CROSS-SITE SCRIPTING ATTACKS AND THE SECURITY OF WEB APPLICATIONS

Ass. Prof. Dr. Petar Halachev
Department of Informatics
University of Chemical Technology and Metallurgy – Sofia, Bulgaria
x3m@mail.orbitel.bg

Abstract: *This report focuses on vulnerabilities on web-applications and web-sites from Cross-Site Scripting attacks (XSS). The different types of XSS attacks are examined: DOM-based, active and passive attacks. The spread of XSS attacks across platforms - government and financial institutions, transportation companies, hospitality and entertainment has been analyzed. Research and analysis of the security of corporate websites and their resistance to XSS attacks have been carried out. The basic guidelines for preventing valuable data theft and unauthorized access to websites and applications from XSS attacks are reviewed and systematized.*

Keywords: WEB-APPLICATION, WEB-SITE, CROSS-SITE SCRIPTING ATTACKS, VULNERABILITY, SECURITY

1. Introduction

Cross-Site Scripting (XSS) is a widespread attack that affects many websites and web applications. XSS vulnerabilities have evolved significantly in the recent years [1]. The number of the DOM-based vulnerabilities, mXSS vulnerabilities [2] and expression-language based XSS attacks increases [3].

Characteristic of the XSS attack is the implementation of malicious code on a page of a website visited by the user who interacts with the server of the attacker. The attacks are realized by luring the user to the infected site through: social engineering; waiting for the user to visit the site himself and more. The results of the XSS attacks are: redirecting users to malicious sites; theft of cookies or credentials; data modification; replacing links or displaying your own ad on the site in question; destruction of the site. Developers often overlook the dangers of XSS attacks because they usually do not require specific user interaction, but only a visitation of the infected site.

The main purpose of the XSS attack is the theft of cookies by users in order to further obtain information and use it for subsequent attacks. The attacker does not attack the user directly but through the vulnerability of the website he is visiting by implementing JavaScript code. XSS executable code is usually written in popular programming languages like JavaScript, Vbscript, and more. The user sees this code in the browser as a part of the site. The visited resource is a conduit for the XSS attack.

If the attacker hits the administrator's cookies, they can access the control panel of the site or to its contents.

A number of popular sites - Facebook, Twitter, MySpace, eBay and Google can be targeted by such attacks.

Compared to SQL injections, the XSS attack is safe for the server but a threat to users of infected pages.

While SQL injection attacks database information from the back end, XSS attacks focus on stealing data from the front end of the website [4]. According to OWASP, the following syntax can be used to perform an XSS attack to steal cookie data, if validation of input data is not used:

```
<SCRIPT type=”text/javascript”>
Var adr = ‘../evil.php?cakemonster=’ + escape(document.cookie);
</SCRIPT>
```

According to P. Wurzinger et al. technically, XSS attacks leverage insufficient input/output validation in the attacked Web application to inject JavaScript code, which is then executed on the victim’s machine within the exploited Web site’s context, thus bypassing the same origin policy. The attacker can craft the injected script such, that it discloses the victim’s confidential information, e.g., a session ID. Then, by hijacking the session, the victim can be impersonated. Also, XSS enables the construction of very powerful phishing pages, since the page contentis actually deliveredby the correct, trusted site [5].

2. Distribution of XSS attacks

Positive Technologies publishes on June 26, 2019, a study of the purpose and distribution of hacking attacks against web applications and sites in various sectors [6]. The main goals of the attacks are: spreading malware, stealing data, posting advertisements and forbidden information, fraud or intrusion. The analysis of statistics on hacking attacks in different sectors could assist security professionals in assessing the risks to which corporate websites are exposed (***Fig.1***).

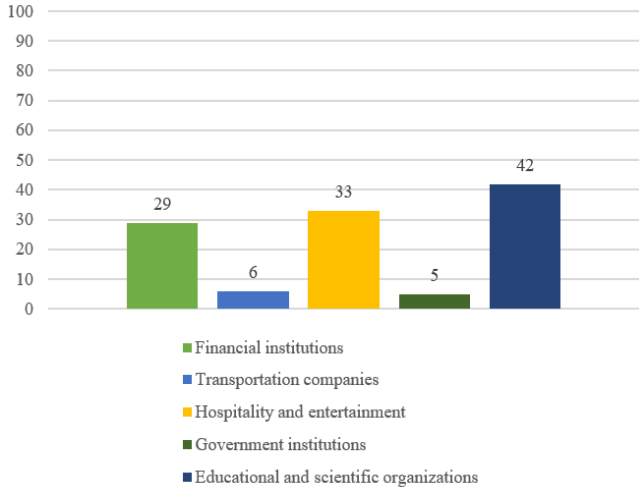


Fig. 1*. Distribution of XSS attacks*

Financial institutions. The XSS attack is one of the most common web application attacks in 2018 - 29% of all attacks. Through e-banking, consumers can manage their finances: paying bills; savings; obtaining loans; transfers and more. In some cases, the bank's official website is also used to distribute malware or stage phishing attacks, and the bank's customers are amongst the first vulnerable.

Transportation companies. XSS attacks are around 6% of the total number of attacks. Sites of transport companies contain forms for registration with personal data of customers, support online payments for reservations and ticket purchase, etc. An example is the theft of customer data by British Airways in August and September 2018. Attackers modify the script on the company's website by adding their own code (called JS sniffer), leading to data theft, including card information of approximately 380,000 customers .

Hospitality and entertainment. About 33% of all attacks against these sites are XSS. Hotel websites contain reservation forms on which the customer enters their personal details and payment information. In this sense, all customer-oriented web applications and their user data should be treated as a valuable asset.

Government institutions. Cross-Site Scripting and SQL Injection are typical attacks on government sites - about 5% of all

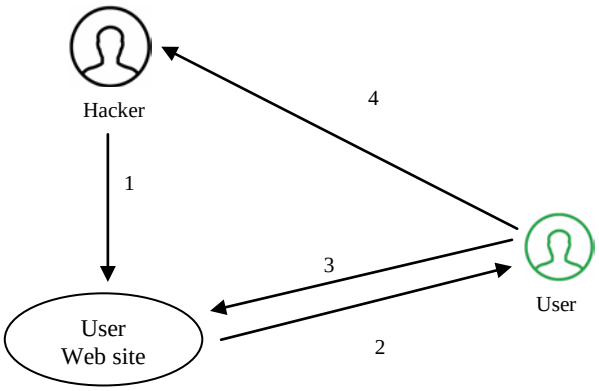
attacks are XSS. In 2018, remote code execution attacks were implemented to gain control of the server. According to Positive Technologies, there are also attacks designed to retrieve information about web applications, as well as many attempts to access .svn or .git directories that store the current source code of the application.

Educational and scientific organizations. About 42% of all attacks against these sites are XSS. Not only the personal data of the students and employees of the organizations, but also information about new studies and research is being attacked.

3. Implementation of XSS attacks

Implementing the XSS attack and implementing malicious JavaScript code is only possible in the user's browser, so the site he visits should be vulnerable. To launch an attack, the attacker initially checks the XSS vulnerability resources through automated scripts or manual search. These are usually standard forms that can send and receive requests (comments, searches, feedback, etc.). Pages with input forms are collected and each one of them is scanned for vulnerability. For example, if there is a "search" on the site's page, to confirm the XSS vulnerability, the following query is sufficient:

<script>alert("cookie: "+document.cookie)</script>



1 – Inject with script
2 – Visit
3 – Inject script
4 - Malicious action

Fig.2. *Cross-Site Scripting attack*

If a message appears on the screen, there is a risk of security breaks. Otherwise, the system displays a page with the search results. Popular CMS systems don't have such problems, but because of the ability to extend their functionality through modules and plugins created by external developers, the risk of XSS attacks is increasing (in systems such as Joomla, Drupal, Bitrix, WordPress, etc.). Most of the XSS vulnerabilities are tested in Internet Explorer.

Another possible option for finding vulnerabilities is to use pages that process GET requests. If we have a request of the type:

http://example.com/search?page=8

In the address bar, instead of the identifier (8), a script is added

"><script>alert("cookie: "+document.cookie)</script>,

Resulting in such an address:

http://example.com/search?page= "><script>alert("cookie: "+document.cookie)</script>.

If there is an XSS vulnerability on the webpage, a message will appear on the screen in the same way as in the first case. There are a huge number of tools, scripts and queries available to look for site vulnerabilities, and if none of them shows that the site is vulnerable, then the resource is protected from such attacks. In ***Fig. 2*** is a schematic of a cross-Site Scripting attack.

4. Classification of XSS Attacks

The XSS attack can be classified into three major types (persistent XSS, non-persistent XSS, and DOM-based XSS [7], [8].

-DOM based XSS - when the injected data remains in the browser and modifies the DOM environment. In this model it is possible to use both reflected XSS and stored XSS. Q Zhang et a. notes that DOM-based XSS uses some DOM object operations of normal Ajax applications to attack user, entirely on client side. [9] DOM based XSS is implemented as follows:

1. The attacker creates in advance a URL that contains malicious code and sends it via email or otherwise to the user.
2. The user follows this link, the infected site accepts the request and executes the malicious code.
3. A code is executed on the user's page, and as a result a malicious script is loaded and the attacker receives cookies.

-Reflected (non-persistent) XSS – According to V. Malviya In reflected XSS attacks the injected code doesn't reside on the web server [10]. The malicious code executes as soon as it is injected and acts as a user request to the infected web site:

1. The attacker creates a URL link in advance that contains malicious code and sends it to the user.
2. The user sends the URL request to the site by following the link.
3. The site automatically takes data from malicious code and places it as a modified URL response to the user's request.
4. As a result, the malicious script contained in the response is executed in the user's browser, and the attacker receives all cookies of that user.

-Stored (persisted) XSS - data is stored by the application and retrieved later in another context, e.g. user profile. One of the most dangerous threats is that it allows the attacker to gain access to the server and to manage malicious code (delete or change the code). Persistent/Stored XSS attack – The attacker injects the code into an input field which lacks proper data validation and sanitization system and which returns back the injected code to the same page or to another page of the web server [11]. Each time somebody logs in the site, a pre-loaded code is executed that works in automatic mode. Such vulnerabilities most often affect forums, portals, blogs where HTML comments are available without restriction. Malicious scripts can be easily embedded in text, in photos, or in pictures.

-Multi-step XSS - when the user have to perform some action on the application (most often in the navigation panel). Within multi-step exploits a vulnerability can be escalated to amore severe vulnerability. Detecting second-order ulnerabilities is crucial to improve the security of web applications [12].

-mXSS attack [13] is characterized by the attacker injecting code that looks secure but is rewritten and modified in the browser while processing the HTML structure of the page [14]. An attack of the type mXSS is difficult to detect and sanitize by the logic of the web site. For example, when changing the order of closing quotes or adding quotes to non-quotation parameters (CSS font-family font).

While the first three categories of XSS threats are well identified by automated Penetration testing tools such as Web application vulnerability scanners [15], Multi-step XSS is challenging for developers [16]. On the one hand, manual testing for XSS attacks becomes more difficult as the number of websites increases and they become more complex. On the other hand, modern automated web application threat detection techniques can test a large percentage of technical threats, but they are limited to evaluate web applications because they lack "any knowledge about the functional behavior and business logic of the application" [17].

There are two basic types of XSS attacks, of the mechanism of the performance [18], [19], [20].

A passive XSS attack requires specific action from the subject of the attack. In order the "malicious code" to be executed, the user

must follow a link. Social engineering is used for this purpose, such as sending an email with an appeal to follow the link or click on a specific area of the site. When the user clicks on the desired object, the malicious script is started. If the user is inactive, the code will not be activated. This type of attack is more difficult to perform because not only technical but also psychological knowledge is required.

With active XSS attacks, the attacker does not need to lure the user through special links, since the code is embedded in the databases or in an executable file on the server. No user activity is required. Typically, an event manager is installed in the input forms, which will activate automatically when you reach this page. As a result, users who click on this link fall victim to the attacker. The attacker is trying to find a vulnerability in the site filter. Using a combination of tags and symbols, the attacker creates a request to the server and, once a "security hole" is detected, malicious code is attached to the request to steal a cookie. An example of such a script is:

Img = new image()
Img.src = http://site.gif?+document.cookie

5. Results and discussions

Researching corporate websites for XSS attacks and guidelines for preventing them

The objects of this study are corporate web sites. Social engineering methods could embed a link with malicious code to the site's hosting and leak personal data, change the site's appearance, introduce its own ad via Javascript, and replace real links with malicious ones.

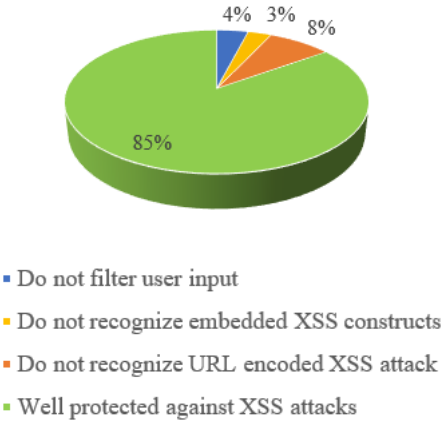


Fig. 3. *Sites vulnerable to XSS attacks*

A small percentage of sites, approximately 4%, do not filter user input, 3% do not recognize embedded XSS constructs, and 8% do not recognize URL encoded XSS attack. Most of the sites, about 85%, are well protected against XSS attacks (***Fig. 3***).

To quickly scan the site for XSS vulnerability special services that scans the page automatically can be used. All URLs where user data is submitted (comment forms, feedback, search) must be verified. For example, http://xss-scanner.com could be used, as well as other similar tools. These services do not provide a full guarantee of safety, and manual verification is advisable, excluding all dangerous special characters and replacing them with safe <,>, `,' which contain all html requests and tags reserved by the language.

For example, the following code can be used to quickly filter and automatically change special characters <,>, `, ':

```
var lt = /</g,
    gt = />/g,
    ap = /'/g,
    ic = /"/g;
userData = userData.toString()
                   .replace(lt, "&lt;")
                   .replace(gt, "&gt;")
                   .replace(ap, "&#39;")
                   .replace(ic, "&#34;");
```

IBM recommend to protect an organization's network from XSS attacks, administrators can do the following [21]:

-Sanitize or validate input;

Theodoor Scholte et al. consider input validation as an additional layer of defense against XSS [22]. Further J. Snehi1 and Dr. R. Dhir considers that the best way to eliminate XSS vulnerabilities is to use defensive coding practices that validate and sanitize inputs to ensures that user inputs conform to a required input format. Replacement methods search for known bad characters and replace them with non-malicious characters and removal methods removes them. [23]

-Perform a thorough code review to identify possible vulnerabilities; Amiangshu Bosu et al. define steps to identify the vulnerable code thorough audit by thoroughly auditing the code [24];

-Disable HTTP trace, which can enable attackers to steal cookie data;

-Users can click on URLs directly when they have doubts about links in websites, emails or messages

One of the basic rules for protecting against XSS attacks is the useage of filters. In the study of corporate sites, almost all were protected, but there were sites that did not use any filtering of the data received (in PHP). Python frameworks (Flask, Django) have filters built in and they just need to be turned on.

The minimal filter protects against amateur attacks, but more serious site protection is required, with more detailed data filtering. The filter must be designed to recognize embedded structures. For example, in embedded construction, malicious JavaScript is placed at the lowest level. The filter blocks the upper level, but the lower one is executed.

When submitting data with incorrectly closed brackets, for example:

>>>>><script{(alert("cookie: "+document.cookie))} ,

the filter sees this and tries to close them, but the embed code executes. This query not only checks the filter for a different number of parentheses, but also determines how the filter responds to different characters, whether it blocks them or skips them. In the survey, some sites did not filter this type of attack.

Another vulnerability is related to the img tag. This tag has many parameters, including dynsrc, which may contain javacsript. This tag must be filtered. If photos will not be used on the site, it is better to exclude them.

Usage example:

<img src="http://example.com/picture.jpg"
dynsrc=" javascript:alert(document.cookie)">

When building a filter, it is necessary to consider the possibility of encoding attacks. There are a number of encryption programs that encrypt the attack so the filter cannot recognize it. In this case, it is necessary to use the filter decryption algorithm before the program executes the request code.

The example above can be encoded in URL notation as follows:

%3Cimg%20src%3D%22http%3A%2F%2Fexample.com%2Fpicture.jpg%22%20%0Adynsrc%3D%22%20javascript%3Aalert%28document.cookie%29%22%3E%0A

Encoding is needed not only to bypass the filter, but also for social engineering. The encoded code can be sent as a link. It is unlikely that anyone will check it. Thus, the attack is encrypted and the filter cannot recognize it. Therefore, it is necessary to use a filter decryption algorithm before the program executes the request code.
It is necessary the employees of the organization to be periodically instructed and informed about the rules for use of the Internet and the risk of XSS attacks, namely: not to open suspicious links; hosting and network administrators to check encrypted links and explore websites for vulnerability using JavaScript.

6. Conclusion

This report examines the nature of the XSS attack, the different types of attacks, and the vulnerability of web applications and sites. A study has been conducted on the stability of corporate sites against XSS attacks. As a result of the security assessment, it was found that most of the surveyed corporate sites, about 85%, were protected by XSS attacks. A certain percentage of the sites surveyed are not well protected and could not withstand XSS attacks, as not all site owners want to invest resources and efforts in enhancing their security.

Following some basic rules could increase the security of web applications from XSS attacks.

The validation of the data entered by the user must be done both from the side of the web server and from the web browser.

Use secure connection via https certificates when user input is allowed on the website. According to A. Neagos turning off the HTTP TRACE can prevent the stealing stealing of cookies [25].

When using popular CMS - Wordpress, Bitrix, Joomla and others, a recent version of the kernel and all installed modules and plugins are required. The most common site management systems are protected by XSS attacks, but the addition of external (additional) plugins from untested sources may contain vulnerabilities.

7. References

[1] Lekies S; Kotowicz, Krzysztof; Groß, Samuel; Nava, Eduardo Vela; Johns, Martin (2017). "Code-reuse attacks for the Web: Breaking Cross-Site Scripting Mitigations via Script Gadgets", CCS 17, October 30 - November 3, 2017, Dallas, TX, USA, pp. 1709-1723

[2] Heiderich M., Schwenk J., Frosch T., Magazinius J., Jang E. Z. mXSS attacks: tacking well-secured web-applications by using innerHTML mutations. In Proceeding of the 2013 ACM SIGSAC conference on Computer and communications security (2013) ACM, pp. 777-788

[3] Heiderich M., Mustache security wiki (online) https&//github.com/cure53/mustache-security.2014

[4] https://keirstenbrager.tech/sql-vs-xxs-injection-attacks-explained/

[5] Peter Wurzinger, Christian Platzer, Christian Ludl, Engin Kirda, Christopher Kruegelk, SWAP: Mitigating XSS attacks using a reverse proxy, Secure Systems Lab - Technical University Vienna, Institute Eurecom France, University of California, Santa Barbara, IWSESS '09 Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems

[6] https://www.ptsecurity.com/ww-en/analytics/web-application-attacks-2019/

[7] M. K Gupta, M. C. Govil, and G. Singh, "Static analysis approaches to detect SQL injection and cross site scripting vulnerabilities in web applications: a survey," in Proceedings of the Recent Advances and Innovations in Engineering (ICRAIE), 2014, pp. 1-5

[8] Vernotte A., Dadeau F., Lebeau F., Legeard B., Peureux F., Piat F. (2014) Efficient Detection of Multi-step Cross-Site Scripting Vulnerabilities. In: Prakash A., Shyamasundar R. (eds) Information Systems Security. ICISS 2014. Lecture Notes in Computer Science, vol. 8880. Springer, Cham DOI https://doi.org/10.1007/978-3-319-13841-1_20

[9] Qianjie Zhang, Hao Chen, Jianhua Sun, The 2nd International Conference on Software Engineering and Data Mining, 2010

[10] Vikas K. Malviya, Saket Saurav, Atul Gupta, 2013 20th Asia-Pacific Software Engineering Conference (APSEC)

[11]Cross-site scripting, Wikipedia, http://en.wikipedia.org/wiki/Cross-site_scripting

[12] Johannes Dahse, Thorsten Holz, Static Detection of Second-Order Vulnerabilities in Web Applications, USENIX Security Symposium, 2014

[13] Mario Heiderich, Jörg Schwenk, Tilman Frosch, Jonas Magazinius, Edward Z. Yang, mXSS Attacks: Attacking well-secured Web-Applications by using innerHTML Mutations, 20th ACM Conference on Computer and Communications Security (CCS), Berlin, Germany, November 2013

[14] Wang, Yi-Hsun, Ching-Hao Mao, Hahn-Ming Lee, Structural Learning of attack vectors for generating mutated XSS attacks, arxiv preprint arxiv: 1009.3711 (2010)

[15] Bau, J., Bursztein, E., Gupta, D., Mitchell, J.: State of the Art: Automated Black-Box Web Application Vulnerability Testing. In: Proc. of the 31st Int. Symp. on Security and Privacy (SP 2010), pp. 332–345. IEEE CS, Oakland (2010)

[16] Doupé, A., Cova, M., Vigna, G.: Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners. In: Kreibich, C., Jahnke, M. (eds.) DIMVA 2010. LNCS, vol. 6201, pp. 111–131. Springer, Heidelberg (2010)

[17] Vernotte A., Dadeau F., Lebeau F., Legeard B., Peureux F., Piat F. (2014) Efficient Detection of Multi-step Cross-Site Scripting Vulnerabilities. In: Prakash A., Shyamasundar R. (eds) Information Systems Security. ICISS 2014. Lecture Notes in Computer Science, vol 8880. Springer, Cham DOI https://doi.org/10.1007/978-3-319-13841-1_20, pp. 359

[18] https://testmatick.com/software-testing-glossary/cross-site-scripting/

[19] Smith M. A., Web application Security: XSS Attacks, Kansas State University, CIS 726, pp. 1-2, http://people.cs.ksu.edu/~mas3773/cis726/report.pdf

[20] Shailendra Rathore, Pradip Kumar Sharma, Jong Hyuk Park, XSSClassifier: An Efficient XSS Attack Detection Approach Based on Machine Learning Classifier on SNSs, Journal of Information Processing Systems, Vol.13, No.4, pp.1014-1028, August 2017, https://doi.org/10,3745/JIPS.03.0079

[21] IBM MSS, Cross-Site Scripting (XSS) Research and Intelligence report, Release data& December 15, 2014 by& Nikita Gupta, p.5

[22] Theodoor Scholte, William Robertson, Davide Balzarotti, Engin Kirda, SAC 2012 Proceedings of the 27th Annual ACM Symposium on Applied Computing, Pages 1419-1426

[23] Jyoti Snehi1, Dr. Renu Dhir, Jalandhar, India, International Journal of Computers & Technology Volume 4 No. 2, March-April, 2013, ISSN 2277-3061

[24] Amiangshu Bosu, Jeffrey C. Carver, Munawar Hafiz, Patrick Hilley, Derek Janni, Identifying the characteristics of vulnerable code changes: an empirical study, FSE 2014 Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering Pages 257-268

[25] Adriana Neagos, Simona Motogna, Security Analysis Regarding Cross-Site Scripting on Internet Explorer, BCI, 2012, Citeseer