Research of the Possibilities of Conducting XSS-attacks and Methods of Countering It

Pavel Razumov[a], Larissa Cherckesova[b] and Olga Safaryan[c]
Don State Technical University, Gagarina sq. 1, Rostov-on-Don, Russia

Keywords: XSS Attack, XSS Vulnerabilities, Cross-Site Scripting, Application Layer, Malicious Script, Exploits Vulnerabilities, Dangerous Code, Virus Code, Preventive Protection Methods, Application Security.

Abstract: One of the most popular types of attacks these days is cross-site scripting using JavaScript. In addition to the problems caused by illegal use of JavaScript, I will discuss security rules that will help prevent a possible XSS attack, as well as provide research related to checking sites for organized website security. In this paper, we have proposed some algorithms that support scanning XSS vulnerabilities, used to create an automated scanning program that will reduce the risk of using Internet resources.

1 INTRODUCTION

One of the most popular attacks these days is cross-site scripting using JavaScript. In addition to the problems caused by illegal use of JavaScript, the security features of the XSS attack related to checking sites for organized website security were prevented (Common Weakness Enumeration).

In this paper, I have proposed some algorithms that scan for XSS vulnerabilities, used to create an automated scanning program that reduced the risk of using Internet resources (Cherckesova, 2020). Research object: web application.

Research subject: search for XSS vulnerabilities and counteraction to them.

The aim of this work is to create a proprietary system for automated testing of web applications for XSS vulnerabilities, capable of combining a high level of coverage and detection, as well as ease of use. An integral part of creating a new system is the use of systems that fulfill similar goals.

The program should be operated by web application developers to control the security level of the project. End users are testers or web application developers.

In this work, the analysis and classification of existing XSS vulnerabilities will be carried out, as well as the algorithm of the program and its implementation will be described.

2 RESEARCH METHODOLOGY

The main task of cross-site scripting is to get user cookies using a script built into the server and then fetch the necessary data using them for the next attacks. The attacker does not attack users directly, but exploits vulnerabilities in the website visited by the victims and injects a special script. In the browser, this code does not appear to stand out in any way, but at the same time this site becomes an accomplice in the attack.

Usually, to exploit such vulnerabilities, contact with the user is necessary: either they are attracted to a hacked site using social engineering, or they are waiting for the moment until they go to this site by themselves.

Web programmers often do not pay due attention to the problems associated with XSS vulnerabilities, but if you do not get rid of them in a timely manner, this can cause global problems and cause irreparable harm to the security of the system. It is important to consider that if the administrator's cookies get to a

[a] https://orcid.org/0000-0003-2454-3600
[b] https://orcid.org/0000-0002-9392-3140
[c] https://orcid.org/0000-0002-7508-913X

Razumov, P., Cherckesova, L. and Safaryan, O.
Research of the Possibilities of Conducting XSS-attacks and Methods of Countering It.
DOI: 10.5220/0010617400003170
In *Proceedings of the International Scientific and Practical Conference on Computer and Information Security (INFSEC 2021)*, pages 54-60
ISBN: 978-989-758-531-9; ISSN: 2184-9862

hacker, he will be able to gain access to the control panel of the site and its contents.

XSS attacks were the most widespread in the past year and the situation has not changed at the present time. In this connection, it should be concluded that cross-site scripting is relevant, but, unfortunately, not enough efforts are being made to solve this problem, and scammers are coming up with more and more new schemes for its implementation, which are harder to deal with every year (Frazer Howard).

2.1 Analysis and Classification of Existing XSS Types

How Cross-Site Scripting Works.

Cross-site scripting (XSS) is one of many attacks on web systems, affecting many web applications and is one of the most common types of hacker attacks at the application layer. In English, the term sounds like Cross-Site Scripting, but it has the abbreviation XSS, so as not to be confused with cascading style sheets, which translate as Cascading Style Sheets (Kozlov, 2010).

This attack allows a hacker to inject malicious code into a specific page of the site so that the victim's Internet browser, when the page is launched, will launch this code.

With the help of cross-site scripting, it is possible to issue modified data, replace links, both visible and hidden, or display your own advertisements on the affected resource. If a hacker can find even one XSS vulnerability, then cross-site scripting will work (Fogie, 2007).

Many websites store their users' data in a database (DB) so that they can be displayed as they are entered. The peculiarity of such attacks is that the virus code can use the authorization form of the site visitor to obtain extended access or user authorization data in the web system. Dangerous code can be embedded into a web page not only through a vulnerability on the user's computer, but also through a vulnerability in the server.

The aforementioned attacks are often made possible by malicious scripts written in JavaScript (JS), with the help of which information is subsequently introduced from third-party sources. The scheme is built in such a way that the malicious code gets into the database, where the username and password of the visitor are located. Thus, when the site displays the user's name on the web page, this code will be executed. This code can do almost anything, given certain conditions, as a result of which the threat becomes quite realistic.

Many website owners neglect to protect against such threats, believing that they cannot be used to steal confidential data that is on the server. This is a common mistake, since a page or cookie can contain quite sensitive data, and where the safety of a cross-site request forgery is at risk, a hacker will be able to perform all actions available to the user. For the operation of the application, and the business itself, the results of XSS attacks can cause significant damage. More than once, many popular websites such as Vkontakte, Facebook, Google, Mail have become victims of such attacks (Semenov).

Consider a standard example of an XSS attack:

- a hacker enters a script into the URL of an active online store;
- the script directs the user to the replaced malicious page;
- a script is executed on the page that gets the value of the visitor's cookies;
- then the necessary information is sent to the hacker, who uses it to intercept the user's session.

Despite the fact that the store was not harmed, the hacker using the vulnerability fraudulently obtained the confidential information of visitors and took control of their sessions. Also, there is an option to create a fake URL by encoding a part of it using any encoding method, which will make it unattractive.

The user will not be suspicious at the sight of the familiar URL and will not pay attention to the subsequent encoded part. Thus, fraud occurs on the Internet (Mitchell).

General classification of XSS attacks.

Not all vulnerabilities are the same, there are many types. Let us consider the classification of such attacks and analyze according to two criteria: in the direction of impact and in the method of impact.

Let's look at and analyze Figure 2 below, where you can visually consider how to use cross-site scripting.

According to the method of impact, attacks are subdivided into active and passive:

- Passive (autonomous) requires the direct intervention of the attacker. The victim needs to do a specific action to call the event handler and load the malicious script in a prepared form. In this case, you can resort to social engineering, for example, send an e-mail with a proposal to follow the link and click on a specific area of the site. When the user completes all these actions, a malicious script will be launched. In case of inaction of the victim, the code will remain inactive.

Example:

<b href = “b” onmauseower = alert (1561) style = ”frontsize: 600pc”>

In this example, we consider a hyperlink that catches the eye of a site visitor, and has a long designation, in connection with which, the chance of clicking on it increases. This link has a hover event handler that contains the attack code. This attack is passive.

Of course, the links provided by hackers do not look like:

http://www.haker.ru/searh?q=ans><script>alert “+ document.cookies) </script>

There are many different ways to encode the content of a link so that it does not look suspicious.

You can encode in base64, hex, or use a third-party server for routing.

Example: data: text / html; base64, aHR0NoLbWw / hYUe3GlcnQoJycpPC9zY3JpcHQ +

- Active XSS attack. The hacker does not need to urge potential victims to go to prepared links, because the code is entered into the database or into a specific active file on the server. Input forms have an event handler installed that will be automatically activated when entering a web page. As a result, all visitors will become victims of cyber fraud.

Example:

<input type = kkkkkk value = a onfocus = alert (2571) AUTOFOCUS>

The above is an input field with automatic focus appearance, which executes the prepared attack code. The focus is set so that its set handler is called. The attack is active and is carried out automatically.

According to the impact vector, XSS attacks are divided into three types: reflected, stored, and based on the DOM model.

- Reflected ("Reflected XSS" - non-persistent). The most common type of attack that requires the user to take certain actions.

This type of attack will work if a user clicks on a specially prepared link that sends a request to a website containing a vulnerability. This vulnerability is often the result of insufficient filtering of incoming requests. All this allows you to manipulate functions and activate malicious scripts to steal data.

If angle brackets are not escaped on the site, then you should convert them to "& lt;" and "& gt;" to get the script on the search results page.

The principle of such an attack can be seen in Figure 1.

Various communication channels are used to distribute the malicious link: e-mail, comments on the site, messages on social networks. Anchor links are more likely to provoke users to click.

Figure 1: Reflected XSS attack.

Next, let's look at an example of exploiting a reflected XSS attack.

For demonstration purposes, let's try to exploit this vulnerability using the example of DVWA.

Damn Vulnerable Web Application (DVWA) is a web application that is intentionally filled with vulnerabilities. The main goal is to help security professionals test their skills and tools without crossing the line of law, to help web developers better understand the process of web application security, and to help both students and teachers learn about web application security. The goal of DVWA is to practice some of the most common web vulnerabilities, with variable levels of difficulty, with a simple and understandable interface for each (Kuznetsov, 2010).

In this project, there are both documented and undocumented vulnerabilities. This is done in order to be able to detect as many vulnerabilities as possible.

Figure 2 shows code that is vulnerable to a reflected XSS attack.

```
<?php

    if(!array_key_exists("name",$_GET) | |$_GET['name'] == NULL ||
$_GET['name']==''){

    $isempty=true;

    }

    else{

    echo '<pre>';

    echo 'Hello' . $_GET['name'];

    echo '</pre>';

    }

    ?>
```

Figure 2: Example of code vulnerable to reflected XSS attack.

As you can see from the example, data is not cleared for the "name" parameter before it is

displayed in the user's browser. Thus, if you inject a JavaScript script into it, this script will be executed.

Let's use the DVWA application to demonstrate this vulnerability. The results are presented below, in Figure 3 and in Figure 4.

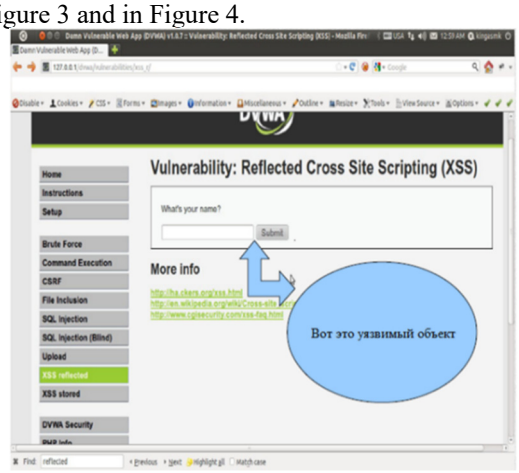

Figure 3: Exploiting Reflected XSS.

Next, inject the code "<script> alert (" xss "); </script>" into the form element:

Figure 4: Result of operating reflected XSS.

- Stored ("Stored XSS" - persistent). For a stored attack to be successful, an attacker needs to find a vulnerability on the site.

3 RESEARCH RESULTS

The program starts its work in parallel with the loading of the site page. Two processes are launched at one moment. The first method looks for vulnerabilities on the current page, and the second method checks all links on the current page, checks against the list of already checked ones, and marks the malicious ones with a marker (Mitchell and Zhukov, 2013).

The algorithm for checking a page for vulnerability consists of 7 mandatory steps.

1. In the DOM-model of the page, elements with the <form /> tag are selected.

2. Inside each of them, look for the child <input /> elements.

3. For each found element, create a data object that includes the value of the name attribute of the <input /> tag, the value of the action attribute and the parent <form /> element. If the attribute is empty, we use the current URL, also enable the current protocol, it can be http or https and the current domain.

4. The array of collected objects is sent to the server.

5. The server generates a malicious link for each dataset and executes it before loading so that it is possible to check in a safe environment whether there will be a response from a potentially vulnerable page.

6. After verification, the status of the current domain is saved to the database and sends a visual element to the client, which will symbolize the status of verification.

7. If along with the data set came a flag that requests a detailed report, then an HTML layout of the report is attached in response to display it on the client's screen.

The described algorithm is also presented as a block diagram in Figure 5.

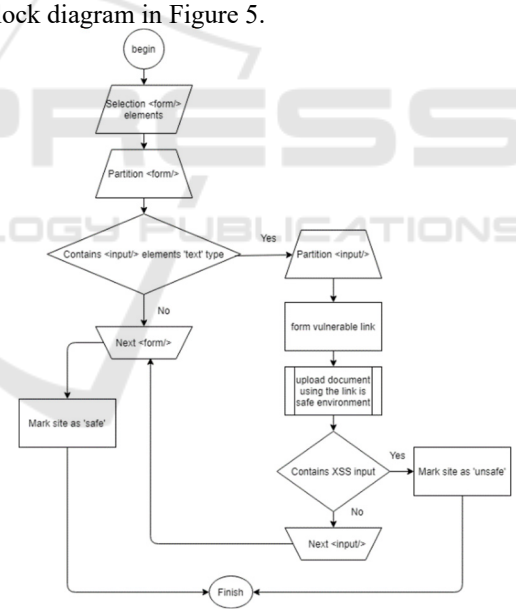

Figure 5: Block diagram of XSS-vulnerability search.

The preventive protection algorithm consists of 5 mandatory steps.

1. In the DOM-model of the page, links are selected, that is, elements with the <a/> tag that have a non-empty href attribute.

2. Each link is marked with an identifier like xss-mark-i, where I is the ordinal number of the link in the DOM model.

3. An array of links is sent to the server, where a selection is made against a table with already checked pages.

4. As a result, the server returns link identifiers that can be potentially dangerous according to the accumulated information.

5. Extension, using CSS, by identifiers from step 2, visually marks dangerous and vulnerable links. Letting the user understand that the link may be dangerous, and the information on it is distorted.

This algorithm is also presented as a block diagram in Figures 6 and 7.

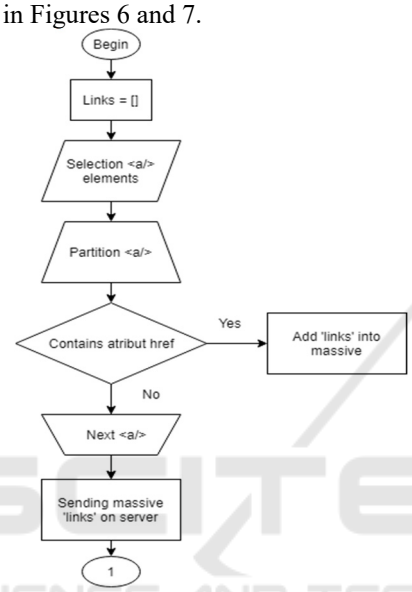

Figure 6: Block diagram of the search for links with XSS vulnerabilities.

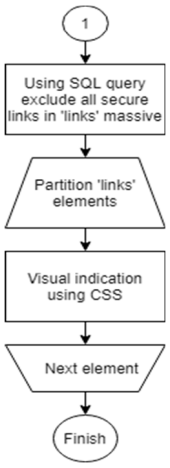

Figure 7: Block diagram of the search for links with XSS vulnerabilities.

Thus, using this extension, thanks to the accumulation of the results of checks and preventive protection methods, it is possible to accumulate a database of dangerous sites much faster and make the Internet safer.

In this chapter, the choice of the programming language and the target platform for the software product was justified, as well as the work with the database, which was implemented in the application, was considered.

Further, algorithms for checking a page for vulnerability and preventive protection were described, as well as their block diagrams were implemented.

4 RESULTS DISCUSSION

To assess the quality of the created software product, it was decided to compare the result of its work with products that are already on the market and are actively used by developers to test their own products: NetSparker (CommunityEdition), an extension for the XSS-Me browser, Wapiti.

Several projects were selected for testing:

- http://www.insecurelabs.org/ created with the support of OWASP to strengthen the skills of finding XSS vulnerabilities;
- http://www.insecurelabs.org/Task, a project consisting of six tasks, the main goal of which is to detect an XSS vulnerability.

Comparison of test results is shown in Table 1.

Table 1: Comparison of test results.

	NetSparker	XSS-ME	Wapiti	XSS checker
http://www.insecurelabs.org/				
Number of vulnerabilities found	2	7	9	12
Number of skipped forms	6/7	0/7	0/7	0/7
http://www.insecurelabs.org/Task				
Number of vulnerabilities found	5	18	6	16
Number of skipped forms	1/6	2/6	3/6	0/6

After analyzing the table, we can conclude that the application created within the framework of this work is a competitive product and, in some respects, surpasses the existing analogues.

4.1 Comparison with Analogues

To assess the quality of the product obtained, it is advisable to compare it by its main characteristics with similar products. Comparison with analogs is presented in table 2.

Table 2: Comparison with analogues.

	NetSparker (CE)	XSS-ME	Wapiti	XSS checker
GUI availability	+	-	+	+
Authorization in the system	-	-	+	-
Parallel work	+	-	+	+
Sitemap creation	-	+	-	+
Stored XSS	-	+	-	+
Cross-platform	-	+	-	-
Total	2	3	3	4

The table clearly reflects the main characteristics of existing solutions and the XSS checker application created within the framework of this work. According to the information from the table, the XSS checker has a number of advantages over existing solutions, which makes it at least a competitive product.

In this chapter, the structure of classes and their relationships was described, a demonstration of the program's operation at all stages of its use was given and its step-by-step installation was considered. A comparative analysis with analogs was also carried out.

5 CONCLUSION

Within the framework of this work, an attack on WEB-applications such as Cross Site Scripting was investigated. The basic principles of its work were studied and the classification was given.

A list of XSS injections was collected from various sources, through which it was decided to check the WEB application for XSS vulnerabilities.

In 2020, the company "We Are Social" conducted a study on the strength and sustainability of sites. As a result of the security assessment, it was found that most of the sites, namely 42%, are poorly protected from this type of attack. Based on this, we can say that many Internet resources are not credible and cannot withstand attacks. This is a gross defect that needs to be corrected, for the sake of security, both of the owner of the site and the data of the site itself, and of its users.

Unfortunately, not all website owners are willing to invest in improving website security. But every year, the severity of the law in relation to disclosure, leakage and damage to personal data is tightening, thereby forcing unscrupulous owners to better monitor the safety of their resources. The size of the fine for violation of the 6th part of Article 13.11 of the Administrative Code is increasing, and with it the protection of personal data is growing.

As part of the work, the existing systems were studied that allow automated testing of application security in the context of XSS vulnerabilities. The main advantages and disadvantages of these systems are revealed.

In accordance with the terms of reference, a multithreaded system for automated testing of a WEB application for Cross Site Scripting was created.

For the convenience of working with the application, a user graphical interface was developed using the JavaFX platform, which allows to inform the user about the incorrectness of the data entered by him, as well as to reflect the current state of the program.

Also, a module was implemented that generates a report on test results using a cascading style sheet and the JavaScript programming language.

To assess the quality of the application, two open source OWASP projects were tested. The results of this application were compared with the results of existing applications. The application developed as part of the qualification work showed an excellent result, in some respects outstripping existing solutions such as Wapti, NetSparker, XSS-Me.

REFERENCES

Common Weakness Enumeration. Official site. https://cwe.mitre.org

Cherckesova, L., Revyakina, Y., Safaryan, Boldyrikhin, N., Ivanov, Y. (2020) Possibilities of conducting XSS-attacks and the development of countermeasures. E3S Web of Conferences, 224, 01040

Flanagan, D. (2012). *Javascript. Detailed guidance*. Symbol-plus, 157 p.

Fogie S., Grossman, J., Hansen, R., Rager, A. and Petkov, Petko D. (2007). *XSS attacks: exploitation and protection*, Syngress, 464 p.

Frazer Howard, Modern Internet Attacks. http://help.yandex.ru/webmaster/?id=1076109

Kozlov, D. D. (2010). *Methods for detecting vulnerabilities in web applications*. - M: Publishing house of the

Faculty of Computational Mathematics and Cybernetics, Moscow State University, pages 146-158.
Kuznetsov, M. V. and Simdyanov, I. V. (2010). Object oriented programming in PHP. - SPb.: BHV-Petersburg, 608 p.
Mitchell, Ch. *Ensuring the security of websites.* http://help.yandex.ru/webmaster/?id=1071330
Semenov, Yu.A. *Overview of vulnerabilities, some types of attacks and protection tools.* http://book.itep.ru/6/intrusion.htm
Shelukhin O.I., Sakalema D. Zh. and Filinov A.S. (2013). *Detection of intrusions in a computer network*. M: Hotline-Telecom, 220 p.
Zhukov, A.I. and Grankov, M.V. (2013). *Using the SQL language to interact with modern DBMS: method. Instructions.* Rostov-on-Don: Publishing Center of DSTU, 203 p.