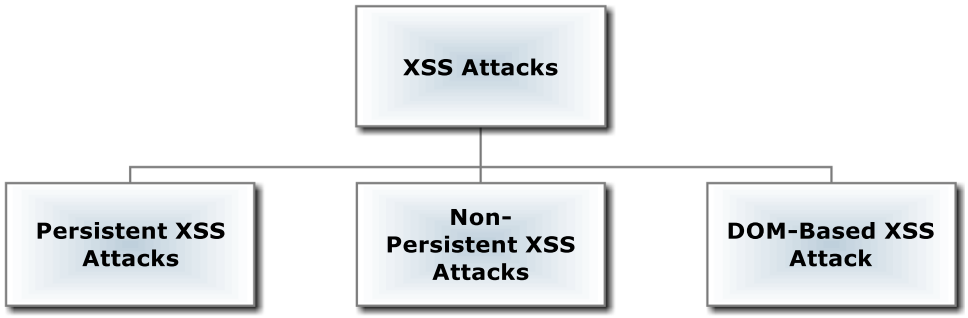


**Fig. 11** Main categories of XSS attacks

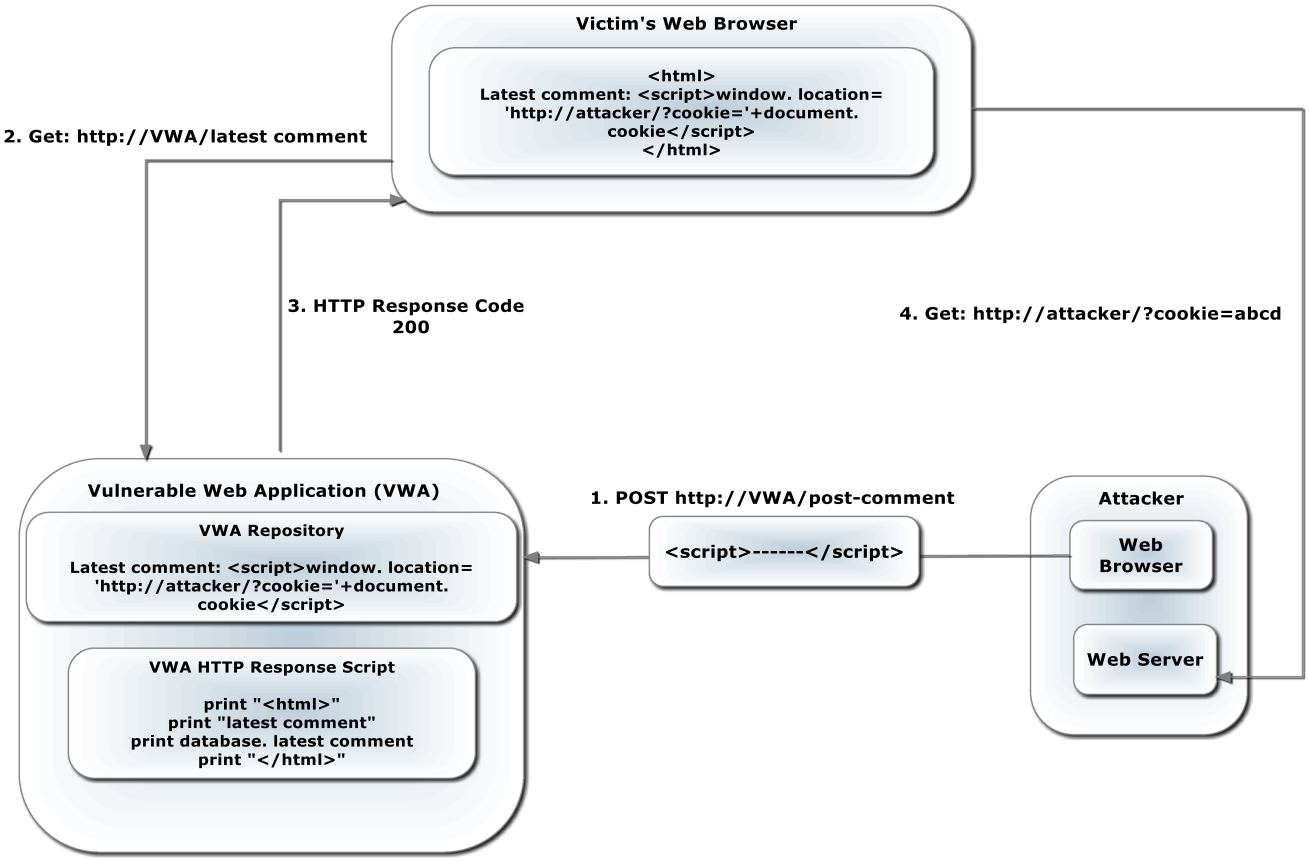


**Fig. 12** Pattern example of stored XSS attack

### 3.2 Non-persistent XSS attack

This XSS attack is also known as Reflected XSS attack where the inserted script is returned back to the server of the victim's web application in the appearance of an error message, outcome of search engines in web applications or any extra response web page or a message that incorporates a few or all part of the input transmitted on the server-side (Avancini and Ceccato 2011; Athanasopoulos et al. 2010). Therefore, in reflected XSS attack, there is no requirement of storing the injected and malicious JavaScript code on the web application of victim. Figure 13 illustrates a pattern scenario of injecting the XSS attack vector payload by exploiting the reflected XSS vulnerability.

Following are the chain of steps of exploiting the vulnerabilities of Non-Persistent XSS attack:

- Initially, the attacker creates a URL including a malicious JavaScript string and transfers it to the web browser.
- The victim is tricked by the attacker into requesting the URL address from the VWA.
- The VWA incorporates the malicious injected JavaScript string from the URL address in the HTTP response.

- The web browser of victim runs the injected JavaScript within the HTTP response message and transmitting the victim's credentials (e.g. passwords, cookies etc.) to the server of the attacker.

### 3.3 DOM-based XSS attack

This XSS attack is an alternative of both Reflected and Stored XSS attack. In this attack, the injected malicious JavaScript string will not execute on the victim's web browser until the benign JavaScript code of VWA gets executed (Tiwari et al. 2008). Figure 14 illustrates the pattern example of DOM-Based XSS attack. Following are the chain of steps of exploiting the vulnerabilities of DOM-Based XSS attacks.

- The attacker creates a URL web address incorporating a malicious JavaScript string and transmits to the browser of the victim.
- The victim is tricked by an attacker into requesting the URL address from the VWA.
- The VWA accepts the HTTP request, although it does not incorporate the malicious injected JavaScript string in the HTTP response message.
- The web browser of the victim runs the benign JavaScript code inside the HTTP response, resulting the malicious JavaScript to be injected into web page.

The web browser of the victim runs the malicious injected JavaScript, transferring the victim's credentials (e.g. cookies) to the web server of an attacker.

## 4 Process to check a website vulnerable to XSS attacks

In Gupta and Gupta (2014), the authors have presented the steps of discovering whether a web application is exposed to XSS attack or not. Figure 15 shows the fixed steps of finding whether your website is exposed to XSS attacks or not. The following sequence of some points gives a precise explanation of the steps depicted in the figure:

- Access a website from your web browser and look for such parts of web application which require input from user side. Such identifications in websites are login forms, bulletin boards, search fields, comments etc.
- Now at this moment just input any string into these spots and submits this string to the web server of the website.
- Following this moment, verify the first condition which states that test the HTTP response web page of the server for the same string which was submitted by the user. If the HTTP response web page includes the similar string, then the web site can be considered as XSS exploited. Otherwise if the HTTP response web

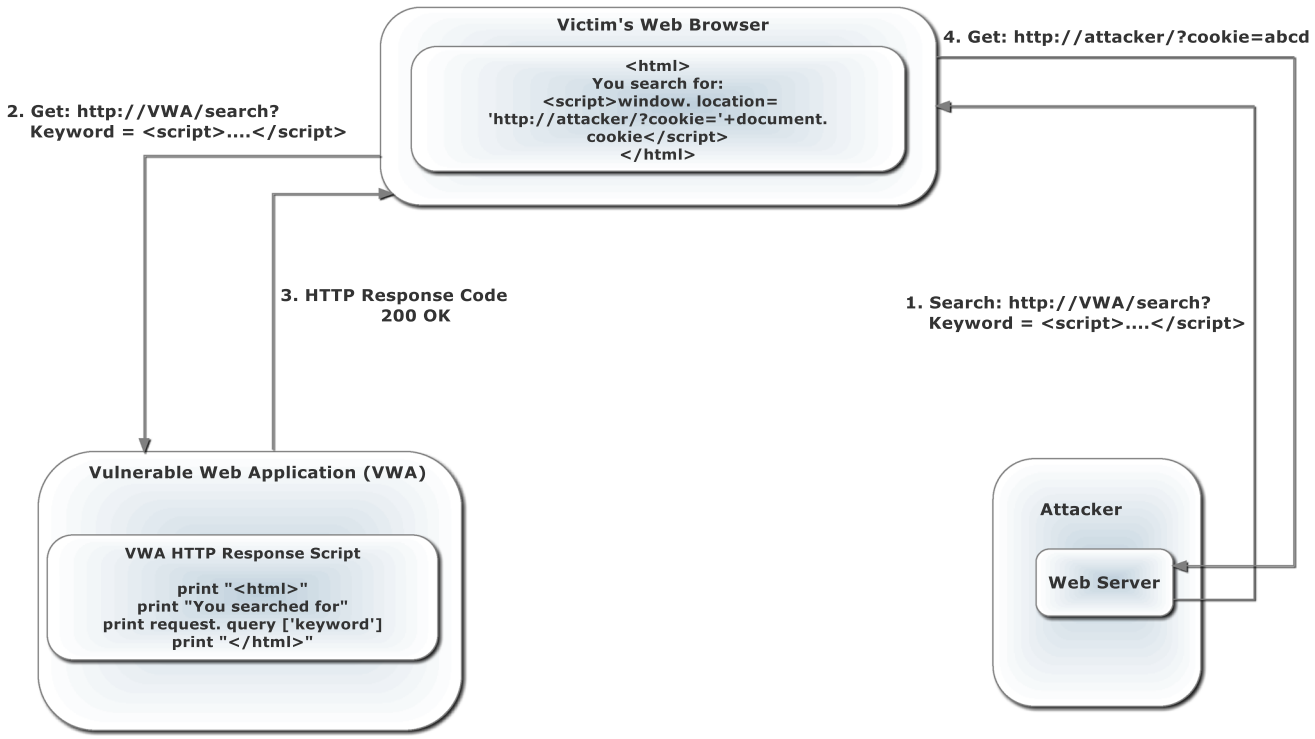


**Fig. 13** Pattern example of reflected XSS attack

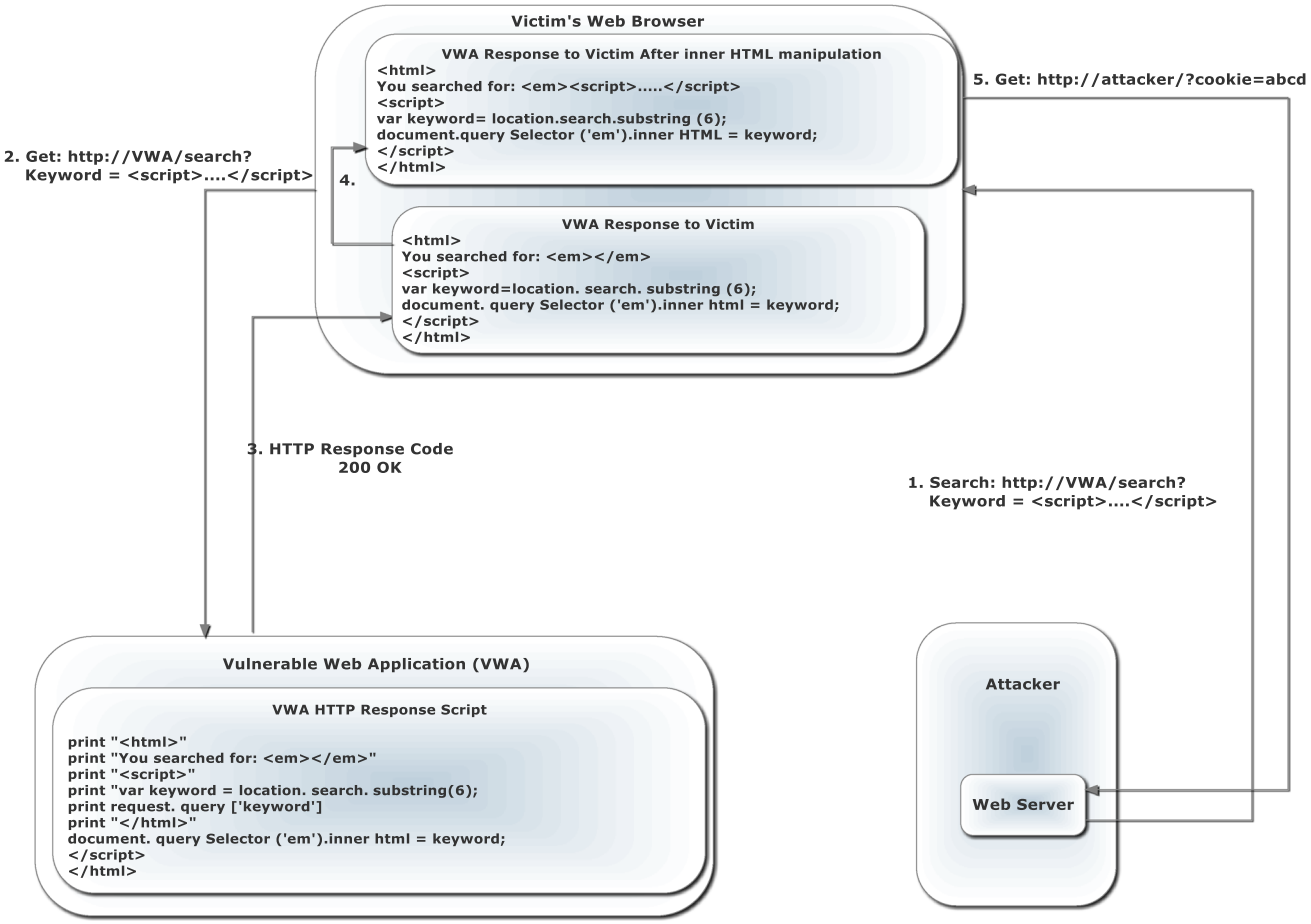


**Fig. 14** Pattern example of DOM-based XSS attack exploit

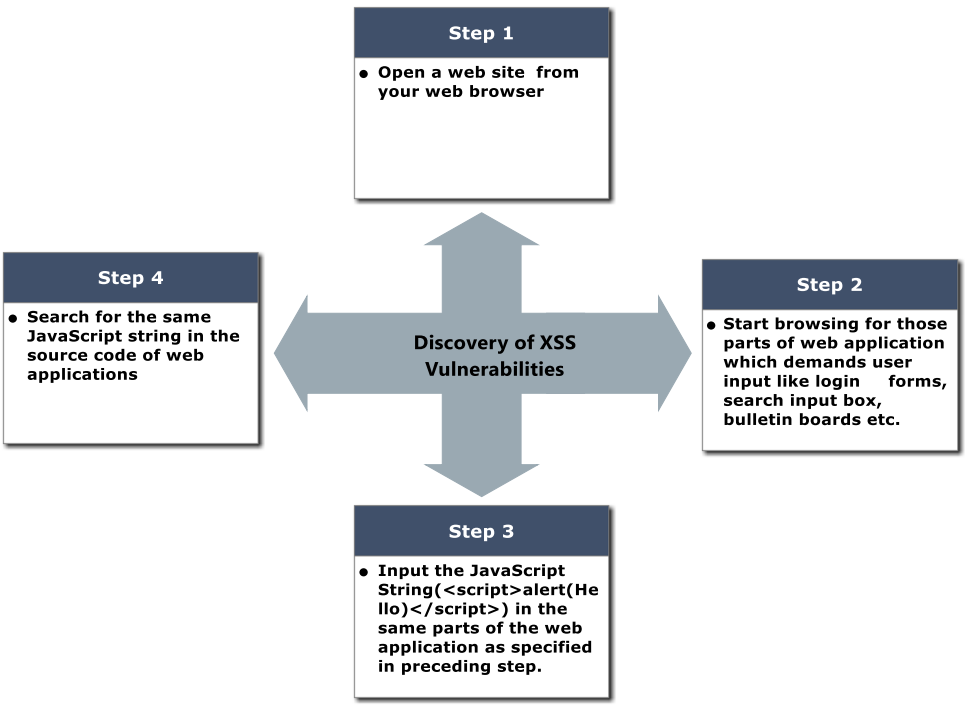


**Fig. 15** Steps to discover XSS attacks in web applications

page does not incorporate any user-submitted string then test for the next condition.

- The next condition says just input any JavaScript string (<script>alert("XSS Exploited")</script>) in the user supplied portions and send this input string to the web server of the website. After transferring this string to the web server, if the web server replies with a "XSS Exploited" keyword in the outline of pop-up window, then your web application is exposed to XSS attacks and if no then test for the next condition.
- The final condition states that just look for the similar input JavaScript string (<script>"XSS Exploit"</script>) in the code of the website. If any ingredient of the string is effectively found, then the web site is susceptible to XSS attacks.

### 4.1 Real-world XSS worms

In the recent survey (Cao et al. 2012), several real-world XSS worms on modern and popular web applications are discussed. Most of these XSS worms exploit the vulnerabilities of Stored XSS attack on real world web applications (like Facebook, Twitter etc.). Table 3 shows the categories of real world XSS worms with specified time period.

The first XSS worm i.e. Samy worm[4] was found in October, 2005 in MySpace web application. It affects more than one million users in just 24 h. The Orkut 'Born Sabado' Virus is a seriously disguised XSS worm. During the course of its occurrence, an Internet user account on Orkut was tainted when he/her just interprets a scrap transferred by his/her tainted friend. JavaScript Yamanner Worm was discovered on Yahoo! Mail. The Xanga XSS worm was found on a real world blogs. The Gaia and the U-Dominion XSS worm were discovered on gaming web applications. The Justin.tv XSS worm was detected on a websites of video hosting. Space Flash XSS worm was also released on MySpace web application.

Samy worm sets a record in the first 24 h of propagation as it was spread over the Internet in a fastest way and affected one million users in just 24 h. In July, 2001, Code Red 1[5] utilizes the benefit of buffer-overflow vulnerability and tainted over 359,000 workstations in 24 h. Meanwhile in the same year, Code Red2[6] comes into picture to exploit the same vulnerability and infected over 275,000 computers in 24 h. In Jan, 2003, Slammer[7] replicates itself on UDP ports by taking the benefit of buffer overflow vulnerability and infected 75,000 victims in just 24 h. In Aug, 2003, Blaster came into existence by exploiting Remote Procedure Call (RPC) against Windows workstations and infected 336,000 machines worldwide.

Figure 16 shows the dissemination totals of each mentioned worm in merely 24 h. It is clearly depicted from the figure that the Samy worm leads the top position by infecting over one million users over the globe.

**Table 3** List of several 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 |

### 4.2 XSS black-box vulnerability scanners

Black-box vulnerability scanners scrutinize only the functionality or input/output results of web application without analyzing the source code of the web applications or their internal structure. Table 4 shows the list of six XSS vulnerability scanners along with their vendor names and corresponding version number. The prices of these scanners generally fall in the range of ten to ten-thousand dollars.

## 5 Detection and mitigation state-of-art techniques of XSS attacks

In this section, we present various XSS attacks detection and mitigation techniques. A brief quantitative discussion of some of the state-of-art techniques of XSS attacks is presented. In addition, Table 5 shows the comparison of summary of related detection and mitigation techniques of XSS vulnerabilities in the form of their deployment location, strengths and weaknesses.

[4] http://www.betanews.com/article/CrossSite_Scripting_Worm_Hits_MySpace/1129232391, http://www.myspace.com/33934660, http://namb.la/popular/tech.html.

[5] http://www.caida.org/analysis/security/code-red/.

[6] http://www.caida.org/outreach/papers/2002/codered/codered.pdf.

[7] http://en.wikipedia.org/wiki/SQLSlammer, http://www.cs.berkeley.edu/~nweaver/sapphire/, http://www.wired.com/wired/archive/11.07/slammer.html.

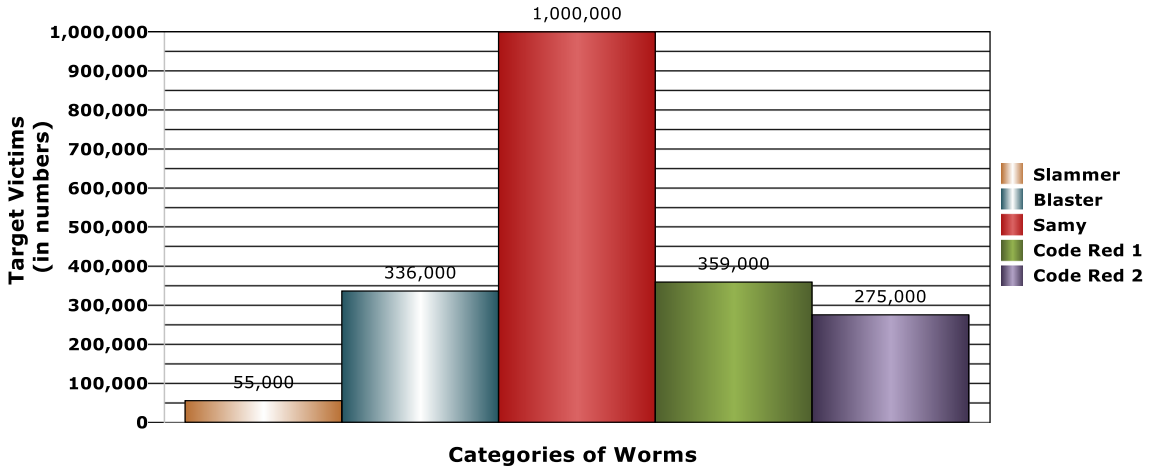


**Fig. 16** Dissemination totals of worms in 24 h

**Table 4** List of XSS black-box vulnerability scanners

| Black-box scanners | Vendor |
|---|---|
| Acunetix | Acunetix |
| AppScan | IBM |
| N-Stalker | N-Stalker |
| Zed Attack Proxy | OWASP |
| WebInspect | HP |

*Noxes: a client-side solution for mitigating XSS attacks* Noxes (Kirda et al. 2006) is the foremost client-side solution that influences the proposal of personal firewalls for preventing the users against XSS attacks. 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 rule repository manually and enter a set of rules. In Firewall prompts, the user can generate a set of policies, every time a connection demand is made that does not match any existing rule. Finally in snapshot mode, the user can automatically generate a set of rules.

*Noncespaces using randomization to defeat Cross-Site Scripting attacks* Noncespaces (Gundy and Chen 2012), an end-to-end mechanism that facilitate 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 its features to discover and mitigate injected malicious script in all documents prior to transferring it to the web browser. 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.

*SWAP: mitigating XSS attacks using a reverse proxy*: SWAP (Secure Web Application Proxy) (Wurzinger et al. 2009) 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. 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.

*XSSDS: server-side detection of Cross-Site Scripting attacks* 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.

*XSS-GUARD: precise dynamic prevention of Cross-Site Scripting attacks* XSS-Guard (Bisht and Venkatakrishnan

**Table 5** Summary of state-of-art techniques of XSS attacks

| State-of-art techniques | Deployment location | Strengths | Weaknesses |
|---|---|---|---|
| Noxes: a client-side solution for mitigating Cross-Site Scripting attacks (Kirda et al. 2006) | Web browser | Noxes support such a mitigation practice of XSS attack that considerably diminishes the amount of connection alert prompts and providing the defense against XSS attacks | Also this tool suffers from low reliability and prohibits the inclusion of benign HTML |
| Noncespaces: using randomization to enforce information flow tracking and thwart Cross-Site Scripting attacks (Gundy and Chen 2012) | Web browser and web server | Evade all the troubles and obscurity occurs with sanitization | Does not provide any defensive mechanism regarding inserted JavaScript code downloaded from remote web site |
| SWAP: mitigating XSS attacks using a reverse proxy (Wurzinger et al. 2009) | Web proxy or web server | It has a fine capability of detecting the deviation between benign and injected JavaScript code | Many categories of XSS attacks cannot be detected by this technique |
| XSSDS: server-side detection of Cross-Site Scripting attacks (Johns et al. 2008) | Web server | It has fine capability to discover XSS attacks by measuring the deviation between the HTTP web request and its associated HTTP response | The method of discovering persistent XSS attack suffers from few false positives and requires a more advanced training phase for the collection of more scripts |
| XSS-GUARD: precise dynamic prevention of Cross-Site Scripting attacks (Bisht and Venkatakrishnan 2008) | Web server | Its main strength is that it can evade illicit script data from being a part of the HTTP response web page | While this technique attempt to sanitize unsafe output, it still influence web browser parsers to infer unsafe HTML data and are vulnerable to threats that utilizes browser parse quirks |
| S2XS2: a server side approach to automatically detect XSS attacks (Shaihriar and Zulkernine 2011a) | Web server | Its main strength lies on the concept of "boundary injection" to encapsulate dynamic-generated content and "policy generation" to validate the user-injected data | Their technique consumes more time in policy checks and thus degrades the attack detection capability |
| Injecting comments to detect JavaScript code injection attacks Shahriar and Zulkernine (2011) | Web server | Their strength relies on discovering the XSS attacks by inserting the comment statements consisting of random generated tokens and characteristics of benign JavaScript code | It has been observed that their proposed technique discovers a part of code injection attacks |
| SessionSafe: implementing XSS immune session handling (Johns 2006) | Web server | It is a server-side translucent tool that does not require any modifications in the source code of web applications and shield against XSS attacks | Such methods need several, highly structured server domains that may be awkward to handle. Also, they can offer only restricted security such as prohibiting access to the sensitive resources of a web application like cookies |
| BLUEPRINT: robust prevention of Cross-Site Scripting attacks for existing browsers (Louw and Venkatakrishnan 2009) | Web server or web browser | Blueprint demonstrates the methods to guarantee the safe construction of the intended HTML parse tree on the web browser. This approach provides security against malicious script injections, and facilitates the support for script-less HTML content | Unfortunately, this technique requires modification both at the client as well as server side. Also the authors had not proposed any idea of handling the unsafe HTML content at the server side |
| SecuBat: a web vulnerability scanner (Kals et al. 2006) | Web server | The main goal of this scanner is to determine and exploit application-level vulnerabilities in a large number of real time web sites without human intervention | The authors have tested this vulnerability scanner on more than 25,000 live web pages. But no ground truth is presented for these web sites |

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. XSS-Guard 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. 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.

*$S^2XS^2$: a server side approach to automatically detect XSS attacks* $S^2XS^2$ (Shaihriar and Zulkernine 2011a) 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.

*Injecting comments to detect JavaScript code injection attacks* Shahriar and Zulkernine (2011) had proposed a server side JavaScript code injection technique which relies on the idea of inserting comment statements consisting of random generated tokens and characteristics of benign JavaScript code. When a HTTP response web page is produced, source code of JavaScript with no or faulty comment is considered as injected code. Also, the legitimate comments are verified for duplicity. The existence of any duplicate comments or a variance between likely code features and perceived features indicates JavaScript code as injected. The authors had also implemented a prototype which inserts JavaScript comments in an automated manner and install inserted JavaScript code detector as a server side filter.

*SessionSafe: implementing XSS immune session handling* Johns (2006) has included three server-side methods in SessionSafe tool, which attempts for XSS-immune sessions to evade XSS session hijacking. This tool utilizes deferred loading, one-time URLs, and sub-domain switching to secure a web application. They thwart the transmission of Session ID (SID), prevent the recreation of session, and restrict vulnerabilities impact to susceptible web pages simply. This technique is not proposed to substitute input and output validation, a key aspect in several Web application security procedures. It is a server-side translucent tool that does not require any modifications in the source code of web applications and shield against XSS Attacks.

*BLUEPRINT: robust prevention of Cross-Site Scripting attacks for existing browsers* BLUEPRINT (Louw and Venkatakrishnan 2009) is a server-side solution to thwart 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. This approach reduces the dependency on web browsers in recognizing unsafe content over the network. 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.

*SecuBat: a web vulnerability scanner* SecuBat (Kals 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. The authors had included four different attack components namely SQL Injection, Simple Reflected XSS Attack, Encoded Reflected XSS Attack and Form-Redirecting XSS Attack. Lastly, the analysis component scans the results returned by the web sites to find out whether an attack was successful or not.

Based on the detailed quantitative discussion of related defensive mechanisms of XSS attacks, a detailed comparison of these techniques has been presented with each other based on some key factors, which is highlighted in Table 6. This has been clearly observed that majority of these techniques cannot mitigate DOM-Based XSS attack. In addition, some of them require major modifications at the client and server side.

In addition, we have also categorized some of the state-of art techniques on XSS attacks into some groups based on some essential key factors (i.e., method utilized, altering browser, browsing source code and altering source code).

Web applications have been emerging extremely fast with novel training frameworks and modern technology skills. Numerous server-side detection and mitigation strategies are present, but such methods have not been fully functional because of their process overhead. On the other hand, several accessible client side XSS defensive strategies disgrace the performance of client's system resulting in poor web surfing experience. In addition to this, it has also been surveyed that majority of the XSS defensive mechanism discussed in Table 7 does not shield against DOM-Based XSS attacks. Moreover several techniques demand major alterations at the client web browser and as well as server-side. Even though several defense mechanism incorporate the sanitization mechanisms in the code of JavaScript for shielding against the XSS attacks. But again, majority of these techniques suffer from wide variety of false positives and false negative rates.

**Table 6** Comparison of existing XSS defensive techniques based on some factors

| Related work techniques | Exploitation location | Discovery site | Scrutinizing mechanism | Persistent XSS attack detection | Non-persistent XSS attack detection | DOM-based XSS attack detection | Client-side web browser alterations | Server-side modifications | JavaScript code amendments |
|---|---|---|---|---|---|---|---|---|---|
| Kirda et al. (2006) | Web browser | Web browser | Active | ✗ | ✔ | ✗ | ✔ | ✗ | ✗ |
| Gundy and Chen (2012) | Web server | Hybrid | Active | ✔ | ✔ | ✗ | ✗ | ✔ | ✗ |
| Wurzinger et al. (2009) | Web proxy or server | Web server | Passive | ✔ | ✔ | ✗ | ✔ | ✗ | ✔ |
| Johns et al. (2008) | Web browser | Hybrid | Passive | ✔ | ✔ | ✗ | ✗ | ✗ | ✔ |
| Bisht and Venkatakrishnan (2008) | Web server or proxy | Web server | Active | ✔ | ✔ | ✔ | ✗ | ✔ | ✗ |
| Shaihriar and Zulkernine (2011a) | Web server | Web server | Active | | ✔ | ✗ | ✗ | ✔ | ✗ |
| Shahriar and Zulkernine (2011b) | Web server | Web server | Active | ✔ | ✔ | ✗ | ✗ | ✔ | ✗ |
| Johns (2006) | Web server | Hybrid | Active | ✔ | ✔ | ✗ | ✔ | ✗ | ✔ |
| Louw and Venkatakrishnan (2009) | Web server or web browser | Web server | Active | ✔ | ✔ | ✗ | ✗ | ✗ | ✔ |
| Kals et al. (2006) | Web server | Hybrid | Passive | ✔ | ✔ | ✗ | ✔ | ✗ | ✗ |

**Table 7** Systematic grouping of existing XSS defensive techniques based on some methods

| State-of-art techniques | Method | AB | BSC | ASC |
|---|---|---|---|---|
| Kirda et al. (2006), Jim et al. (2007) | Black-white list | ✔ | ✔ | ✗ |
| Gundy and Chen (2012), Wurzinger et al. (2009), Johns et al. (2008), Shaihriar and Zulkernine (2011a, b) | String injection | ✔ | ✔ | ✔ |
| Bisht and Venkatakrishnan (2008) | Rewriting | ✗ | ✔ | ✗ |
| Wassermann and Su (2008), Gupta and Gupta (2015), Johns (2006) | Static analysis | ✗ | ✔ | ✗ |
| Kals et al. (2006), Vogt et al. (2007), Zhang et al. (2010) | Taint and flow analysis | ✗ | ✔ | ✗ |
| Choi et al. (2012), Nunan et al. (2012), Shar and Tan (2012) | Machine learning | ✗ | ✗ | ✗ |

*AB* altering browser, *BSC* browsing source cod, *ASC* altering source code

The necessity to install updates or additional components on each user's web browser or workstation also degrade the performance of client side solutions. The XSS defensive solutions revealed above concerning mitigation of XSS attack cannot thwart the XSS attack totally. A recent statistics from White Hat Security's Website Security Statistics Report showed that nearly 80 % of web applications found on the Internet have at least one serious vulnerability. Moreover, XSS vulnerability is considered to be a plague for the modern Internet-based web applications.

## 6 Future scope

The most essential technique of mitigating XSS attacks is to carry out safe input handling. Encoding mechanisms should be exploited each time user input is incorporated in a web page (Weinberger et al. 2011; Sharath Chandra and Selvakumar 2011).In several other cases, encoding mechanisms has to be substituted with input validation mechanisms. Safe handling of user-supplied input has to take into consideration that which perspective of a web page the user-supplied input is injected into. Therefore, to thwart all types of XSS attacks, safe handling of user-supplied input has to be exploited in both client-side and server-side source code of web applications. Taking into consideration all the research gaps discovered in the existing defensive techniques of XSS attacks, a novel XSS defensive technique is required which fulfills all the following requirements:

- Automated process to distinguish between legitimate JavaScript code from a malicious injected code.
- Efficient and fast policy checks are needed to optimize the speed of XSS attack detection capability.
- Placement of Context Sensitive Sanitization routines is required to be introduced in the source code of web applications.
- XSS defensive solution should be capable enough to discover all browser-specific XSS attacks.
- The method must possess the updated white-list and blacklist of scripts to avoid the rate of false positives and false negatives.
- Powerful web crawler components must be utilized to execute the deep scanning of web pages to discover possible target links and user input forms.

## 7 Conclusion

The escalating utilization of the web paradigm for the advancement of omnipresent web applications all around the WWW is an opportunity for novel security vulnerabilities against the infrastructure after such web applications. The developers of the websites must be required to utilize the defensive strategies to assure protected coding practices, safe programming frameworks and secure skeleton of construction for the exploitation of protected web applications. However attackers keep on administering new technologies to take advantage of the loop-hole of web applications. The importance of such attacks can be reflected globally by the persistent existence of the web applications in different organizations like health care, banking, government administration, and so on.

Therefore, in this paper, we have presented a detailed comprehensive survey on one of the most serious vulnerability (i.e., XSS attacks). It is considered to be a serious infection for the modern web applications. XSS attacks permit an attacker to inject the malicious payload on the victim's web application resulting in information leakage, stealing of cookies, credit card numbers, passwords, etc. We have also presented the major concerns for web applications and Internet-based services which are persistent in several web applications of diverse organizations like banking, health care, financial service, retail, etc. We

have also discussed several state-of-art techniques based on XSS attacks, discovered their research gaps and discuss future scope.

## References


A Firefox PDF plug-in XSS vulnerability. http://lwn.net/Articles/216223/

Alexa Ranking Tool. http://developers.evrsoft.com/find-traffic-rank.shtml

Athanasopoulos E, Krithinakis A, Markatos EP (2010) Hunting cross-site scripting attacks in the network. In: W2SP 2010: web 2.0 security and privacy workshop

Avancini A, Ceccato M (2011) Security testing of web applications: a search-based approach for cross-site scripting vulnerabilities. In: 2011 IEEE 11th international working conference on source code analysis and manipulation, pp 85–94

Bisht P, Venkatakrishnan VN (2008) XSS-GUARD: precise dynamic prevention of cross-site scripting attacks. In: Conference on detection of intrusions and malware & vulnerability assessment

CAIDA analysis of Code-Red. http://www.caida.org/analysis/security/code-red/

Cao Y, Yegneswaran V, Possas P, Chen Y (2012) Pathcutter: severing the self-propagation path of XSS JavaScript worms in social web networks. In: Proceedings of the 19th network and distributed system security symposium (NDSS), San Diego, CA, USA

Choi JH, Choi C, Ko BK, Kim PK (2012) Detection of cross site scripting attack in wireless networks using n-Gram and SVM. Mob Inf Syst 8(3):275–286

Code-Red: a case study on the spread and victims of an Internet worm. http://www.caida.org/outreach/papers/2002/codered/codered.pdf

Cross-site scripting worm hits MySpace. BetaNews, 13 Oct 2005. http://www.betanews.com/article/CrossSite_Scripting_Worm_Hits_MySpace/1129232391

Flanagan D (2001) JavaScript: the definitive guide, 4th edn. O'Reilly, Sebastopol

Frenz C, Yoon J (2012) XSSmon: a perl based IDS for the detection of potential XSS attacks. In: Systems, applications and technology conference (LISAT), Proceedings of 2012 IEEE Long Island, pp 1–4, May 2012

Frenz CM, Yoon JP (2012) XSSmon: a perl based IDS for the detection of potential XSS attacks. In: 2012 IEEE Long Island systems, application and technology conference (LISAT), pp 1–4

Gundy MV, Chen H (2012) Noncespaces: using randomization to defeat cross-site scripting attacks. Comput Secur 31(4):612–628

Gupta S, Sharma L (2012) Exploitation of cross-site scripting (XSS) vulnerability on real world web applications and its defense. Int J Comput Appl 60:28–33

Gupta S, Gupta BB (2014) BDS: browser dependent XSS sanitizer. Book on cloud-based databases with biometric applications. IGI-Global's advances in information security, privacy, and ethics (AISPE) series. IGI-Global, Hershey, pp 174–191

Gupta S, Gupta BB (2015) PHP-sensor: a prototype method to discover workflow violation and XSS vulnerabilities in PHP web applications. In: Proceedings of the 12th ACM international conference on computing frontiers (CF'15), Ischia, Italy

Gupta S, Sharma L et al (2012) Prevention of cross-site scripting vulnerabilities using dynamic hash generation technique on the server side. Int J Adv Comput Res 2(5):49–54

Jim T, Swamy N, Hicks M (2007) Defeating script injection attacks with browser-enforced embedded policies. In: WWW'07: proceedings of the 16th international conference on World Wide Web, pp 601–610

Johns M (2006) SessionSafe: implementing XSS immune session handling. In: Proceedings of European symposium on research in computer security

Johns M, Engelmann B, Posegga J (2008) XSSDS: server- side detection of cross-site scripting attacks. In: Proceedings of the ACSAC, California, pp 335–344

Kallin J, Valbuena IL. A comprehensive tutorial on cross-site scripting. http://excess-xss.com/

Kals S, Kirda E, Kruegel C, Jovanovic J (2006) SecuBat: a web vulnerability scanner. In: 15th international World Wide Web conference (WWW), UK, May 2006

Kirda E, Kruegel C, Vigna G, Jovanovic N (2006) Noxes: a client-side solution for mitigating cross-site scripting attacks. In: SAC'06: proceedings of the 2006 ACM symposium on applied computing, pp 330–337

Klein A (2005) DOM based cross site scripting or XSS of the third kind. Technical report, Web application security consortium

Louw MT, Venkatakrishnan V (2009) Blueprint: robust prevention of cross-site scripting attacks for existing browsers. In: Proceedings of the IEEE symposium on security and privacy

MacDonald M, Szpuszta M (2005) Pro ASP.NET 2.0 in C# 2005, 1st edn. Apress, New York. ISBN 1-59059-496-7

Martin M, Lam MS (2008) Automatic generation of XSS and SQL injection attacks with goal-directed model checking. In: Proceedings of the USENIX security symposium (USENIX)

Meyerovich L, Livshits B (2010) ConScript: specifying and enforcing fine-grained security policies for JavaScript in the browser. In: Proceedings of the IEEE symposium on security and privacy

Nunan A, Souto E, dos Santos EM, Feitosa E (2012) Automatic classification of cross-site scripting in web pages using document based and URL based features. In: IEEE symposium on computers and communications (ISCC), pp 702–707

Putthacharoen R, Bunyatnoparat P (2011) Protecting cookies from cross site script attacks using dynamic cookies rewriting technique. In: 13th international conference on advanced communication technology ICACT2011, pp 1090–1094

Samy's cancelled MySpace profile. http://www.myspace.com/33934660

Shahriar H, Zulkernine M (2009) MUTEC: mutation-based testing of cross site scripting. In: Proceedings of the 5th international ICSE workshop on software engineering for secure systems. IEEE CS Press, Vancouver, pp 47–53, May 2009

Shaihriar H, Zulkernine M (2011a) S2XS2: a server side approach to automatically detect XSS attacks. In: Ninth international conference on dependable, automatic secure computing. IEEE, pp 7–17

Shaihriar H, Zulkernine M (2011b) Injecting comments to detect JavaScript code injection attacks. In: Proceedings of the 6th IEEE workshop on security, trust, and privacy for software applications, Munich, Germany, pp 104–109

Shar LK, Tan HBK (2012) Predicting common web application vulnerabilities from input validation and sanitization code patterns. In: IEEE/ACM international conference on automated software engineering, pp 310–313

Sharath Chandra V, Selvakumar S (2011) Bixsan: browser independent XSS sanitizer for prevention of XSS attacks. ACM SIGSOFT Softw Eng Notes 36(5):1

Slammed! Wired, July 2003. http://www.wired.com/wired/archive/11.07/slammer.html

SQL Slammer (computer worm). http://en.wikipedia.org/wiki/SQLSlammer

Technical explanation of the MySpace worm. http://namb.la/popular/tech.html

The spread of the Sapphire/Slammer worm. http://www.cs.berkeley.edu/~nweaver/sapphire/

Tiwari S, Bansal R, Bansal D (2008) Optimized client side solution for cross site scripting. In: 2008 16th IEEE international conference on networks, pp 1–4

Van-Acker S, Nikiforakis N, Desmet L, Joosen W, Piessens F (2012) FlashOver: automated discovery of cross-site scripting vulnerabilities in rich internet applications. In: ASIACCS'12: proceedings of the 7th ACM symposium on information, computer and communications security, pp 12–13

Vogt P, Nentwich F, Jovanovic N, Kirda E, Kruegel C, Vigna G (2007) Cross site scripting prevention with dynamic data tainting and static analysis. In: Proceeding of the network and distributed system security symposium (NDSS), San Diego, CA, February 2007

Wang S, Chang Y, Chiang W, Juang W (2007) Investigations in cross-site script on web-systems gathering digital evidence against cyber-intrusions. In: Future generation communication and networking (FGCN 2007), vol 2, pp 125–129

Wang Y, Li Z, Guo T (2011) Program slicing stored XSS bugs in web application. In: 2011 fifth international conference on theoretical aspects of software engineering, pp 191–194

Wassermann G, Su Z (2008) Static detection of cross-site scripting vulnerabilities. In: ICSE'08: proceedings of the 30th international conference on software engineering, pp 171–180

Weinberger J, Saxena P, Akhawe D, Finifter M, Shin R, Song D (2011) A systematic analysis of XSS sanitization in web application frameworks. In: Proceedings of the European symposium on research in computer security (ESORICS), Leuven, Belgium

WhiteHat (2013) WhiteHat website security statistic report 2013. https://www.whitehatsec.com/resource/stats.html

Wurzinger P, Platzer C, Ludl C, Kirda E, Kruegel C (2009) SWAP: mitigating XSS attacks using a reverse proxy. In: ICSE workshop on software engineering for secure systems. IEEE Computer Society

XSS Worm on Renren Social Network (2009). http://issmall.isgreat.org/blog/archives/2

Zhang Z, Wang Z (2010) A static analysis tool for detecting web application injection vulnerabilities for ASP program. In: 2nd international conference on e-business and information security (EBISS), pp 1–5

Zhang Q, Chen H, Sun J (2010) An execution-flow based method for detecting cross-site scripting attacks. In: 2nd international conference on software engineering and data mining (SEDM), pp 160–165. IEEE

Zhenyu Q, Jing X, Baoguo L, Fang T (2007) MBDS: model-based detection system for cross site scripting. In: IET conference on wireless, mobile and sensor networks, pp 849–852