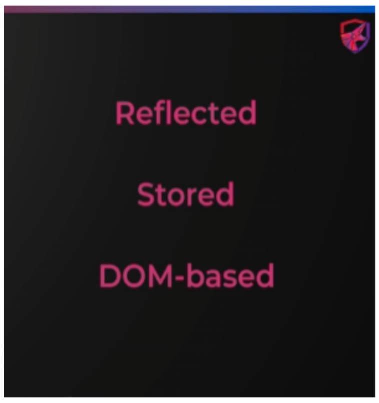
05 - XSS Intro

A.K.A - Cross Site Scripting

In short, it allows us to execute JavaScript in a victims browser. This is a very flexible vulnerability that can be used to trigger other vulnerabilities.

There are 3 types:

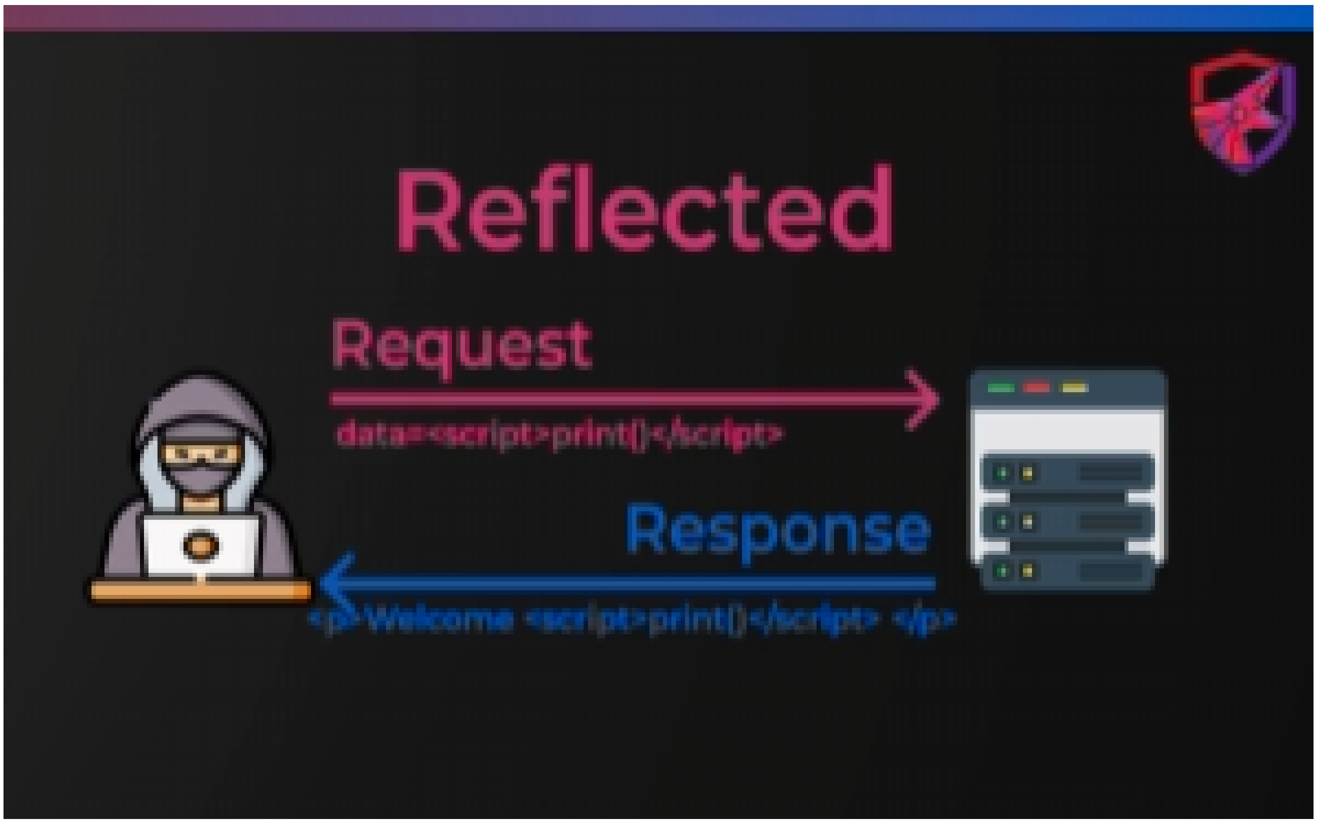


Reflected

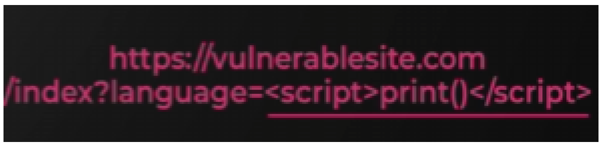
Stored

DOM-based

1)Reflected Cross Site Scripting.



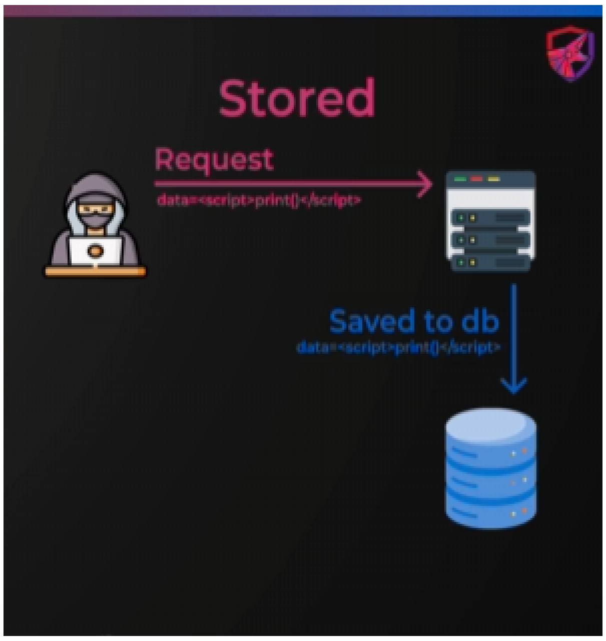
Reflected
Request
data=<script>print()</script>
Response
<p>Welcome <script>print()</script> </p>



https://vulnerablesite.com
/index?language=<script>print()</script>

Reflected: is where the scripting we are trying to inject comes from the current http request. We send a request, and we get a response, and the script is included in the response. This type of attack we can only target ourself, unless the payload is via the URI, and then maybe we can "entice"/trick a user to click on the link.

2)Stored Cross Site Scripting.



This is a much more "powerful" attack. In this attack, the script is stored in a database of some other software of some functionality, and retrieved later. This vulnerability allows us to attack other users, so the impact of this vulnerability is much higher.

2)DOM-Based Cross Site Scripting

DOM-based
Request
Everything happens locally in the browser.

"The client side has vulnerable JavaScript that uses untrusted input, instead of having a vulnerability server side."

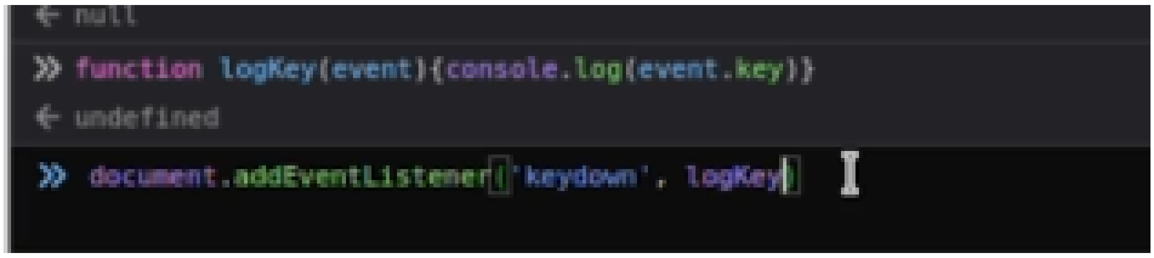
Notes:

Here, execution is what we are after.

"Later on, we actually need to find a place where we can inject our code rather than just executing JavaScript manually in the client."

Quick Tip:

Browsers are picking up on this, so use "print()" or "promt('hello')" to test instead of "alert(1)".



```
← null
» function logKey(event){console.log(event.key)}
← undefined
» document.addEventListener('keydown', logKey)
```

By replacing "console.log" with "fetch.api" for example, and try to exfiltrate information that way.