Write Up

HACKTHEBOX

UcokGallagher

Institut Teknologi Bandung

Agriweb

Solver:

Diberikan kode web berikut

```
import express from 'express';
import path from 'path';
import { fileURLToPath } from 'url';
import { dirname } from 'path';
import cookieParser from 'cookie-parser';
import { registerUser, loginUser } from './routes/auth.js';
import { updateProfile, updateSettings, getUser } from
'./routes/profile.js';
import { verifyToken, getTokenFromCookie, setTokenCookie,
clearTokenCookie } from './utils/jwt.js';

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);

const app = express();
```

```
// Middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());

// Static files
app.use('/challenge/css', express.static(path.join(__dirname,
'./static/css')));
app.use('/challenge/js', express.static(path.join(__dirname,
'./static/js')));

// Authentication middleware
const authenticate = (req, res, next) => {
    const token = req.cookies.auth_token;

    if (!token) {
        if (req.path.startsWith('/challenge/api/')) {
            return res.status(401).json({ success: false, error: 'Not
authenticated' });
        }
        return res.sendFile(path.join(__dirname,
'./templates/login.html'));
    }

    try {
        req.user = verifyToken(token);
        next();
    } catch (error) {
        res.clearCookie('auth_token');
        if (req.path.startsWith('/challenge/api/')) {
            return res.status(401).json({ success: false, error:
'Invalid token' });
        }
        return res.sendFile(path.join(__dirname,
'./templates/login.html'));
    }
};
```

```
// Admin middleware
const isAdmin = (req, res, next) => {
    try {
        if (!req.user.isAdmin) {
            if (req.path.startsWith('/challenge/api/')) {
                return res.status(403).json({ success: false, error:
'Not authorized' });
            }
            return res.sendFile(path.join(__dirname,
'./templates/unauthorized.html'));
        }
        next();
    } catch (error) {
        return res.sendFile(path.join(__dirname,
'./templates/unauthorized.html'));
    }
};

// Routes
app.get('/challenge', authenticate, (req, res) => {
    res.sendFile(path.join(__dirname, './templates/index.html'));
});

app.get('/challenge/admin', authenticate, isAdmin, (req, res) => {
    res.sendFile(path.join(__dirname, './templates/admin.html'));
});

app.post('/challenge/api/register', async (req, res) => {
    try {
        const { username, email, password } = req.body;
        const { user, token } = await registerUser(username, email,
password);

        setTokenCookie(res, token);

        res.json({
            success: true,
            message: 'Registration successful',
```

```
            user
        });
    } catch (error) {
        res.status(400).json({ success: false, error: error.message
});
    }
});

app.post('/challenge/api/login', async (req, res) => {
    try {
        const { username, password } = req.body;
        const { user, token } = await loginUser(username, password);

        setTokenCookie(res, token);

        res.json({
            success: true,
            message: 'Login successful',
            user
        });
    } catch (error) {
        res.status(400).json({ success: false, error: error.message
});
    }
});

app.post('/challenge/api/logout', (req, res) => {
    clearTokenCookie(res);
    res.json({ success: true, message: 'Logged out successfully' });
});

app.post('/challenge/api/profile', authenticate, async (req, res) =>
{
    try {
        const updatedProfile = await updateProfile(req.user.id,
req.body);
        res.json({
            success: true,
```

```
            message: 'Profile updated successfully',
            profile: updatedProfile
        });
    } catch (error) {
        res.status(500).json({ success: false, error: error.message
});
    }
});

app.get('/challenge/api/user', authenticate, async (req, res) => {
    try {
        const user = await getUser(req.user.id);
        res.json({ success: true, user: user });
    } catch (error) {
        res.status(500).json({ success: false, error: error.message
});
    }
});

app.post('/challenge/api/settings', authenticate, async (req, res) =>
{
    try {
        const updatedSettings = await updateSettings(req.user.id,
req.body);
        res.json({
            success: true,
            message: 'Settings updated successfully',
            settings: updatedSettings
        });
    } catch (error) {
        res.status(500).json({ success: false, error: error.message
});
    }
});

const PORT = process.env.PORT || 8000;
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
```

```
});
```

Dari membaca sebentar, kita melihat ada endpoint yang berbahaya yaitu

```
app.post('/challenge/api/profile', authenticate, async (req, res) => { const updatedProfile = await updateProfile(req.user.id, req.body); // Direct req.body });

app.post('/challenge/api/settings', authenticate, async (req, res) => { const updatedSettings = await updateSettings(req.user.id, req.body); // Direct req.body });
```

Dimana server tidak melakukan sanitasi dari input user, sehingga user bisa menggunakan prototype pollution attack yang dapat mengubah behavior object.

Kemudian di file exploit.py , dapat dilihat payloadnya

```
data1 = {
   "favoriteCrop": "wheat",
   "experienceLevel": "intermediate",
   "farmSize": 501,
   "__proto__": {          # ← PAYLOAD
      "isAdmin": True
   }
}

data2 = {
   # ... normal profile data
   "prototype": {          # ← PAYLOAD
      "constructor": {
         "isAdmin": True
      }
   }
}
```

Kelemahan ini dapat diatasi dengan melakukan patch yaitu memberi sanitasi untuk input dari pengguna.
Berikut patch yang dilakukan

```
import express from 'express';
import path from 'path';
import { fileURLToPath } from 'url';
import { dirname } from 'path';
import cookieParser from 'cookie-parser';
```

```
import { registerUser, loginUser } from './routes/auth.js';
import { updateProfile, updateSettings, getUser } from
'./routes/profile.js';
import { verifyToken, getTokenFromCookie, setTokenCookie,
clearTokenCookie } from './utils/jwt.js';

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);

const app = express();

// Middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());

// Static files
app.use('/challenge/css', express.static(path.join(__dirname,
'./static/css')));
app.use('/challenge/js', express.static(path.join(__dirname,
'./static/js')));

// Authentication middleware
const authenticate = (req, res, next) => {
    const token = req.cookies.auth_token;

    if (!token) {
        if (req.path.startsWith('/challenge/api/')) {
            return res.status(401).json({ success: false, error: 'Not
authenticated' });
        }
        return res.sendFile(path.join(__dirname,
'./templates/login.html'));
    }

    try {
        req.user = verifyToken(token);
        next();
    } catch (error) {
        res.clearCookie('auth_token');
        if (req.path.startsWith('/challenge/api/')) {
            return res.status(401).json({ success: false, error:
```

```
'Invalid token' });
        }
        return res.sendFile(path.join(__dirname,
'./templates/login.html'));
    }
};

// Admin middleware - FIXED: Prevent prototype pollution with
explicit checks
const isAdmin = (req, res, next) => {
    try {
        // Multiple layers of protection against prototype pollution
        const userIsAdmin =
Object.prototype.hasOwnProperty.call(req.user, 'isAdmin') &&
req.user.isAdmin === true;

        if (!userIsAdmin) {
            if (req.path.startsWith('/challenge/api/')) {
                return res.status(403).json({ success: false, error:
'Not authorized' });
            }
            return res.sendFile(path.join(__dirname,
'./templates/unauthorized.html'));
        }
        next();
    } catch (error) {
        return res.sendFile(path.join(__dirname,
'./templates/unauthorized.html'));
    }
};

// Routes
app.get('/challenge', authenticate, (req, res) => {
    res.sendFile(path.join(__dirname, './templates/index.html'));
});

app.get('/challenge/admin', authenticate, isAdmin, (req, res) => {
    res.sendFile(path.join(__dirname, './templates/admin.html'));
});

app.post('/challenge/api/register', async (req, res) => {
    try {
```

```
        const { username, email, password } = req.body;
        const { user, token } = await registerUser(username, email,
password);

        setTokenCookie(res, token);

        res.json({
            success: true,
            message: 'Registration successful',
            user
        });
    } catch (error) {
        res.status(400).json({ success: false, error: error.message
});
    }
});

app.post('/challenge/api/login', async (req, res) => {
    try {
        const { username, password } = req.body;
        const { user, token } = await loginUser(username, password);

        setTokenCookie(res, token);

        res.json({
            success: true,
            message: 'Login successful',
            user
        });
    } catch (error) {
        res.status(400).json({ success: false, error: error.message
});
    }
});

app.post('/challenge/api/logout', (req, res) => {
    clearTokenCookie(res);
    res.json({ success: true, message: 'Logged out successfully' });
});

app.post('/challenge/api/profile', authenticate, async (req, res) =>
{
```

```
    try {
        //  sanitization
        const sanitizeObject = (obj) => {
            if (obj === null || typeof obj !== 'object') {
                return obj;
            }

            const sanitized = {};
            for (const [key, value] of Object.entries(obj)) {
                if (['__proto__', 'constructor', 'prototype',
'__defineGetter__', '__defineSetter__', '__lookupGetter__',
'__lookupSetter__'].includes(key)) {
                    continue;
                }

                if (typeof value === 'object' && value !== null) {
                    sanitized[key] = sanitizeObject(value);
                } else {
                    sanitized[key] = value;
                }
            }
            return sanitized;
        };

        const sanitizedData = sanitizeObject(req.body);
        const updatedProfile = await updateProfile(req.user.id,
sanitizedData);
        res.json({
            success: true,
            message: 'Profile updated successfully',
            profile: updatedProfile
        });
    } catch (error) {
        res.status(500).json({ success: false, error: error.message
});
    }
});

app.get('/challenge/api/user', authenticate, async (req, res) => {
    try {
        const user = await getUser(req.user.id);
        res.json({ success: true, user: user });
```

```
    } catch (error) {
        res.status(500).json({ success: false, error: error.message
});
    }
});

app.post('/challenge/api/settings', authenticate, async (req, res) =>
{
    try {
        const sanitizeObject = (obj) => {
            if (obj === null || typeof obj !== 'object') {
                return obj;
            }

            const sanitized = {};
            for (const [key, value] of Object.entries(obj)) {
                if (['__proto__', 'constructor', 'prototype',
'__defineGetter__', '__defineSetter__', '__lookupGetter__',
'__lookupSetter__'].includes(key)) {
                    continue;
                }

                if (typeof value === 'object' && value !== null) {
                    sanitized[key] = sanitizeObject(value);
                } else {
                    sanitized[key] = value;
                }
            }
            return sanitized;
        };

        const sanitizedData = sanitizeObject(req.body);
        const updatedSettings = await updateSettings(req.user.id,
sanitizedData);
        res.json({
            success: true,
            message: 'Settings updated successfully',
            settings: updatedSettings
        });
    } catch (error) {
        res.status(500).json({ success: false, error: error.message
});
```

```
    }
});

const PORT = process.env.PORT || 8000;
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

Pada dua endpoint tersebut kita melakukan sanitasi serta menambah blockir untuk keyword keyword tertentu yang berbahaya, setelah di cek, dapat flag

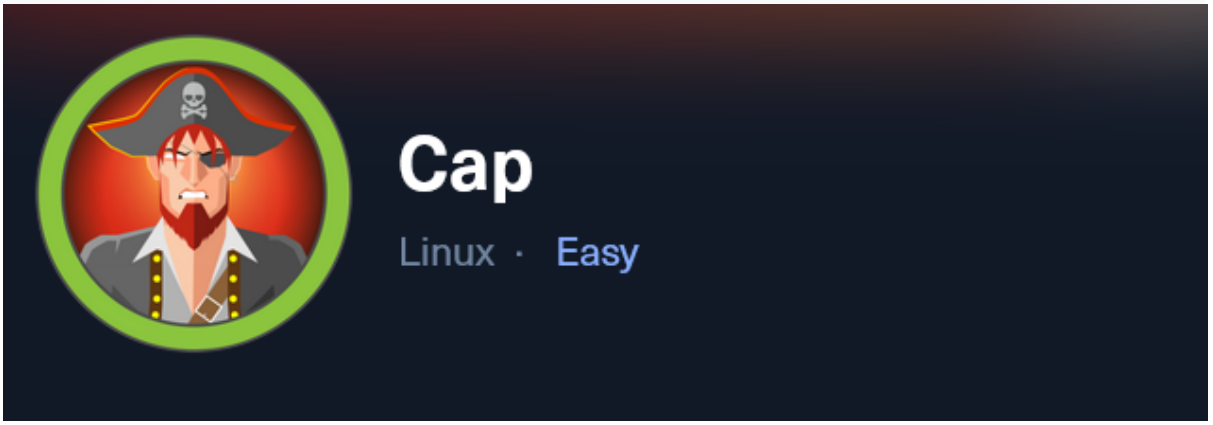
FLAG:HTB{prototyp3_pollut1on_to_4uth_byp4s5}

Remediation:
Dari challenge ini saya menyadari, sepertinya bahaya terbesar tidak hanya dari eksternal, namun juga dri internal, seperti ini, dimana terlalu mempercayai inputan pengguna sehingga dapat diexploit. Sehingga penting untuk clean code serta pentesting di staging. Selain itu ini pertama saya mengerjakan chall seperti!

Jika kasus ini benar2 terjadi di dunia nyata, dampaknya bisa besar, seperti bocornya data2 internal dan infrastruktur internal dapat ketahuan, yang akan mengarah ke kerugian besar.

Contoh RL : Lodash CVE-2019-10744, Dampaknya adalah didapat RCE melalui prototype pollution. Mempengaruhi Facebook, Netflix dll. dengan estima kerugian puluhan juta dollar.

Cap



*Klarifikasi, Saat saya menulis WU ini ternyata playboxnya habis, ga bisa make machine htb, jdi ngga ada gambar gitu

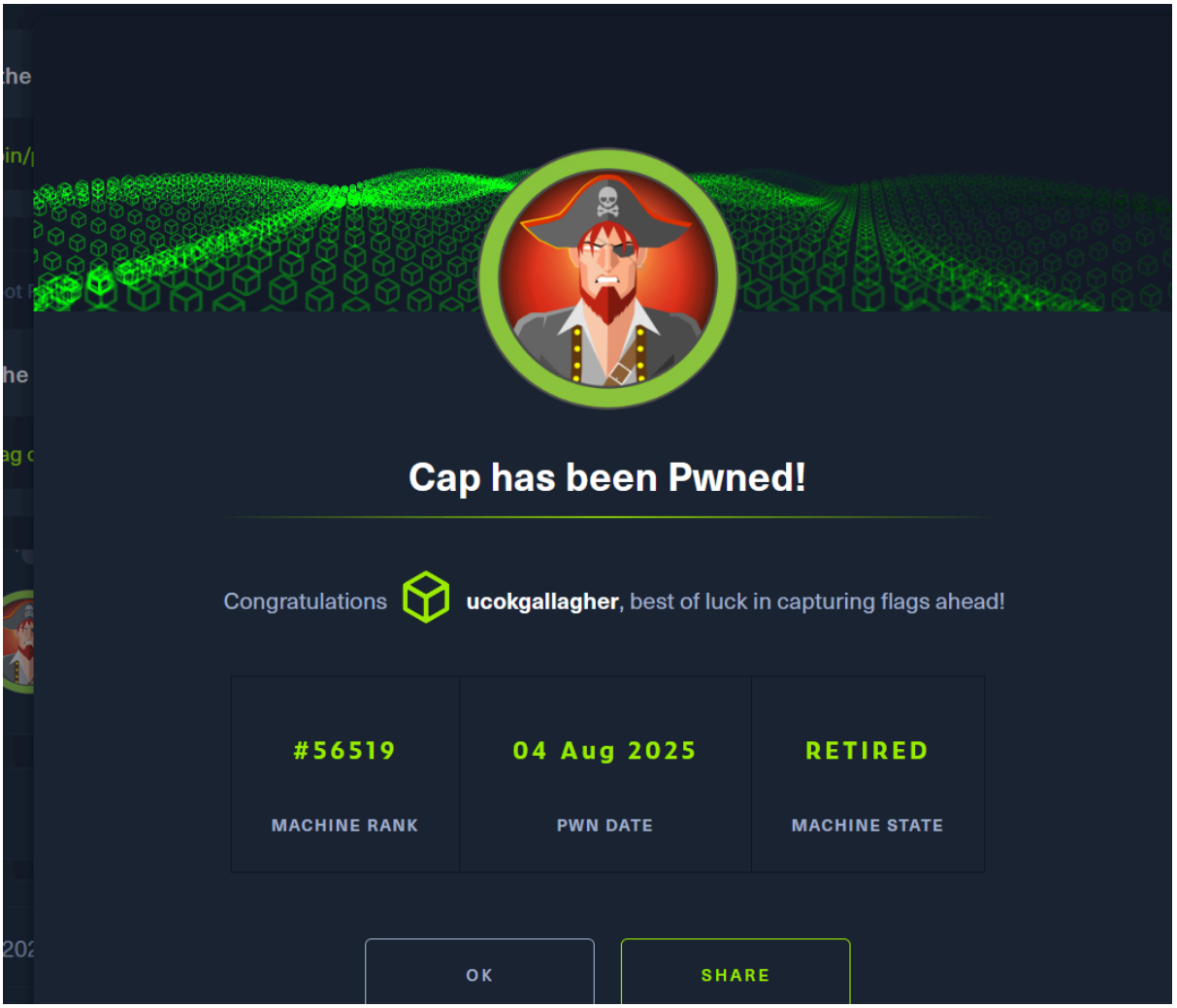
Karena ditanya berapa port dibuka, jadi langsung saja enumerate pake nmap

```
nmap -p$ports -Pn -sC -sV 10.10.10.245
```

Didapat ada 3 yaitu, SSH, FTP, dan HTTP

Kemudian saya mencoba mengakses HTTP yaitu port 80, dan didapat landing page semacam security monitor dashboard (?) idk. Kemudian ke fase terlama, yaitu ngotak ngatik, setelah beberapa lama ketemu page Security Snapshot yang isinya seperti live monitor. Dicoba download didapat file pcapng, dimana didapat URL `10.10.10.245/data/1` setelah konsultasi dengan Claude disarankan IDOR, jadi karena claude sarankan untuk coba2 url, kita cobalah `10.10.10.245/data/0` and boom! opened new page.

Di page baru sama, kita download pcapng, dan ketika di cek, ada protocol FTP dengan message “Login Successful” ketika di follow, didapat creds plaintext, sehingga dicoba untuk creds FTP, dan berhasil!, didapat user.txt. kemudian langsung konek SSH, dan ada juga ternyata user.txt :v. keliling2, kok ngga ada yang mencurigakan. kembali konsul ke claude. dan didapat bahwa ada script linpeas.sh dimana itu adalah tool untuk mencari celah yang bisa di exploit. langsung kita run saja. dan kemudian hasil run kita cek, didapat /usr/bin/python3.8 memiliki konfigurasi yang bisa ganti uid tanpa root privilege, langsung saja masuk ke interactive python3 dan kita set uid ke root (0) dengan os.setuid(0), kemudian kita langsung saja os.system(“sh”) dan kita memiliki shell session sebagai root dan didapat rootflag


Cap has been Pwned!
Congratulations ucokgallagher, best of luck in capturing flags ahead!
#56519
MACHINE RANK
04 Aug 2025
PWN DATE
RETIRED
MACHINE STATE
OK
SHARE