Co-funded by the
Erasmus+ Programme
of the European Union

ISSES – Information Security Services
Education in Serbia

Supported by the Erasmus+ Capacity Building in the field of Higher Education (CBHE) grant
N° 586474-EPP-1-2017-1-RS-EPPKA2-CBHE-JP

WEB HACKING

Information Security Services Education in Serbia (ISSES), Erasmus+ Key Action 2 – Capacity Building in the field of Higher Education (CBHE), University of Novi Sad, 2017 – 2020

ISSES 2017-2020
Novi Sad, Serbia

Vision

- From Zero to Hero in web hacking

- Web intro
- Injection
- Broken authentication & authorization
- Cross-site scripting
- Vulnerable 3rd party components

WEB INTRO

Loading a webpage

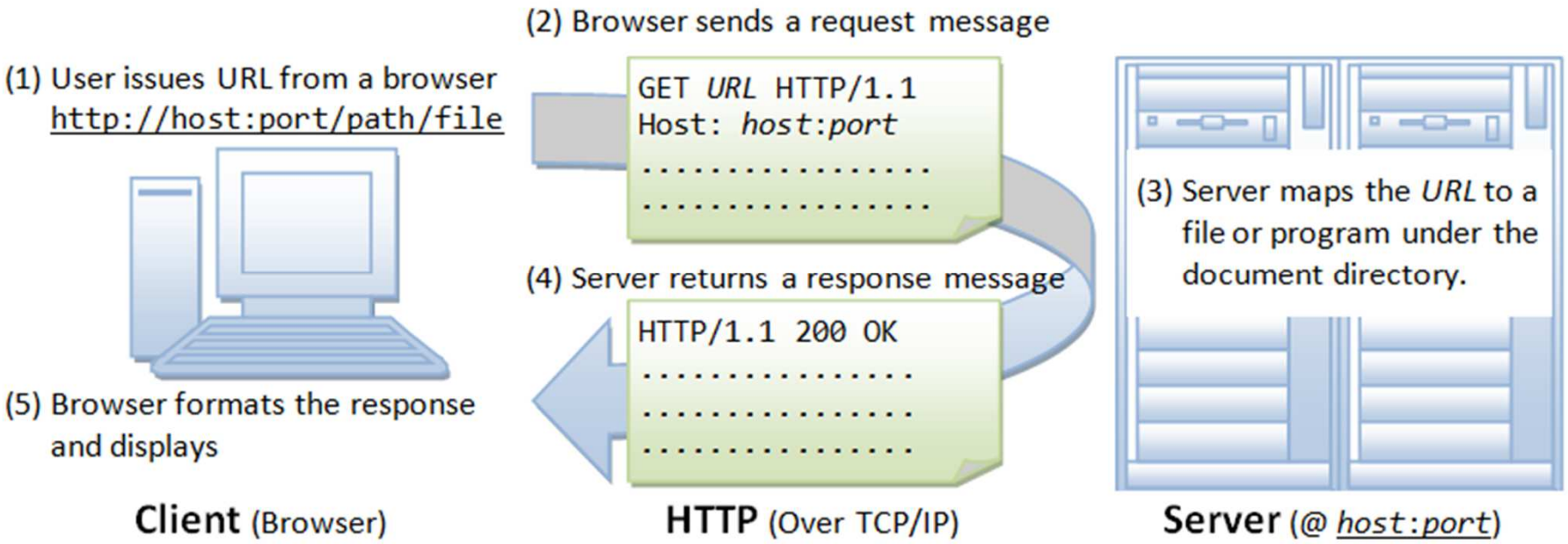
1. Fetch the given URL (Universal Resource Locator)
   a) Resolve the domain name to an IP address
   b) TCP connect to the IP address on port 80 (server)
   c) Send a HTTP GET request for the given path
   d) Receive the HTTP response with HTML data inside
2. Result: HTML (Hyper Text Markup Language) format data
3. Parse the HTML
   a) Request sub-resources as encountered by parser
      - Images
      - Stylesheets
      - Scripts
      - Frames
   b) Execute JavaScript as encountered by parser
4. Display content
5. Run event loop, dispatch events to JS

Uniform Resource Locator (URL)

https://www.b92.net/zivot/vesti.php?yyyy=2019&mm=12&dd=26&nav_id=1635357

Component name	URL component value
scheme/protocol used	http(s)://
domain name	www.b92.net
port (defaults to 80)	:80 (not shown above!)
path	/zivot/
document	vesti.php
query string	?yyyy=2019&mm=12&dd=26&nav_id=1635357
anchor	#internal_link (not present in the above example)

HTTP example



https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

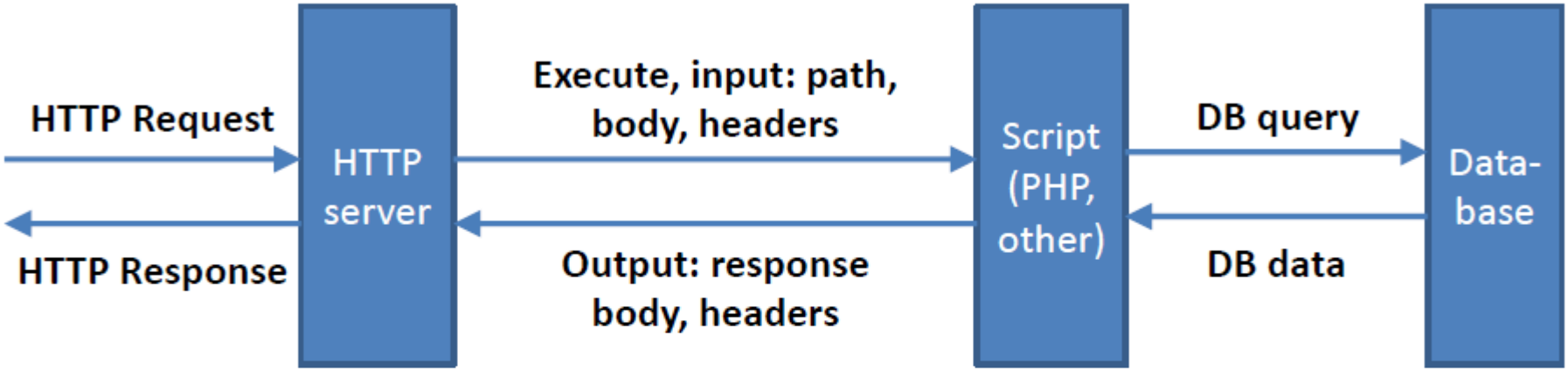
HTTP request & response

```
GET /zivot/vesti.php?yyyy=2019&mm=12&dd=26&nav_id=1635357 HTTP/1.1
Host: www.b92.net
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

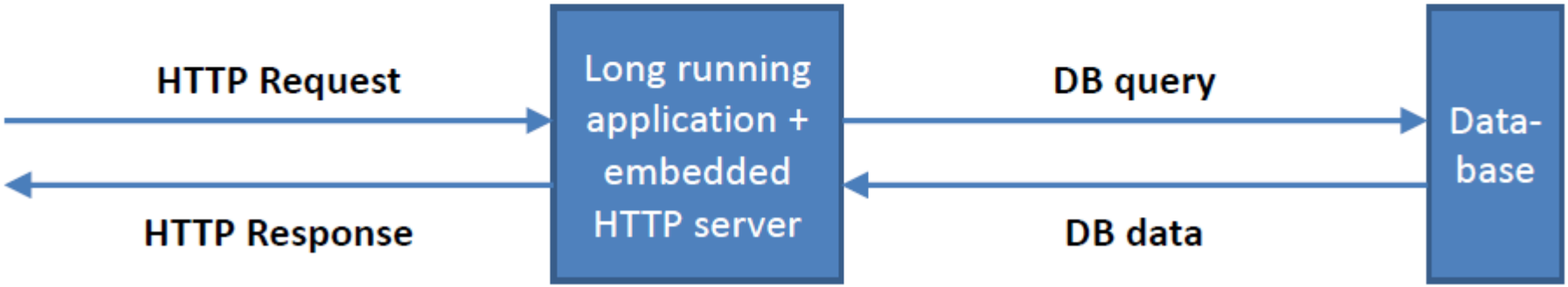
```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

Server-side architecture

- PHP and CGI (Common Gateway Interface):



- Python Tornado, Node.js, etc. (Event Driven):



- + load balancers, TLS terminators, distributed DBs, CDNs, etc.

HyperText Markup Language (HTML)

```
<html>
<head>
  <title>Example</title>
  <link href="/style.css" rel="stylesheet"
        type="text/css">
</head>
<body>
  <img src="/photo.jpg">
  <script src="/code.js">
  </script>
  <iframe src="/comments.html" id="comments">
  </iframe>
</body>
</html>
```

Parsing, processing ↓

HTTP GET /style.css

HTTP GET /photo.jpg

HTTP GET /code.js

HTTP GET /comment.html

HTML forms

```
<form action="/form.php" method="POST">
  <input type="text" name="name">
  <input type="checkbox" name="newsletter">
  <input type="submit">
</form>
```

Form submission when method="GET":

```
GET /form.php?name=joe&newsletter=on HTTP/1.1
```

Form submission when method="POST":

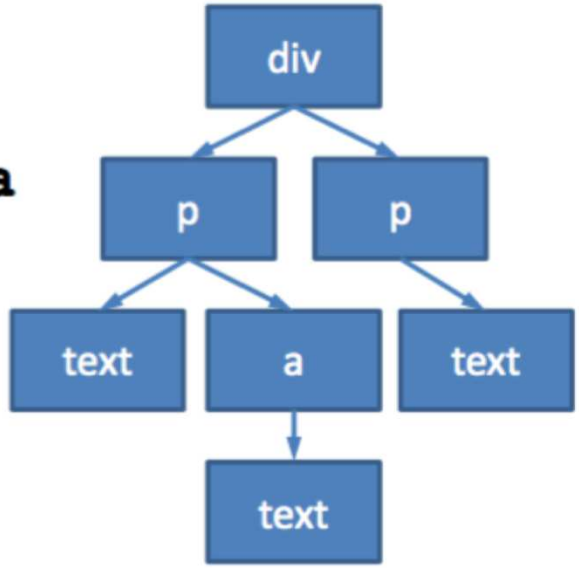
```
POST /form.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 19

name=joe&newsletter=on
```

Cookies

```
GET /index.php HTTP/1.1

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 100
Set-Cookie: PHPSESSID=asdfghjkl1234567890
…

GET /anything_on_same_server.php HTTP/1.1
Cookie: PHPSESSID=asdfghjkl1234567890
```

Document Object Model (DOM)

```
<div>
  <p>
    This is a paragraph with a
    <a>link</a>
  </p>
  <p>second paragraph</p>
</div>
```



- Document Object Model (DOM)
  - In-memory representation of HTML
  - Tree data structure
  - Set of JavaScript APIs for modification
- The jQuery JS library is an abstraction layer on DOM

JAVASCRIPT

JavaScript

- JavaScript is language created in 10 days
- Designed by Brendan Eich (Netscape) in 1995

- ECMA standardized it in 1996-1997
  - European Computer Manufacturers Association (ECMA)
- ECMAScript Edition 10 published in June 2019 → Active use and active development

Occurances

- Inline HTML:
  <script>...</script>

- HTML elements:

- Remote:
  <script src="example.com/glob.js"></script>

- CSS:
  BODY{background:url("javascript:alert('XSS')")}

Everything is Global (!)

Global language

- Variables have global scope by default (unless declared locally in a function)
- Functions have global scope
- Objects inherit from global prototypes
- No block scoping

Consequences

- Every script from the same origin can access and modify every variable, function, object and page content
- And remote scripts embedded with <script> are equal with other scripts

Danger! Danger!

- Different scripts can access each other's variables
- Different scripts can redefine each other's functions
- Scripts can override native methods
- Transmit data anywhere
- Watch keystrokes
- Steal cookies
- User click is equivalent to JavaScript click

Protecting JavaScript code

- Obfuscation methods
  - Renaming functions, variables, … everything
  - Structural transformations (changing layout, command order)
  - Data transformations (e.g. [][] → [], encoding all text)
  - Control transformations (e.g. regroup statements)

- In case of native code: automatic decompilation is harder
- JavaScript: it just makes the human understanding harder

- But obfuscation is just security by obscurity at the end

JavaScript security

- JavaScript is part of a Web browser and is executed in the client environment

- To minimize the risk posed by a malicious JavaScript code, browsers implement the following restrictions
  - Scripts run in a sandbox to stop a malicious web site from attacking your computer
  - Scripts are constrained by the Same Origin Policy, so a malicious web site cannot interact with another web site

SAME ORIGIN POLICY

Same Origin Policy

- Prevents elements on one page from accessing other pages
  - Origin refers to: domain name, protocol and port
  - Elements on a page without a specific origin (like e.g. JavaScript or data) inherit their origin from the page

- Cross-origin accesses
  - Cross-origin write (sending): allowed to any origin
    - Examples: links, redirects, form submission
    - Without this pages would only to link to themselves
    - CSRF and clickjacking is still possible (not in scope of SOP)
  - Cross-origin read (receiving): permitted only from same origin
    - But is typically circumvented by embedding
  - Cross-origin embedding: allowed from any origin
    - Examples: JavaScript (the <script> tag), CSS (<link…>), images (<img>), <audio>, <video>, <iframe>, …

Same Origin Policy → origin checking rules

URL	Result	Reason
http://www.example.com/dir/file.html	OK	Same protocol and host
http://www.example.com/dir2/file2.html	OK	Same protocol and host
http://www.example.com:8080/dir/file.html	X	Different port
https://www.example.com/dir/file.html	X	Different protocol
http://en.example.com/dir/file.html	X	Different host
http://example.com/dir/file.html	X	Different host

OWASP Top 10

OWASP Top 10 - 2017

- A1 – Injection
- A2 – Broken Authentication
- A3 – Sensitive Data Exposure
- A4 – XML External Entities
- A5 – Broken Access Control
- A6 – Security Misconfiguration
- A7 – Cross-Site Scripting (XSS)
- A8 – Insecure Deserialization
- A9 – Using Known Vulnerable Components
- A10 – Insufficient Logging & Monitoring

OWASP Top 10 progress

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

BROKEN AUTHENTICATION

Introduction

- Cybercriminals and other adversaries have access to millions of valid username/password combinations → e.g. **credential stuffing**, in which stolen username/password combinations are tried & used to gain unauthorized access
- Useful reference @ OWASP Top 10: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication

Web presence vulnerabilities

- No protection against credential stuffing
- Lack of protection against brute force attacks, i.e. username and/or password guessing
- Lack of proper password policy allowing weak passwords
- Weak or otherwise insecure password recovery policies
- Use of plain text passwords and/or weak cryptographic solutions for hashing password
- Exposed Session IDs and other sensitive information in the URL
- Improper session ID rotation and invalidation techniques → session re-use or hijacking

Protection

- Multi-factor authentication wherever possible, e.g. username & password + SMS code received on mobile
- Include protection against brute force attacks by locking out or delaying users after N invalid login attempts
- Do not allow default admin or any other credentials → enforce users to set up strong passwords as soon as they start using systems + check passwords against databases of known weak passwords
- Protect registration and password recovery API functions against discovery & misuse
- Incorporate strong session ID management on the server side

Useful challenges

- Avatao: OWASP Top 10 Intro → Broken Auth → Common passwords

BROKEN ACCESS CONTROL

Intro

- Access control allows system owners/operators to set up the access privileges for different users and user categories
- Broken access control allows adversaries to breach
  - Confidentiality → unauthorized data/service access
  - Integrity → unauthorized modification of data/services
  - Availability → rendering data/services unavailable via exploiting unauthorized (usually) administrative access

- OWASP Top 10 reference: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control

Vulnerabilities

- Web content allows adversaries to bypass access control via simply modifying the URL or by using a (usually custom) API access tool
- Elevation of privilege
  - Guest user acts as a logged in user
  - Logged in user acts as a (site) administrator
- Manipulating hidden data, e.g. JSON Web Tokens (JWT), cookies, hidden fields
- Access API with missing authentication for HTTP operations POST, PUT or DELETE.

Protection

- Deny access by default (in backend code)
- Disable web server directory listing
- Implement access control centrally and re-use in all-through the web applications
- Log access control failures → be creative and detect enumeration and repeated failed access attempts
- Thoroughly invalidate access control tokens after logout

Useful challenges

- Avatao: OWASP Top 10 Intro → Broken Access Control → Who is that?

INJECTION

Intro

- Injection flaws occur when an adversary is allowed to submit unwanted data to an interpreter, via environment variables, URL, HTML forms, etc.
  - E.g. in command injection attacks against web content the (web) server is tricked into executing OS commands
- Injection can result in the loss of
  - Confidentiality via gaining unauthorized access to data
  - Integrity via unauthorized data modification
  - Availability via unauthorized data deletion
- OWASP Top 10 reference: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A1-Injection

Vulnerability

- Improper (user) input validation & sanitization → dynamic queries or API calls are allowed without context-aware, strict checks
- Injections usually occur via SQL, NoSQL, OS command or LDAP

Protection

- Keep data and queries strictly separated
- Use safe APIs which are sufficiently opaque → adversaries are not able, or it is very expensive for them to develop injection(s)
- If parameterized queries are necessary, escape all special characters everywhere
- Implement measures against mass disclosure of records, e.g. LIMIT 1 in SQL

Useful challenges

- Avatao: OWASP Top 10 Intro → SQLi → Stay out of my database
- Avatao: OWASP Top 10 Intro → Command Injection → Ping the World

SQLi vulnerability – fix the code!

- Avatao: OWASP Top 10 Intro → SQLi → Stay out of my database

```
import sqlite3
from settings import DBPATH


def delete_item(row_id: str, user_id: str):
    #query = "DELETE FROM items WHERE rowid = '" + row_id + "' AND
    userid = '" + user_id + "';"
    query = "DELETE FROM items WHERE rowid = ? AND userid = ?"

    with sqlite3.connect(DBPATH) as connection:
        cursor = connection.cursor()
        cursor.execute(query, (row_id, user_id))

    # Please make sure to always return the query string for
    testing purposes!
    return query
```

Information Security Services Education in Serbia (ISSES)

CROSS-SITE SCRIPTING (XSS)

XSS steps

- **DEF:** Cross-site scripting is an HTML/script injection vulnerability
- If attackers can make a malicious script appear and run in a page (HTML) or data sent to victim's browser or application
- XSS usually relies on malicious JavaScript code
- Optionally it can be also HTML/HTML5 code, Flash or other executable code

Persistent XSS

- The script is put in a database (persistent store), e.g. a SQL table
- Script gets to the client and executes when visiting a site that renders some data from this database (e.g. forum)

Reflected XSS

- Relies on social engineering
- The user is persuaded to send a request with malicious data (script), which is put back to the response without validation

DOM-based XSS

- Also referred to as type-0 XSS
- Many websites process parts of the URL client-side – AJAX sites typically store page state information after the #, since that is never sent to the server
- If a part of the URL is processed in an unsafe manner, an attacker may be able to inject arbitrary DOM elements, including <script> tags

XSS actions on objects

- An attacker can do anything that the user can do with its session
- Completely change the look of the website (replace text, images, links, etc.)
- Steal your credit card info
- Do anything on the vulnerable site on your behalf (send/read messages, buy items, etc.)
- Mine cryptocurrencies using your computing power
- Log your activity

Useful XSS challenges

- Avatao: OWASP Top 10 Intro → XSS → The missing item

VULNERABLE 3RD PARTY LIBRARIES AND COMPONENTS

Intro

- Adversaries target the web presence by detecting that it uses/relies on components (usually outdated) with known vulnerabilities for which there are known exploits
- The risk is higher in complex environments relying on many different components and libraries, e.g. Wordpress website with many addons and libraries

Vulnerability

- Lack of proper component and library **inventory** tools → the web presence owner/operator does not know which libs and components are used by their system
- Lack of **proper and timely version control** leading to the continued use of outdated software with known vulnerabilities (for which there are known exploits)
- Lack of action when important security bulletins about the libs and/or components are issued → even worse, not even being aware of the security bulletins

Protection

- Implement proper lib and component inventory management, i.e. know what is used
- Obtain 3rd party libs and components from official & secure sources only
- Implement proper version control with emergency updates if needed → if an urgent security bulletin is issued, do not wait a whole month or quarter until next planned update, but update immediately instead
- Implement monitoring tools which identify unused libs and components → remove them

Useful challenges

- Avatao: OWASP Top 10 Intro → Using Components with Known Vulnerabilities → And then there were None

Summary

- Web intro
- Injection
- Broken authentication & authorization
- Cross-site scripting
- Vulnerable 3rd party components

Thank you for your attention!