LakeCTF Finals Writeup

phishing

by maitai

Overview

```
.
├── fishing-bot/
│   ├── bot.py
│   └── main.py
└── fishing-web/
    ├── models/
    │   ├── post.js
    │   └── user.js
    ├── routes/
    │   ├── admin.js
    │   ├── auth.js
    │   ├── moderator.js
    │   └── posts.js
    ├── server.js
    └── views/
        ├── register.ejs
        ├── login.ejs
        ├── post.ejs
        ├── new-post.ejs
        ├── logs.ejs
        ├── index.ejs
        └── flag.ejs
```

Speed pwning gone wrong

- Quickly grepped for <%-
- Quickly looked at the bot source code
- Quickly understood that this should took a while
- Quickly found out that there were no obvious bugs
- Not-so-quickly changed challenge

(this is exactly how it went at EPFL)

Upsolving

- There are quite a lot of source files
- We can narrow down what we need to do by understading the bot behaviour

```
def visit(postId):
  chrome_options=webdriver.ChromeOptions()
  chrome_options.add_argument("--headless")
  chrome_options.add_argument("--no-sandbox")
  driver = webdriver.Chrome(chrome_options=chrome_options)
  driver.get("http://fishing-web:3000/posts/"+postId)
  wait(driver, 3)
  driver.execute_script('document.querySelectorAll("a").forEach(a => {if (a.innerText == "[Login to view this content]") {a.click(); return}})')
  wait(driver, 3)
  if not driver.current_url.endswith("/login"):
    driver.quit()
    return
  usernameEl = driver.find_element_by_id("username")
  usernameEl.send_keys(os.environ['MODERATOR_USERNAME'])
  passwordEl = driver.find_element_by_id("password")
  passwordEl.send_keys(os.environ['MODERATOR_PASSWORD'])
  butEl = driver.find_element_by_id("submitbut")
  butEl.click()
  wait(driver, 10)
  driver.quit()
```

The Bot

- Visits a post
- Selects all the anchor tag and click only on those containing [Login to view this content]
- Waits for 3 seconds
- Checks if the page he is current in ends with /login
- Insert the username and password, in order to login
- Waits for 10 seconds

Deh Bot

- It's pretty obvious that our post it's the entrypoint for the exploit

```
create: (title, author, content) => {
        id = crypto.randomBytes(16).toString('hex')
        const insertPostQuery = `
            INSERT INTO posts (id, title, author, content, approved) VALUES (?, ?, ?, ?, ?)
        `;
        const window = new JSDOM('').window;
        const DOMPurify = createDOMPurify(window);
        const sanitizedContent = DOMPurify.sanitize(content, {ALLOWED_TAGS: ['a'], ALLOWED_ATTR: ['href']});
        return new Promise((resolve, reject) => {
            db.run(insertPostQuery, [id, title, author, sanitizedContent, false], function (err) {
                if (err) {
                    reject(err);
                } else {
                    resolve(id);
                }
            });
        });
    }
```

- Impossible not to have DOMPurify

Deh ✕ 💪 Bot

- DOMPurify allows only the anchor tag with the href attribute
- This is more than enough to trick the bot right?

```
router.get('/:postId', async (req, res) => {
    const postId = req.params.postId;
    const isAuth = req.session.user && (await User.findByUsername(req.session.user)) && (await User.findByUsername(req.session.user)).permissions.includes('user')
    const isMod = req.session.user && (await User.findByUsername(req.session.user)) && (await User.findByUsername(req.session.user)).permissions.includes('moderator')
    try {
        const post = await Post.getById(postId);
        if (!post) {
            return res.status(404).send('Post not found');
        }

        const dom = new JSDOM("<p id=content>"+post.content+"</p>");
        const content = dom.window.document.getElementById('content');
        content.querySelectorAll('a').forEach(a => {
            try {
                if (parse(a.href).hostname !== parse(req.headers.host).hostname && !isAuth) {
                    a.text = "[Login to view this content]"
                    a.href = "/login";
                }
            } catch (e) {
                a.innerText = "[Login to view this content]"
                a.href = "/login";
            }
        });

        res.render('post', { post, content: content.innerHTML, isMod });
    } catch (error) {
        console.error(error);
        return res.status(500).send('Internal Server Error');
    }
});
```

Oday (?)

- Here is where we got stucked
- Essentially we cannot usa any html tag besides the anchor
- Even though we can use it we are not able to phish the admin to our malicious domain due to this parse function

First Bug - Parser Differential

- We need to find a valid syntax for the parse function which passes the check
- At the same time we need to find a valid syntax for Chrome that will lead the bot to our domain

```
\\super-bligo:9999\x@fishing-web:3000/../
```

- The function will return fishing-web as host while Chrome will change alle the \ into / and keep super-bligo as host
- Neat trick we should keep that in mind for the future

It's phishing right?

- Now the admin is on our domain, we can just steal his credentials and login as him, right?

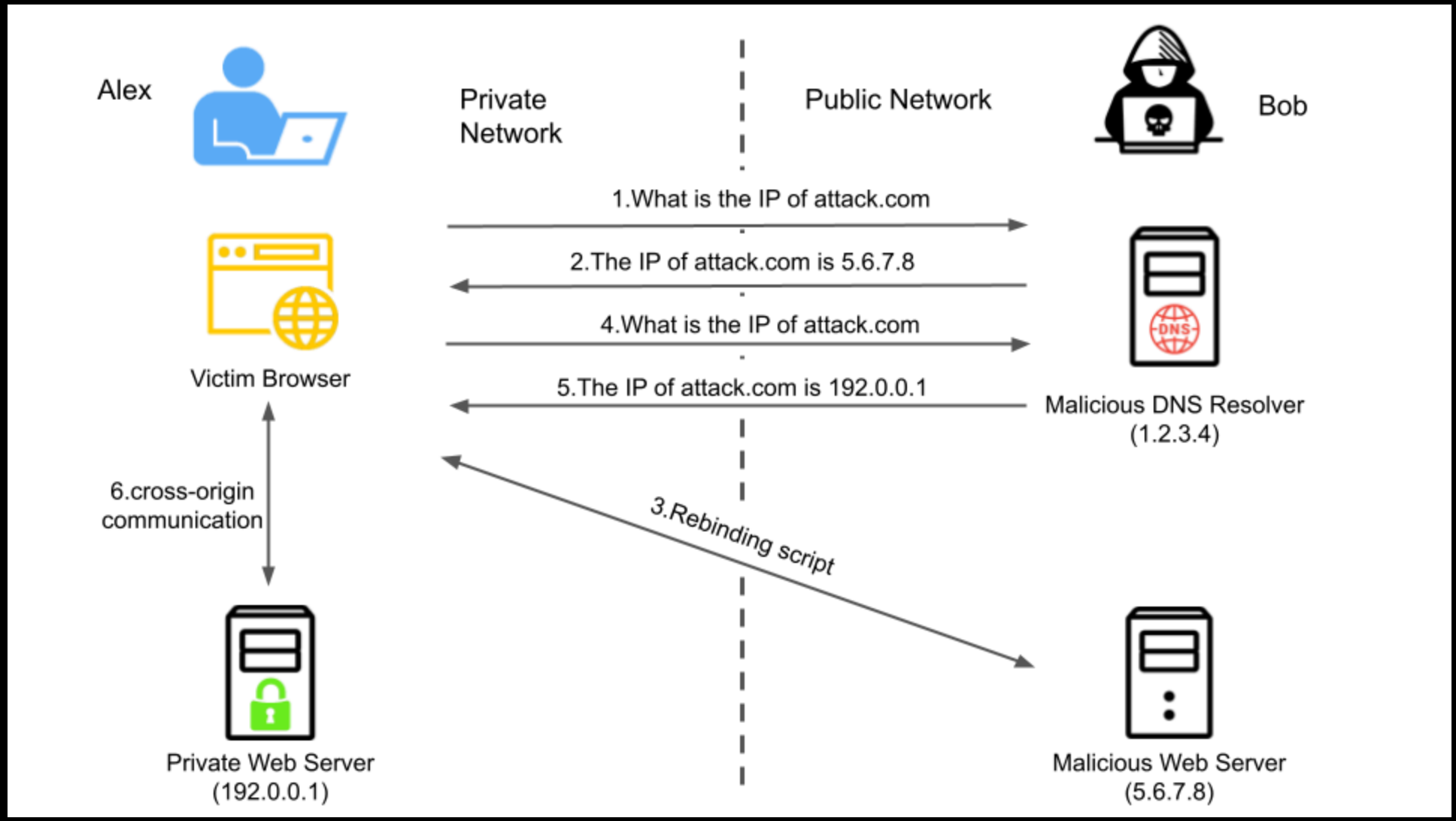
```
router.post('/login', async (req, res) => {
    const { username, password } = req.body;
    if (typeof username !== 'string' || typeof password !== 'string') {
        return res.status(400).send('{"error":"Bad Request"}');
    }
    try {
        let user = await User.findByUsername(username);
        if (!user) {
            user = {}
            user.permissions = JSON.stringify(["user"])
            user.password = "$2b$10$XeKD8ih3RR3aZUA7iHhZfe.MiOKRfkf7ViY0qr2h2lv/AD9OU2msK" // error out but keep the bcrypt check to avoid side channel, this hash is not brute-foceable
        }
        user.permissions = JSON.parse(user.permissions)
        const match = await bcrypt.compare(password, user.password);
        if (match && user.ip == req.socket.remoteAddress.replace(/^.*:/, '')) {
            req.session.user = user.username;
            return res.status(200).send("{}");
        } else {
            return res.status(401).send('{"error":"Invalid username or password or ip"}');
        }
    } catch (error) {
        console.error(error);
        res.status(500).send('{"error":"Internal Server Error"}');
    }
});
```

- Impossible not to have an IP check

IP Check Bypass ? ❌ 💪

- We are now supposed to make the admin perform some actions but first he needs to login.
- We cannot steal the cookie since is SameSite=Strict and HttpOnly
- Actually we need to find a way for the cookie to be sent, that we can partially control
- Enter: DNS Rebinding

DNS Rebinding



Deh ✕ 💪 Rebinding

- This could allow us to bypass all of the restriction imposed
- This is enough to fetch the flag, right?

```
const createUserTable = async () => {
    const createUserTableQuery = `
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT NOT NULL UNIQUE,
            password TEXT NOT NULL,
            permissions TEXT NOT NULL,
            ip TEXT NOT NULL
        );
    `;
    db.run(createUserTableQuery);
    await new Promise(resolve => setTimeout(resolve, 1000));
    User.findByUsername(MODERATOR_USERNAME).then(user => {
    if (!user) {
      db.run(`INSERT INTO users (username, password, permissions, ip) VALUES ('${MODERATOR_USERNAME}', '${MODERATOR_PASSWORD_HASH}', '["user", "moderator"]', '${MODERATOR_IP}');`)
        }
    });
};
```

Seriously pilvar?

- The admin is not allowed to fetch the flag

```
router.use(async (req, res, next) => {
    if (!req.session.user || !(await User.findByUsername(req.session.user)) || !((await User.findByUsername(req.session.user)).permissions.includes('administrator'))) {
        return res.status(401).send('Unauthorized');
    }
    next();
});

router.get('/flag', async (req, res) => {
    res.render('flag', {flag: FLAG});
})
```

- But he is allowed to promote users

Impossible not to have < insert meme here >

```
router.post('/promote', async (req, res) => {
    const user = req.body.username;
    const permission = req.body.permission;
    if (typeof user !== 'string' || typeof permission !== 'string') {
        return res.status(400).send('{"error":"Bad Request"}');
    }
    if (permission.includes('administrator')) {
        return res.status(500).send('{"error":"Not allowed"}');
    }
    const currentPermissions = JSON.parse((await User.findByUsername(user)).permissions);
    const newPermissions = JSON.stringify([...currentPermissions, permission]);
    User.editPermission(user, newPermissions).then(() => {
        res.status(200).send('{}');
    }).catch(() => {
        res.status(500).send('{"error":"Internal Server Error"}');
    });
});
```

After all this time? ✖️ 💪

- The thing here is subtle

```
if (permission.includes('administrator')) {
        return res.status(500).send('{"error":"Not allowed"}');
}
const currentPermissions = JSON.parse((await User.findByUsername(user)).permissions);
const newPermissions = JSON.stringify([...currentPermissions, permission]);
```

```
!((await User.findByUsername(req.session.user)).permissions.includes('administrator'))
```

```
< `\u001administrator`.includes('administrator')
> false
< JSON.stringify([`\u001administrator`]).includes("administrator")
> true
```

Time for the exploit

not by maitai