

OFFENSIVE SECURITY

RECAP

PMF Kragujevac | OWASP Top 10

OFFENSIVE SECURITY RECAP

- Enumeration is the key
- Think like an attacker
- Google is the tool
- Penetration Testing
- Vulnerabilities



OWASP TOP 10

The Open Worldwide Application Security Project

A01:2021 Broken Access Control

A02:2021 Cryptographic Failures

A03:2021 Injection

A04:2021 Insecure Design

A05:2021 Security Misconfiguration

A06:2021 Vulnerable and Outdated Components

A07:2021 Identification and Authentication Failures

A08:2021 Software and Data Integrity Failures

A09:2021 Security Logging and Monitoring Failures

A10:2021 Server-Side Request Forgery

A01:2021

BROKEN ACCESS CONTROL

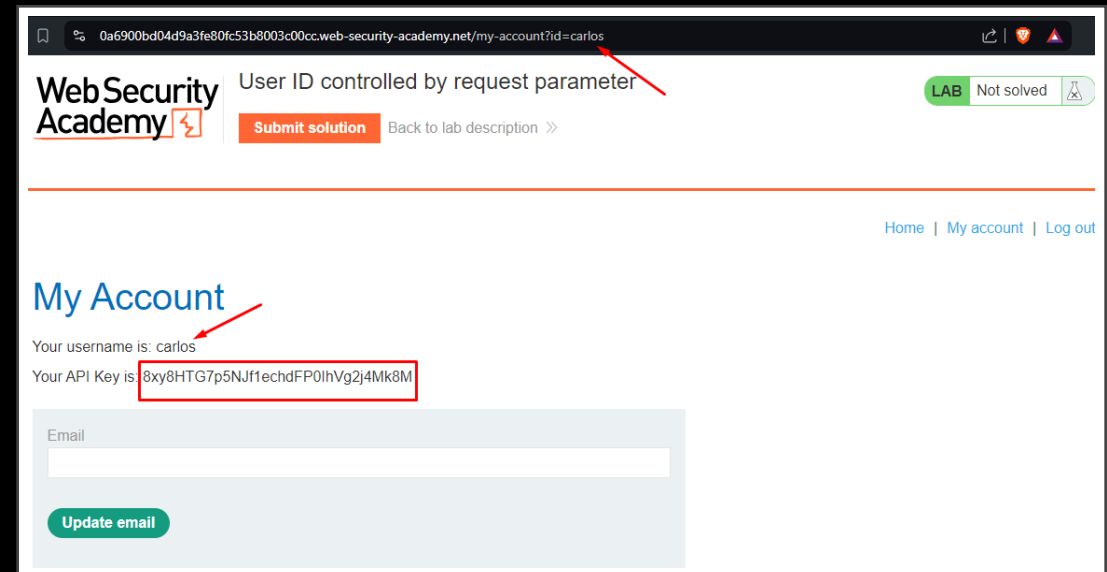
Lab: “User ID controlled by request parameter” - the account page is loaded based on an id value taken directly from the URL, with no check that it belongs to the logged-in user.



A01:2021

BROKEN ACCESS CONTROL

By simply changing the id parameter to another username (carlos), the attacker can view that user's account page and API key - exploiting the missing access control.



A02:2021

CRYPTOGRAPHIC FAILURES

- Use HTTPS instead of HTTP
- Do not store sensitive data unnecessarily
- Disable caching for responses that contain sensitive data
- Store passwords using strong adaptive and salted hashing functions (Argon2, bcrypt, scrypt)

A02:2021

CRYPTOGRAPHIC FAILURES - HASHING

How do hashing functions work?

A hashing function is a one-way function - a hash can't be "dehashed".

```
Data:
  1337haxor
  ↓
Algorithm does algorithmic things
  ↓
Hash:
  4d4b02cc5030d633823a26b9c9f69d2c
```

Hash (SHA-256): one-way

```
Scrambled data:
Travhf jvgubhg rqhpngvba vf yvxr
fvyire va gur zvar.
↓
(Caesar cipher) Key: ROT13
↑
Plaintext:
Remember to subscribe if you
haven't already
```

Encoding (ROT13): reversible

A02:2021

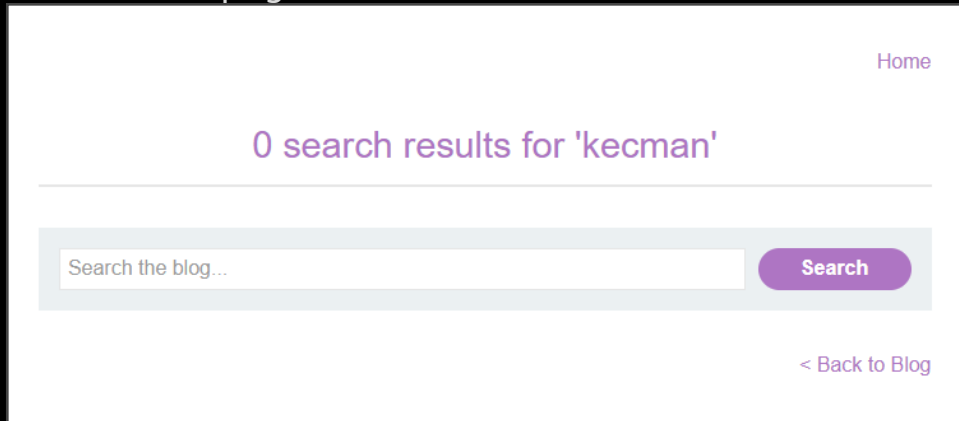
STORE PASSWORDS WITH STRONG HASHING

```
(Database)Username, PasswordHash  
  
snoopdog,  
55b45bc3c10804cd7014b675d377134c  
  
willywonker,  
654974d39ab36b76f8f3536f7d89c998  
  
aliG,  
be73f79cf63a15e48e23cdfa18e711df
```

A03:2021

INJECTION - CROSS-SITE SCRIPTING (XSS)

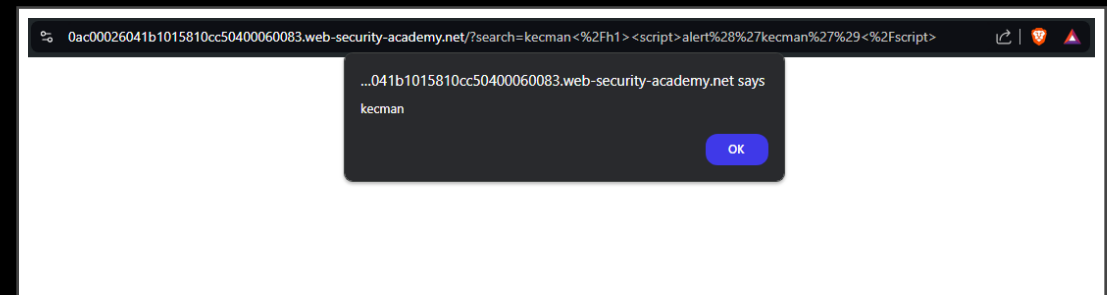
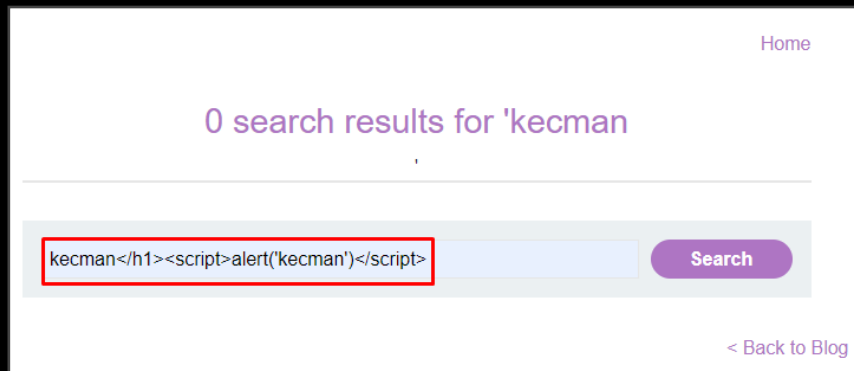
A search for 'kecman' returns no results - and the value is reflected back into the page's HTML, as visible in the DOM below.



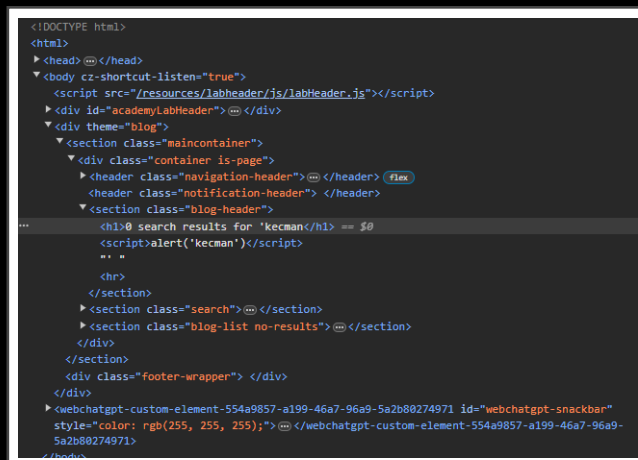
```
<!DOCTYPE html>
<html>
  <head> </head>
  <body cz-shortcut-listen="true">
    <script src="/resources/labheader/js/labHeader.js"></script>
    <div id="academyLabHeader"> </div>
    <div theme="blog">
      <section class="maincontainer">
        <div class="container is-page">
          <header class="navigation-header"> </header> flex
          <header class="notification-header"> </header>
          <section class="blog-header">
            <hr>
            <h1>0 search results for 'kecman'</h1> == $0
          </section>
          <section class="search"> </section>
          <section class="blog-list no-results"> </section>
        </div>
      </section>
      <div class="footer-wrapper"> </div>
    </div>
    <webchatgpt-custom-element-5d95cff5-9666-4df2-b135-2037ba2d9cbc id="webchatgpt-snackbar"
      style="color: rgb(255, 255, 255);"> </webchatgpt-custom-element-5d95cff5-9666-4df2-b135-
      2037ba2d9cbc>
  </body>
</html>
```

A03:2021

INJECTION - REFLECTED XSS EXPLOITED



Injecting a payload into the search parameter breaks out of the HTML and injects a `<script>` tag, which executes in the victim's browser and triggers a JavaScript alert.



A03:2021

INJECTION - SQL INJECTION

Login

Username

kecman

Password

kecman123

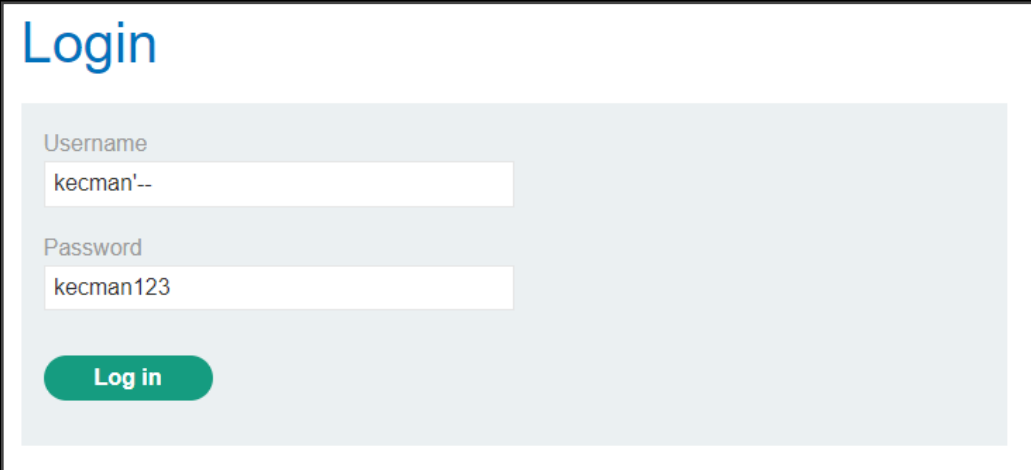
Log in

The submitted username and password are inserted directly into a SQL query:

```
SELECT * FROM users WHERE username = 'kecman' AND password = 'kecman123'
```

A03:2021

INJECTION - SQL INJECTION BYPASS



Login

Username
kecman'--

Password
kecman123

Log in

Appending -- comments out the rest of the query, bypassing the password check:

```
SELECT * FROM users WHERE username = 'kecman'--' AND password = 'kecman123'
```

A04:2021

INSECURE DESIGN

A secure design can still have implementation defects leading to vulnerabilities that may be exploited. An insecure design cannot be fixed by a perfect implementation, as by definition, needed security controls were never created to defend against specific attacks.

Example Attack Scenario

A credential recovery workflow might include “questions and answers,” which is prohibited by NIST 800-63b, the OWASP ASVS, and the OWASP Top 10. Questions and answers cannot be trusted as evidence of identity, since more than one person can know the answers. Such code should be removed and replaced with a more secure design.

A decorative horizontal bar at the top left of the slide, featuring a gradient from yellow to orange to red, with a teal section on the far left. The bar is partially cut off by the right edge of the slide.

A05:2021

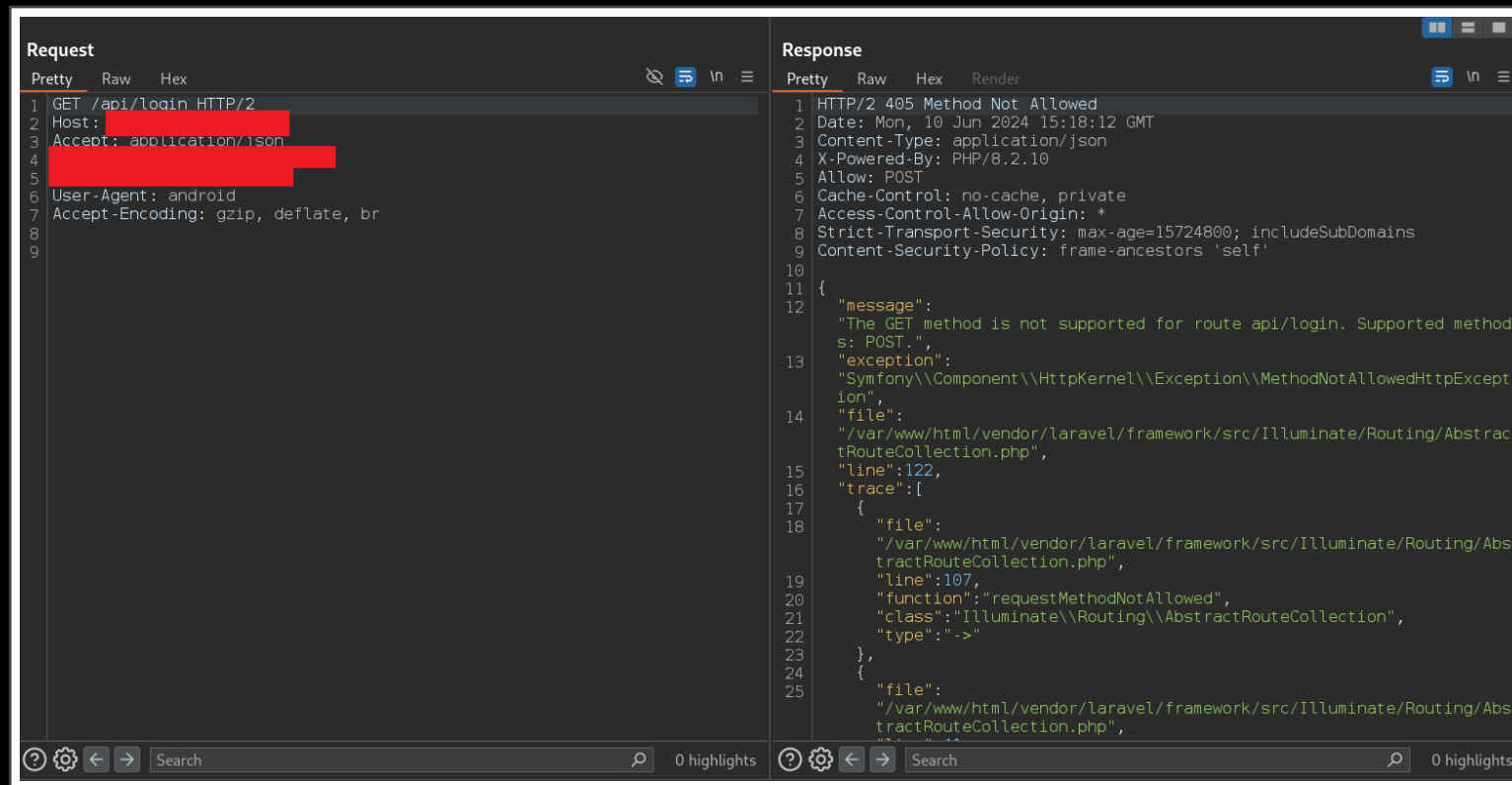
SECURITY MISCONFIGURATION

- Using default credentials
- Bad error handling
- Weak password policy
- Unnecessary features enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges)

A05:2021

SECURITY MISCONFIGURATION - BAD ERROR HANDLING

Verbose error messages leak internal file paths, framework details, and stack traces:



A06:2021

VULNERABLE AND OUTDATED COMPONENTS

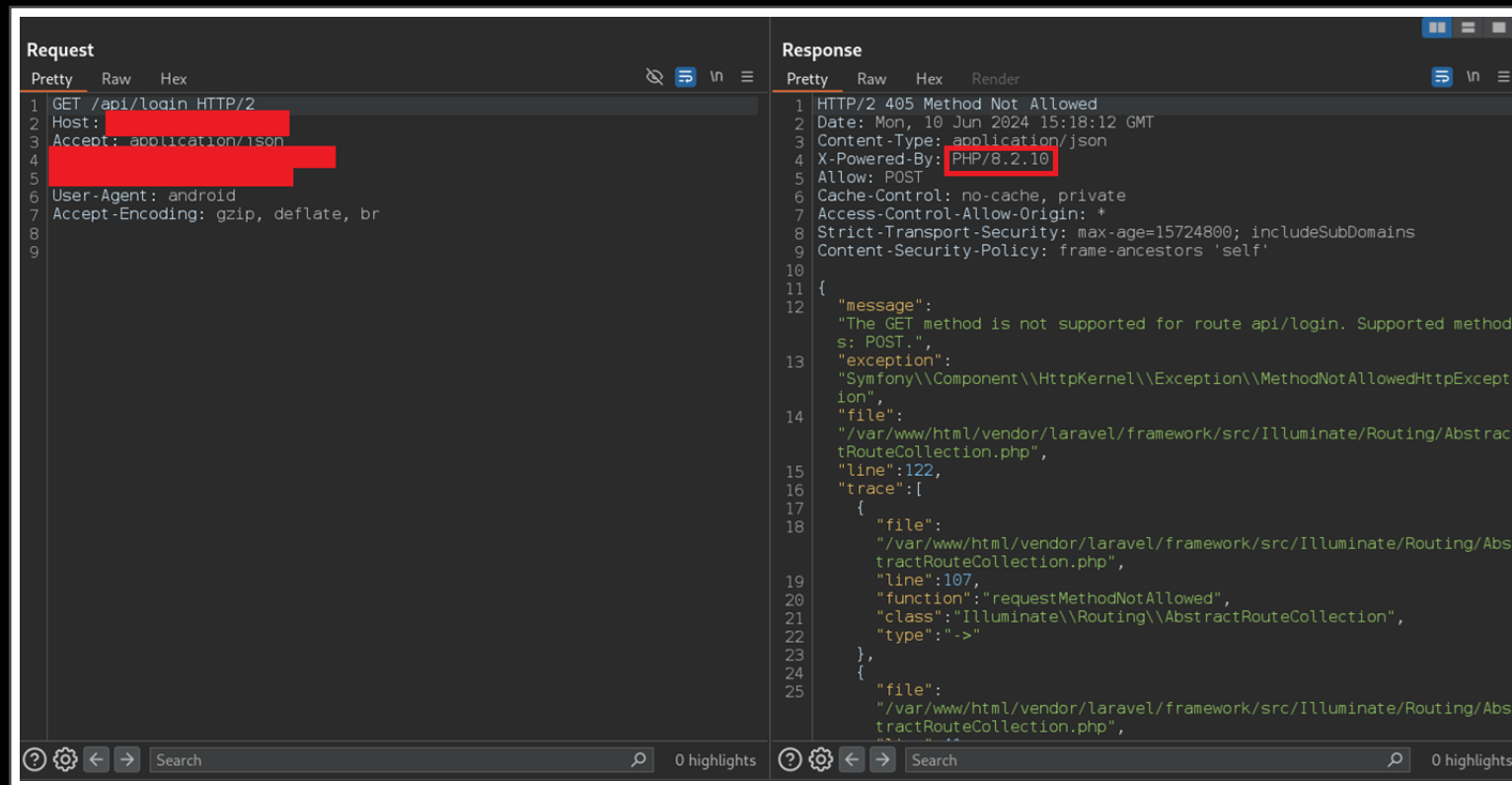
Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g., a coding error) or intentional (e.g., a backdoor in a component).

- CVE-2017-5638 - a Struts 2 remote code execution vulnerability enabling arbitrary code execution on the server, blamed for significant breaches
- IoT devices are frequently difficult or impossible to patch, yet the importance of patching them can be great (e.g., biomedical devices)
- Automated tools such as Shodan help attackers find unpatched or misconfigured systems, e.g. devices still vulnerable to Heartbleed (patched April 2014)

A06:2021

VULNERABLE COMPONENTS - FINGERPRINTING

The X-Powered-By response header reveals the exact PHP version in use:



A06:2021

VULNERABLE COMPONENTS - KNOWN CVEs

That version maps directly to publicly disclosed, critical CVEs:

PHP 8.2.x < 8.2.18 Multiple Vulnerabilities

CRITICAL

Web App Scanning Plugin ID 114271

Synopsis

PHP 8.2.x < 8.2.18 Multiple Vulnerabilities

Description

According to its self-reported version number, the version of PHP installed on the remote host is 8.1.x prior to 8.1.28, 8.2.x prior to 8.2.18, or 8.3.x prior to 8.3.6. It is, therefore, affected by multiple vulnerabilities:

- A command injection via array-ish \$command parameter of proc_open. (CVE-2024-1874)
- __Host-/_Secure- cookie bypass due to partial CVE-2022-31629 fix. (CVE-2024-2756)
- password_verify can erroneously return true, opening ATO risk. (CVE-2024-3096)
- mb_encode_mimeheader runs endlessly for some inputs. (CVE-2024-2757)

Note that the scanner has not tested for these issues but has instead relied only on the application's self-reported version number.

Solution

Upgrade to PHP version 8.2.18 or later.

A07:2021

IDENTIFICATION AND AUTHENTICATION FAILURES

- Permits brute force or other automated attacks (missing rate limiting)
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin"
- Uses weak or ineffective credential recovery processes, such as "knowledge-based answers"
- Uses plain text, encrypted, or weakly hashed password data stores
- Has missing or ineffective multi-factor authentication
- Exposes the session identifier in the URL, or reuses it after a successful login

A07:2021

AUTHENTICATION FAILURES - SCENARIOS

Scenario #1

Credential stuffing, using lists of known passwords, is a common attack. If an application does not implement automated threat or credential-stuffing protection, it can be used as a password oracle to determine whether credentials are valid.

Scenario #2

Most authentication attacks occur due to continued reliance on passwords as the sole factor. Password rotation and complexity requirements, once considered best practice, actually encourage users to reuse weak passwords. NIST 800-63 recommends multi-factor authentication instead.

Scenario #3

Application session timeouts that aren't set correctly can leave a user authenticated on a shared or public computer long after they've walked away, allowing an attacker to take over the session.

A08:2021

SOFTWARE AND DATA INTEGRITY FAILURES

Scenario #1

Update without signing: Many home routers, set-top boxes, and device firmware do not verify updates via signed firmware. Unsigned firmware is a growing target, and there is often no remediation mechanism other than a future fix.

Scenario #2

The SolarWinds Orion attack: nation-state actors subverted the vendor's build and update integrity process, distributing a highly targeted malicious update to more than 18,000 organizations - one of the most far-reaching breaches of its kind.

A09:2021

SECURITY LOGGING AND MONITORING FAILURES

Scenario #1

A children's health plan provider could not detect a breach due to a lack of monitoring and logging. An external party discovered the attacker had accessed and modified sensitive health records of more than 3.5 million children - the breach may have been ongoing since 2013, over seven years.

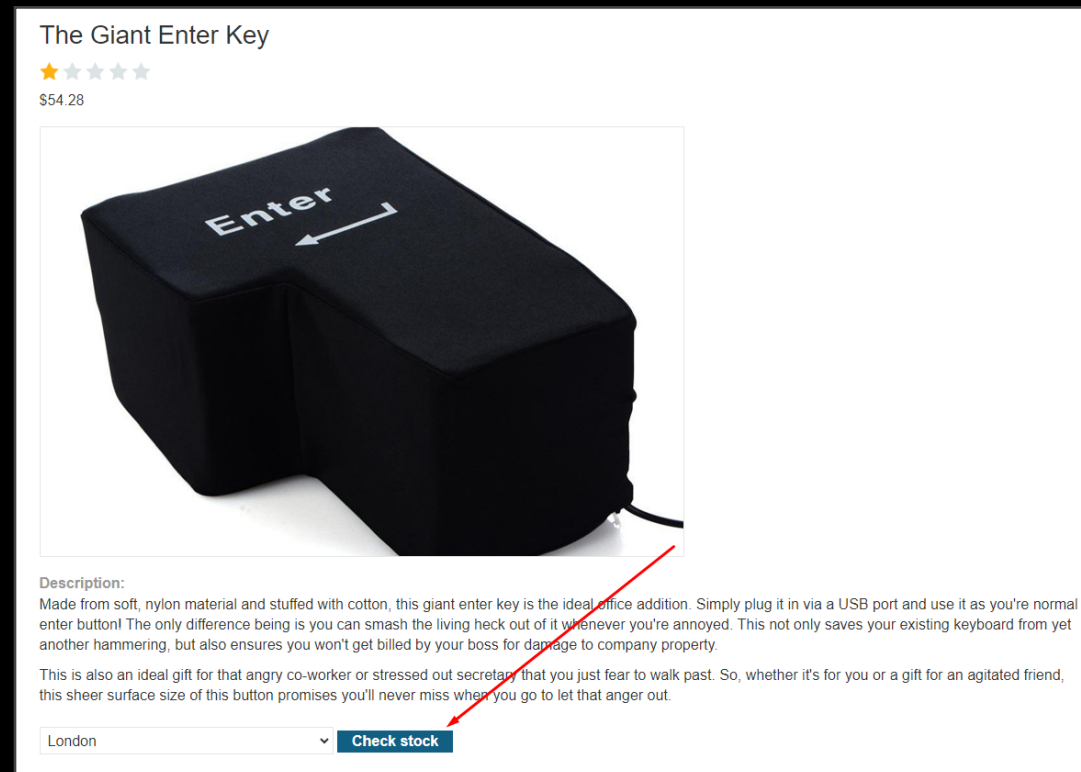
Scenario #2

A major Indian airline suffered a data breach involving more than ten years of passenger data, including passport and credit card details. The breach occurred at a third-party cloud hosting provider, which only notified the airline some time later.

A10:2021

SERVER-SIDE REQUEST FORGERY (SSRF)

A product page includes a “Check stock” feature that queries stock by store location:



A10:2021

SSRF - INSPECTING THE REQUEST

The stock check is performed server-side via a stockApi parameter containing a full URL:

```
POST /product/stock HTTP/2
Host: 0a3c00ca039d39a5812ca7a4004b00c2.web-security-academy.net
Cookie: session=ZDuvFlqYDWe1xVzZss6IX0qQ7ns1IMsm
Content-Length: 87
Sec-Ch-Ua: "Not/A)Brand";v="8", "Chromium";v="126", "Brave";v="126"
Sec-Ch-Ua-Platform: "Windows"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: */*
Sec-Gpc: 1
Accept-Language: en-US,en;q=0.7
Origin: https://0a3c00ca039d39a5812ca7a4004b00c2.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://0a3c00ca039d39a5812ca7a4004b00c2.web-security-academy.net/product?productId=1
Accept-Encoding: gzip, deflate, br
Priority: u=1, i

stockApi=
http://stock.weliketoshop.net:8080/product/stock/check?productId=1%26storeId=1
```

```
1 HTTP/2 200 OK
2 Content-Type: text/plain; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 3
5
6 182
```

A10:2021

SSRF - REACHING INTERNAL SERVICES

Replacing the stockApi URL with http://localhost/admin reaches an internal-only admin panel:

The screenshot displays a web browser interface with two main panes: 'Request' and 'Response'.

Request Pane: Shows the details of an HTTP POST request to `/product/stock`. The request includes various headers such as `Host`, `Cookie`, `Content-Length`, `Sec-Ch-Ua`, `Sec-Ch-Ua-Platform`, `Sec-Ch-Ua-Mobile`, `User-Agent`, `Content-Type`, `Accept`, `Sec-Gpc`, `Accept-Language`, `Origin`, `Sec-Fetch-Site`, `Sec-Fetch-Mode`, `Sec-Fetch-Dest`, `Referer`, `Accept-Encoding`, and `Priority`. The body of the request is `stockApi=http://localhost/admin`, which is highlighted with a red box.

Response Pane: Shows the response from the internal admin panel. The title is 'Basic SSRF against the local server'. The response includes a 'Web Security Academy' logo and a 'Users' section with links for 'Home', 'Admin panel', and 'My account'. The 'Users' section lists 'wiener' and 'carlos' with 'Delete' links.

A10:2021

SSRF - ATTACK OVERVIEW



A10:2021

SSRF - EXAMPLE ATTACK SCENARIOS

Scenario #1

Port scan internal servers: on an unsegmented network, attackers can map internal hosts and determine open or closed ports from SSRF connection results or timing.

Scenario #2

Sensitive data exposure: attackers can reach local files or internal services, e.g. `file:///etc/passwd` or `http://localhost:28017/`.

Scenario #3

Access cloud metadata storage: most cloud providers expose metadata at addresses such as `http://169.254.169.254/`, which an attacker can read for sensitive information.

Scenario #4

Compromise internal services: attackers can abuse internal services for further attacks such as Remote Code Execution (RCE) or Denial of Service (DoS).

EXERCISE 3

- Find all interesting directories of the target website
- Find out if there are any subdomains of the target

HOMEWORK

- Install Burp Suite
- Install OWASP Juice Shop locally

OPTIONAL HOMEWORK

Develop a small custom app that includes an example of:

- Stored XSS
- Reflected XSS
- Self XSS
- DOM Based XSS

QA

damjan@thecyberprotect.com