Network Security Project

Title - XSS Scripting

Indian Institute of Information Technology, Allahabad

Group Number:- 42

Student Details -

Name	Roll no.
Tushar Gupta	IIT2019043
Ankit Chauhan	IIT2019086
Saloni	IIT2019128
Jay Kumar Pal	IIT2019170
Vikram Singh	IIT20190213

Abstract:-[1][3]

This study discusses the XSS attack and incidence. In addition, the paper presents the necessary information of the XSS attack , how it works and prevention of XSS attack.

XSS Scripting, also called Cross-Site Scripting, is a type of injection where malicious scripts are injected into trusted websites. When malicious code, usually in the form of browser side script, is injected using a web application to a different end user, an XSS attack is said to have taken place.

Cross-site scripting presents one entry point for attackers to access and manipulate control systems networks. It takes advantage of Web servers that return dynamically generated Web pages or allow users to post viewable content in order to execute arbitrary HTML and active content such as JavaScript, ActiveX, and VBScript on a remote machine browsing the site within the context of a client-server session. This potentially allows the attacker to redirect the Web page to a malicious location, hijack the client-server session, engage in network reconnaissance, and plant a backdoor program.

Flaws which allow success to this attack are remarkably widespread and occur anywhere a web application handles the user input without validating or encoding it. A study carried out by Symantic states that more than 50% of the websites are vulnerable to the XSS attack. Security engineers of Microsoft coined the term "Cross-Site Scripting" in January of the year 2000. But even if it was coined in the year 2000, XSS vulnerabilities have been reported and exploited since the beginning of 1990's, whose prey have been all the (then) tech-giants such as Twitter, Myspace, Orkut, Facebook and YouTube. Hence the name "Cross-Site" Scripting.

This attack could be combined with other attacks such as phishing attack to make it more lethal but it usually isn't necessary, since it is already extremely difficult to deal with from a user perspective because in many cases it looks very legitimate as it's leveraging attacks against our banks, our shopping websites and not some fake malicious website.

The background details of the attack-

Cross-site scripting, often abbreviated as XSS, is a type of attack in which malicious scripts are injected into websites and web applications for the purpose of running on the end user's device. It mainly occurs when attackers inject their own code into a web page, typically accomplished by exploiting a vulnerability on the website’s software, attackers can then inject their own script, which is executed by the victim’s browser.
In many cases, XSS is performed in a more direct way, such as in an email message. An XSS attack can turn a web application or website into a vector for delivering malicious scripts to the web browsers of unsuspecting victims.

Depending on the goals, bad actors can use cross-site scripting in a number of different ways. Following are some types of XSS attack :

1: Stored (Persistent) Cross-Site Scripting

Stored cross-site scripting attacks occur when attackers stores their payload on a compromised server, causing the website to deliver malicious code to other visitors.

Since this method only requires an initial action from the attacker and can compromise many visitors afterwards, this is the most dangerous and most commonly employed type of cross-site scripting.

Examples of stored cross-site scripting attacks include the profile fields such as your username or email, which are saved on the server and displayed on your account page.

2: Reflected Cross-Site Scripting

Reflected cross-site scripting attacks occur when the payload is stored in the data sent from the browser to the server.

Examples of reflected cross-site scripting attacks include when an attacker stores malicious script in the data sent from a website's search or contact form.

A typical example of reflected cross-site scripting is a search form, where visitors sends their search query to the server, and only they see the result.

Attackers typically send victims custom links that direct unsuspecting users toward a vulnerable page. From this page, they often employ a variety of methods to trigger their proof of concept.

3: Self Cross-Site Scripting

Self cross-site scripting occurs when attackers exploit a vulnerability that requires extremely specific context and manual changes. The only one who can be a victim is yourself.

These specific changes can include things like cookie values or setting your own information to a payload.

4: Blind Cross-Site Scripting

Blind cross-site scripting attacks occur when an attacker can't see the result of an attack. In these attacks, the vulnerability commonly lies on a page where only authorized users can access.

This method requires more preparation to successfully launch an attack; if the payload fails, the attacker won't be notified.

To increase the success rate of these attacks, hackers will often use polyglots, which are designed to work into many different scenarios, such as in an attribute, as plain text, or in a script tag.

An example of a blind cross-site scripting attack would be when a username is vulnerable to XSS, but only from an administrative page restricted to admin users.

5: DOM-Based Cross-Site Scripting

DOM-based cross-site scripting attacks occur when the server itself isn't the one vulnerable to XSS, but rather the JavaScript on the page is.

As JavaScript is used to add interactivity to the page, arguments in the URL can be used to modify the page after it has been loaded. By modifying the DOM when it doesn't sanitize the values derived from the user, attackers can add malicious code to a page.

An example of DOM-based cross-site scripting attack would be when the website changes the language selection from the default one to one provided in the URL.

Impact of XSS vulnerabilities

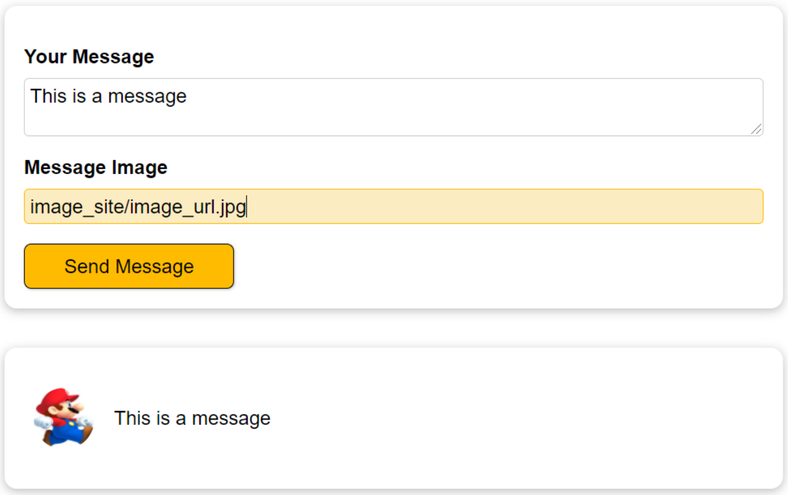
XSS can have huge implications for a web application and its users. User accounts can be hijacked, credentials could be stolen, sensitive data could be exfiltrated, and lastly, access to your client computers can be obtained. Attackers can impersonate or masquerade as the victim user and will be able to carry out any action that the user is able to perform. But the actual impact of an XSS attack generally depends on the nature of the application, its functionality and data, and the status of the compromised user.
For example:
In a brochureware application, where all users are anonymous and all information is public, the impact will often be minimal.
In an application holding sensitive data, such as banking transactions, emails, or healthcare records, the impact will usually be serious.
If the compromised user has elevated privileges within the application, then the impact will generally be critical, allowing the attacker to take full control of the vulnerable application and compromise all users and their data.

<u>Injection and Explanation of the Attack:-</u>[2][5]

So, In simple words we can say that XSS scripting or Cross-site scripting is a type of injection which is due to some malicious code injected to the website through different means and in this tutorial we are going to explain how XSS scripting works.

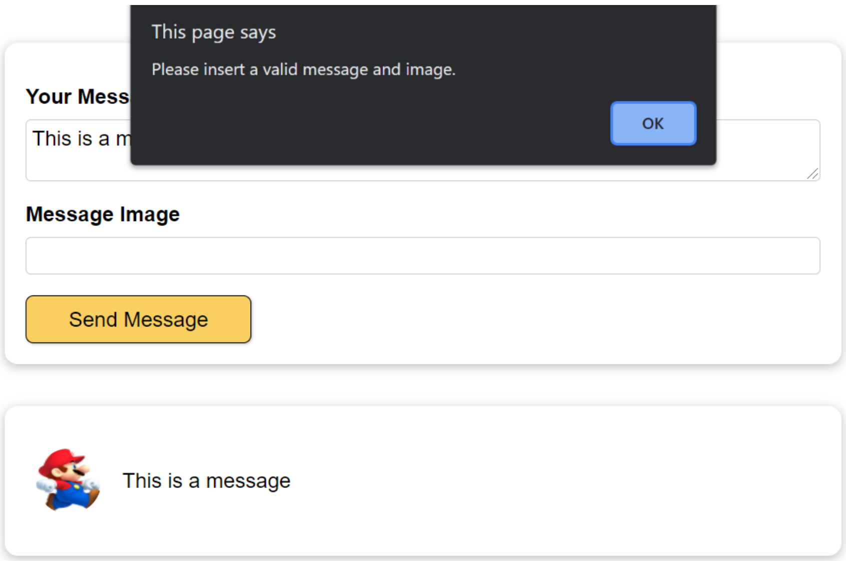
We have made our own simple web Page which we will use to describe the insertion, detection and prevention techniques for XSS scripting.

1) We have created a simple web Page which just takes two parameters: message and image url. In case any of the two parameters left empty the page will show an alert.



Img1:- Simple web page

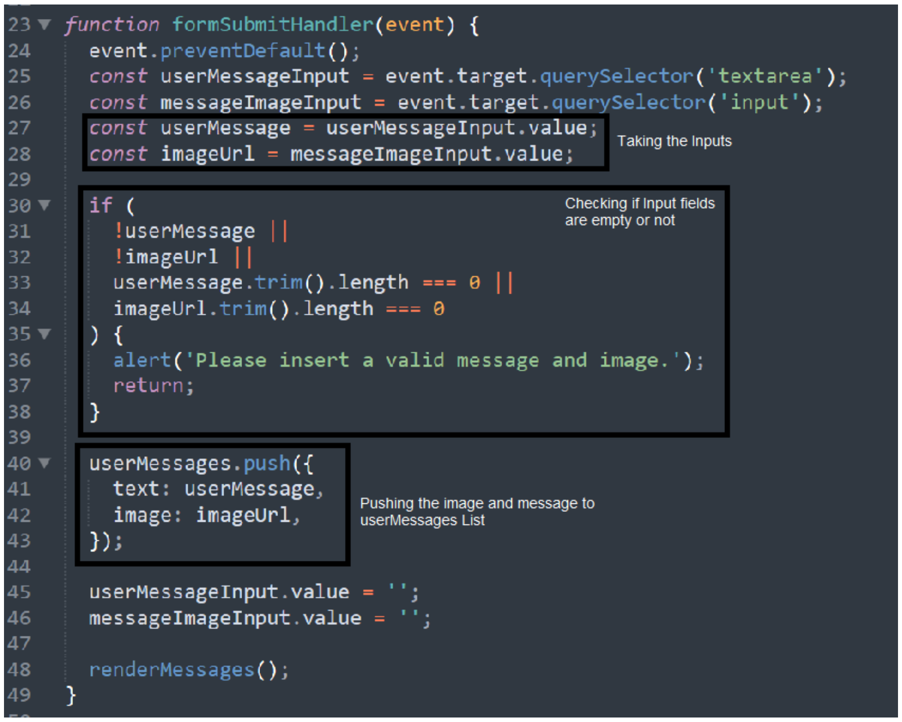
And an Alert will pop up in case you have left any of the two parameters empty as below.


This page says

Please insert a valid message and image.

Img2:- page showing alert

2) Now as this simple web page just takes only two parameters message and image url now let's have a look at the code base that how it will take these parameters and evaluate it to display it to the user.

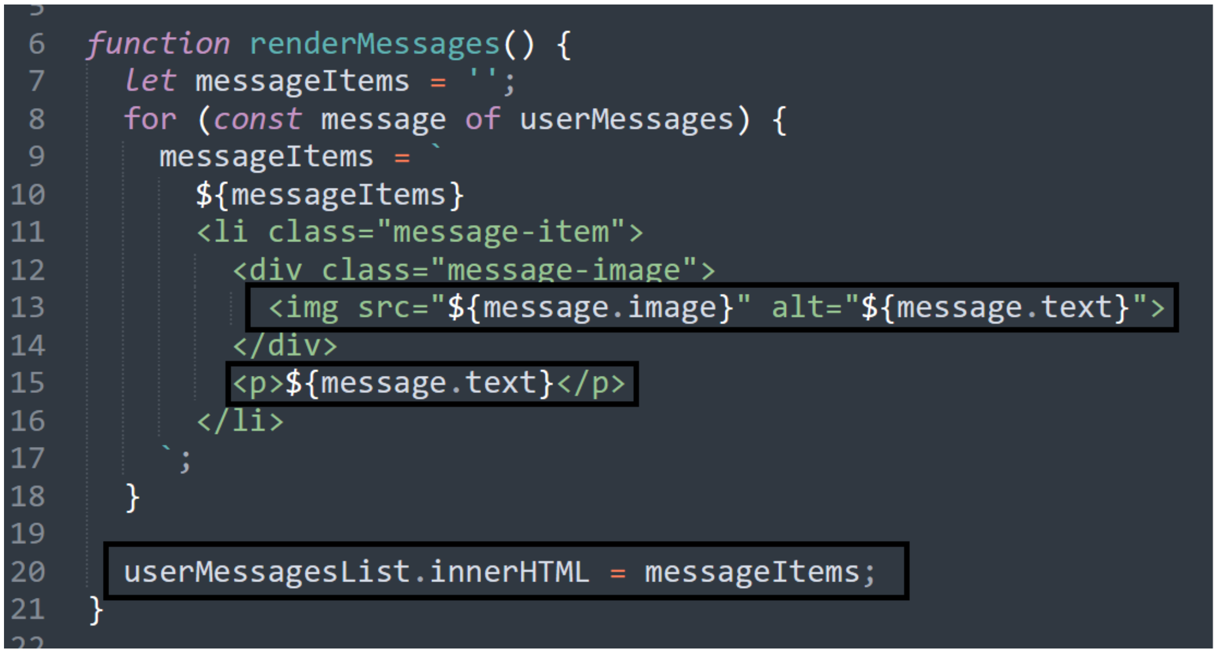


```javascript
function formSubmitHandler(event) {
  event.preventDefault();
  const userMessageInput = event.target.querySelector('textarea');
  const messageImageInput = event.target.querySelector('input');
  const userMessage = userMessageInput.value;
  const imageUrl = messageImageInput.value;

  if (
    !userMessage ||
    !imageUrl ||
    userMessage.trim().length === 0 ||
    imageUrl.trim().length === 0
  ) {
    alert('Please insert a valid message and image.');
    return;
  }

  userMessages.push({
    text: userMessage,
    image: imageUrl,
  });

  userMessageInput.value = '';
  messageImageInput.value = '';

  renderMessages();
}
```

Taking the Inputs
Checking if Input fields are empty or not
Pushing the image and message to userMessages List

Img3:- Taking inputs from the user

As we can see while taking input from the user we just check if the fields are just empty or not but we didn't check if there's any other malicious code in the input or any other exceptions that the code is unable to execute. And this is the one of the basic reasons for making the XSS scripting attack so easy to execute.



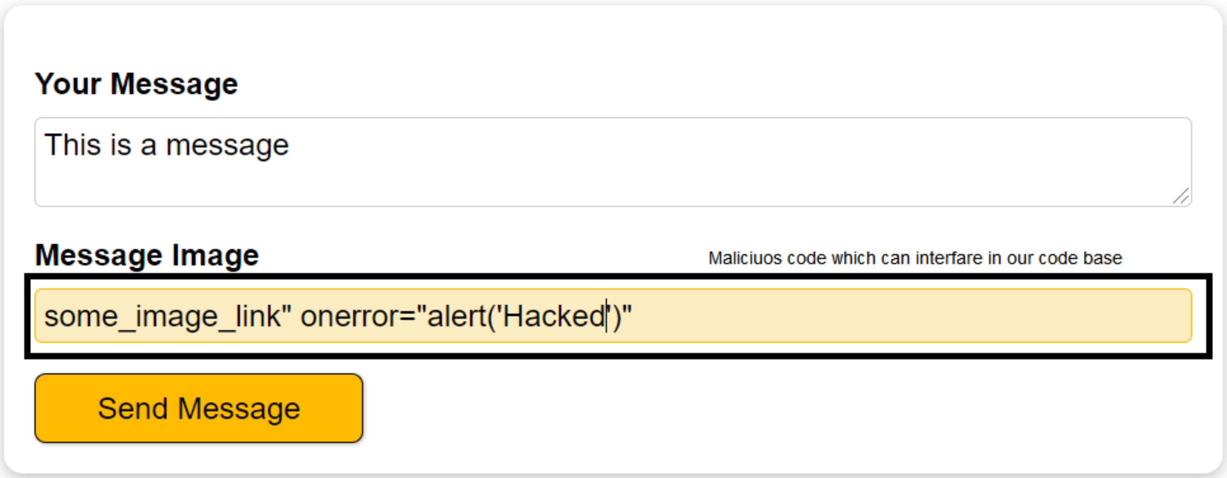
```javascript
function renderMessages() {
  let messageItems = '';
  for (const message of userMessages) {
    messageItems = `
      ${messageItems}
      <li class="message-item">
        <div class="message-image">
          <img src="${message.image}" alt="${message.text}">
        </div>
        <p>${message.text}</p>
      </li>
    `;
  }

  userMessagesList.innerHTML = messageItems;
}
```

Img4:- Displaying the data

Now while displaying the data we are making the same mistake we aren't following any exceptions handling techniques. Which makes our code more vulnerable to XSS scripting attacks.

So, now lets see an example of an xss scripting which can occur due to this code.

3) Adding some malicious code in the parameters.



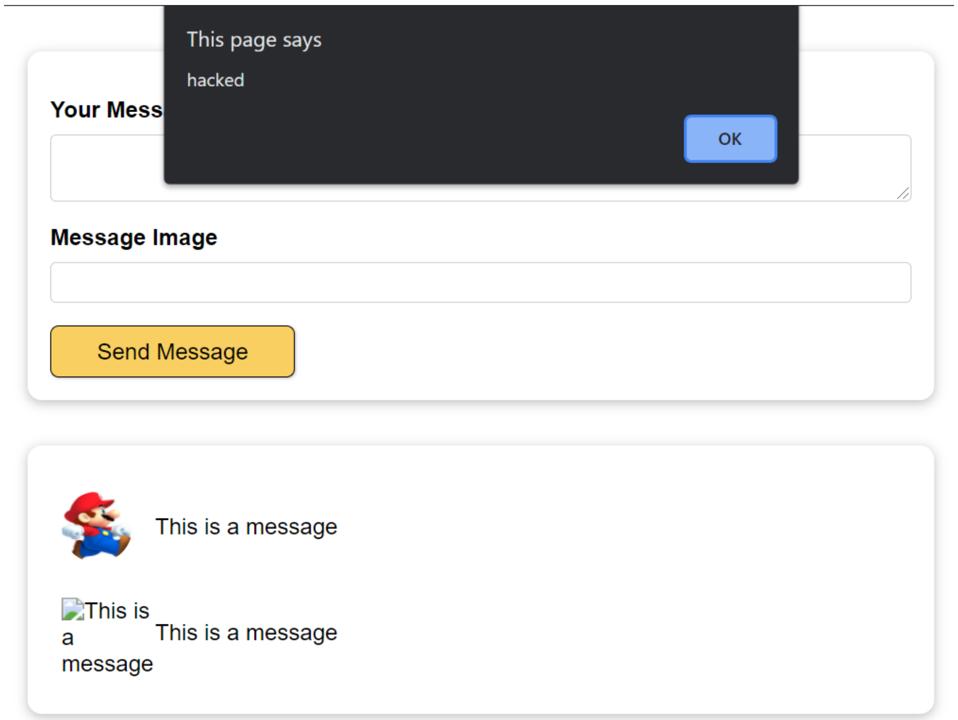
This is a message

Maliciuos code which can interfare in our code base

```
some_image_link" onerror="alert('Hacked')"
```

Img5:- Adding a malicious code

" Some_image_link" onerror="alert('Hacked')" " This text is a malicious code which can interfere in our code so, this can lead to some exception that our code is not designed for.



This page says
hacked

This is a message

This is a message

This is a message

Img6:- Alert box saying "Hacked"

As you can see an alert box appears saying "Hacked" which is not a part of our code but still it appeared due to the malicious code written in the image parameters. This is a simple demonstration of the XSS scripting attack but such code can make it hard to the code we imagine. Hence proper exception handling and proper security measures are necessary to prevent such types of attack.

Detection of XSS Attack :- [1][4]

Cross-site scripting or XSS is a specially crafted URL that includes attack code that will cause information that a user enters into their web browser to be sent to the attacker.

In an identity theft based attack, an attacker will find a web server that is vulnerable to XSS and send a legitimate looking URL with XSS attack code appended to the end of the URL. The malicious URL is often sent in a phishing email message.

The best defense is to log everything on our web servers and use resources such as the ha.ckers.org Cross Site Scripting cheat sheet for testing and detection methods (http://ha.ckers.org/xss.html).

Nothing replaces due diligence and frequent log scouring, however. So what do you look for? Anything out of the ordinary. How do you know what is ordinary? By checking our logs frequently.

Here is one sample web access log entry that is a sign of an XSS attack.

```
192.168.0.252 – – [05/Aug/2009:15:16:42 -0400]
"GET/%27%27;!–%22%3CXSS%3E=&{()} HTTP/1.1″ 404 310 "-" "Mozilla/5.0 (X11;
U;Linux x86_64; en-US; rv:1.9.0.12)Gecko/2009070812 Ubuntu/8.04 (hardy)
Firefox/3.0.12″
```

Prevention of XSS Attack:- [1][4]

Attackers leverage a variety of methods to exploit website vulnerabilities. As a result, there is no single strategy to mitigate the risk of a cross-site scripting attack.
The concept of cross-site scripting relies on unsafe user input being directly rendered onto a web page. If user inputs are properly sanitized, cross-site scripting attacks would be impossible. There are multiple ways to ensure that user inputs can not be escaped on our websites.

To protect our website, we will harden our web applications with the following protective measures.

1. **Allowlist Values -** Restrict user input to a specific allowlist. This practice ensures that only known and safe values are sent to the server. Restricting user input only works if we know what data we will receive, such as the content of a drop-down menu, and is not practical for custom user content.
2. **Avoid and Restrict HTML in inputs -** While HTML might be needed for rich content, it should be limited to trusted users. If we do allow styling and formatting on an input, we should consider using alternative ways to generate the content such as Markdown.

   Finally, if we do use HTML, make sure to sanitize it by using a robust sanitizer such as DOMPurify to remove all unsafe code.

3. **Sanitize Values -** When we are using user-generated content to a page, ensure it won't result in HTML content by replacing unsafe characters with their respective entities. Entities have the same appearance as a regular character, but can't be used to generate HTML.

Post-Hack Action :- [1][4]

In the event of cross-site scripting, there are a number of steps we can take to fix our website.

1. **Locate Vulnerable Code -** The first step in recovering from cross-site scripting is to identify where the vulnerability is located.
2. **Remove malicious Content and Backdoors -** Once we have obtained information about the location of the malware, remove any malicious content or bad data from our database and restore it to a clean state. We'll also want to check the rest of our website and file systems for backdoors.
3. **Patch the Vulnerability -** Vulnerabilities in databases, applications, and third-party components are frequently exploited by hackers. Once we have identified the vulnerable software, apply patches and updates to the vulnerable code along with any other out-of-date components.
4. **Updates Our Credentials -** When a compromise occurs, it is important to change all of our passwords and application secrets as soon as the vulnerability is patched. Prevent reinfection by cleaning up our data to ensure that there are no rogue admin users or backdoors present in the database.

Conclusion:-

XSS is a versatile attack vector which opens the door to a large number of social-engineering and client-side attacks. As shown, it could be used to steal sensitive information, such as session tokens, user credentials or commercially valuable data, as well as to perform sensitive operations.

Most of the vulnerable areas include search and login pages that return a response or an error message to the browser - as well as comment fields that allow script tags.

As security consultants, we should do our best to avoid such attacks.
Some effective countermeasures include input validation to verify the user's input meets the expected format, output encoding to instruct the browser to interpret certain characters as data instead of executing them as code, and a content security policy (CSP) to restrict foreign scripts from loading.

References:-

1) Cross site Scripting (XSS) , KirstenS
2) Cross Site Scripting, web Security Academy, PostSwigger
3) "Symantec Internet Security Threat Report: Trends for July–December 2007 (Executive Summary)". Symantec Corp. April 2008
4) Grossman, Jeremiah (July 30, 2006). "The origins of Cross-Site Scripting (XSS)".
5) Running a XSS Attack + How to defend
6) https://sucuri.net/guides/what-is-cross-site-scripting/