# Web Security

**Today: Florin Stancu**

Asst. Prof. Mihai Chiroiu

SYSTEMS LABORATORY

Computer Science & Engineering Department

# 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

<optional body>

**Server -> Client:**

HTTP/1.0 200 OK
Header1: value1
Header2: value2

<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>
```

**Generates**:

```
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]

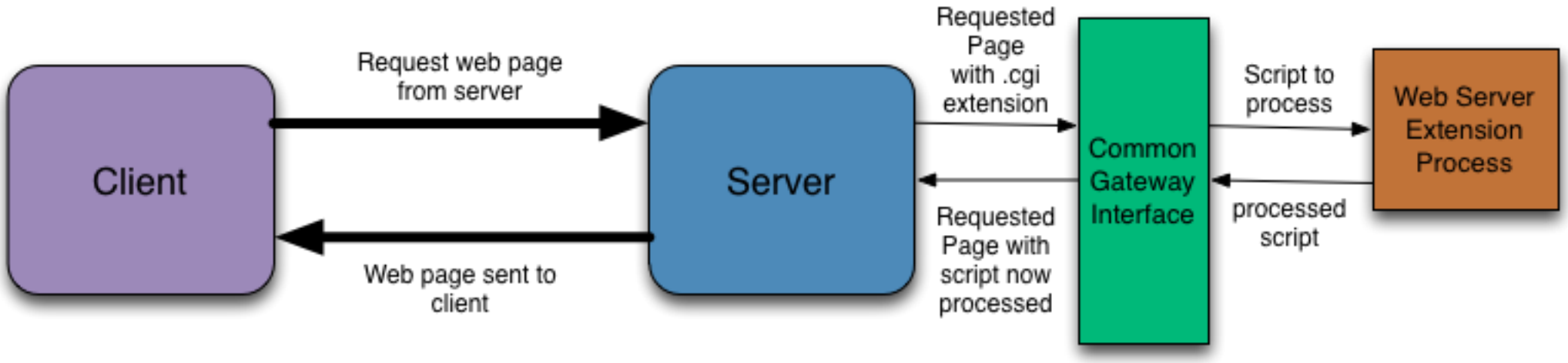
- Based on Secure Sockets Layer / Transport Layer Security
- Creates a private channel between the client and the server
- The server authenticates itself using certificates and PKI
- Diffie-Hellman for forward secrecy
- Cipher negotiation: RC4, DES, AES CBC, AES GCM etc.
- Target of numerous attacks

# TLS / HTTPS Attacks [4]

- Compression attacks (CRIME, BREACH)
- RC4 weaknesses
- Man-in-the-middle (Malicious Certificates, SSL stripping)
- Downgrade attacks (FREAK, Logjam, POODLE)
- Implementation bugs (e.g. Heartbeat, Cloudflare)

# Server-side Processing

- Server generates dynamic content
- Scripting interfaces: CGI (legacy) / FastCGI / apache2 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;
?>
```

# Server-side Injection (1)

## SQL Injection [6]

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

*POST: email=admin'--*

- => **SELECT * FROM** users **WHERE**
  - user='admin'-- AND password='`

# Server-side Injection (2)

- **File upload attack**
- **Example:**
  - A site allows image submissions with minimal verification
  - The user uploads image.gif.php with malicious code
  - User finds out the path to the image and requests it:
    - GET /uploads/image_9876.gif.php
  - Server executes our script (if badly configured ;) )

# Server-side Injection (3)

- **Preventing injection:**
  - **Do not trust tutorials [7]**
  - **Always sanitize user input!**
  - Try not to use shell execution / script evaluation
  - 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

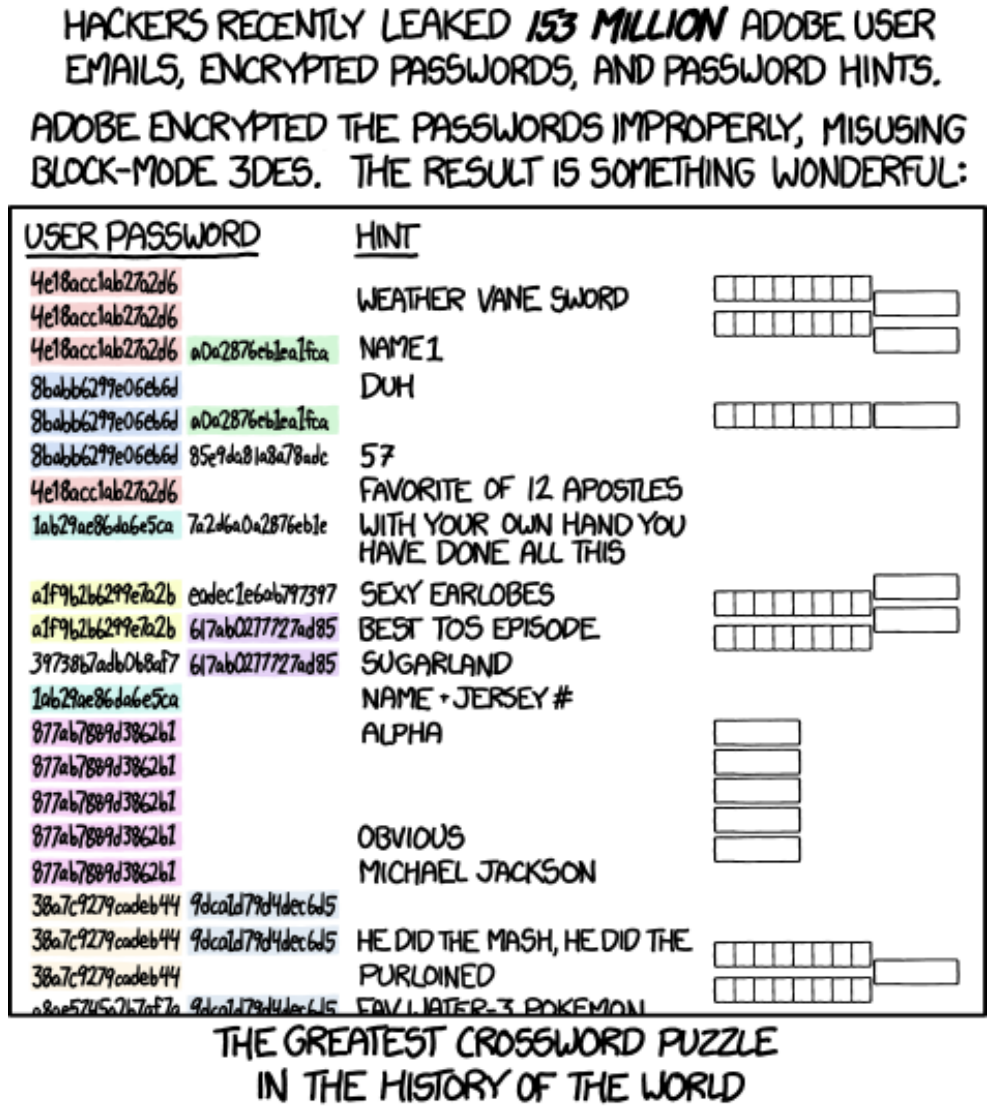
- Broken Authentication System [8]
  - Predictable / insecure session IDs
  - Unencrypted passwords [9]
- Authorization Vulnerabilities
  - Improper access verification
  - Example: /delete_user.php?id=5368
  - Direct object reference: /admin/list_users.php
- Vulnerable Frameworks / Plugins (e.g. many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

# Server Misconfiguration [9]

- **Again: do not trust tutorials**
  - 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** – check it home!
- **Yahoo!** (2012 – SQL Injection, 2013, 2014 – forged cookies)
  - 1 bilion accounts exposed!
- **LinkedIn** (hacked 2012, exposed in 2016)
- **Adobe** (2013): broken encryption =))
- **Dropbox** (2012):
  - SHA1 and salted passwords ;)






SYSTEMS LABORATORY

Computer Science & Engineering Department

# 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.

# 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
  - Malicious.com makes a request for 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: http://domain.com *(for HTTPS)*
  - Origin: http://domain.com *(for HTTP)*
- 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

# XSS [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 "<", ">" to ">", quotes to """ 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:** Dont do this unless you know what you're doing!
  - Use a library designed to do this (e.g. htmlpurifier.org)

# CSRF [16]

- Cross-Site Request Forgery
- A malicious website tricks the browser / user into accessing a cross-origin URL
- Example (on malicious.com):
  - `<img src="https://www.facebook.com/post/?msg=PWNED!"/>`
- Defenses:
  - Don’t execute critical actions on GET requests!
  - Use CSRF tokens
  - Check headers (Referer, X-Requested-With etc.)

# Browser Privacy [17]

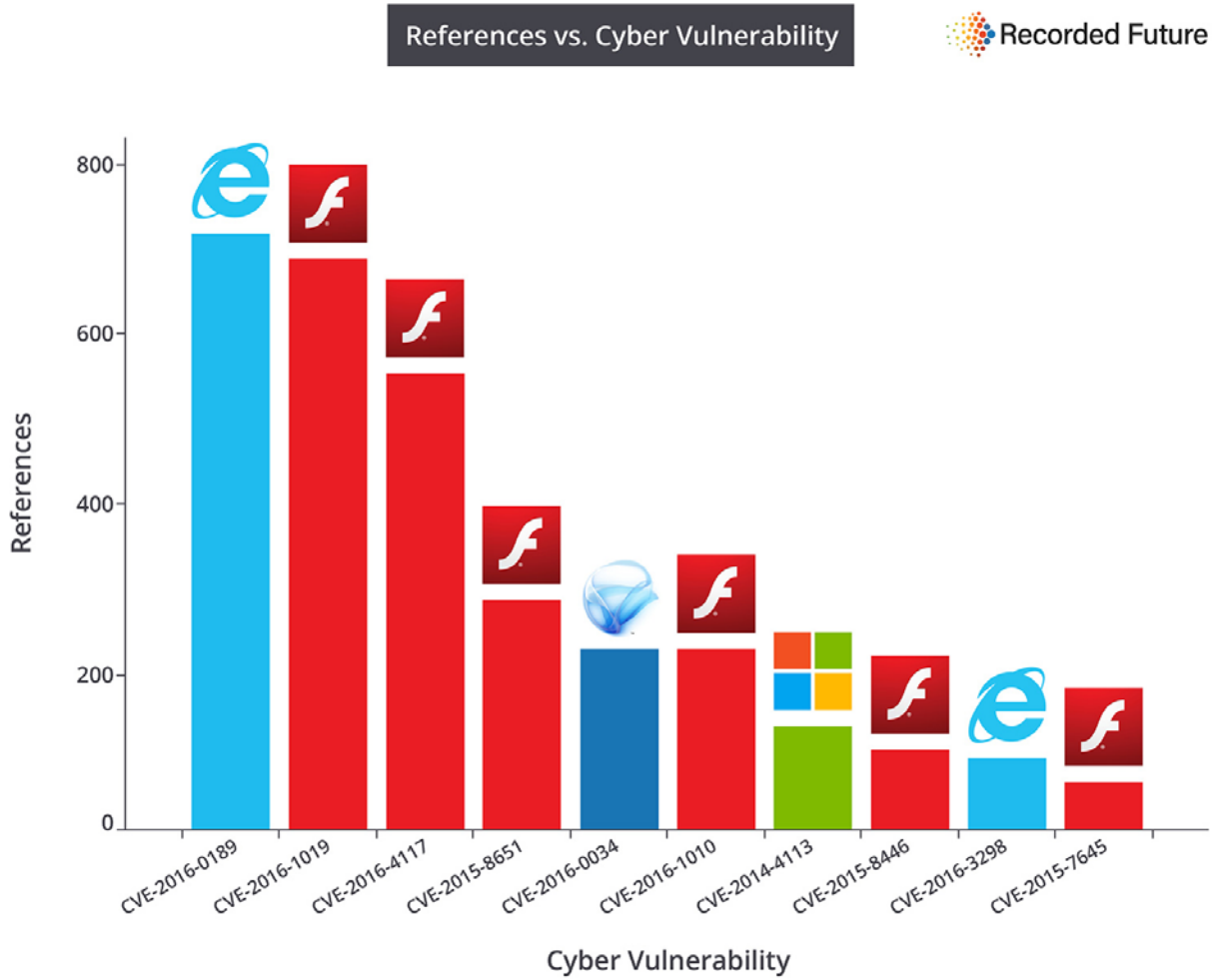
- Websites can track the user across multiple domains!
  - Cookies
  - Invisible objects or scripts that do remote requests
  - e.g.: Google Adsense, Google Analytics, Facebook etc.
- Browser Fingerprinting [19]
  - Test yourselves using EFF's Panopticlick [18]
- Tracking servers can become attack vectors!
- 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, Nuclear etc.



# Browser Vulnerabilities (3)

- **Pwn2Own**: security competition for hacking browsers
- **2016** results [23]:
  - 5 bugs in the Windows operating system
  - 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

# Secure Browsers

- If you want a secure browser:
  - Don’t use Microsoft’s Internet Explorer!
  - 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 even uses LXC namespaces on Linux! [24]

# OWASP [25]

- The Open Web Application Security Project
- OWASP Top 10 for **2017** (preview [26]):

1. Code Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Broken Access Control
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Insufficient Attack Protection
8. Cross-site Request Forgery
9. Using components with known vulnerabilities
10. Underprotected APIs

# 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.rfc-editor.org/rfc/pdfrfc/rfc7457.txt.pdf

[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

# 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

[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

[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://lifehacker.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

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

[26] https://raw.githubusercontent.com/OWASP/Top10/master/2017/OWASP%20Top%2010%20-%202017%20RC1-English.pdf