Ethical Hacking Lab

Scapy For Packet Crafting

Networking

Packet Crafting

# **Packet Crafting using Scapy**

**Background:**

Packet Crafting Plays an important role in bypassing IDS, IPS and firewalls. As they are programmed to allow only certain type of traffic to pass through in this lab, we aim to learn the basics of creating our own packet using a tool called scapy!

**About Scapy**

Scapy is a Python program that enables the user to send, sniff, dissect and forge network packets. This capability allows construction of tools that can probe, scan or attack networks.

In other words, Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. It can replace hping, arpspoof, arp-sk, arping, p0f and even some parts of Nmap, tcpdump, and tshark.

**Lab - Packet Crafting with Scapy**

**Objectives**

In this lab, you will use Scapy, a Python-based packet manipulation tool, to craft custom packets. These custom packets will be used to perform reconnaissance on a target system.

* Part 1: Investigate the Scapy Tool.
* Part 2: Use Scapy to Sniff Network Traffic.
* Part 3: Create and Send an ICMP Packet.
* Part 4: Create and Send TCP SYN Packets.

**Background / Scenario**

Penetration testers and ethical hackers often use specially-crafted packets to discover and/or exploit vulnerabilities in clients’ infrastructure and systems. You have used Nmap to scan and analyze vulnerabilities in a computer attached to the local network.

In this lab, you will perform further reconnaissance on the same computer using custom ICMP and TCP packets.

**Required Resources**

* Kali VM
* Internet Access

**Part 1: Investigate the Scapy Tool**

Scapy is a multi-purpose tool originally written by Philippe Biondi. In this part, you will load the Scapy tool and explore some of its capabilities. Tools like Scapy should only be used to scan or communicate with machines that you own or have written permission to access.

**Step 1: Investigate Scapy documentation and resources.**

Scapy can be run interactively from the Python shell or can be incorporated into Python programs by importing the python-scapy module. The Scapy tool has extensive documentation online at <https://scapy.readthedocs.io/en/latest/introduction.html>. This customized Kali Linux is distributed with both Python and Scapy pre-installed.

1. Start the Kali VM and login.
2. Open the browser and navigate to <https://scapy.readthedocs.io/en/latest/introduction.html>. Read the Introduction page written by the Scapy creator, Philippe Biondi.

**Step 2: Use Scapy interactive command mode.**

Enter the **scapy** command in a terminal window to load the Python interpreter. By using this command, the interpreter runs pre-loaded with the Scapy classes and objects. You will enter Scapy commands interactively and receive the output. Scapy commands can also be embedded in a Python script.

1. The commands to craft and send packets require root privileges to run. Use the **sudo su** command to obtain root privileges before starting Scapy.

┌──(kali㉿kali)-[~]

└─$ **sudo su**

[sudo] password for kali:

┌──(root㉿kali)-[/home/kali]

└─#

1. Load the Scapy tool using the **scapy** command. The interactive Python interpreter will load and present a screen image similar to that shown.

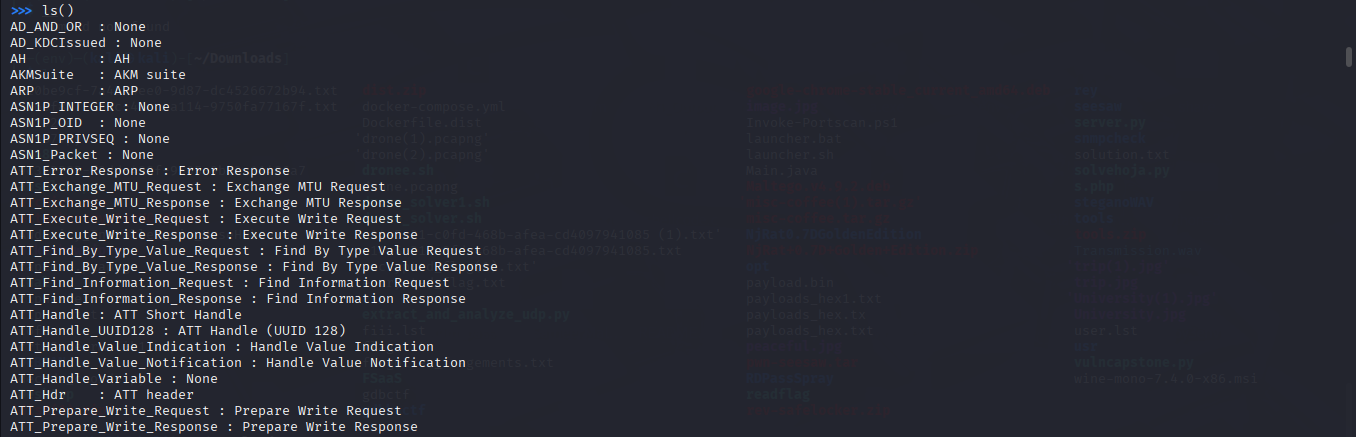
┌──(root㉿kali)-[/home/kali]

└─# **scapy**

1. At the >>> prompt within the Scapy shell, enter the **ls()** function to list all of the available default formats and protocols included with the tool. The list is quite extensive and will fill multiple screens.

>>> **ls()**

TFTP is a protocol used to send and receive files on a LAN segment. It is commonly used to back up configuration files on networking devices. Scroll up to view the available TFTP packet formats.



To list the modifiable fields inside a specific type of packet use this syntax >>> ls(<type>).

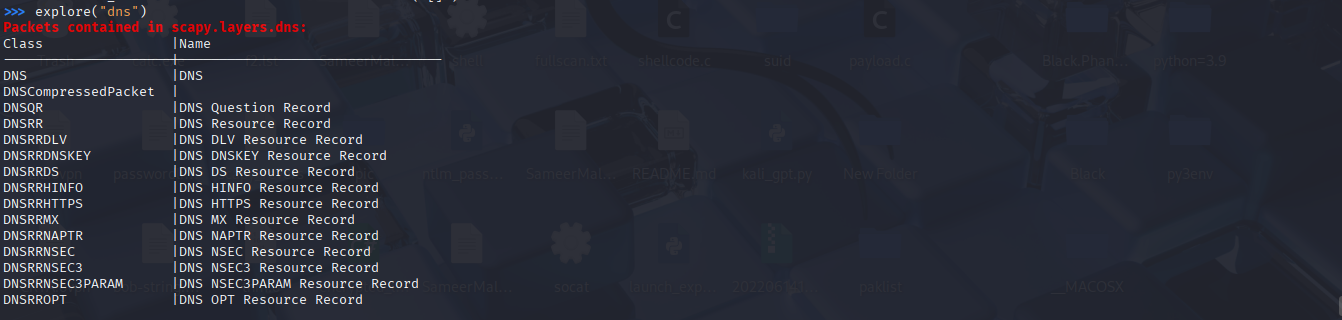
For example to see the TCP fields that can be modified by using scapy we use the following command:



 The DNS packet fields that can be modified by Scapy:



Moreover we can use the Explore function to see the more DNS packets in scapy using the command below:



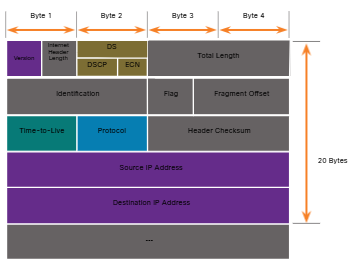
**Step 3: Examine the fields in an IPv4 packet header.**

1. It is important to understand the structure of an IP packet before creating and sending custom packets over the network. Each IP packet has an associated header that provides information about the structure of the packet. Review this information before continuing with the lab.

**IPv4 Packet Header Fields**

The binary values of each field identify various settings of the IP packet. Protocol header diagrams, which are read left to right, and top down, provide a visual to refer to when discussing protocol fields. The IP protocol header diagram in the figure identifies the fields of an IPv4 packet.

**Fields in the IPv4 Packet Header**



*Significant fields in the IPv4 header include the following:*

* ***Version****- Contains a 4-bit binary value set to 0100 that identifies this as an IPv4 packet.*
* ***Differentiated Services or DiffServ (DS)****- Formerly called the type of service (ToS) field, the DS field is an 8-bit field used to determine the priority of each packet. The six most significant bits of the DiffServ field are the differentiated services code point (DSCP) bits and the last two bits are the explicit congestion notification (ECN) bits.*
* ***Time to Live (TTL)****- TTL contains an 8-bit binary value that is used to limit the lifetime of a packet. The source device of the IPv4 packet sets the initial TTL value. It is decreased by one each time the packet is processed by a router. If the TTL field decrements to zero, the router discards the packet and sends an Internet Control Message Protocol (ICMP) Time Exceeded message to the source IP address. Because the router decrements the TTL of each packet, the router must also recalculate the Header Checksum.*
* ***Protocol****- This field is used to identify the next level protocol. This 8-bit binary value indicates the data payload type that the packet is carrying, which enables the network layer to pass the data to the appropriate upper-layer protocol. Common values include ICMP (1), TCP (6), and UDP (17).*
* ***Header Checksum****- This is used to detect corruption in the IPv4 header.*
* ***Source IPv4 Address****- This contains a 32-bit binary value that represents the source IPv4 address of the packet. The source IPv4 address is always a unicast address.*
* ***Destination IPv4 Address****- This contains a 32-bit binary value that represents the destination IPv4 address of the packet. The destination IPv4 address is a unicast, multicast, or broadcast address.*

**Part 2: Use Scapy to Sniff Network Traffic**

Scapy can be used to capture and display network traffic, similar to a tcpdump or tshark packet collection.

**Step 1: Use the sniff() function.**

1. Use the **sniff()** function to collect traffic using the default eth0 interface of your VM. Start the capture with the **sniff()** function without specifying any arguments.

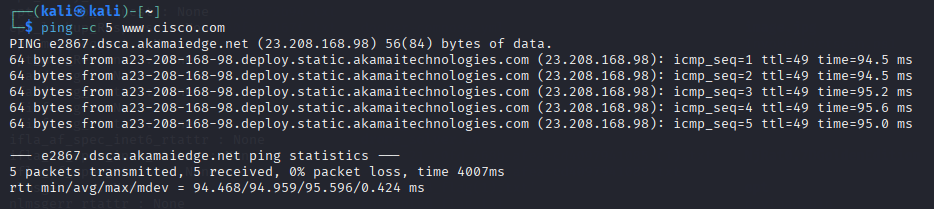
>>> **sniff()**



1. Open a second terminal window and **ping** an internet address, such as **www.cisco.com**. Remember to specify the count using the **-c** argument.

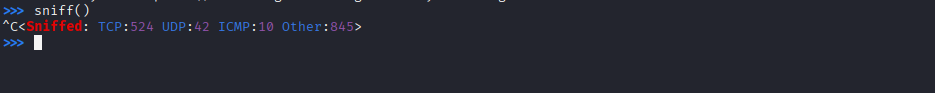
┌──(kali㉿kali)-[~]

└─$ **ping -c 5** [**www.cisco.com**](http://www.cisco.com)

****

1. Return to the terminal window that is running the Scapy tool. Press **CTRL-C** to stop the capture. You should receive output similar to what is shown here:

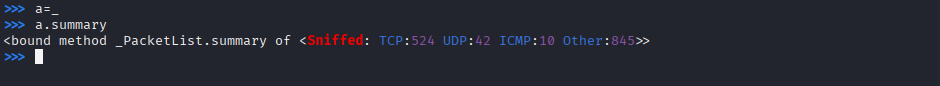
^C<Sniffed: TCP:525 UDP:42 ICMP:10 Other:845>



1. View the captured traffic using the **summary()** function. The **a=\_** assigns the variable **a**to hold the output of the **sniff()** function. The underscore ( \_ ) in Python is used to temporarily hold the output of the last function executed.

>>> **a=\_**

>>> **a.summary()**

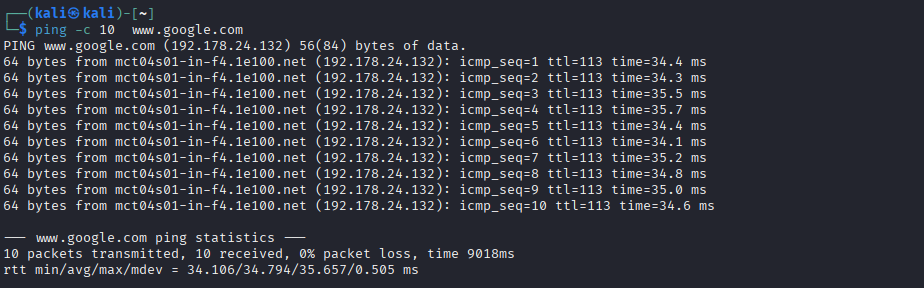


The output of this command can be extensive, depending on the applications running on the network.

1. syntax is **sniff(iface="***interface name***", filter = “***protocol***", count =** *integer***)**.

>>> **sniff(iface="br-internal",filter = “icmp",count = 10)**

1. Open a second terminal window and ping



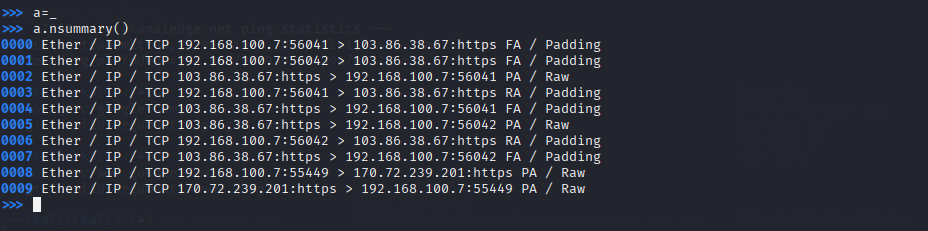
1. Return to the terminal window running the Scapy tool. The capture automatically stopped when 10 packets were sent or received. View the captured traffic with line numbers using the **nsummary()** function.



>>> **a=\_**

>>> **a.nsummary()**

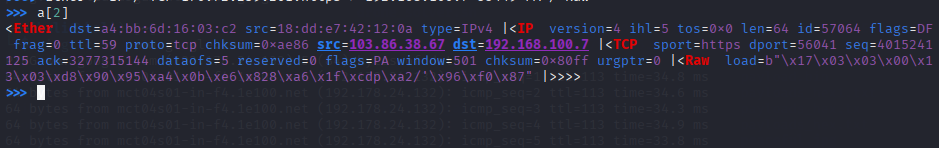
The summary should only contain 10 lines because the capture count is equal to 10.



1. To view details about a specific packet in the series, refer to the blue line number of the packet. Do not include the leading zeros.

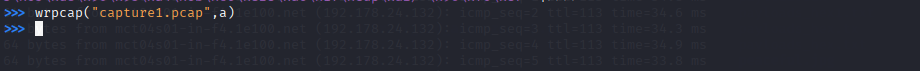
>>> **a[2]**

The detail output shows the layers of information about the protocol data units (PDUs) that make up the packet. The protocol layer names appear in red in the output. Examine the source (src) and destination (dst) addresses as well as the raw data (load=) portion of the collected packet.



1. Use the **wrpcap()** function to save the captured data to a pcap file that can be opened by Wireshark and other applications. The syntax is **wrpcap(“***filename*.*pcap*", *variable name***),**in this example the variable that you stored the output is “**a**".

>>> **wrpcap(“capture1.pcap", a)**



1. The .pcap file will be written to the default user directory. Use a different terminal window to verify the location of the **capture1.pcap** file using the Linux **ls** command.



**Part 3: Create and Send an ICMP Packet.**

ICMP is a protocol designed to send control messages between network devices for various purposes. There are many types of ICMP packets, with echo-request and echo-reply the most familiar to IT technicians. To see a list of the message types that can be sent and received using ICMP, navigate to<https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml>**.**

**Step 1: Use interactive mode to create and send a custom ICMP packet.**

1. In a Scapy terminal window, enter the command to sniff traffic from the interface connected to the network.

>>> **sniff(iface=”eth0”)**



1. Open another terminal window, enter **sudo su** to perform packet crafting as root. Start a second instance of Scapy. Enter the **send**() function to send a packet with a modified ICMP payload.

┌──(kali㉿kali)-[~]

└─$ **sudo su**

[sudo] password for kali:

┌──(root㉿kali)-[/home/kali]

└─# **scapy**

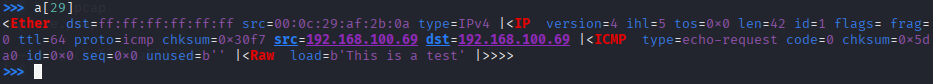
>>> **send(IP(dst="IP")/ICMP()/"This is a test")**



1. Return to the first terminal window and press **CTRL-C**. You should receive a response similar to this:

^C<Sniffed: TCP:13 UDP:2 ICMP:1 Other:0>

And this is our captured packet.

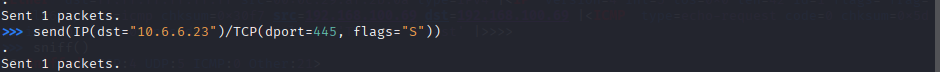


**Part 4: Create and Send a TCP SYN Packet.**

**Step 1: Start the packet capture on the internal interface.**

1. In the original Scapy terminal window, begin a packet capture on the internal interface attached to the network. Use the interface name that you obtained previously.
2. Navigate to the second terminal window. Create and send a TCP SYN packet using the command shown.

>>> **send(IP(dst="10.6.6.23")/TCP(dport=445, flags="S"))**

****



1 packet sent

This command sent an IP packet to the host with IP address 10.6.6.23. The packet is addressed to TCP port 445 and has the S (SYN) flag set.

1. Close the terminal window.
2. In the original Scapy terminal window, begin a packet capture on the internal interface attached to the network. Use the interface name that you obtained previously.
3. Navigate to the second terminal window. Create and send a TCP SYN packet using the command shown.

>>> **send(IP(dst="10.6.6.23")/TCP(dport=445, flags="S"))**

1 packet sent

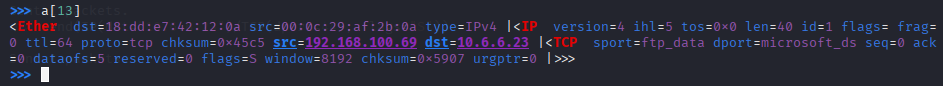


This command sent an IP packet to the host with IP address 10.6.6.23. The packet is addressed to TCP port 445 and has the S (SYN) flag set.

1. Close the terminal window.



To check our particular packet we used the following command:

****

## **Conclusion:**

This Lab provided a generic Introduction to Packet Crafting using Scapy, Scapy is a very potent tool if deployed correctly to maximize on its benefits it is essential to have a solid foundation that is exactly what this lab aimed to provide. So I Hope you guys learnt something from this.