HACK THE BOX

Penetration Test Report

Machine: Browsed

Candidate Name: Samurai
Target IP: 10.129.244.79
Date: February 16, 2026
Version: 1.0

Contents

1 Executive Summary

1.1 Approach

This report documents the findings of a black-box penetration test conducted against the target system "Browsed" (10.129.244.79). The assessment followed standard penetration testing methodologies, starting with reconnaissance and moving through enumeration, exploitation, and post-exploitation.

1.2 Scope

The scope of this assessment was limited to the single host at IP address **10.129.244.79**. No other systems or networks were targeted.

1.3 Assessment Overview

The assessment revealed critical vulnerabilities that allowed for a complete system compromise.

- **Initial Access:** Achieved via a Server-Side Request Forgery (SSRF) vulnerability in the "Chrome Extension Upload" functionality. This was chained with a Command Injection vulnerability in an internal Flask application running on localhost.
- **Privilege Escalation:** Achieved by exploiting a misconfigured Python environment. A sudo-enabled script allowed for Python Bytecode Hijacking via a world-writable `__pycache__` directory.

1.4 Recommendations

- **Critical:** Sanitize input in the internal `routines.sh` script to prevent command injection.
- **Critical:** Restrict permissions on the `/opt/extensiontool/__pycache__` directory to prevent unauthorized modification of compiled Python files.
- **High:** Implement strict validation on uploaded files to prevent SSRF via malicious extension manifests.

2 Internal Network Compromise Walkthrough

2.1 Reconnaissance & Enumeration

An Nmap scan identified two open ports: SSH (22) and HTTP (80). Accessing the web server revealed a "Browsed" landing page with functionality to upload Chrome extensions. Virtual host enumeration identified a second domain: `browsedinternals.htb`, which hosted a Gitea instance.

2.2 Initial Access (The Exploit Chain)

The application on `browsed.htb` allowed users to upload a ZIP file containing a Chrome extension. Analysis revealed the server "emulates" the extension, executing its background scripts.

1. **SSRF via Chrome Extension:** A malicious extension was crafted with a `manifest.json` requesting `<all_urls>` permissions. The `background.js` script was configured to fetch data from the internal interface (`127.0.0.1`).
2. **Internal Discovery:** Using the SSRF, we probed the internal network and discovered a Flask application running on port 5000. Source code obtained from the Gitea instance revealed a route `/routines/<rid>` that passed input to a bash script `routines.sh`.
3. **Command Injection (RCE):** The `routines.sh` script used the bash arithmetic operator `[[ "$1" -eq 0 ]]`, which is vulnerable to injection if the variable contains array syntax. A payload was constructed using `a[$(encoded_command)]` to bypass the Flask router's slash restrictions.

Exploit Payload

```
http://browsed.htb/routines/a[$(echo BASE64_PAYLOAD | base64 -d | bash)]
```

This resulted in a reverse shell as the user **larry**.

2.3 Privilege Escalation

Enumeration of the user `larry` revealed sudo privileges for `/opt/extensiontool/extension_tool.py`. Analysis of the directory showed that while the scripts were owned by root, the `__pycache__` directory was world-writable (permissions 777).

Exploit Method:

1. Created a malicious Python script spawning a root shell.
2. Compiled it to bytecode (`.pyc`).
3. Forged the `.pyc` header to match the timestamp and file size of the legitimate `extension_utils.py`.
4. Replaced the legitimate cached file in `__pycache__`.
5. Executed the tool via `sudo`, triggering the execution of the cached malicious bytecode.

This resulted in full **root** access.

3 Technical Findings Details

Command Injection in Internal App **Severity: Critical (9.8)**

RCE via Bash Arithmetic Injection

The internal script `routines.sh` improperly handles user input within a bash arithmetic context. This allows an attacker to execute arbitrary system commands by injecting array syntax.

- **Impact:** Full control over the web server user.
- **Remediation:** Sanitize input before passing it to `routines.sh` or use safer comparison methods in bash.

Python Bytecode Hijacking **Severity: High (8.8)**

Privilege Escalation via Weak Permissions

The directory `/opt/extensiontool/__pycache__` is world-writable. This allows any user to replace compiled Python files. Since the parent script runs with `sudo`, this leads to root compromise.

- **Impact:** Complete system takeover (Root access).
- **Remediation:** Change ownership of the `__pycache__` directory to root and remove write permissions for others.

Server-Side Request Forgery (SSRF) **Severity: High (7.5)**

Access to Internal Services

The Chrome Extension upload feature allows the execution of arbitrary JavaScript in the context of the server's browser, enabling access to internal network resources (localhost).

- **Impact:** Exposure of internal services (Flask, Gitea) and data exfiltration.
- **Remediation:** Run the browser in a strictly isolated container with no network access to internal interfaces.

4 Appendix

4.1 A.1 Host & Service Discovery

Port	Service	Notes
22	SSH	OpenSSH 9.6p1 Ubuntu
80	HTTP	Nginx 1.24.0 (Hosts: browsed.htb, browsedinternals.htb)

4.2 A.2 Compromised Users

Username	Type	Method
larry	User	RCE via Flask/Bash Injection
root	Admin	Python Bytecode Hijacking