XSS & CSRF

Alex Inführ

Whoami

- Alex Inführ
- @insertscript
- Cure53
- MS IE Team
- Browser
- PDF / file formats
- B Shark movies

Topics of today

- XSS
  - Cross Site Scripting
- XSRF/CSRF
  - Cross Site Request Forgery
- Browser Hackers Handbook

Cross Site Scripting

Why XSS

- https://blogs.msdn.microsoft.com/dross/2009/12/15/happy-10th-birthday-cross-site-scripting/

- „Unauthorized Site Scripting, Unofficial Site Scripting, URL Parameter Script Insertion, Cross Site Scripting, Synthesized Scripting,Fraudulent Scripting“

- „Let's hope that ten years from now we'll be celebrating the death, not the birth, of Cross-Site Scripting!“ (2000 – 2009 - now)

What is XSS

- User sends data to server
- Server includes user controlled data in response
- User controlled data contains HTML code
- It gets interpreted and parsed as HTML and executed

What is XSS

- JavaScript

- Non JavaScript
  - CSS
  - Not as powerful

What is XSS

- Access to website origins
- Cookie Theft
- Keylogging
- Phishing

- Beef

XSS Intro

```
<!DOCTYPE html>
 <body> <h1>
You searched for
<?php echo $_GET['search']; ?>
</h1>
</body>
```

http://website/xss.php?search=abcd

```
<!DOCTYPE html>
 <body> <h1>
You searched for
abcd
</h1>
</body>
```

http://website/xss.php?
search=<script>alert(1)</script>

```
<!DOCTYPE html>
 <body> <h1>
You searched for
<script>alert(1)</script>
</h1>
</body>
```

Types of XSS

- Reflected XSS
- Stored XSS
- DOM XSS
  - framework XSS
- (UXSS)

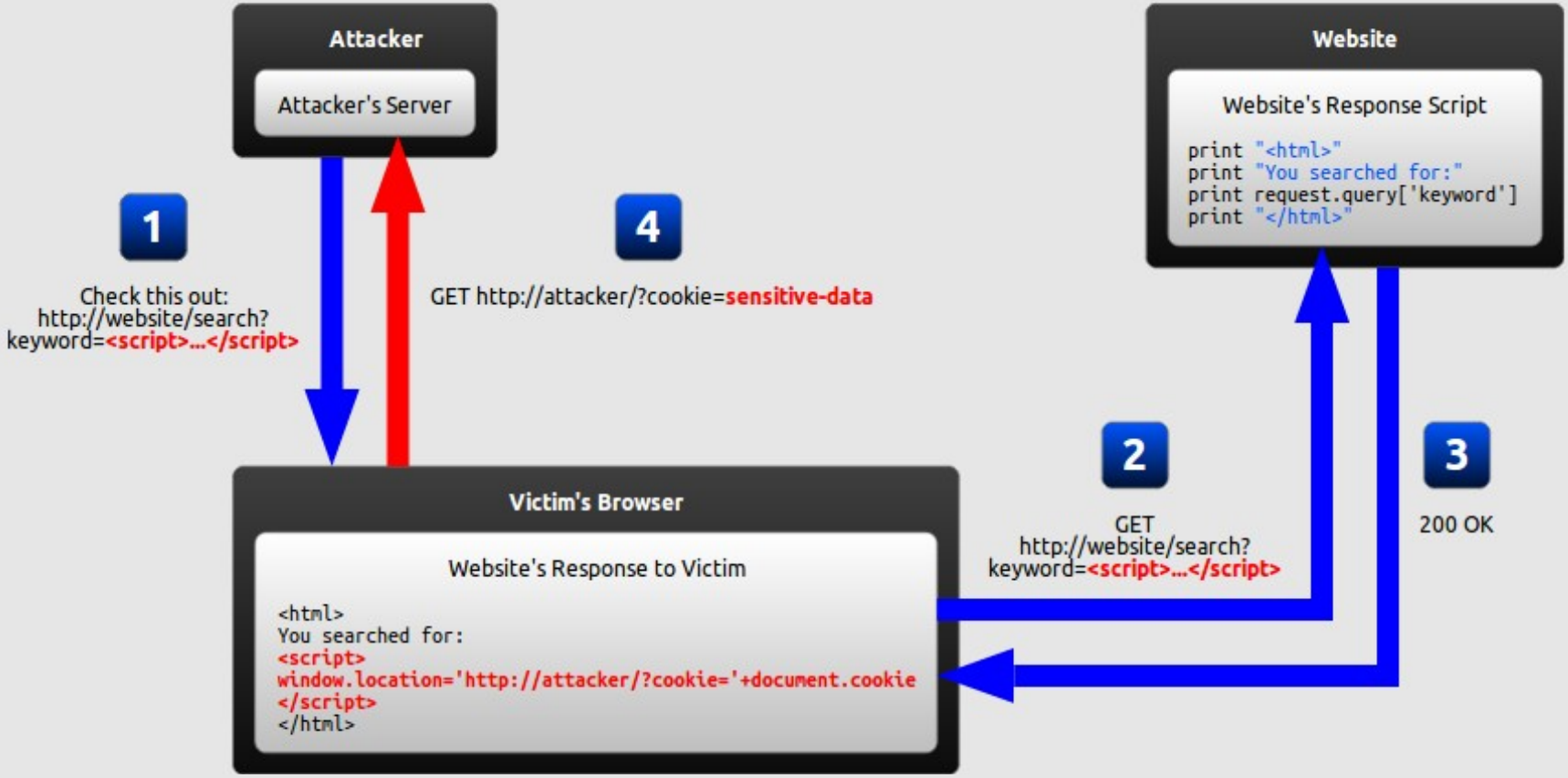
Reflected XSS

The attacker crafts a HTTP request/URL, which contains the malicious HTML payload.

The victim is tricked by the attacker into requesting the URL from the website.

The website includes the malicious string from the URL in the response.

The victim's browser executes the malicious script inside the response, sending the victim's cookies to the attacker's server.



* Source: https://excess-xss.com/reflected-xss.png

Reflected XSS

- Victim has to send the request
- Distribution: Email/Malicious Website/Social Media

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”

- Persistent
- Can affect all users (crypto miner, malware download…)

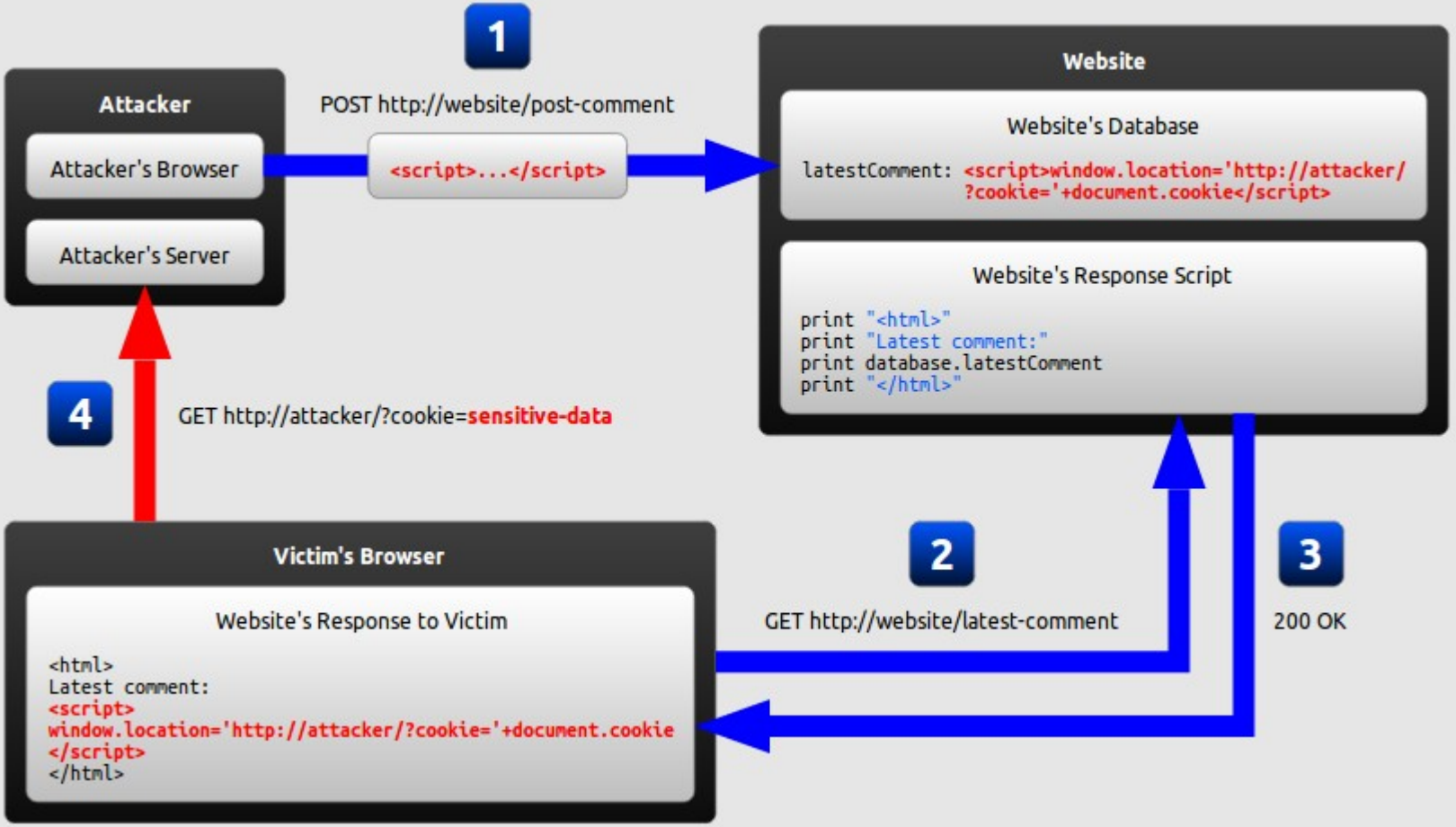
Stored XSS

The attacker uses one of the website's forms to insert a malicious string into the website's database.

The victim requests a page from the website.

The website includes the malicious string from the database in the response and sends it to the victim.

The victim's browser executes the malicious script inside the response, sending the victim's cookies to the attacker's server.



* Source: https://excess-xss.com/persistent-xss.png

Stored XSS

- Can cause huge damage (assume XSS on youtube)

- DDos
- - sohu.com
- - 20 Million GET requests (22 000 users)

- Crypto Miner

DOM XSS

- JavaScript
  - ≠ server side
- Mix of reflected/stored

- Document Object Model
  - innerHTML

- Difficult to detect

DOM XSS

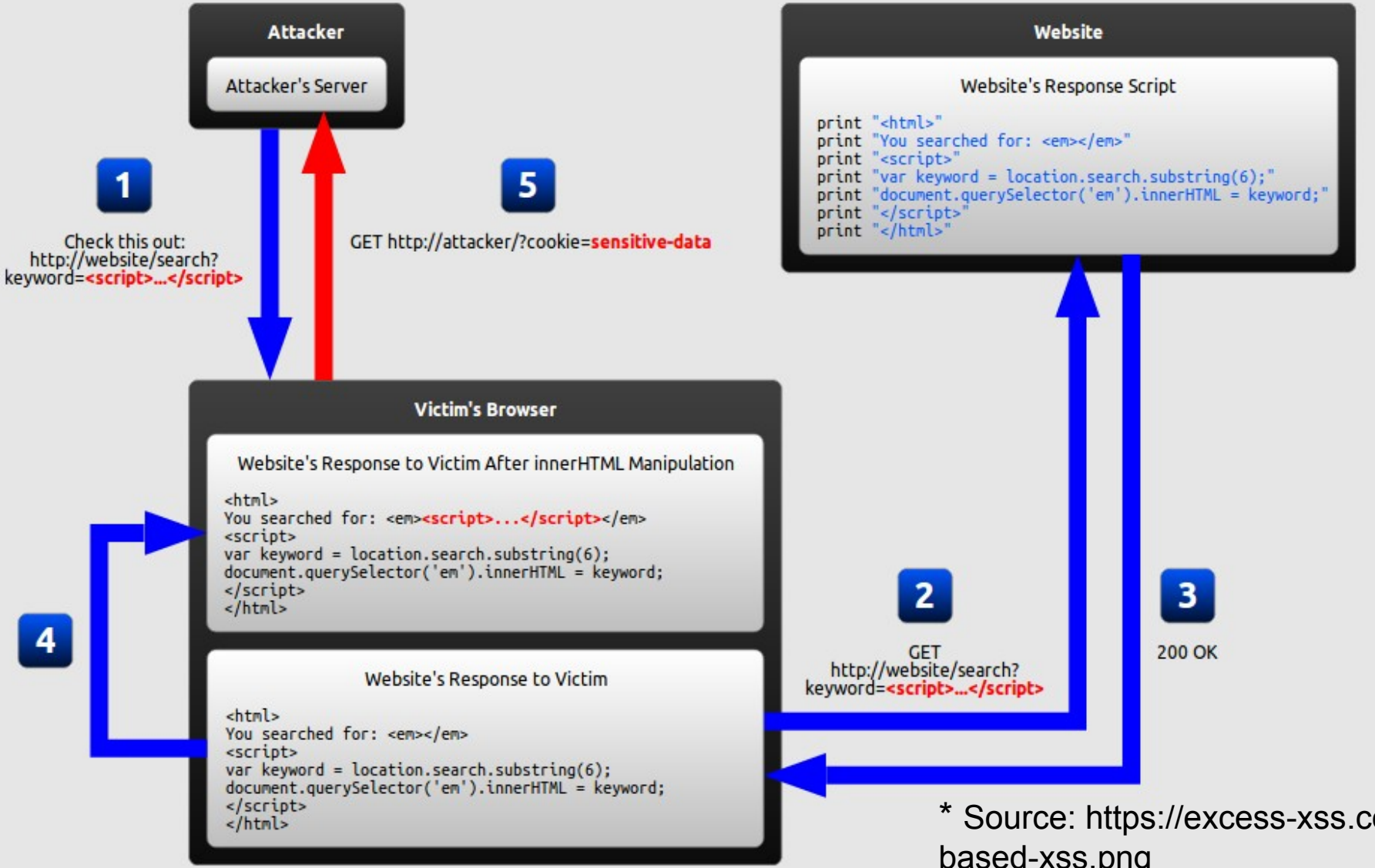
1. The attacker crafts a URL containing a malicious string and sends it to the victim.

2. The victim is tricked by the attacker into requesting the URL from the website.

3. The website receives the request, but does not include the malicious string in the response.

4. The victim's browser executes the legitimate script inside the response, causing the malicious script to be inserted into the page.

5. The victim's browser executes the malicious script inserted into the page, sending the victim's cookies to the attacker's server.



* Source: https://excess-xss.com/dom-based-xss.png

DOM XSS

- Browser encode values
    - location.search/hash
- Step to prevent DOM XSS

- DOM XSS in 2018 (AngularJS example)

DOM XSS

- Often difficult to detect
  - Frameworks/MBs of JavaScript
- Server Side != Client Side
  - Third Party scripts

Preventing XSS

Strategies

- Encoding
  - <img> <img>
- Escaping
  - <img> \<img\>
- Validation/Filtering

Strategies

- Context
- Inbound/Outbound
- Client/Server input handling

Context	Example Code
HTML Element content	UserInput
HTML attributes	<input value="UserInput"/>
HTML URLs	link
JavaScript value	<script>var a ='UserInput'</script>
Cascading style sheet	<style> * { color: 'UserInput'} </style>

Inbound vs Outbound

- Server side
- Inbound
  - One point of encoding
  - Not context aware
- **Outbound**
  - Multiple points of encoding
  - Context aware

HTML encoding

- Encodes certain characters to HTML entity:

  - [<>“ ‘A] => [<>"'A]

- Text representation

```php
<?php
$user_input = '"\'>ee<script>aaaa\\</script>';
echo htmlentities($user_input,ENT_QUOTES);
?>

//Output:
&amp;quot;&amp;#039;&amp;gt;ee&amp;lt;script&amp;gt;aaaa\&amp;lt;/script&amp;gt;
```

UserInput = '';\<script>

Context	Example Code
HTML Element content	"'>\<script>
HTML attributes	<input value=""'> \<script>"/>
HTML URLs	 \<script>">link
JavaScript value	<script>var a ='"'> \<script>,</script>
Cascading style sheet	<style>* { color: '"'> \<script>'}</style>

UserInput = '';\<script>

Context	Example Code
HTML Element content	"'>\<script></ div>
HTML attributes	<input value=""'> \<script>"/>
HTML URLs	 \<script>">link
JavaScript value	<script> var a ='"'> \<script>, </script>
Cascading style sheet	<style>* { color: '"'> \<script>'} </style>

HTML and URLs

- Linking/Loading of other resources

  - a,iframe,script …

- Context: Inside HTML attribute

URL attribute	Request URL
a	GET http://website/
b	GET http://website/

HTML and URLs

- „Standard“ protocols
  - http,https,mailto,ftp
- Pseudo protocols
  - vbscript (you are missed)
  - data:
  - javascript:

Context	Example Code
Standard	
HTML encoded	
HTML5 Entities	
HTML5 Entities in protocol	
URL encoding	<a href="
javas cript:alert%281)">aaaa

HTML and URLs

- Server side needs to parse URL
- Follow behavior of browser
- NodeJS

  -new URL

Script encoding

- \ often not encoded
- Allows to ‚extend' JavaScript strings
- Often requires multiple inputs

```
<script>
var input1 = 'UserInp1'; var input2 = 'UserInp2';
</script>
```

Script encoding

- UserInp1 = \
- UserInp2 = +alert(1)//

```
<script>
var input1 = '\'; var input2 = '+alert(1)//';
</script>
```

Solution: \ => \\

Validation

- Allow certain HTML (styling for comments etc.)
- Blacklist
  - Used in the past
  - Prone to bypasses (MySpace)
  - Living standard
- Whitelist
  - Browser/Server (DOMPurify)

Validation

- Rejection
  - Malicious HTML is rejected
  - false positives

- Sanitisation
  - remove malicious part
  - more user friendly
  - can introduce bypasses (<scrip<script>>)

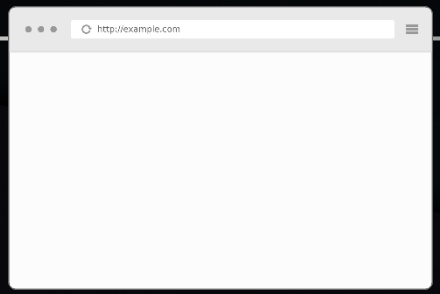
XSS – TL;DR;

- Attacker advantage
  - Spray and pray
  - HTML context
- JavaScript librarys
  - Jquery, Angular, **React**
- „XSS will be dead“
  - Context aware frameworks

Cross-Site Request Forgery

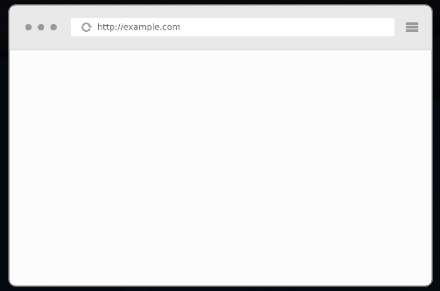
HTTP Cookies

- Cookies allow to store „data“ in the user browsers
- Linked to the domain/subdomain
- Used for authentication/authorization
- Appended automatically to every HTTP request



POST http://site/login.php

HTTP 200 OK
Set-Cookie: auth=123



GET http://site.com/index
Cookie: auth=123

HTTP Cookies

- HTTP Set-Cookie
- JavaScript: document.cookie
- Browser **always** send Cookies associated with a domain

Attack example

- Victim is logged in at facebook.com
- Victim visits attacker.com
- Attacker.com triggers a GET/POST/PUT request to facebook.com in the victims browsers
- Cookies of facebook.com are sent along
- Facebook.com sees a correct authenticated request
  - Change password/email etc.

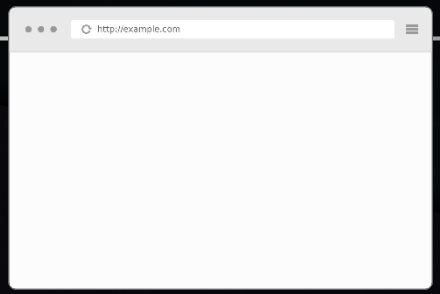
Attacker.com

```
<form action="https://facebook.com/changeEmail" method="POST">

<input type="text" name="newEmail" value="attacker@evil.com">

</form>

<script>
document.forms[0].submit();
</script>
```

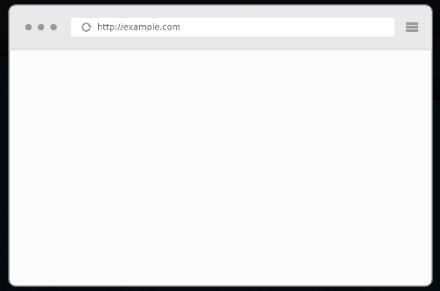


POST http://site/email.php

Cookie: auth=123

Referer: http://attacker.com

NewEmail=attacker@evil.com



HTTP/1.1 200 OK

Protection ideas

- Referer Check

  - Referer Header contains domain triggering the request

  - Browser bugs

  - Missing Referers

- Solution:

  - Tokens

CSRF Tokens

- Server generates random token for each HTML form
  - state changing request
  - token associated to user session
- User sends requests
  - Cookie must be present
  - Token must be present
- No token / token wrong => request rejected

CSRF Tokens

```
<form action="https://facebook.com/changeEmail" method="POST">

<input type="text" name="newEmail" value="attacker@evil.com">
<input type= "hidden" name= "csrf_token" value="rokefwokfewokfewijh"/>
</form>
```

- Attacker.com can‘t know the random token
- No way to send request in behalf of other user

CSRF Tokens – second solution

- Real Life Question for a job:

„Our client wants to implement CSRF. They have many users. Storing CSRF tokens requires too much server memory – How can the client implement CSRF without memory problems“

- I was lucky – read 2 days before a blogpost about that

CSRF Tokens – second solution

- Authenticated user requests a web page
- The CSRF token is set in the HTML form (as usual)
- CSRF token is also set in the cookie (Set-Cookie)

- Legit user sends request
  - CSRF token in Cookie matches form token
- Attacker.com
  - CSRF token in Cookie is present but not in form

CSRF Tokens

- Support in all major frameworks
  - tokens completely random
  - no huge development effort

- Solved… ?

Additional Topics

- Same Site Cookies
- Content-Security Policies
- Browser XSS filters
- Iframe sandbox
- Service workers
- Electron (XSS => RCE)
- Just have fun with it: Browsers Hackers Handbook/Tangled Web/html5sec.org/XSS challenges

Additional Topics

- https://excess-xss.com
- https://html5sec.org
- AngularJS XSS Portswigger
- Cure53 XSS challenges
- Web Application Obfuscation
- Web Hackers Handbook