SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur – 603 203

DEPARTMENT OF CYBER SECURITY

LAB MANUAL

V SEMESTER

CY3566 – ETHICAL HACKING PRACTICES LABORATORY

Regulation – 2023

Academic Year 2025-2026 (ODD SEMESTER)

Prepared by

Dr.V.Sandhana Marichamy / Assoc. Prof

EXP NO: 1 **NETWORK RECONNAISSANCE TOOLS – WHOIS,**
DATE: **TRACEROUTE, DIG AND NSLOOKUP**

AIM : To study how to gather information about the networks by using different n/w reconnaissance tools.

Tools required : Kalilinux, WHOIS, traceroute, nslookup

Theory :

traceroute:

The traceroute, tracert, or tracepath command is similar to ping, but provides information about the path a packet takes. traceroute sends packets to a destination, asking each Internet router along the way to reply when it passes on the packet. This will show you the path packets take when you send them between your location and a destination.

In computing, traceroute is a computer network diagnostic tool for displaying the route (path) and measuring transit delays of packets across an Internet Protocol (IP) network. The history of the route is recorded as the round-trip times of the packets received from each successive host (remote node) in the route (path); the sum of the mean times in each hop indicates the total time spent to establish the connection. Traceroute proceeds unless all (three) sent packets are lost more than twice, then the connection is lost and the route cannot be evaluated.

Installation: sudo apt-get install traceroute Commands: traceroute google.com

whois:

The whois command looks up the registration record associated with a domain name. This can show you more information about who registered and owns a domain name, including their contact information.

Installation: :sudo apt-get install whois Commands: whois google.com

The WHOIS protocol had its origin in the ARPANET NICNAME protocol and was based on the NAME/FINGER Protocol, described in RFC 742 (1977). The NICNAME/WHOIS protocol was first described in RFC 812 in 1982 by Ken Harrenstien and Vic White of the Network Information Centre at SRI International.

WHOIS was originally implemented on the Network Control Program (NCP) but found its major use when the TCP/IP suite was standardized across the ARPANET and later the Internet.

nslookup

The nslookup command will look up the IP addresses associated with a domain name. For example, you can run nslookup howtogeek.com to see the IP address of How-To Geek's server

Installation: :sudo apt-get install nslookup Commands: nslookup google.com

dig

Dig is a networking tool that can query DNS servers for information. It can be very helpful for diagnosing problems with domain pointing and is a good way to verify that your configuration is working.

Installation: :sudo apt-get install dig Commands: dig google.com

The dig command output has the following sections:

- Header: This displays the dig command version number, the global options used by the dig command, and few additional header information.
- QUESTION SECTION: This displays the question it asked the DNS. i.e This is your input. Since we said 'dig redhat.com', and the default type dig command uses is A record, it indicates in this section that we asked for the A record of the redhat.com website
- ANSWER SECTION: This displays the answer it receives from the DNS. i.e This is your output. This displays the A record of redhat.com
- AUTHORITY SECTION: This displays the DNS name server that has the authority to respond to this query. Basically this displays available name servers of redhat.com
- ADDITIONAL SECTION: This displays the ip address of the name servers listed in the AUTHORITY SECTION.
- Stats section at the bottom displays few dig command statistics including how much time it took to execute this query.

To view all the record types (A, MX, NS, etc.), use ANY as the record type as shown below

Result: Hence we have studied network reconnaissance tools.

EXP NO: 2 **PACKET SNIFFER TOOLS –WIRESHARK, ETHEREAL,**
DATE: **TCPDUMP**

AIM : To study packet sniffer tools like wireshark, tcpdump, etc. and observe the performance in promiscuous as well as non-promiscuous mode and to find packets based on different filters.

Tools Required : Kali linux, wireshark, tcpdump

Theory :

Wireshark :-

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues.

Wireshark is cross-platform, using the GTK+ widget toolkit in current releases, and Qt in the development version, to implement its user interface, and using pcap to capture packets; it runs on Linux, OS X, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows. There is also a terminal-based (non-GUI) version called TShark. Wireshark, and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License.

Functionality of wireshark:-

Wireshark lets the user put network interface controllers that support promiscuous mode into that mode, so they can see all traffic visible on that interface, not just traffic addressed to one of the interface's configured addresses and broadcast/multicast traffic. However, when capturing with a packet analyzer in promiscuous mode on a port on a network switch, not all traffic through the switch is necessarily sent to the port where the capture is done, so capturing in promiscuous mode is not necessarily sufficient to see all network traffic.

Port mirroring or various network taps extend capture to any point on the network. Simple passive taps are extremely resistant to tampering If a remote machine captures packets and sends the captured packets to a machine running Wireshark using the TZSP protocol or the protocol used by OmniPeek, Wireshark dissects those packets, so it can analyze packets captured on a remote machine at the time that they are captured

Features of wireshark:-

Wireshark is software that "understands" the structure (encapsulation) of different networking protocols. It can parse and display the fields, along with their meanings as specified by different networking protocols.

Wireshark uses pcap to capture packets, so it can only capture packets on the types of networks that pcap supports.

1. Data can be captured "from the wire" from a live network connection or read from a file of already-captured packets.
2. Live data can be read from a number of types of network, including Ethernet, IEEE 802.11, PPP, and loopback.
3. Captured network data can be browsed via a GUI, or via the terminal (command line) version of the utility, TShark.
4. Captured files can be programmatically edited or converted via command-line switches to the "editcap" program.
5. Data display can be refined using a display filter.
6. Plug-ins can be created for dissecting new protocols.
7. VoIP calls in the captured traffic can be detected. If encoded in a compatible encoding, the media flow can even be played.
8. Raw USB traffic can be captured.
9. Wireless connections can also be filtered as long as they transverse the monitored Ethernet.
10. Various settings, timers, and filters can be set that ensure only triggered traffic appear.

Packets captured in promiscuous mode.(Packets are captured in promiscuous mode by default.)

Tcpdump :

tcpdump is a common packet analyzer that runs under the command line. It allows the user to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached. Distributed under the BSD license, tcpdump is free software.

Tcpdump works on most Unix-like operating systems: Linux, Solaris, BSD, OS X, HP- UX, Android and AIX among others. In those systems, tcpdump uses the libpcap library to capture packets. The port of tcpdump for Windows is called WinDump; it uses WinPcap, the Windows port of libpcap.

Result : Hence, we have studied packet sniffer tools like wireshark and tcpdump and learned to implement them.

EXP NO 3 **NMAP –SCAN OPEN PORTS,OS FINGERPRINTING,PING**
DATE: **SCAN,TCP AND UDP PORT SCAN**

AIM: To study and implement various scanning techniques using Nmap.

Materials required: Kali linux, Nmap, Internet.

Theory:

Nmap (*Network Mapper*) is a security scanner originally written by Gordon Lyon (also known by his pseudonym *Fyodor Vaskovich*) used to discover hosts and services on a computer network, thus creating a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host and then analyzes the responses.

The software provides a number of features for probing computer networks, including host discovery and service and operating system detection. These features are extensible by scripts that provide more advanced service detection, vulnerability detection, and other features. Nmap is also capable of adapting to network conditions including latency and congestion during a scan. Nmap is under development and refinement by its user community.

Nmap ("Network Mapper") is a free and open source (license) utility for network discovery and security auditing. Many systems and network administrators also find it useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics.

It was designed to rapidly scan large networks, but works fine against single hosts. Nmap runs on all major computer operating systems, and official binary packages are available for Linux, Windows, and Mac OS X. In addition to the classic command-line Nmap executable, the Nmap suite includes an advanced GUI and results viewer (Zenmap), a flexible data transfer, redirection, and debugging tool (Ncat), a utility for comparing scan results (Ndiff), and a packet generation and response analysis tool (Nping).

Nmap was named "Security Product of the Year" by Linux Journal, Info World, LinuxQuestions.Org, and Codetalker Digest. It was even featured in twelve movies, including The Matrix Reloaded, Die Hard 4, Girl With the Dragon Tattoo, and The Bourne Ultimatum.

Features:

Nmap features include:

- Host discovery – Identifying hosts on a network. For example, listing the hosts that respond to TCP and/or ICMP requests or have a particular port open.
- Port scanning – Enumerating the open ports on target hosts.
- Version detection – Interrogating network services on remote devices to determine application name and version number.
- OS detection – Determining the operating system and hardware characteristics of network devices.
- Scriptable interaction with the target – using Nmap Scripting Engine (NSE) and Lua programming language.

Nmap can provide further information on targets, including reverse DNS names, device types, and MAC addresses.

Uses of Nmap:

- Auditing the security of a device or firewall by identifying the network connections which can be made to, or through it.
- Identifying open ports on a target host in preparation for auditing.
- Network inventory, network mapping, and maintenance and asset management.
- Auditing the security of a network by identifying new servers.
- Generating traffic to hosts on a network.
- Find and exploit vulnerabilities in a network.

Port scanning:

A port scan or port scanning can be defined as a process that sends client requests to a range of server port addresses on a host, with the goal of finding an active port. While not a nefarious process in and of itself, it is one used by hackers to probe target machine services with the AIM of exploiting a known vulnerability of that service, however the majority of uses of a port scan are not attacks and are simple probes to determine services available on a remote machine.

OS fingerprinting:

TCP/IP stack fingerprinting is the passive collection of configuration attributes from a remote device during standard layer 4 network communications. The combination of parameters may then be used to infer the remote machine's operating system (aka, **OS fingerprinting**), or incorporated into a device fingerprint.

Certain parameters within the TCP protocol definition are left up to the implementation. Different operating systems, and different versions of the same operating system, set different defaults for these values. By collecting and examining these values, one may differentiate among various operating systems, and implementations of TCP/IP. The TCP/IP fields that may vary include the following:

- Initial packet size (16 bits)
- Initial TTL (8 bits)
- Window size (16 bits)
- Max segment size (16 bits)
- Window scaling value (8 bits)
- "don't fragment" flag (1 bit)
- "sackOK" flag (1 bit)
- "nop" flag (1 bit)

These values may be combined to form a 67-bit signature, or fingerprint, for the target machine. Just inspecting the Initial TTL and window size fields is often enough in order to successfully identify an operating system, which eases the task of performing manual OS fingerprinting.

TCP scan:

TCP connect scan is the default TCP scan type when SYN scan is not an option. This is the case when a user does not have raw packet privileges. Instead of writing raw packets as most other scan types do, Nmap asks the underlying operating system to establish a connection with the target machine and port by issuing the connect system call. This is the same high- level system call that web browsers, P2P clients, and most other network-enabled applications use to establish a connection. It is part of a programming interface known as the Berkeley Sockets API. Rather than read raw packet responses off the wire, Nmap uses this API to obtain status information on each connection attempt.

UDP scan:

While most popular services on the Internet run over the TCP protocol, UDP services are widely deployed. DNS, SNMP, and DHCP (registered ports 53, 161/162, and 67/68) are three of the most common. Because UDP scanning is generally slower and more difficult than TCP, some security auditors ignore these ports. This is a mistake, as exploitable UDP services are quite common and attackers certainly don't ignore the whole protocol. Fortunately, Nmap can help inventory UDP ports.

UDP scan is activated with the -sU option. It can be combined with a TCP scan type such as SYN scan (-sS) to check both protocols during the same run.

UDP scan works by sending a UDP packet to every targeted port. For some common ports such as 53 and 161, a protocol-specific payload is sent to increase response rate, but for most ports the packet is empty unless the --data, --data-string, or -- data-length options are specified. If an ICMP port unreachable error (type 3, code 3) is returned, the port is closed. Other ICMP unreachable errors (type 3, codes 0, 1, 2, 9, 10, or 13) mark the port as filtered. Occasionally, a service will respond with a UDP packet, proving that it is open. If no response is received after retransmissions, the port is classified as open|filtered. This means that the port could be open, or perhaps packet filters are blocking the communication. Version detection (-sV) can be used to help differentiate the truly open ports from the filtered ones.

Nmap detects rate limiting and slows down accordingly to avoid flooding the network with useless packets that the target machine will drop. Unfortunately, a Linux-style limit of one packet per second makes a 65,536-port scan take more than 18 hours. Ideas for speeding your UDP scans up include scanning more hosts in parallel, doing a quick scan of just the popular ports first, scanning from behind the firewall, and using --host-timeout to skip slow hosts.

Result:

Thus, the AIM to study and implement Nmap which include port scanning, OS fingerprinting, tcp scan and udp scan is accomplished.

EXP NO: 4 **ARP–SPOOFING USING ARPWATCH TOOL**
DATE:

AIM: To find ARP spoofing using the open source tool ARPWATCH.

Tools Required: Kali linux, ARPWATCH

Theory:

arpwatch is a computer software tool for monitoring Address Resolution Protocol traffic on a computer network. It generates a log of observed pairing of IP addresses with MAC addresses along with a timestamp when the pairing appeared on the network. It also has the option of sending an email to an administrator when a pairing changes or is added.

Network administrators monitor ARP activity to detect ARP spoofing, network flip-flops, changed and new stations and address reuse.

arpwatch was developed by Lawrence Berkeley National Laboratory, Network Research Group, as open-source software and is released under the BSD license.

Installing arpwatch:

$ sudo apt-get install arpwatch

Here are some of the most important arpwatch files, the location of the files are slightly differ based on your operating system.

- **/etc/rc.d/init.d/arpwatch : The arpwatch service for start or stop daemon.**
- **/etc/sysconfig/arpwatch : This is main configuration file...**
- **/usr/sbin/arpwatch : Binary command to starting and stopping tool via the terminal.**
- **/var/arpwatch/arp.dat : This is main database file where IP/MAC addresses are recorded.**
- **/var/log/messages : The log file, where arpwatch writes any changes or unusual activity to IP/MAC.**

Type the following command to start the arpwatch service.

 # chkconfig --level 35 arpwatch on # /etc/init.d/arpwatch start

Arpwatch Commands and Usage

To watch a specific interface, type the following command with '-i' and device name.

 # arpwatch -i eth0

So, whenever a new MAC is plugged or a particular IP is changing his MAC address on the network, you will notice syslog entries at '/var/log/syslog' or '/var/log/message' file.

 # tail -f /var/log/messages

Result: Hence, we have detected ARP spoofing using ARPWATCH.

EXP NO: 5 **NESSUS TOOL TO SCAN THE NETWORK FOR**
DATE: **VULNERABILITIES**

AIM: use of nessus tool to scan the network for vulnerabilities.

Theory:

NESSUS Vulnerability Scanner – Basics

If you are looking for a vulnerability scanner, you might have came across several expensive commercial products and tools, with wide range of features and benefits. If a full featured free vulnerability scanner is on your mind, then it's time to know about Nessus. The article covers installation, configuring and select policies, starting a scan, analyzing the reports using NESSUS Vulnerability Scanner.

Nessus was founded by Renuad Deraison in the year 1998 to provide to the Internet community a free remote security scanner. It is one of the full fledged vulnerability scanners which allow you to detect potential vulnerabilities in the systems. Nessus is the world's most popular vulnerability scanning tool and supported by most of the research teams around the world.

The tool is free of cost and non-commercial for non-enterprises. Nessus uses web interface to set up, scan and view repots. It has one of the largest vulnerability knowledge bases and because of this KB the tool is very popular.

Nessus supports wide range of operating systems that include Windows XP/7, Linux, Mac OS X, Sun Solaris, etc.

Key Features:

- Identifies Vulnerabilities that allow a remote attacker to access sensitive information from the system.
- Checks whether the systems in the network has the latest software patches.
- Tries with Default passwords, common passwords, on systems account
- Configuration audits.
- Vulnerability analysis.
- Mobile Device audits.
- Customized reporting.

Installation & Configuration:

1. You can download the Nessus home feed (free) or professional feed from Nessus website.
2. Once you download the Nessus home tool, you need to register for generating an activation key. The activation key will be sent to your email id.
3. Install the tool (Installation of nessus tool will be quite confusing and the installation guide comes handy).
4. Open the Nessus in the browser, normally it runs on the port 8834 –

http://localhost:8834/WelcomeToNessus-Install/welcome and follow the screen.

5. Create an account with Nessus.
6. Enter the activation code you have obtained by registering with the Nessus website. Also you can configure the proxy if needed by giving proxy hostname, proxy username and password.
7. Then scanner gets registered and creates the user account.
8. Then downloads the necessary plugins (It takes some time for downloading the plugins).
9. Once the plug-ins are downloaded then it will automatically redirects you to a login screen. Provide the Username and password that you have created earlier to login.

Running the Tool:

Nessus gives you lots of choices when it comes to running the actual vulnerability scan. You'll be able to scan individual computers, ranges of IP addresses or complete subnets. There are over 1200 vulnerability plugins with Nessus using which you'll be able to specify individual or set of vulnerabilities to test for. In contrast to other tools Nessus won't assume for explicit services running on common ports instead it will try to exploit the vulnerabilities.

One of the foundations for discovering the vulnerabilities in the network are:

- Knowing which systems exist
- Knowing which ports are open and which listening services are available in those ports
- Determining which Operating System is running in the remote machine

Once you log into the Nessus using web-interface, you will be able to see different options like,

- Policies –Using which you can configure the options required for scan
- Scans -for adding different scans
- Reports -for analyzing the results

Basic workflow of Nessus tool is to Login, Create or Configure the Policy, Run the Scan and Analyze the Results.

POLICIES:

Policies are nothing but the vulnerability tests that you can perform on the target machine. By default Nessus has 4 policies.

External Network Scan:

The policy is pre-configured in such a way that Nessus scans externally facing hosts, which provides services to the host. It scans all 65,535 ports of the target machine. It is also configured with Plugins required for web application vulnerabilities tests like XSS.

Internal Network Scan:

This policy is configured to scan large internal networks with many hosts, services, embedded systems like printers, etc... This policy scans only standard ports instead of scanning all 65,535 ports.

Web App Tests:

Nessus uses this policy to detect different types of vulnerabilities exist in the web applications. It has the capability to spider the entire web site and discovers the content and links in the application. Once the spider process has been completed then Nessus starts to discover the vulnerabilities that exist in the application.

Prepare for PCI DSS audits:

This policy consists of PCI DSS (Payment Card Industry Data Security Standards) enabled.

Nessus compares the results with the standards and produces a report for the scan. The

scan doesn't guarantee for a secure infrastructure. Industries or Organizations preparing for PCI-DSS can use this policy to prepare their network and systems.

Apart from these pre-configured policies you can also upload a policy by clicking on "Upload" or configure your own policy as per your scan requirement by clicking on "New Policy".

Configuring the Policy:

- Click on the policies tab on the top of the screen
- Click on the New Policy button to create a new policy

Under the General settings tab select the "setting type" based on scan requirement, like Port Scanning, Performance scanning etc... Based on the type Nessus prompts different options that has to be filled. For example 'Port Scanning' has the following options

Enter the port scan range. By default Nessus scans all the TCP ports in /etc/services file. You can limit the ports by specifying it manually (like 20-30). You have different scanners like Nessus SNMP scanner, SSH scanner, ping remote host, TCP Scanner, SYN scanner, etc.... Enable by selecting the check box as per the scan requirement.

- Enter the credentials for scan to use. You can use single set of credentials or multiple set of credentials if you have. You can also work it out without entering the credentials.

- The plugins tab has number of plugins. By default Nessus will have all the plugins enabled. You can enable or disable all the plugins at a time or enable few from the plug-in family as per the scan you'd like to perform. You can also disable some unwanted

plugins from the plug-in family by clicking on particular plug-in.

- In the Preferences, you are provided with a drop down box to select different types of plugins. Select the plugin based on the scan requirement and specify the settings as per the plugins requirement. Click finish once completed. For example: configure the database.

SCANS:

Once you are done with configuring the policies as per your scan requirement, you need to configure the scan details properly. You can do it under Scan tab.

Under the Scan tab, you can create a new scan by clicking *New Scan* on the top right. Then a pop up appears where you need to enter the details like Scan Name, Scan Type, Scan Policy & Target.

- Scan Name: The name that you are willing to give to the scan.
- Scan Type: You have options to RUN the scan instantly by selecting *RUN NOW*. Or you can make a template which you can launch later when you are willing to run. All the templates are moved under the TEMPLATE tab beside the SCAN tab.
- Scan Policy: Select the policy that you have configured previous in the policies section.
- Select Target: Enter the target machine which you are planning to test. Depending upon the targets Nessus takes time to scan the targets.

Results:

Once the scanning process has been completed successfully, results can be analyzed from RESULTS menu.

- Once the scan has been completed, you can see the name of the scan under the results section. Click on the name to see the report.
- Hosts: Specifies all the target systems that you have scanned.
- Vulnerabilities: Displays all the vulnerabilities on the target machine that has been tested.
- Export Results: You can export the results into difference formats like html, pdf, etc… You can also select an individual section or complete result to export based on your requirement.

Let us try out an example now-

I have configured a policy named *Basic Scan*. We have many options while configuring or building the policy like port scanners, performance of the tool, Advanced etc.

You don't need credentials now, so skip the credentials tab and move to Plugins tab. You need to configure the specific plug-in as per the scan requirement that you are willing to perform on remote machine.

I have configured the scan to run instantly with the policy that I have created earlier. And the scan target specify the IP address I am willing to scan.

Once all the details has been entered click on *Create Scan* which shows the Scan is running as shown in the below Figure.

Once the scanning has been completed then you can see the results in Results tab.

Double clicking on the title displays the scan results.

The above figure shows the Hosts details. It includes all the targets that you have scanned during the test. Double clicking on the host address displays the vulnerabilities Nessus have identified during the test. You can also click on Vulnerabilities tab to check out the vulnerabilities.

Based on the Risk Nessus marks it as high, medium, info etc... Clicking on the Vulnerability gives you brief description of it.

For example let us go with Netstat portscanner, displays you the following information

In the same manner you can analyze complete details by clicking on the vulnerabilities. Nessus also suggests the solutions or remedies for the vulnerabilities with few references.

Result:

Nessus is a tool which automates the process of scanning the network and web applications for the vulnerabilities also suggests solutions for the vulnerabilities that are identified during the scan.

EXP NO: 6 **SIMULATE BUFFER OVERFLOW ATTACK**
DATE:

AIM: Implement a code to simulate buffer overflow attack.

Materials Required: Kali linux, gcc compiler

Theory:

In computer security and programming, a buffer overflow, or buffer overrun, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites memory in adjacent locations. This is a special case of the violation of memory safety.

Buffer overflows can be triggered by inputs that are designed to execute code, or alter the way the program operates. This may result in erratic program behaviour, including memory access errors, incorrect results, a crash, or a breach of system security. Thus, they are the basis of many software vulnerabilities and can be maliciously exploited.

Programming languages commonly associated with buffer overflows include C

And C++, which provide no built-in protection against accessing or overwriting data in any part of memory and do not automatically check that data written to an array (the built-in buffer type) is within the boundaries of that array. Bounds checking can prevent buffer overflows.

Although tons of research has been done to tackle buffer overflow attacks, existing defences are still quite limited in meeting four highly desired requirements:

1. simplicity In maintenance;
2. transparency to existing (legacy) server OS, application software, and hardware;
3. resiliency to obfuscation;
4. economical Internet-wide deployment.

As a result, although several very secure solutions have been proposed, they are not pervasively deployed, and a considerable number of buffer overflow attacks continue to succeed on a daily basis. Existing defences are limited in meeting these four requirements. Existing buffer overflow defences are categorized into six classes.

(A) Finding bugs in source code.

(B) Compiler extensions.

(C) OS modifications.

(D) Hardware modifications.

(E) Defence-side obfuscation.

(F) Capturing code running symptoms of buffer overflow attack

OBJECTIVE:

Buffer overflow problems are well known. Several studies have been conducted to detect and prevent buffer overflows. The purpose of this work is to study security problems and to detect buffer overflow attack using various techniques.

The following are the major objective:

1. To study the vulnerabilities and attacks such as buffer overflow.
2. To study some of the existing techniques used for detecting and preventing buffer overflow.
3. To develop a new efficient techniques for detecting and preventing the buffer overflow.
4. To analyse this techniques. What is the stack used for?

The stack works according to a LIFO model (Last In First Out). Since the spaces within the stack are allocated for the lifetime of a function, only data that is active during this lifetime can reside there. Only this type of structure results from the essence of a structural approach to programming, where the code is split into many code sections called functions or procedures. When a program runs in memory, it sequentially calls each individual procedure, very often taking one from another, thereby producing a multi-level chain of calls. Upon completion of a procedure it is required for the program to continue execution by processing the instruction immediately following the CALL instruction. In addition, because the calling function has not been terminated, all its local variables, parameters and execution status require to be "frozen" to allow the remainder of the program to resume execution immediately after the call. The implementation of such a stack will guarantee that the behaviour described here is exactly the same.

Function calls

The program works by sequentially executing CPU instructions. For this purpose the CPU has the Extended Instruction Counter (EIP register) to maintain the sequence order. It controls the execution of the program, indicating the address of the next instruction to be executed. For example, running a jump or calling a function causes the said register to be appropriately modified. Suppose that the EIP calls itself at the address of its own code section and proceeds with execution. What will happen then?

When a procedure is called, the return address for function call, which the program needs to resume execution, is put into the stack. Looking at it from the attacker's point of view, this is a situation of key importance. If the attacker somehow managed to overwrite the return address stored on the stack, upon termination of the procedure, it would be loaded into the EIP register, potentially allowing any overflow code to be executed instead of the process code resulting from the normal behaviour of the program. We may see how the stack behaves after the code of Listing 1 has been executed.

Preventing buffer overflow attacks

The most straightforward and effective solution to the buffer overflow problem is to employ secure coding. On the market there are several commercial or free solutions available which effectively stop most buffer overflow attacks. The two approaches here are commonly employed:

- library-based defences that use re-implemented unsafe functions and ensure that these functions can never exceed the buffer size. An example is the Libsafe project.
- library-based defences that detect any attempt to run illegitimate code on the stack. If the stack smashing attack has been attempted, the program responds by emitting an alert.

This is a solution implemented in the SecureStack developed by SecureWave. Another prevention technique is to use compiler-based runtime boundaries, checking what recently became available and hopefully with time, the buffer overflow problem will end up being a major headache for system administrators. While no security measure is perfect, avoiding programming errors is always the best solution.

Before and after the doit() function, we have two calls to function printf().

The program between the two defined printf() calls displays the content of the buffer, which is filled with data entered by the user.

If our binary application is in ELF format, then we are able to use an objdump program to analise it and find necessery information to exploit the buffer overflow error.

Below is output produced by the objdump. From that output we are able to find addresses, where printf() is called (0x80483d6 and 0x80483e7).

There is no control over the size of the copied buffer into the previously declared one. In this example we overwrite the EIP register with address 0x080483f9, which is in fact a call to ret in the last phase of the program execution.

Result: Hence, we have implemented a buffer overflow attack.

EXP NO: 7 **SET UP IPSEC UNDER LINUX**
DATE:

AIM: To set up IPSEC under LINUX.

Tools Required: Kali linux, vpn configuration, firewall, racoon

Theory:

Many organizations across the world use every available physical connection method to link up their individual offices. The option chosen can be both dedicated digital lines and Virtual Private Networks (VPN), which are significantly cheaper than their physical equivalents. VPNs, which generally deploy the same approaches as dedicated lines, can combine several LANs into one and encrypt the traffic to conceal the data being transmitted. When encryption is deployed in VPN technology, open standards are generally used. This involves the traffic being transmitted on top of IP and using datagrams as the transport level.

From a technical perspective, VPNs can be implemented using both software and hardware. In Linux®, FreeS/Wan technology has often been deployed, using the standard implementation of the security protocol IPSEC (Internet Protocol Security). These solutions, which are implemented using both software and hardware, operate like routers at the ends of the VPN connections. When a packet is transmitted by the client, it is sent to this dedicated router, which adds an Authentication Header (AH) to it. After this data is encrypted and instructions are added for decrypting and processing it, the packet is transferred to the dedicated router at the other end. Once the packet is received, the end router decrypts it, discarding the header, and transmits the clear packet to the user at the destination.

If encryption is used between the networks, the host in the LAN receives the packet already decrypted and starts processing it in the normal way. This means that when encryption is used between networks, the entire encryption/decryption process becomes transparent to the network's end host.

Several levels of authentication and encryption are deployed in VPNs, thus making them secure and effective enough to combine multiple remote nodes into a single intranet.

IPSEC is a popular implementation of the VPN standard which is reliable enough to meet the requirements of various customers in terms of connecting their branches or remote users to their networks.

General overview of IPSEC

IPSEC is generally used to support secure connections between nodes and networks throughout the Internet. It can operate in a node-to-node configuration (with one computer connected to another) or in a network-to-network configuration (with one LAN/WAN connected to another). IPSEC is implemented using the Internet Key Exchange (IKE) protocol developed by the Internet Engineering Task Force (IETF) for the mutual authentication and comparison of security parameters between systems or networks connecting to each other.

The IPSEC connection process is split into two logical phases. During the first phase, the IPSEC node establishes a connection with a remote node or network. The remote node/network verifies the credentials of the requesting node and both sides agree on the authentication method

to be used in the connection. An algorithm with a pre-shared key is usually used for the IPSEC node's authentication. If the IPSEC connection uses a pre-shared key, both nodes must use the same key. It will then be possible to proceed to the second phase of establishing a connection.

The second phase of establishing an IPSEC connection between nodes is carried out using Security Association (SA). This involves configuration data, such as the encryption method, means of exchanging the session's secret keys and a few other parameters being imported into the SA database. This phase also manages the IPSEC connection between the nodes and networks distributed throughout the WAN.

Now consider the implementation of IPSEC based on the example of the CentOS Linux distribution. To deploy IPSEC on all the machines in the network (in the case of a node-to-node configuration) or on the routers (in the case of a network-to-network configuration), you must set up the relevant packets for managing the IPSEC configuration. These packets must include basic libraries, daemons, and configuration files that help establish the IPSEC connection, including the /lib/libipsec.so library containing the interface for managing the trusted key, PF_KEY, between the Linux kernel and the IPSEC implementation being used in CentOS Linux. In this case:

- /sbin/setkey provides key management and the IPSEC security attributes in the kernel. This program complies with the racoon daemon managing the keys. Further information about setkey is available in the setkey(8) man page (see Resources).
- /sbin/racoon is the daemon managing the IKE keys and supervises the key exchange and security association between the computers running IPSEC. This daemon can be set up after editing the file/etc/racoon/racoon.conf. Further information about racoon is available from the racoon(8) man page (see Resources).
- /etc/racoon/racoon.conf is configuration file where various IPSEC connection parameters are set, including the authentication methods and encryption algorithms.

Setting up IPSEC for node-to-node configuration

IPSEC can be used to connect one workstation to another, based on a node-to-node connection. With this kind of connection, the network to which both nodes are connected is used to create a secure tunnel. The nodes only need a permanent connection to the Internet or another constantly operating network to establish the IPSEC connection.

The following data is required to create a node-to-node connection:

- IP addresses for both nodes
- A unique name for the IPSEC connection, differentiating it from the other devices or connections (ipsec0)
- An encryption key which is permanent or created automatically using racoon
- An authentication pre-shared key used to establish the connection and exchange encryption keys during the connection session

Now look at the scenario where two hosts establish a connection with each other. They will do this using the shared key my_keyand the daemon racoonto automatically create and exchange an authentication key. The connection name will be ipsec0.

The workstation uses the ifcfg file, in Listing 1, to establish the IPSEC node-to-node connection with the other workstation. Listing 1 shows the format of the file

/etc/sysconfig/network-scripts/ifcfg-ipsec0.

On the first computer the letters *X.X.X.X* must be replaced by the second computer's IP address, and vice versa on the second computer. This connection is established during boot- up (ONBOOT=yes) and uses the pre-shared key authentication method (IKE_METHOD=PSK).

The file with the shared key (/etc/sysconfig/network-scripts/keys-ipsec0), shown in Listing 2, is needed by both computers for mutual authentication. The content of this file must be the same on both computers, and only the root user must be able to access it: IKE_PSK=my_key.

To change authentication key at any time, you must edit the keys-ipsec0 file on both computers. Both keys must be identical to establish the connection.

Listing 3 examines the configuration process for the first phase in connecting to a remote node. This file is called *X.X.X.X.conf* (*X.X.X.X* is replaced by the IP address of the remote IPSEC router). Remember that this file is generated automatically when the IPSEC tunnel is activated and is not edited manually.

- he command remote *X.X.X.X* indicates that the following strings in the configuration file are only applied to the remote node being assigned to the address *X.X.X.X*.
- exchange_mode aggressive in the default IPSEC configuration in CentOS Linux is an authentication method that allows different IPSEC connections with multiple nodes.
- my_identifier address defines the identification method that will be used for node authentication. CentOS Linux identifies the nodes at the IP addresses.
- encryption_algorithm 3des defines the encryption algorithm used for authentication. The default method is the Triple Data Encryption Standard (3DES).
- hash_algorithm sha1 indicates the hash calculation algorithm used during the first phase of the connection.
- authentication_method pre_shared_key defines the authentication method used when synchronizing nodes.
- Bdh_group 2 indicates the number of the Diffie-Hellman group for selecting the dynamically generated session keys. The 1024-bit group is used by default.

The /etc/racoon/racoon.conf must also be identical at all the IPSEC nodes, *apart from* the operator include "/etc/racoon/X.X.X.X.conf", which will have a different IP address. Both the

operation and the file are generated when the IPSEC tunnel is activated. Listing 4 shows the typical racoon.conf produced when the IPSEC connection is established:

The description of all the parameters for the configuration files falls outside the scope of this article. You can obtain them from the appropriate manuals.

To establish the connection, either reboot the computer or execute the following command at each node on behalf of root: /sbin/ifup ipsec0.

Launch the utility tcpdump to check the IPSEC connection. When doing so, the packet must contain the AH and ESP data. ESP means that encryption is working. Listing 5 shows an example tcpdump:

Result: Hence we have studied IPSec under Linux environment.

EXP NO: 8 **INSTALL IDS (SNORT) AND STUDY THE LOGS**
DATE:

AIM: To study and test the working of IDS SNORT.

Materials Required: Kali linux, SNORT pre-requisites and installationfiles.

Theory:

Snort has four main pre-requisites:

Pcap	(libpcap-dev)	available from the Ubuntu repository
PCRE	(libpcre3-dev)	available from the Ubuntu repository
Libdnet	(libdumbnet-dev)	available from the Ubuntu repository DAQ
	(www.snort.org/downloads)	compiled from source

First we want to install all the tools required for building software. The build-essentials package does this for us:

sudo apt-get install -y build-essential

Once our build tools are installed, we install all Snort pre-requisites that are available from the Ubuntu
Repositories

sudo apt-get install -y libpcap-dev libpcre3-dev libdumbnet-dev

The Snort DAQ (Data Acquisition library)has a few pre-requisites that need to be installed:

sudo apt-get install -y bison flex

Download and install the latest version of DAQ from the Snort website (please check thewebsite to ensure
you are getting the latest version).

wget https://www.snort.org/downloads/snort/daq 2.0.4.tar.gztar -xvzf daq-2.0.4.tar.gz

cd daq-2.0.4

./configuremake

sudo make install

To install Snort on Ubuntu, there is one other additional Snort pre-requisite that needs to beinstalled that is not mentioned in the documentation, zlibg, a compression library:

sudo apt-get install -y zlib1g-dev

We are now ready to download the Snort source tarball, compile, and then install. The --

enable-sourcefire option gives Packet Performance Monitoring (PPM), which lets us do performance monitoring for rules and pre-processors, and builds Snort the same way that the Snort team does:

cd ~/snort_src
wget https://www.snort.org/downloads/snort/snort-2.9.7.0.tar.gz tar -xvzf snort-2.9.7.0.tar.gz

cd snort-2.9.7.0
./configure --enable-sourcefiremake

sudo make install

Run the following command to update shared libraries (you'll get an error when you try torun Snort if you
skip this step):

sudo ldconfig

Place a symlink to the Snort binary in /usr/sbin:

sudo ln -s /usr/local/bin/snort /usr/sbin/snort

Test Snort by running the binary as a regular user, passing it the -V flag (which tells Snort to verify itself and any configuration _les passed to it). You should see output similar to what is shown below (although exact version numbers may be slightly different).

user@snortserver:~$ snort -V
,,_ -*> Snort! <*-

o")~ Version 2.9.7.0 GRE (Build 149)
'''' By Martin Roesch & The Snort Team: http://www.snort.org/contact#teamCopyright (C) 2014 Cisco and/or its affiliates. All rights reserved.

Copyright (C) 1998-2013 Sourcefire, Inc., et al.Using libpcap version 1.6.2
Using PCRE version: 8.35 2014-04-04Using ZLIB version: 1.2.8 user@snortserver:~$

Configuring Snort to Run in NIDS Mode

Since we don't want Snort to run as root, we need to setup an unprivileged account and group for the daemon to run under (snort: snort). We will also create a number of _les anddirectories required by Snort, and set permissions on those _les. Snort will keep all configurations and rule _les in /etc/snort, and all alerts will be written to /var/log/snort.

sudo groupadd snort
sudo useradd snort -r -s /sbin/nologin -c SNORT_IDS -g snortsudo mkdir /etc/snort

sudo mkdir /etc/snort/rules
sudo mkdir /etc/snort/preproc_rules

sudo touch /etc/snort/rules/white_list.rules

/etc/snort/rules/black_list.rules
/etc/snort/rules/local.rulessudo mkdir /var/log/snort

sudo mkdir /usr/local/lib/snort_dynamicrulessudo chmod -R 5775 /etc/snort
sudo chmod -R 5775 /var/log/snort

sudo chmod -R 5775 /usr/local/lib/snort_dynamicrules sudo chown -R snort:snort /etc/snort
sudo chown -R snort:snort /var/log/snort

sudo chown -R snort:snort /usr/local/lib/snort_dynamicrules

Snort needs the following configuration _les copied from the Snort tarball into the /etc/snort folder:

_ classification.con_g
_ reference.con_g
_ snort.conf
_ threshold.conf
_ gen-msg.map
_ unicode.map

To copy the configuration _les, run the following commands:

sudo cp ~/snort_src/snort-2.9.7.0/etc/*.conf* /etc/snort sudo cp ~/snort_src/snort-2.9.7.0/etc/*.map /etc/snort

We now need to edit Snort's main configuration _le, /etc/snort/snort.conf. When we run Snort with
this file as an argument, it tells Snort to run in NIDS mode.
Before we run Snort in NIDS mode, we need to make a few edits to the default configuration _le. We need to comment out all of the individual rule _les that are referenced in the Snort configuration _le, since instead of downloading each _le individually, we will use PulledPork to manage our rulesets, which combines all the rules into a single _le. The following line will comment out all rulesets in our snort.conf file:

sudo sed -i 's/include \$RULE_PATH/#include \$RULE_PATH/' /etc/snort/snort.conf

We will now manually change some settings in the snort.conf _le, using your favorite editor:

sudo vi /etc/snort/snort.conf

Change the following lines to meet your environment:

Line 45, HOME NET should match your internal (friendly) network, and line 48, EXTERNAL NET should be all other networks. In the below example our HOME NET is
10.0.0.0 with a 24-bit subnet mask (255.255.255.0):

ipvar HOME_NET 10.0.0.0/24

ipvar EXTERNAL_NET !$HOME_NET

Set the following _le paths, beginning at line 104:

var RULE_PATH /etc/snort/rules
var SO_RULE_PATH /etc/snort/so_rules

var PREPROC_RULE_PATH /etc/snort/preproc_rules var WHITE_LIST_PATH /etc/snort/rules
var BLACK_LIST_PATH /etc/snort/rules

In order to make testing snort easy, we want to enable the local.rules _le, where we can add rules that Snort can alert on, which is good for testing. Un-comment (remove the hash symbol) from line 545:

include $RULE_PATH/local.rules

Once the configuration file is ready, we will have Snort verify that it is a valid _le, and all necessary files it references are correct. We use the -T flag to test the configuration _le, and we use the -c flag to tell Snort which configuration _le to use. Run the following command and look for the following output:

user@snortserver:~$ sudo snort -T -c /etc/snort/snort.conf(...)
Snort successfully validated the configuration!

Snort exiting user@snortserver:~$

At this stage, Snort does not have any rules loaded (our rule _les referenced in snort.conf are empty).
You can verify that Snort has not loaded any rules if you scroll up through the output from the previous command. To test Snort, let's create a simple rule that will cause Snort to generate an alert whenever Snort sees an ICMP \Echo request" or \Echo reply" messages, which is easy to generate with the ubiquitus ping utility (this makes for easy testing of the rule). Paste the following line into the empty local rules _le: /etc/snort/rules/local.rules:

alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:10000001; rev:001;)

When you un-commented line 545 above (include $RULE PATH/local.rules) you were telling Snort that the local.rules fie should be loaded by Snort. When Snort loads that _le on start-up, it will see the rule you created, and use that rule on all traffic the interface sees. In this case, when we created the rule, we told Snort that it should generate an alert when it sees an ICMP ping. Since we made changes to the Snort configuration, we should test the configuration file again:

sudo snort -T -c /etc/snort/snort.conf

This time if you scroll up through the output, you will find that one rule (the one we created in local.rules, and loaded by the include in snort.conf) has been loaded:

+-------------------[Rule Port Counts]-------------------------------------
| tcp udp icmp ip

| src 0 0 0 0
| dst 0 0 0 0

| any 0 0 1 0
| nc 0 0 1 0

| s+d 0 0 0 0
+--

Now that we know that Snort correctly loads our rule and our configuration, we can start snort in NIDS
mode, and tell it to output any alerts right to the console. We will run Snort from thecommand line, using the following flags:

-A console	The `console' option prints fast mode alerts to stdout
-q Quiet mode.	Don't show banner and status report.
-u snort	Run Snort as the following user after startup
-g snort	Run Snort as the following group after startup
-c /etc/snort/snort.conf	The path to our snort.conf file
-i eth0	The interface to listen on

$ sudo /usr/local/bin/snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i eth0

Snort is now running, processing all packets that arrive on eth0, comparing them to the rules it has loaded (in this case our single ICMP Ping rule), and sending all alerts generated when apacket matches our rule to the console. From another computer, ping the IP address of eth0on the Snort computer (or alternately ping from the Snort host to another machine, or to its own eth0, but not loopback interface), and you should see console output similar to what is displayed below (in the below example, the Snort server is listening on eth0 with and IP address of 10.0.0.218, and the computer generating the ping is 10.0.0.169).

Use ctrl-c to stop Snort from running. Note that Snort has saved a copy of this information in
/var/log/snort, with the name **snort.log.nnnnnnnnn** (the numbers may be different). At this point Snort is running correctly in NIDS mode and generating alerts.

Result: Hence, we have studied IDS using snort and studied the logs

EXP NO:9 **USE OF IPTABLES IN LINUX TO CREATE FIREWALL**
DATE:

AIM: To study use of iptables in linux to create firewalls

Materials Required: Kali linux,

Theory:

iptables is a command-line firewall utility that uses policy chains to allow or block traffic. When a connection tries to establish itself on your system, iptables looks for a rule in its list to match it to. If it doesn't find one, it resorts to the default action.

iptables almost always comes pre-installed on any Linux distribution. To update/install it, just retrieve the iptables package:

sudo apt-get install iptables

iptables uses three different chains: input, forward, and output.

Input – This chain is used to control the behavior for incoming connections. For example, if a user attempts to SSH into your PC/server, iptables will attempt to match the IP address and port to a rule in the input chain.

Forward – This chain is used for incoming connections that aren't actually being delivered locally. Think of a router – data is always being sent to it but rarely actually destined for the router itself; the data is just forwarded to its target. Unless you're doing some kind of routing, NATing, or something else on your system that requires forwarding, you won't even use this chain.

There's one sure-fire way to check whether or not your system uses/needs the forward chain.

The screenshot above is of a server that's been running for a few weeks and has no restrictions on incoming or outgoing connections. As you can see, the input chain has processed 11GB of packets and the output chain has processed 17GB. The forward chain, on the other hand, has not needed to process a single packet. This is because the server isn't doing any kind of forwarding or being used as a pass-through device.

Output – This chain is used for outgoing connections. For example, if you try to ping howtogeek.com, iptables will check its output chain to see what the rules are regarding ping and howtogeek.com before making a decision to allow or deny the connection attempt.

Before going in and configuring specific rules, you'll want to decide what you want the default behavior of the three chains to be. In other words, what do you want iptables to do if the connection doesn't match any existing rules?

To see what your policy chains are currently configured to do with unmatched traffic.

As you can see, we also used the grep command to give us cleaner output. In that screenshot, our chains are currently figured to accept traffic.

More times than not, you'll want your system to accept connections by default. Unless you've changed the policy chain rules previously, this setting should already be configured. Either way, here's the command to accept connections by default:

```
iptables --policy INPUT ACCEPT
iptables --policy OUTPUT ACCEPT
iptables --policy FORWARD ACCEPT
```

By defaulting to the accept rule, you can then use iptables to deny specific IP addresses or port numbers, while continuing to accept all other connections. We'll get to those commands in a minute.

If you would rather deny all connections and manually specify which ones you want to allow to connect, you should change the default policy of your chains to drop. Doing this would probably only be useful for servers that contain sensitive information and only ever have the same IP addresses connect to them.

```
iptables --policy INPUT DROP
iptables --policy OUTPUT DROP
iptables --policy FORWARD DROP
```

Connection-specific Responses

With your default chain policies configured, you can start adding rules to iptables so it knows what to do when it encounters a connection from or to a particular IP address or port. In this guide, we're going to go over the three most basic and commonly used "responses".

Accept – Allow the connection.

Drop – Drop the connection, act like it never happened. This is best if you don't want the source to realize your system exists.

Reject – Don't allow the connection, but send back an error. This is best if you don't want a particular source to connect to your system, but you want them to know that your firewall blocked them.

The best way to show the difference between these three rules is to show what it looks like when a PC tries to ping a Linux machine with iptables configured for each one of these settings

Allowing or Blocking Specific Connections

With your policy chains configured, you can now configure iptables to allow or block specific addresses, address ranges, and ports. In these examples, we'll set the connections to DROP, but you can switch them to ACCEPT or REJECT, depending on your needs and how you configured your policy chains.

Note: In these examples, we're going to use iptables -A to append rules to the existing chain. iptables starts at the top of its list and goes through each rule until it finds one that it matches. If you need to insert a rule above another, you can use iptables -I [chain] [number] to specify the number it should be in the list.

Connections from a single IP address

This example shows how to block all connections from the IP address 10.10.10.10.

iptables -A INPUT -s 10.10.10.10 -j DROP

Connections from a range of IP addresses

This example shows how to block all of the IP addresses in the 10.10.10.0/24 network range. You can use a netmask or standard slash notation to specify the range of IP addresses.

iptables -A INPUT -s 10.10.10.0/24 -j DROP

or

iptables -A INPUT -s 10.10.10.0/255.255.255.0 -j DROP

Connections to a specific port

This example shows how to block SSH connections from 10.10.10.10.

```
iptables -A INPUT -p tcp --dport ssh -s 10.10.10.10 -j DROP
```

You can replace "ssh" with any protocol or port number. The -p tcp part of the code tells iptables what kind of connection the protocol uses. If you were blocking a protocol that uses UDP rather than TCP, then -p udp would be necessary instead.

This example shows how to block SSH connections from any IP address.

```
iptables -A INPUT -p tcp --dport ssh -j DROP
```

Connection States

As we mentioned earlier, a lot of protocols are going to require two-way communication. For example, if you want to allow SSH connections to your system, the input and output chains are going to need a rule added to them. But, what if you only want SSH coming into your system to be allowed? Won't adding a rule to the output chain also allow outgoing SSH attempts?

That's where connection states come in, which give you the capability you'd need to allow two way communication but only allow one way connections to be established. Take a look at this example, where SSH connections FROM 10.10.10.10 are permitted, but SSH connections TO 10.10.10.10 are not. However, the system is permitted to send back information over SSH as long as the session has already been established, which makes SSH communication possible between these two hosts.

```
iptables -A INPUT -p tcp --dport ssh -s 10.10.10.10 -m state --state
NEW,ESTABLISHED -j ACCEPT

iptables -A OUTPUT -p tcp --sport 22 -d 10.10.10.10 -m state --state
ESTABLISHED -j ACCEPT
```

Saving Changes

The changes that you make to your iptables rules will be scrapped the next time that the iptables service gets restarted unless you execute a command to save the changes. This command can differ depending on your distribution:

Ubuntu:

```
sudo /sbin/iptables-save
```

Red Hat / CentOS:

```
/sbin/service iptables save
```

Or

```
/etc/init.d/iptables save
```

Other Command

List the currently configured iptables rules:

```
iptables –L
```

Adding the -v option will give you packet and byte information, and adding -n will list everything numerically. In other words – hostnames, protocols, and networks are listed as numbers.

To clear all the currently configured rules, you can issue the flush command

Result: Hence, we have studied the use of iptables in linux to create firewalls.