Module Malware

**Netcat**

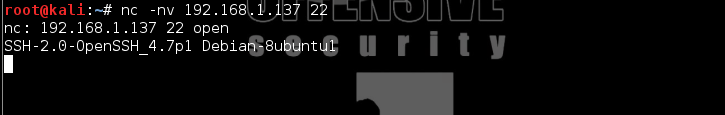
Netcat is a versatile tool that has been dubbed the Hackersk Swiss Army Knife and exists as both Linux and Windows binaries. The simplest definition of Netcat is “a tool that can read and write to TCP and UDP ports.” This dual functionality suggests that Netcat runs in two modes: client and server.

***Connecting to a TCP/UDP Port***

Connecting to a TCP/UDP port can be useful in several situations:

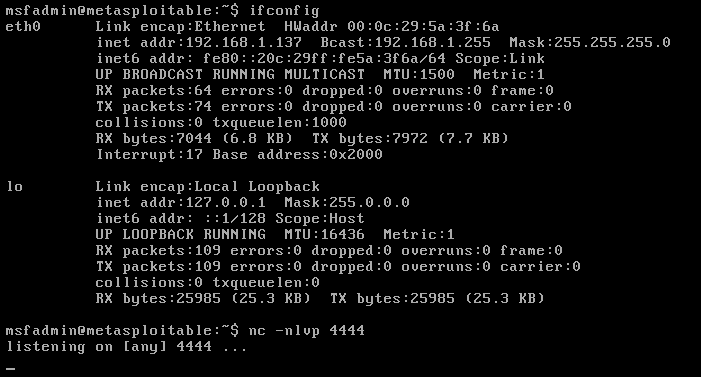
* To check if a port is open or closed.
* To read a banner from the port.
* To connect to a network service manually.

Let’s begin by using netcat to check if TCP port 22 (the SSH mail service) is open on Metasploitable 2 machine.



***Listening on a TCP/UDP Port***

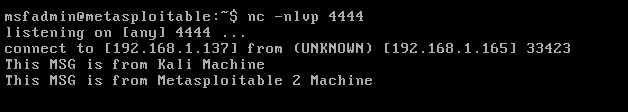
Listening on a TCP/UDP port using netcat is useful for network debugging client applications, or otherwise receiving a TCP/UDP network connection. Letks try implementing a simple chat involving two machines, using netcat both as a client and as a server. We’ll set up netcat to listen for incoming&connections on TCP port 4444, on a Metasploitable 2 machine (with IP address 192.168.1.137).



Once we have bound port 4444 on the Windows machine to Netcat, we can connect to that port from the Kali machine to interact with it.

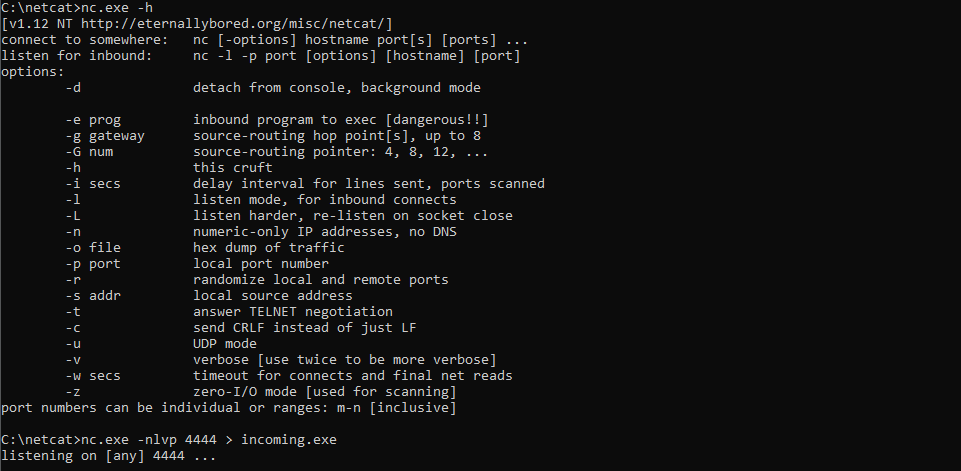


Our text is sent to the Metasploitable 2 machine over TCP port 4444 and we can continue the “chat” from the Metasploitable 2 machine as shown below.



***Transferring Files with Netcat***

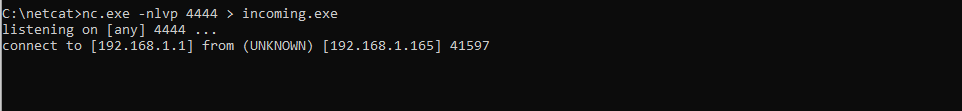
Netcat can also be used to transfer files, both text and binary, from one computer to another. To send a file from the Kali machine to the Windows 10 machine, we initiate a setup that is similar to the previous chat example, with some slight differences. On the Windows machine, we will set up a netcat listener on port 4444 and redirect any incoming input into a file called incoming.exe.



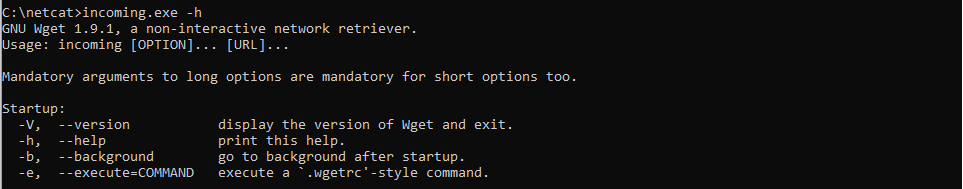
On the Kali system, we will push the wget.exe file to the Windows machine through TCP port 4444.



The connection is received by netcat on the Windows machine as shown below:



Notice that we haven’t received any feedback from netcat about our file upload progress. In this case, since the file we are uploading is small, we can just wait for a few seconds and then check wheter it has been fully uploaded to the Windows machine, by running the executable:



***Remote Administration with Netcat***

One of the most useful features of netcat is its ability to do command redirection. Netcat can take an executable file and redirect the input, output, and error messages to a TCP/UDP port rather than the default console. To further explain this, consider the cmd.exe executable. By redirecting the stdin, stdout, and stderr to the network, you can bind cmd.exe to a local port. Anyone connecting to this port will be presented with a command prompt belonging to this computer. To further drive this home, consider the following scenarios, involving Bob and Alice.

*Netcat Bind Shell Scenario*

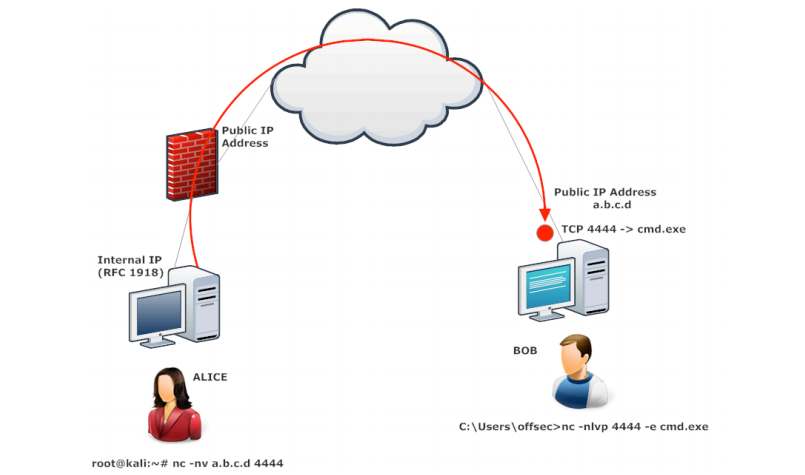
In our first scenario, Bob (running Windows) has requested Alice’s assistance (running Kali) and has asked her to connect to his computer and issue some commands remotely. Bob has a public IP address, and is directly connected to the Internet. Alice, however, is behind a NAT connection, and has an internal IP address. To complete the scenario, Bob needs to bind cmd.exe to a TCP port on his public IP address, and ask Alice to connect to this particular IP and port. Bob will proceed to issue the following command with netcat.



Netcat has bound TCP port 4444 to cmd.exe and will redirect any input, output, or error messages from cmd.exe to the network. In other words, anyone connecting to TCP port 4444 on Bob’s machine, hopefully Alice, will be presented with Bob’s command prompt.



The following image depicts the bind shell scenario where Alice gets remote command prompt access on Bob’s Windows machine:



*Reverse Shell Scenario*

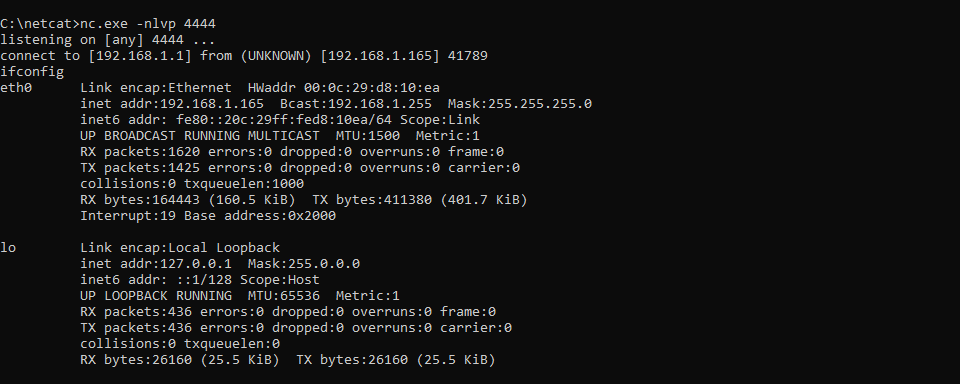
In our second scenario, Alice needs help from Bob. However, Alice has no control over the router in her office, and therefore cannot forward traffic from the router to her internal machine. Is there any way for Bob to connect to Aliceks computer, and solve her problem? Here we discover another useful feature of Netcat, the ability to send a command shell to a listening host. In this situation, although Alice cannot bind a port to /bin/bash locally on her computer and expect Bob to connect, she can send control of her command prompt to Bobks machine, instead. This is known as a reverse shell. To get this working, Bob needs to set up netcat to listen for an incoming shell. We’ll use port 4444 in our example:



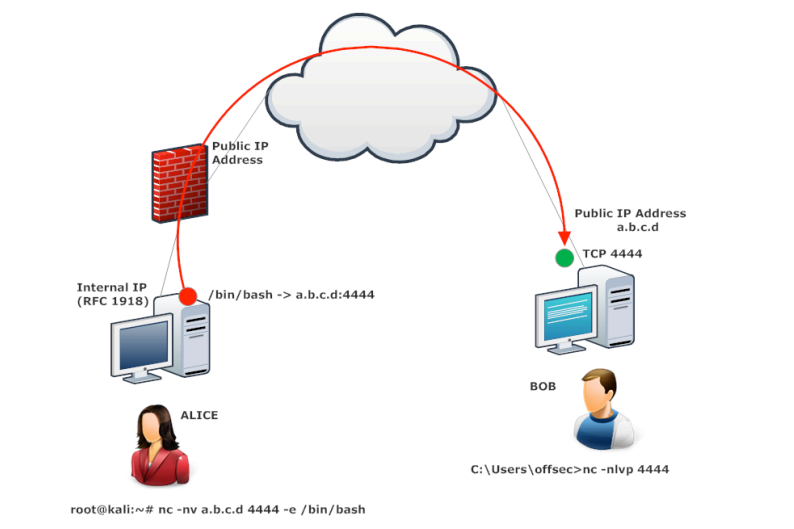
Now, Alice can send a reverse shell from her Linux machine to Bob:



Once the connection is established, Alice’s netcat will have redirected input, output, and error from /bin/bash, to Bob’s machine, on port 4444.



The! following!image!depicts! the reverse! shell! scenario!where!Bob! gets!remote!shell!access!on!Alice’s!Linux!machine,!traversing!the!corporate!firewall.



**Metasploit Payload**

***Staged vs Non-Staged Payloads***

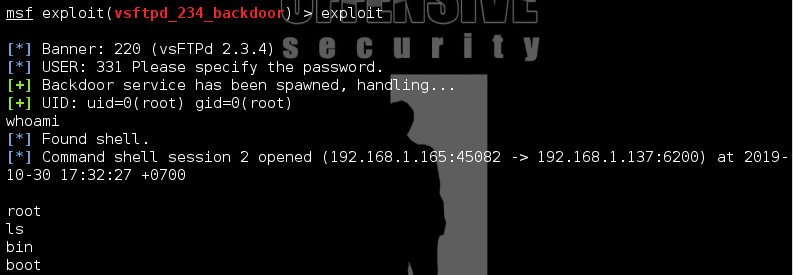
One of the first distinctions that are important to make in Metasploit payloads are between staged and non-staged shellcode. A good example of this is evident when comparing the following two payloads:

**windows/shell\_reverse\_tcp - Connect back to attacker and spawn a command shell windows/shell/reverse\_tcp - Connect back to attacker, Spawn cmd shell (staged)**

A non-staged payload is a payload that is sent in its entirety in one go – as we’ve been doing up to now. A staged payload is usually sent in two parts. The first part is a small primary payload, which causes the victim machine to connect back to the victim, accept a longer secondary payload containing the rest of the shellcode, and then execute it. There are several situations where we would prefer to use staged shellcode over non-staged:

* The vulnerability we are exploiting does not have enough buffer space to hold a full payload. As the first part of a staged payload is typically smaller than a full payload, these smaller payloads can often save us in tight situations.
* Antivirus software is detecting embedded shellcode in an exploit. By replacing the embedded shellcode with a staged payload, we will be removing most of the malicious part of the shellcode and injecting it directly into the victim machine memory.

As described on the Metasploit site, the Meterpreter is a staged, multi-function payload that can be dynamically extended at run-time. In practice, this means that the Meterpreter shell provides more features and functionality than a regular command shell by having inbuilt functionality, such as file upload and downloads, keyloggers, and many more builtLin routines to interact with the victim machine. These are useful in a post-exploitation phase. This additional functionality makes Meterpreter the favorite and most commonly used payload in the MSF. Let’s explore the Meterpreter payload by using it on the vsftpd server on Metasploitable 2 lab machine.



There are several interesting things to notice in this output:

* We didn’t select a payload. If the user does not specify a payload for an exploit, a reverse Meterpreter payload is used by default.
* Meterpreter is a staged payload. The second stage is a 750k DLL file that is injected directly into memory. As the DLL file never touches the victim file system, it is less likely to be detected by antivirus software

The best way to get to know the features of Meterpreter is to test them out. Let’s start with a few simple commands such as sysinfo, getuid, and search.

Next, let’s try some easy uploads and downloads using built in commands in Meterpreter. Take note that, due to shell escaping, it is necessary to use two \ characters for the destination path as shown below.

**Meterpreter > upload /usr/share/windows-binaries/nc.exe c:\\Users\\Offsec**

From the Meterpreter session, let’s invoke a command shell.



<https://samsclass.info/123/proj10/123p15fire.htm>