XSS and CSRF attacks

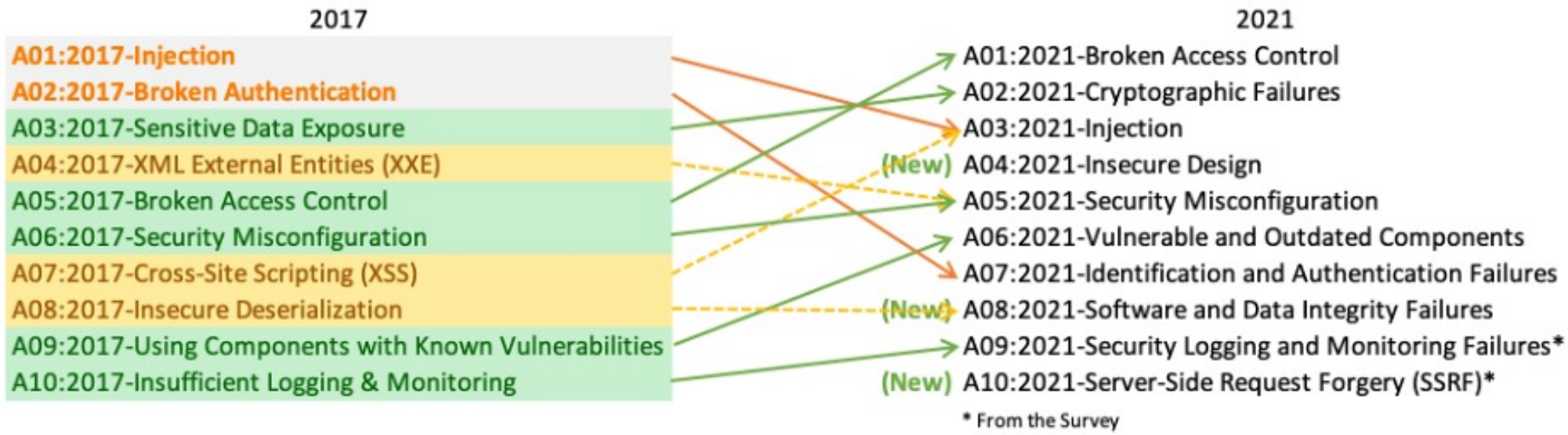
Jaroslav Hlaváč

User Input = EVIL

Catch-22

https://www.thecatch.cz/

OWASP TOP 10


2017
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring
2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

Cross-Site Scripting - XSS

- Malicious script injected into benign website
- 3 types
  - Reflected XSS
  - Stored XSS
  - DOM based XSS

Same-origin policy

- Security mechanism restricting how items are loaded to a website
- Prevents malicious script accessing resources on other origins (e.g. logged in Facebook in different session)
- Can be modified to allow specific exceptions

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Same origin	Only the path differs
`http://store.company.com/dir/inner/another.html`	Same origin	Only the path differs
`https://store.company.com/page.html`	Failure	Different protocol
`http://store.company.com:81/dir/page.html`	Failure	Different port (`http://` is port 80 by default)
`http://news.company.com/dir/page.html`	Failure	Different host

source

Dangers of XSS

- it can lead to:
  - account impersonation
  - observing user behaviour
  - loading external content
  - stealing sensitive data
  - … and more

Dangers of XSS

- it can lead to:
  - account impersonation -> stealing cookies, secrets and usernames
  - observing user behaviour
  - loading external content
  - stealing sensitive data
  - … and more

Dangers of XSS

- it can lead to:
  - account impersonation -> stealing cookies, secrets and usernames
  - observing user behaviour -> what actions he takes
  - loading external content
  - stealing sensitive data
  - … and more

Dangers of XSS

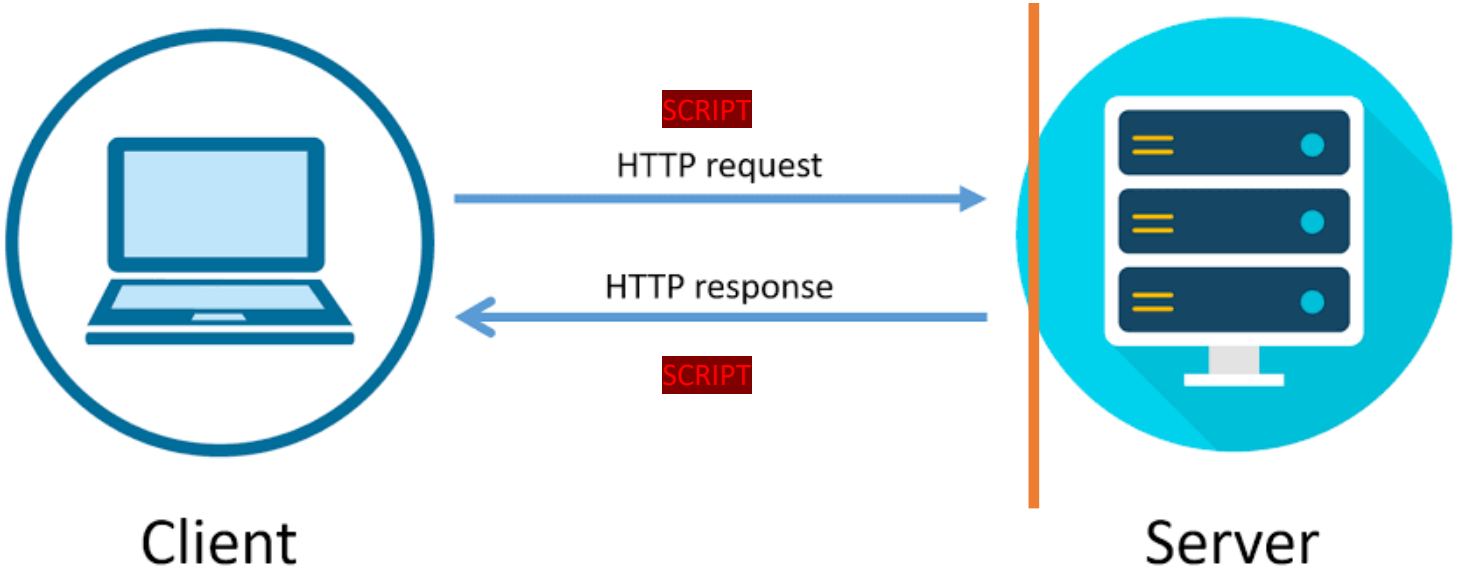
- it can lead to:
  - account impersonation -> stealing cookies, secrets and usernames
  - observing user behaviour -> what actions he takes
  - loading external content -> adds, coin miners
  - stealing sensitive data
  - … and more

Dangers of XSS

- it can lead to:
  - account impersonation -> stealing cookies, secrets and usernames
  - observing user behaviour -> what actions he takes
  - loading external content -> adds, coin miners
  - stealing sensitive data -> exfiltrating what he views
  - … and more

Reflected XSS

- the script is reflected of the website to the browser
- script is not stored on the server



Reflected XSS

Request:

```
https://evil.com/search?query=ponozky
```

Returns:

```
<p> You have queried: ponozky </p>
```

What will happen:

```
https://evil.com/search?query=<script>alert(XSS)<\script>
```

Reflected XSS

Request:

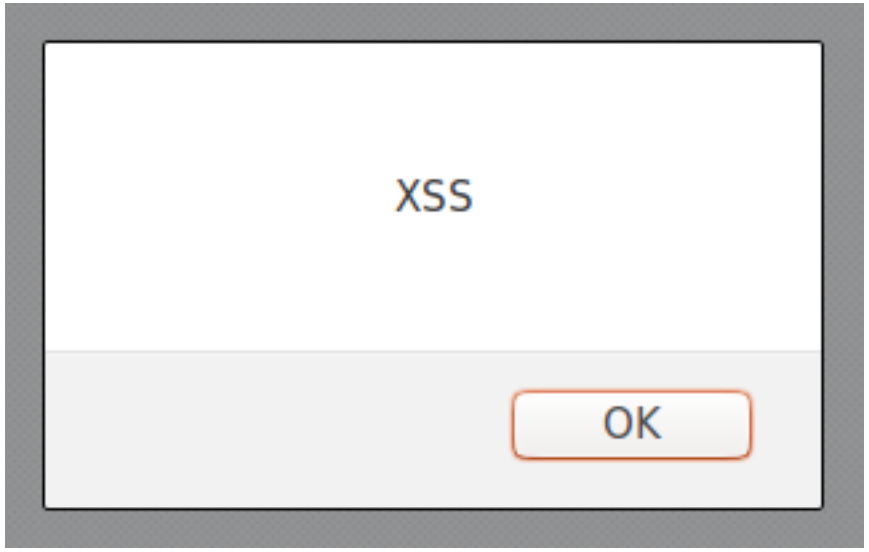
```
https://evil.com/search?query=ponozky
```

Returns:

```
<p> You have queried: ponozky </p>
```

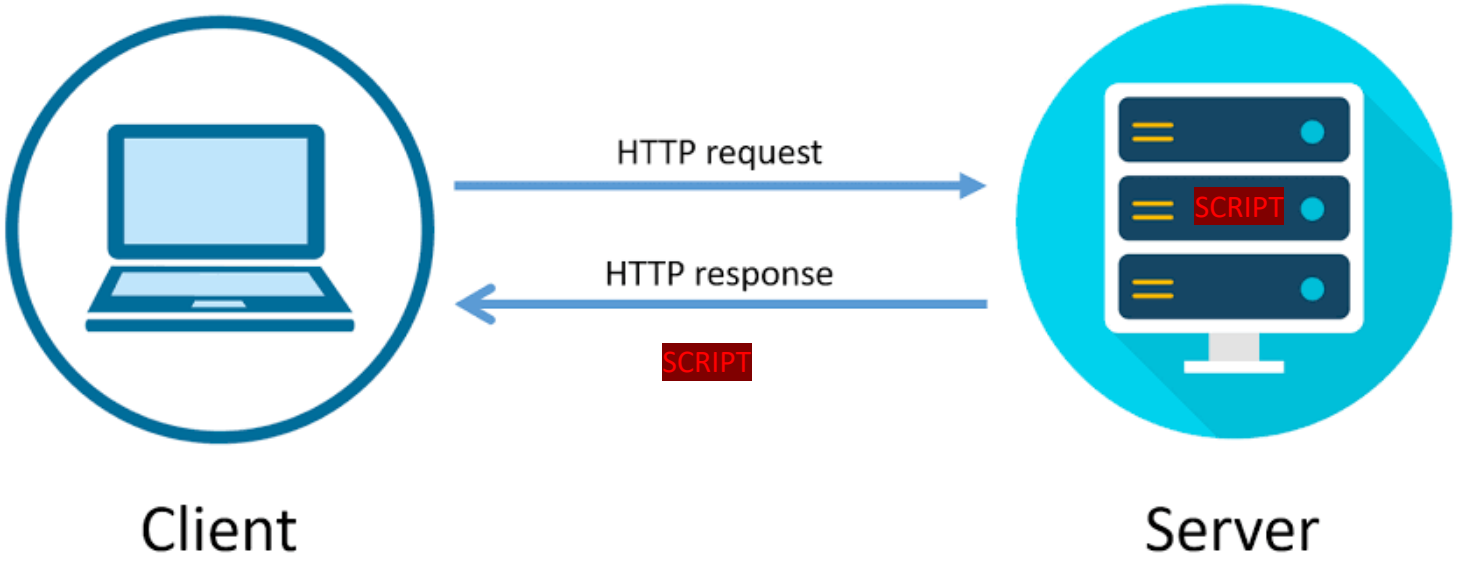
What will happen:

```
https://evil.com/search?query=<script>alert(XSS)<\script>
```



Persistent XSS

- Script stored on the server
- Trusted source – browser loads it



Serverside vs. Clientside

Where untrusted data is used

Data Persistence	XSS	Server	Client
	Stored	Stored Server XSS	Stored Client XSS
	Reflected	Reflected Server XSS	Reflected Client XSS

- **DOM-Based XSS is a subset of Client XSS (where the data source is from the client only)**
- **Stored vs. Reflected only affects the likelihood of successful attack, not nature of vulnerability or defense**

source

Hands-on

https://google-gruyere.appspot.com/

Prevention

- Modern web frameworks help – but cannot protect you from yourself
- Output Encoding
  - encode every user input
  - carefully chose where you allow user input
  - e.g. HTML encoding:
    - Script: <script>alert(“You have been attacked!”)</script>
    - Encoded: <script>alert("you are attacked")</script>
  - different encoding types here
- HTML Sanitization
  - when you need to let user input html
  - https://github.com/cure53/DOMPurify

Cross-Site Request Forgery - CSRF

- trick user to submit malicious request
- ALL cookies to the relevant site are sent
- Usual attack scenario
  - Figure out how the request should look like
  - Craft the attack
  - Trick user into execution

CSRF - usual attack scenario

1. Figure out how the request should look like

```
GET http://bank.com/transfer.do?acct=MAMA&amount=100
HTTP/1.1
```

2. Craft the attack
   1. Malicious GET

```
GET http://bank.com/transfer.do?acct=JARDA&amount=10000000
HTTP/1.1
```

   2. Hide it!

```
<img src="http://bank.com/transfer.do?acct=MARIA&amount=100000
" width="0" height="0" border="0">
```

3. Trick user into execution – email with HTML

Hi Mama,

I just want to tell you I am doing fine and my grades are good.

Bye.

Hands-on

https://google-gruyere.appspot.com/

Prevention

- Tokens
  - per session and generated by server – ideally per request
  - secret and unpredictable
  - not transferred by cookies

Now go and do the homework!

kbe.felk.cvut.cz