

# Introduction to Computer Security Lecture Slides

© 2024 by [Mihai Chiroiu](#) & [Florin Stancu](#)

is licensed under [Attribution-NonCommercial-ShareAlike 4.0  
International](#)

# Web Security

# Contents

- HTTP Security
  - Cookies, Sessions
  - HTTPS
- Server-side Security
  - Injection
  - Session Hijacking
- Client-side / Browser Security

# HTTP Protocol [1]

- Stateless, text-based request-response protocol

**Client** -> **Server**:

GET /index.html HTTP/1.0

Header1: value1

Header2: value2



**Server** -> **Client**:

HTTP/1.0 200 OK

Header1: value1

Header2: value2

<optional body>

<html><head>...</head>

<body>...</body></html>

# HTTP Methods

- **GET:** fetch a resource, may have *query strings*:  
<http://domain.com/browse.php?list=users&name=john>  
*generates*  
GET /browse.php?list=users&name=john HTTP/1.0
- **PUT / POST:** create or edit a resource (only POST is widely used)
- **DELETE:** delete resources (not used in practice)
- **HEAD:** like GET, but server responds with the headers only
- **OPTIONS:** determine options for a resource
- GET, HEAD and OPTIONS should be idempotent

# HTTP Methods & HTML Forms

- Links typically use a **GET** request for opening pages
- HTML forms can generate **GET** and **POST** requests:

```
<form action="/login.php?user_type=regular" method="post">  
  User: <input type="text" name="username">  
  Password: <input type="password" name="pass">  
</form>
```

=>

```
POST /login.php?user_type=regular HTTP/1.0  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 30 <-- the length of the body
```

username=<username>&pass=<user's password>

# Cookies

- Small piece of data that the browser stores and sends back to the server on future requests
- Can be used to remember user preferences, server sessions etc.

## Response header example:

```
HTTP/1.0 200 OK  
Set-Cookie: c1=val1  
Set-Cookie: c2=val2
```

## Request example:

```
GET / HTTP/1.1  
Cookie: cook1=val1;cook2=val2
```

# Cookie Security

- Cookies are insecure:
  - The user can freely read & modify them
  - They can be intercepted unless HTTPS is used for transport
- Must add confidentiality and integrity guarantees:
  - Using cryptography: encryption & HMAC [2]
  - Server-side sessions
- Privacy implications:
  - Cookies can be used to track users (e.g. by analytics & ad servers)



# Server Sessions

- Also known as server-side cookies
- Server generates a random, unique session ID:  
**4125a859778b1bf9b9b778a236f01e01**
- Server uses database to store secrets associated with a session ID
- Persisted as cookie / passed using GET / POST parameters

Cookie: **PHPSESSID=4125a85...**

*or*

**show.php?phpsessid=4125a85...**

# HTTPS [3]

- Based on Secure Sockets Layer / Transport Layer Security
- Creates a private channel between the client and the server
- The server authenticates itself using certificates and PKI
- Diffie-Hellman for forward secrecy
- Cipher negotiation: RC4, DES, AES CBC, AES GCM etc.
- Target of numerous attacks

# TLS / HTTPS Attacks [4]

- Compression attacks (CRIME, BREACH - 2013)
  - Compression Length Oracle
- Crypto weaknesses (RC4 - broken, 3DES - Sweet32, RSA - ROBOT)
  - ROBOT: Return Of Bleichenbacher's Oracle Threat (2018)
  - Lucky13: Timing padding oracle in CBC-mode
- Man-in-the-middle (Malicious Certificates, SSL stripping)
- Downgrade attacks (FREAK, Logjam, POODLE - 2014)
- Implementation bugs, e.g.:
  - Heartbleed (CVE-2014-0160)
  - Cloudflare parser bug (2017)

# TLS Testing Services

Servers:

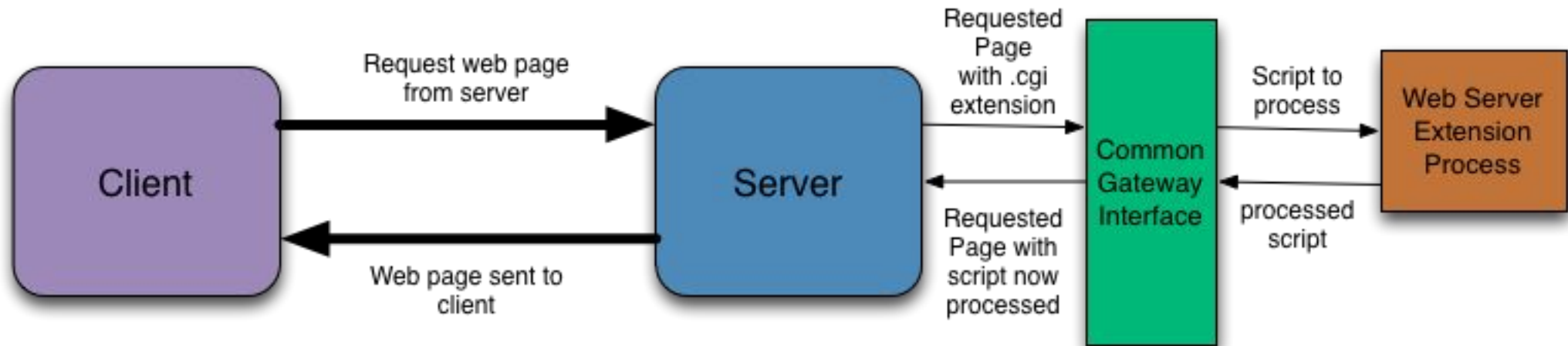
<https://www.immuniweb.com/ssl/>

For both clients / servers:

<https://www.ssllabs.com/ssltest/>

# Server-side Processing

- Server generates dynamic content
- Scripting interfaces: CGI (legacy) / FastCGI / integrated modules [5]



# Sample Directory Layout

## **/var/www**

- |-- index.html
- |-- login.php
- |-- css/style.css
- |-- images/
  - |-- logo.png
  - |-- map.png

## **Example requests:**

- > GET /index.html HTTP/1.0
- > GET /images/logo.png HTTP/1.0
- > POST /login.php HTTP/1.0

# Server-side Processing

## *Example (PHP)*

```
<?php
$name = $_GET["name"];
$curDate = date("l");
?>
<p>Hello, <i><?=$name?></i>.
The date is
<b><?=$curDate?></b>
<?php
echo $message;
?>
```

## *Example (Python / Flask)*

```
@app.route("/")
def index():
    cur_date =
datetime.now().strftime("%d.%m.%Y")
    return f"Hello,
{request.args.get('name')}<br />
Date is {cur_date}!"
```

# SQL Intro

## Querying:

```
$query = "SELECT * FROM employees WHERE name LIKE 'florin%';"  
$result = mysql_query($conn, $query);
```

## Modification queries:

```
INSERT INTO employees (name, emp_date, status)  
VALUES ('Florin S.', '2023-10-01 08:00', 'active');
```

```
UPDATE employees SET status='terminated' WHERE id=2;
```



# SQL Injection (1)

Code:

```
$query = "SELECT * FROM users WHERE user='" .  
    $_POST["user"] . "' AND password='" .  
    hash($_POST["password"]) . "'";  
$result = mysql_query($conn, $query);
```

**What if user input is:** *admin' -- comment here*

```
=> SELECT * FROM users WHERE  
    user='admin' -- comment here' AND password=''  
    ^<- injected input    ->^
```

# SQL Injection (2)

***This doesn't work very often:***

```
SELECT * FROM users WHERE  
    user=' ' ; DROP DATABASE app -- commented'  
    ^<-      user-injected input      ->^
```

*Sql servers' standard query() functions only execute **one single statement!***

There are multi\_query() like functions, but seldom used!

# SQL Injection (3)

## ***Error Reporting Abuse***

```
SELECT * FROM users WHERE  
user='asdf' OR 1/(select password from users) --' ...
```

## ***Boolean statements***

```
SELECT * FROM users WHERE user='' OR user LIKE '%admin%' ...
```

## ***Time-based attacks***

```
SELECT * FROM users WHERE user='  
OR IF(user LIKE '%adm%'), SLEEP(10), 'false')' ...
```

# Code Injection (File Upload)

- A site allows image submissions with minimal verification.
- The hacker1337 uploads [image.gif.php](#) with malicious code.
- Find out the path to the image and requests it:

**GET /uploads/image\_9876.gif.php**

- Server executes our script (if badly configured)!
  - **Remote Code Execution** :(

# Code Injection (2)

**Multipart File Upload with **path traversal** bug:**

-----5191859754266

Content-Disposition: form-data; name=" ../ ../index.php"

```
<?php die("PWNED")
```

# Code Injection (3)



```
total 216944
-rw-r--r-- 1 rl courses 73616201 Nov 13 2016 PacketTracer533_i386_no_tutorials_installer-deb.bin
-rwxr-xr-x 1 rl courses 57573696 Nov 13 2016 Packet_Tracer_6.2.exe
-rwxr-xr-x 1 rl courses 90770084 Nov 13 2016 Packet_Tracer_6.2_Ubuntu.tar.gz
-rwxr-xr-x 1 rl courses 385 Oct 29 23:38 gen.php
-rwxr-xr-x 1 rl courses 284 Oct 29 22:48 generate.php
-rwxr-xr-x 1 rl courses 0 Nov 13 2016 index.html
drwxr-sr-x 2 rl courses 4096 Nov 13 2016 js
-rwxr-xr-x 1 rl courses 312 Nov 13 2016 style.css
-rwxr-xr-x 1 rl courses 8241 Nov 13 2016 tema1
-rwxr-xr-x 1 rl courses 1655 Nov 13 2016 tema1.c
-rwxr-xr-x 1 rl courses 2094 Nov 13 2016 tema1.php
-rw-r--r-- 1 rl courses 144542 Oct 29 22:14 tema1.pkt
```

# Preventing Injection

- **Do not trust tutorials / ChatGPT [7]**
- **Always sanitize user input!**
- Try not to use **exec()** / **eval()**
- For SQL, use prepared statements:

```
$stmt = $mysqli->prepare("INSERT INTO table  
    (name) VALUES (?");  
$stmt->bind_param("s", $id); // "s" for string  
$stmt->execute();
```

# Application-Specific Vectors

- Broken Authentication System [8]
  - Predictable / insecure session IDs
  - Unencrypted passwords [9]
- Authorization Vulnerabilities
  - Improper access verification
  - Example: `/delete_user.php?id=5368`
  - Direct object reference: `/admin/list_users.php`
- Vulnerable Frameworks / Plugins (e.g. Wordpress)



# Server Misconfiguration [9]

- **Again: do not trust tutorials & ChatGPT**
  - Nginx & PHP FastCGI configuration vulnerability [10]
- Exposed files (e.g. password files, backups) / directory listings
- Bad permissions
- Debugging enabled in production
- System software vulnerabilities:
  - E.g. ShellShock (BASH vulnerability) [11]

# Pwned Websites

- [Haveibeenpwned.com](https://haveibeenpwned.com) – account breach checker
- **Yahoo!** (2012 – SQL Injection, 2013, 2014 – forged cookies)
  - 3 bilion accounts exposed!
- **LinkedIn** (hacked 2012, exposed in 2016, 2021 Dark Web DB sale)
- **Adobe** (2013, 2019): broken encryption :|
- **Dropbox** (2012):
  - SHA1 and salted passwords ;)

HACKERS RECENTLY LEAKED **153 MILLION** ADOBE USER EMAILS, ENCRYPTED PASSWORDS, AND PASSWORD HINTS. ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:

USER	PASSWORD	HINT	
4e18acc1ab27a2d6		WEATHER VANE SWORD	<input type="text"/>
4e18acc1ab27a2d6			<input type="text"/>
4e18acc1ab27a2d6	a0a2876eb1ea1fca	NAME 1	<input type="text"/>
8babb6279e06eb6d		DUH	<input type="text"/>
8babb6279e06eb6d	a0a2876eb1ea1fca		<input type="text"/>
8babb6279e06eb6d	85c9da81a8a78adc	57	
4e18acc1ab27a2d6		FAVORITE OF 12 APOSTLES	
1ab29ae86da6e5ca	7a2d6a0a2876eb1e	WITH YOUR OWN HAND YOU HAVE DONE ALL THIS	
a1f9b2b6299e7a2b	eadec1e6ab797397	SEXY EARLOBES	<input type="text"/>
a1f9b2b6299e7a2b	617ab027727ad85	BEST TOS EPISODE	<input type="text"/>
39738b7adb0b8af7	617ab027727ad85	SUGARLAND	
1ab29ae86da6e5ca		NAME + JERSEY #	
877ab7889d3862b1		ALPHA	<input type="text"/>
877ab7889d3862b1			<input type="text"/>
877ab7889d3862b1			<input type="text"/>
877ab7889d3862b1			<input type="text"/>
877ab7889d3862b1		OBVIOUS	<input type="text"/>
877ab7889d3862b1		MICHAEL JACKSON	
38a7c9279cadeb44	9dca1d79d4dec6d5		
38a7c9279cadeb44	9dca1d79d4dec6d5	HE DID THE MASH, HE DID THE	<input type="text"/>
38a7c9279cadeb44		PURLOINED	<input type="text"/>
a8ae5745a7b7af7a	9dca1d79d4dec6d5	FAV. LATER-3. POKEMON	<input type="text"/>

THE GREATEST CROSSWORD PUZZLE  
IN THE HISTORY OF THE WORLD

# Pwned Websites (2)

- **Equifax** (2017): credit reporting agency
- **Starwood / Marriott** (2018, 500 million guests)
- **Twitter** (2018): user passwords were logged in plaintext
- **MyFitnessPal** (2018): user diet data, securely hashed passwords
- **Facebook** (2019): user data leaks (146 gigabytes)
- **Twitch** (2021): source code stolen ;)
- **Graff** (2021): jewellery, data on high-profile clients (Trump, Beckham, Oprah)
- **23andme** (2023): USA DNA testing...

# Client-side Security

- Client-side Scripting (JavaScript)
  - Isolated execution, resource policies
  - AJAX
- Websites affecting client-side security:
  - Cross-site scripting (XSS)
  - Cross-site request forgery (CSRF)
  - Tracking & Advertisements
- Browser vulnerabilities
- Legacy plugins: ActiveX, Java, Flash

# JavaScript

- The most popular ECMAScript implementation [12]
- Used for webpage scripting (dynamic content, animations)
  - Document Object Model
- It can also be used for server scripting (NodeJS)
- Sandboxed execution (e.g. cannot: read user's files, run external programs)
- Modern web applications rendered entirely in JavaScript
  - Angular, React, Polymer etc.

# XSS Attack [15]

- Cross-Site Scripting / client-side code injection
- E.g.: a messaging board website that allows HTML rich text:
- Someone posts:
  - I just wanted to say hello!
  - `<script>pwnThisSucker();</script>`
- If the target website doesn't filter this, the code will execute on any visitor's browser
- Code can steal data, infect the victims using a browser exploit etc.

# XSS Prevention

- Escape HTML before rendering
  - Convert "<" to "&lt;", ">" to "&gt;", quotes to "&quot;" etc.
  - Use a template engine that does this
- If rich text is required, use a whitelist-based HTML processor to sanitize!
  - Example: strip out dangerous tags like script, embed, iframe etc.
  - **WARNING:** Don't do this unless you know what you're doing!
  - Use a library designed to do this (e.g. [htmlpurifier.org](http://htmlpurifier.org))

# AJAX [13]

- Asynchronous JavaScript and XML
- XMLHttpRequest - API for issuing background HTTP requests
- Used to build modern, responsive applications
- XHR re-sends cookies for the requested domain!

```
var xhr = new XMLHttpRequest();  
xhr.open('get', 'ajax.php');  
xhr.onreadystatechange = function() { /*...*/ };  
xhr.send(null);
```



# Same / Cross Origin Policies [14]

- Same Origin = Same protocol + domain + port
  - Example: <http://domain.com> vs <https://www.domain.com>
- Used to prevent cross-domain data stealing
  - For example, a user enters [malicious.com](https://malicious.com)
  - [Malicious.com](https://malicious.com) makes a request for [facebook.com](https://facebook.com)
  - The request is made, but the response is discarded
- Does not prevent information leakage!
- CORS – Cross-Origin Resource Sharing

# CORS

- CORS – Cross-Origin Resource Sharing
- The target server sends special response headers:  
**Access-Control-Allow-Origin:** `https://*example.com`
- If the requester's domain matches this ACL, the browser accepts it
- Otherwise, the XHR will receive an error and the response text will be discarded

# CSRF [16]

- Cross-Site Request Forgery
- A malicious website tricks the browser / user into accessing a cross-origin URL
- Example (on [malicious.com](#)):

```

```

- Defenses:
  - Don't execute critical actions on GET requests!
  - Use CSRF tokens
  - Check headers (Referer, X-Requested-With etc.)

# Browser Privacy [17]

- Websites can track the user across multiple domains!
  - Cookies
  - Invisible objects or scripts that do remote requests
  - e.g.: Google AdSense, Google Analytics, Facebook etc.
- Browser Fingerprinting [19]
  - Test yourselves using EFF's [Panopticklick \[18\]](#)
- Tracking servers can become attack vectors!
- Extensions that block such requests [20]

# Browser Vulnerabilities

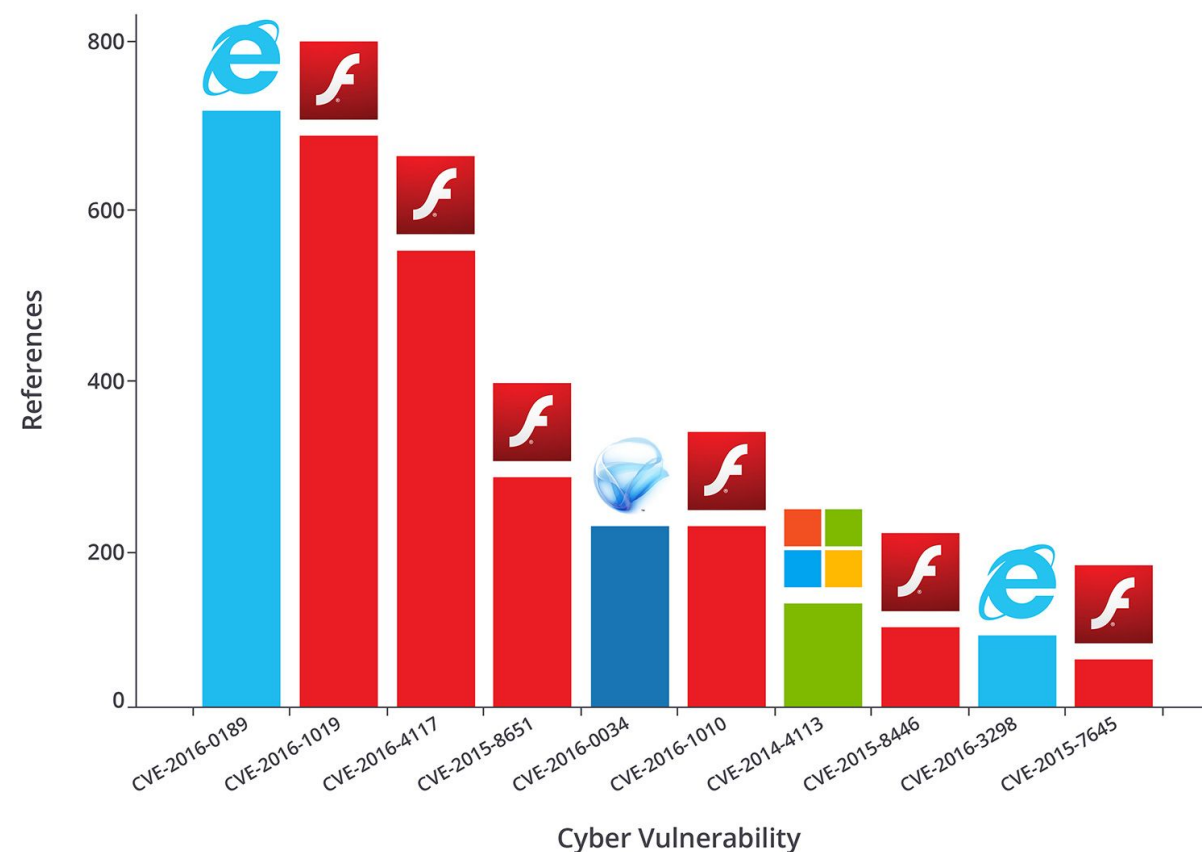
- Browsers are a complex piece of software
- May have vulnerabilities that allow attackers to escape sandboxing
- Attack vectors:
  - Malicious websites
  - Code injection on trusted websites (e.g. XSS)
  - Browser plugins: Flash, Java, ActiveX etc.



# Browser Vulnerabilities (2)

- **2015:** Adobe Flash had 96 vulnerabilities [21]!
- **2016:**
  - Flash most featured in exploit kits!
  - Internet Explorer second place [22]
- **Exploit kits:**
  - Angler, RIG, GrandSoft etc.

References vs. Cyber Vulnerability



# Browser Vulnerabilities (3)

- **Pwn2Own**: security competition for hacking browsers

- **2016** results [23]:

- 4 bugs in Internet Explorer 11
- 3 bugs in Mozilla Firefox
- 3 bugs in Adobe Reader
- 3 bugs in Adobe Flash
- 2 bugs in Apple Safari
- 1 bug in Google Chrome

- **2018** results [23]:

- 5 Apple Safari bugs
- 4 Microsoft Edge bugs
- 1 bug in Mozilla Firefox
- 1 bug in Google Chrome (unsuccessful exploitation)

- **2021**: Apple Safari exploit, Zoom Messenger, Chrome & MS Edge

# Secure Browsers

- If you want a secure browser:
  - Don't use Microsoft's Internet ExploDer!
  - Block all plugins by default
  - Always use the latest version of a browser
- Modern browsers employ multi-process sandboxing
  - One process per tab with no access to the user's system
  - Coordinate with a main browser process
  - Chromium uses namespaces + seccomp on Linux! [24]



# OWASP [25]

- The Open ~~Web~~ Worldwide Application Security Project
- OWASP Top 10 for **2021** (preview [26]):
  1. Broken Access Control
  2. Cryptographic Failures
  3. Injection (SQL, XSS etc.)
  4. Insecure Design
  5. Security Misconfiguration
  6. Vulnerable and Outdated Components
  7. Identification and Authentication Failures
  8. Software and Data Integrity Failures
  9. Security Logging and Monitoring Failures
  10. Server-Side Request Forgery

<https://owasp.org/www-project-top-ten/>

# References

- [1] HTTP <https://tools.ietf.org/html/rfc2616>
- [2] Murdoch, Steven J. "Hardened stateless session cookies." International Workshop on Security Protocols. Springer Berlin Heidelberg, 2008.
- [3] TLS protocol version 1.2, <https://tools.ietf.org/html/rfc5246> (2008)
- [4] TLS attacks,  
<https://www.cloudinsidr.com/content/known-attack-vectors-against-tls-implementation-vulnerabilities/>
- [5] Common Gateway Interface, <https://tools.ietf.org/html/rfc3875>
- [6] Clarke-Salt, Justin. SQL injection attacks and defense. Elsevier, 2009.
- [6] Flawed Tutorials, <https://arxiv.org/pdf/1704.02786.pdf>
- [7] Session Fixation: [http://www.acros.si/papers/session\\_fixation.pdf](http://www.acros.si/papers/session_fixation.pdf)

# References (2)

- [8] <https://fishbowl.pastiche.org/archives/docs/PasswordRecovery.pdf>
- [9] <http://www.pcmag.com/article2/0,2817,11525,00.asp>
- [10] Common Nginx + PHP Misconfiguration <http://bit.ly/1kAK8xu>
- [11] ShellShock, <http://www.securityfocus.com/bid/70103>
- [12] ECMA-262, <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- [13] <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>
- [14] [https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin\\_policy](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy)
- [15] <https://blogs.msdn.microsoft.com/dross/2009/12/15/happy-10th-birthday-cross-site-scripting/>

# References (3)

- [16] [https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/csrf\\_paper.pdf](https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/csrf_paper.pdf)
- [17] <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6234427>
- [18] <https://panopticlick.eff.org/>
- [19] How unique is your browser? <https://kabijo.de/files/13/14/5641571611600.pdf>
- [20] <http://lifelacker.com/the-best-browser-extensions-that-protect-your-privacy-479408034>
- [21] <https://heimdalsecurity.com/blog/adobe-flash-vulnerabilities-security-risks/>

# References (4)

[22] <https://www.recordedfuture.com/top-vulnerabilities-2016/>

[23] <https://venturebeat.com/2016/03/18/pwn2own-2016-chrome-edge-and-safari-hacked-460k-awarded-in-total/>

[24] [https://chromium.googlesource.com/chromium/src/+master/docs/linux\\_sandboxing.md](https://chromium.googlesource.com/chromium/src/+master/docs/linux_sandboxing.md)

[25] <https://www.owasp.org/>

[26] <https://owasp.org/Top10/>

[27] <https://www.thezdi.com/blog/2018/3/15/pwn2own-2018-day-two-results-and-master-of-pwn>