SecureSet Prep
Introduction to Network Security 3

Glenn Webb, CISSP, GSEC

Table of Contents

ARP packets

Use Wireshark in your Kali VM to capture and analyze an ARP packet.
How often does it appear that ARP table entries time out?

ARP tables

run this command `arp -a` in a terminal window to see your ARP table.

ARP Spoofing Lab

1. Start VirtualBox Manager
2. This step creates an internal virtual network in VirtualBox
    a. Click File Menu
    b. Preferences
    c. Network
    d. Click green plus sign
    e. click OK
3. Download the 64 bit Kali Linux template for VirtualBox on this page
    a. https://www.offensive-security.com/kali-linux-vmware-virtualbox-image-download/
    b. Click on the "Kali Linux VirtualBox Images" tab
    c. Click on the **Kali Linux 64 bit VBox** link (not the Torrent link)
4. In your VirtualBox Manager
    a. File
    b. Import Appliance...
    c. Navigate to where you saved it and select it
    d. Scroll through the "Appliance settings" to familiarize yourself with it
    e. Check the "Reinitialize the MAC address" checkbox
    f. Click "Import"
    g. After it completes you will have a Kali VM, with root/toor login credentials
5. Clone your Ubuntu VM
    a. Right click on the VM you want to clone
    b. Choose Clone
    c. Give it a new name
    d. Click "Reinitialize the MAC address"
    e. Next
    f. Click "Full Clone"
    g. Click Clone
6. Click on (highlight) your original Ubuntu VM in the left panel of the VirtualBox Manager
    a. Click on Settings button at top

   b. Click on Network
   c. Change "Attached to:" to be NAT Network
   d. Name: NatNetwork
   e. Click OK
7. Do step 6 on the new clone you created
8. Do step 6 on the Kali VM
9. Start all 3 VM's
10. Log into all 3 VM's
11. In the original Ubuntu VM bring up a terminal
    a. run `sudo apt-get install net-tools xinetd telnetd`
    b. run `sudo systemctl stop ufw`
    c. run `ip addr` and make a note of the IP address
12. In the cloned Ubuntu VM bring up a terminal window
    a. run `sudo apt-get install net-tools xinetd telnetd`
    b. run `sudo systemctl stop ufw`
    c. run `ip addr` and make a note of the IP address
13. On the Kali VM bring up 3 terminal windows
    a. In the first terminal run `arpspoof -t first_IP_address second_IP_address`
    b. In the second term run `arpspoof -t second_IP_address first_IP_address`
    c. In the third terminal run `echo 1 > /proc/sys/net/ipv4/ip_forward`
    d. In the third terminal run `dsniff`
14. In the original Ubuntu VM run `telnet second_IP_address`
    a. log in with your regular username
15. In the Kali VM check to see if dsniff captured your login credentials

https://null-byte.wonderhowto.com/how-to/hack-like-pro-conduct-simple-man-middle-attack-0147291/

Netcat

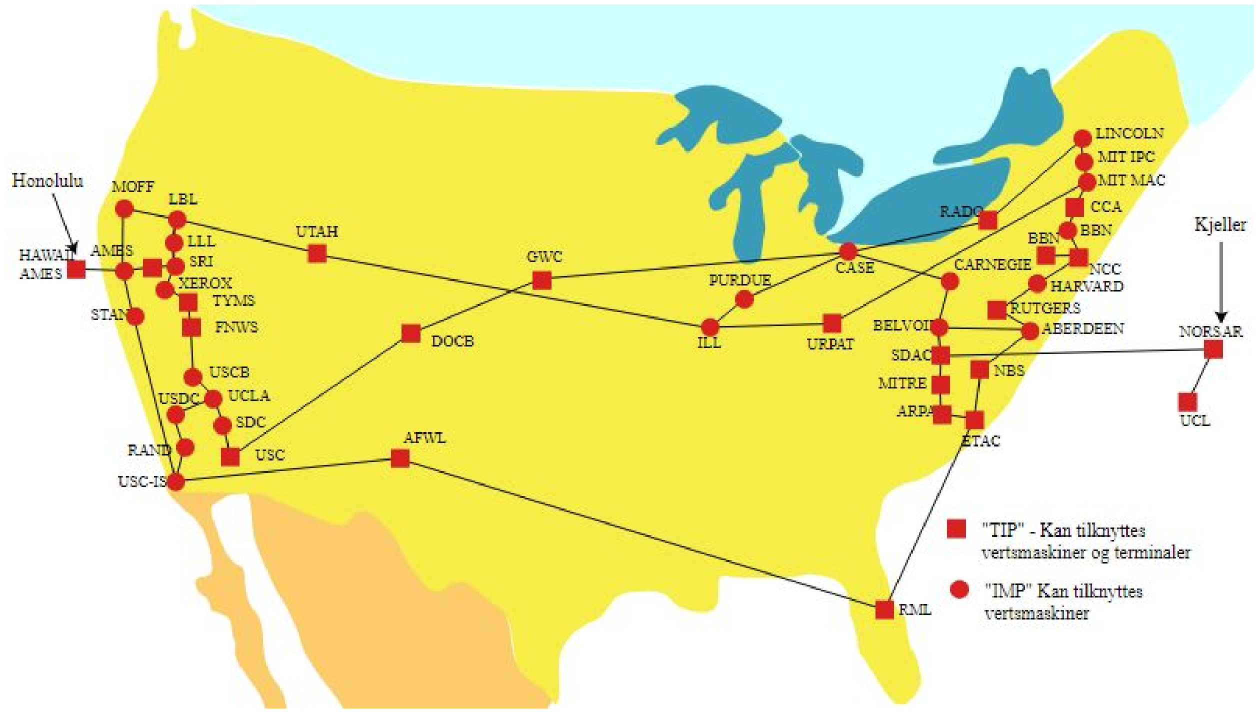
Netcat (often abbreviated to nc) is a computer networking utility for reading from and writing to network connections using TCP or UDP. Netcat is designed to be a dependable back-end that can be used directly or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and investigation tool, since it can produce almost any kind of connection its user could need and has a number of built-in capabilities.

https://en.wikipedia.org/wiki/Netcat
https://www.sans.org/security-resources/sec560/netcat_cheat_sheet_v1.pdf

1. Bring up Firefox in your main Ubuntu VM
2. In the URL bar type in the following but don't hit Enter yet
   a. `http://IP_address_of_Ubuntu_clone:8080`

3. In your Ubuntu Clone VM bring up a terminal window and type the following
   a.
   ```
   { printf 'HTTP/1.0 200 OK\r\nContent-Length: %d\r\n\r\n' "$(wc -c
   < /usr/share/wireshark/rawshark.html)"; cat
   /usr/share/wireshark/rawshark.html; } | nc -l 8080
   ```
   b. hit `Enter`
4. In your Ubuntu VM, hit Enter for the URL you entered
   a. You should see a man page which was served by your temporary netcat webserver

ARPANET, TCP/IP, and DNS



The **Advanced Research Projects Agency Network** (**ARPANET**) was an early packet switching network and the first network to implement the protocol suite TCP/IP. Both technologies became the technical foundation of the Internet.

As the project progressed, protocols for internetworking were developed by which multiple separate networks could be joined into a network of networks. Access to the ARPANET was expanded in 1981 when the National Science Foundation (NSF) funded the Computer Science Network (CSNET). In 1982, the Internet protocol suite (TCP/IP) was introduced as the standard networking protocol on the ARPANET. In the early 1980s the NSF funded the establishment for national supercomputing centers at several universities, and provided interconnectivity in 1986 with the NSFNET project, which also created network access to the supercomputer sites in the United States from research and education organizations. The ARPANET was decommissioned in 1990.

https://en.wikipedia.org/wiki/ARPANET

In the good old days, the mapping between hostname and addresses was kept in a single text file that was managed centrally and distributed to all the hosts on the ARPANET. Hostnames were not hierarchical and the procedure for naming a computer included verify that no one else in the world had taken the name you wanted. Updates consumed a large portion of the ARPANET's bandwidth, and the file was constantly out of date. It soon became clear that although a static host table was reasonable for a small networks it was inadequate for the large and growing ARPANET. DNS solves the problems of a static table by using two key concepts: hierarchical hostnames and distributed responsibility. In 1985, Kevin Dunlap produced the first version of BIND (Berkeley Internet Name Domain system).

The DNS namespace is a tree of domains. Each domain represents a distinct chunk of the namespace and is loosely managed by a single administrative entity. The root of the tree is called ". " or dot, and beneath it are the top-level (or root-level) domains. The top-level domains are relatively fixed.

One branch of the naming tree maps hostnames to IP addresses, and a second branch maps IP addresses back to hostnames. The former branch is called the "forward mapping" and the BIND data files associated with it are called "forward zone files". The address-to-hostname branch is the "reverse mapping" and its data files are called "reverse zone files".

Within the DNS system, fully qualified names are terminated by a dot, for example, "boulder.colorado.edu**.**". The lack of a final dot indicates a relative address. The final dot convention is generally hidden from everyday users of DNS. In fact, some systems (such as mail) will break if you supply the dot yourself. A domain can be subdivided into subdomains, for example, along departmental lines, i.e. anchor.cs.colorado.edu. The creation of subdomains must be coordinated with the administrators of the domain above to guarantee uniqueness.

BIND software

The BIND system has three components:

- a daemon called **named** that answers queries
- library routines that resolve host queries by contacting the servers of the DNS distributed database
- command-line interfaces to DNS: **nslookup**, **dig**, and **host**

named

named answers queries about hostnames and IP addresses. If named doesn't know the answer to a query, it asks other servers and caches their responses. named also performs "zone transfers" to copy data among the servers of a domain. Nameservers deal with zones and a zone is a domain minus its subdomains.

Types of named servers

Type of server	Description
authoritative	An official representative of a zone
master	The primary repository of a zone's data; gets data from a disk file
slave	Copies its data from the master
nonauthoritative	Answers a query from cache; doesn't know if the data is still valid
caching	Caches data from previous queries; usually has no local zones
forwarder	Performs queries on behalf of many clients; builds a large cache
recursive	Queries on your behalf until it returns either an answer or an error
nonrecursive	Refers you to another server if it can't answer a query

Master, slave, and caching-only servers

Master, slave, and caching-only servers are distinguished by two characteristics: where the data comes from and whether the server is authoritative for the domain. Each zone has one master name server. The master keeps the official copy of the zone's data on disk.

A slave server gets its data from the master server through a "zone transfer" operation. A zone can have several slave name servers and ***must*** have at least one. It is fine for the same machine to be both a master server for your zones and a slave server for other zones.

A caching-only name server loads the addresses of the servers for the root domain from a startup file and accumulates the rest of its data by caching answers to the queries it resolves. A caching-only name server has no data of its own and is not authoritative for any zone.

An authoritative answer from a name server is guaranteed to be accurate; a non-authoritative answer might be out of date (but not typically). Master and slave servers are authoritative for their own zones but not for information they have cached about other domains.

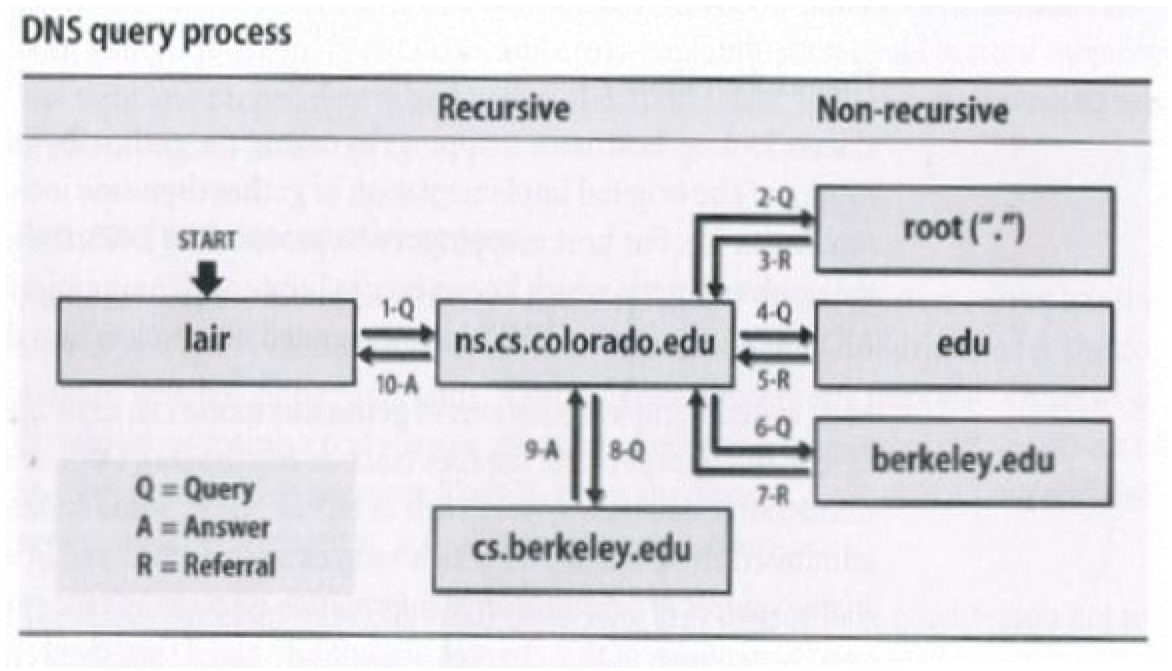
Although they are not authoritative, caching-only servers can reduce the latency seen by your users and the amount of DNS traffic on your internal networks. Consider putting a caching-only server on each subnet for quickly answering user's queries.

Recursive and nonrecursive servers

Name servers are either recursive or nonrecursive. If a **nonrecursive** server has the answer to a query cached from a previous transaction or is authoritative for the domain to which the query pertains, it provides an appropriate response. Otherwise, instead of returning a real answer, it returns a referral to the authoritative servers of another domain that are more likely to know the answer. Root servers and top-level domain servers are all nonrecursive because of the number of queries they receive.

A **recursive** server returns only real answers or error messages. It follows referrals itself, relieving the client of this responsibility.

DNS Query Process



Suppose we want to look up the address for vangogh.cs.berkeley.edu from lair.cs.colorado.edu. We assume that none of the required information has been cached before the query except for the names and IP addresses of the root servers.

1. The host lair asks its local name server ns.cs.colorado.edu for the answer.
2. The local name server ns.cs.colorado.edu (which is a recursive server) doesn't know anything about cs.berkeley.edu, berkeley.edu, or even edu so it queries a root server about vangogh.cs.berkeley.edu.
3. The root server refers us to the servers for the edu domain
4. ns.cs.colorado.edu asks the edu server about vangogh.cs.berkeley.edu

5. The edu servers refer us to the berkeley.edu servers
6. ns.cs.colorado.edu asks the berkeley.edu servers about vangogh.cs.berkeley.edu
7. The berkeley.edu servers refer us to the cs.berkeley.edu servers
8. ns.cs.colorado.edu asks the cs.berkeley.edu servers about vangogh.cs.berkeley.edu
9. The cs.berkeley.edu servers return us the IP address of vangogh.cs.berkeley.edu
10. ns.cs.colorado.edu returns the IP address of vangogh.cs.berkeley.edu to lair

Resolver configuration

Each host on the network has a file called /etc/resolv.conf that lists the DNS server the host should query. The format is as follows:

search *domainname* *(up to eight space separated domainnames)*
nameserver *IPaddr* *(up to three lines)*

Example /etc/resolv.conf

```
search cs.colorado.edu colorado.edu ee.colorado.edu
nameserver 128.138.243.151      ; ns
nameserver 128.138.204.4        ; piper
nameserver 128.138.240.1        ; anchor
```

The search line lists the domains to query if a hostname is not fully qualified. The nameservers listed in `/etc/resolv.conf` must be recursive since the resolver client on your workstation does not understand referrals. The nameservers are contacted in order and if the first one continues answering queries the others are ignored.

The file `/etc/nsswitch.conf` has a "hosts" line like so:

```
hosts:     files dns
```

This means that when your system is trying to resolve a name to IP address mapping it will first look in `/etc/hosts` and if the mapping is not in the hosts file it will look in DNS for the resolution.

Reference: UNIX System Administration Handbook, Nemeth, Snyder, Seebass, and Hein

named Server Configuration

1. Configure DNS on glenn-X60 (Ubuntu 16.0.4 system)
    a. `apt-get update`
    b. `apt-get upgrade`
    c. `apt-get install bind9 bind9utils bind9-doc`

d. `vi /etc/bind/named.conf.options`

```
forwarders {
    8.8.8.8;
};
```

e. `systemctl restart bind9`

f. `vi /etc/bind/named.conf`

```
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
```

g. `vi /etc/bind/named.conf.local`

```
zone "linuxlab.local" {
        type master;
        file "/etc/bind/forward.linuxlab.local";
        allow-update { none; };
};

zone "0.0.10.in-addr.arpa" {
        type master;
        file "/etc/bind/reverse.linuxlab.local";
        allow-update { none; };
};
```

h. `vi /etc/bind/forward.linuxlab.local`

```
$TTL 86400
@   IN  SOA     glenn-X60.linuxlab.local.
root.linuxlab.local. (
        2011071001  ;Serial
        3600        ;Refresh
        1800        ;Retry
        604800      ;Expire
        86400       ;Minimum TTL
)
@       IN  NS          glenn-X60.linuxlab.local.
RonCentOS       IN      A       10.0.0.101
ron             IN      CNAME   RonCentOS
AndrewCentOS    IN      A       10.0.0.102
andrew          IN      CNAME   AndrewCentOS
ckHost          IN      A       10.0.0.103
chris           IN      CNAME   ckHost
chev            IN      A       10.0.0.104
chaveal         IN      CNAME   chev
```

```
MarkVM          IN      A       10.0.0.105
mark            IN      CNAME   MarkVM
bobbyCentOS     IN      A       10.0.0.106
bobby           IN      CNAME   bobbyCentOS
neutron         IN      A       10.0.0.199
nova1           IN      A       10.0.0.201
nova2           IN      A       10.0.0.202
cinder          IN      A       10.0.0.203
glenn-X60       IN      A       10.0.0.204
```

i. `$TTL` — Sets the default *Time to Live (TTL)* value for the zone. This is the length of time, in seconds, a zone resource record is valid.
ii. The `@` symbol places the zone's name as the namespace being defined by this `SOA` resource record. The hostname of the primary nameserver that is authoritative for this domain is the <*primary-name-server*> directive, and the email of the person to contact about this namespace is the <*hostmaster-email*> directive.
iii. `NS` — NameServer record, which announces the authoritative nameservers for a particular zone.
iv. `A` — Address record, which specifies an IP address to assign to a name
v. `CNAME` - Canonical Name, an alias referring to the Address record

i. `vi /etc/bind/reverse.linuxlab.local`

```
$TTL 86400
@   IN  SOA     glenn-X60.linuxlab.local. root.linuxlab.local. (
        2011071002  ;Serial
        3600        ;Refresh
        1800        ;Retry
        604800      ;Expire
        86400       ;Minimum TTL
)
@       IN  NS          glenn-X60.linuxlab.local.
@       IN  PTR         linuxlab.local.

RonCentOS       IN      A       10.0.0.101
AndrewCentOS    IN      A       10.0.0.102
ckHost          IN      A       10.0.0.103
chev            IN      A       10.0.0.104
MarkVM          IN      A       10.0.0.105
bobbyCentOS     IN      A       10.0.0.106
neutron         IN      A       10.0.0.199
nova1           IN      A       10.0.0.201
nova2           IN      A       10.0.0.202
cinder          IN      A       10.0.0.203
glenn-X60       IN      A       10.0.0.204

101     IN      PTR     RonCentOS.linuxlab.local.
102     IN      PTR     AndrewCentOS.linuxlab.local.
```

```
103     IN      PTR     ckHost.linuxlab.local.
104     IN      PTR     chev.linuxlab.local.
105     IN      PTR     MarkVM.linuxlab.local.
106     IN      PTR     bobbyCentoOS.linuxlab.local.
199     IN      PTR     neutron.linuxlab.local.
201     IN      PTR     nova1.linuxlab.local.
202     IN      PTR     nova2.linuxlab.local.
203     IN      PTR     cinder.linuxlab.local.
204     IN      PTR     glenn-X60.linuxlab.local.
```

j. `chmod -R 755 /etc/bind`

k. `chown -R bind:bind /etc/bind`

l. `named-checkconf /etc/bind/named.conf`

m. `named-checkconf /etc/bind/named.conf.local`

n.
```
named-checkzone linuxlab.local
/etc/bind/forward.linuxlab.local
zone linuxlab.local/IN: loaded serial 2011071001
OK
```

o.
```
named-checkzone linuxlab.local
/etc/bind/reverse.linuxlab.local
zone linuxlab.local/IN: loaded serial 2011071002
OK
```

p. `vi /etc/network/interfaces`

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
dns-nameservers 10.0.0.204
```

q.
```
echo "nameserver 10.0.0.204
search linuxlab.local" | resolvconf -a  ens2.inet
```

r. on the CentOS VMs edit the `/etc/resolv.conf` file

   i. `vi /etc/resolv.conf`

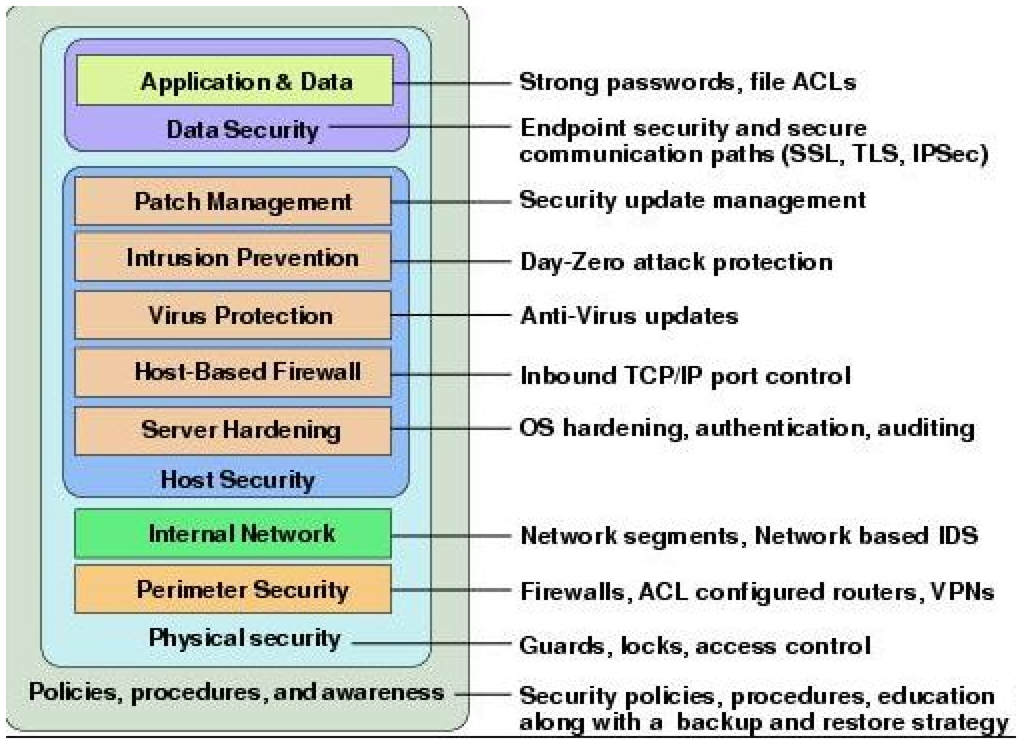
```
nameserver 10.0.0.204
search linuxlab.local
```

DoS/DDoS and Network Security

Defense in Depth

Defenses are deployed in a layered fashion. What would defense in depth look like in a corporate setting or your school setting?

Here is an example of defense in depth


Application & Data
Data Security
Patch Management
Intrusion Prevention
Virus Protection
Host-Based Firewall
Server Hardening
Host Security
Internal Network
Perimeter Security
Physical security
Policies, procedures, and awareness
Strong passwords, file ACLs
Endpoint security and secure communication paths (SSL, TLS, IPSec)
Security update management
Day-Zero attack protection
Anti-Virus updates
Inbound TCP/IP port control
OS hardening, authentication, auditing
Network segments, Network based IDS
Firewalls, ACL configured routers, VPNs
Guards, locks, access control
Security policies, procedures, education along with a backup and restore strategy

https://www.pinterest.com/pin/96616354489566017/

Discuss each layer of defense.

CIA can be affected in ways not related to hacking. What are some ways?

Attacks

Land attack - nmap a victim server to find a listening port. Then send a SYN to the target with the source:port and destination:port set to the victim (spoofed). This would cause the server to enter into a loop and eventually lock up. Describe the packet mechanics.
Ping of death - Early implementations of TCP/IP were written to only expect ICMP ping messages to be 32 bytes in length. A large ping (ICMP) packet would be created by the hacker and then sent to the target system. If the target system couldn't handle a large packet it would cause a buffer overflow condition in the kernel and thus crash the system.

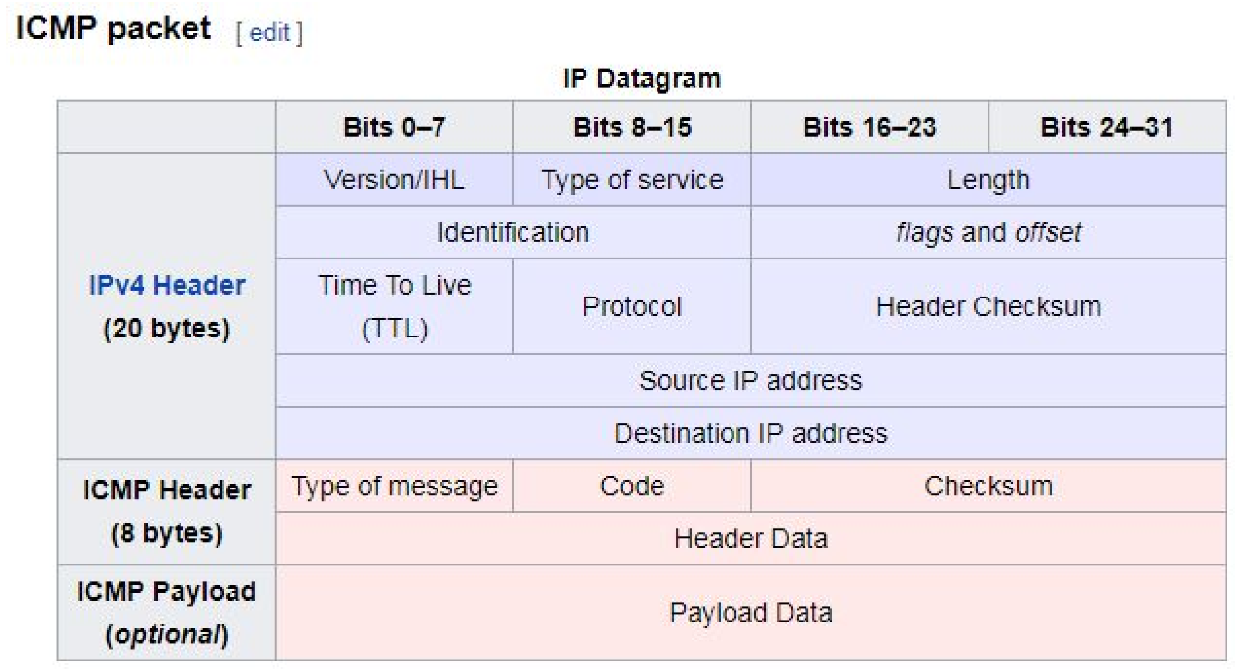
What is a buffer overflow?

What does it mean for a system to crash?

Infected USB - what would the average person do if they found a really nice new USB drive? What if it was strategically place in the path of a person who had a system we wanted to target?

Smurf attack - this is a network distributed denial of service (DDOS) attack. An attacker gets the IP address of the system he wants to DDOS. He then sends an ICMP Echo Request (a ping) to the broadcast address on the LAN segment. (Why only on the LAN segment?) with the victim's IP address in the source field. What happens next?

Contents of a ping packet



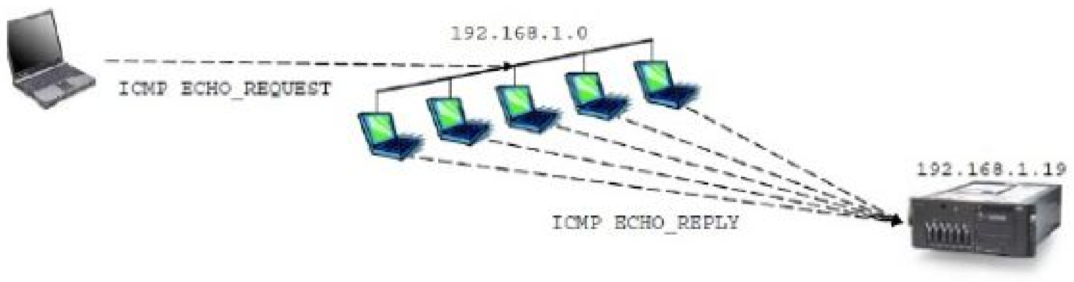
ICMP packet [edit]

IP Datagram

	Bits 0–7	**Bits 8–15**	**Bits 16–23**	**Bits 24–31**
IPv4 Header (20 bytes)	Version/IHL	Type of service	Length	
	Identification		*flags* and *offset*	
	Time To Live (TTL)	Protocol	Header Checksum	
	Source IP address			
	Destination IP address			
ICMP Header (8 bytes)	Type of message	Code	Checksum	
	Header Data			
ICMP Payload (*optional*)	Payload Data			

Smurf attack

- A smurf attack is an exploitation of the Internet Protocol (IP) broadcast addressing to create a denial of service.



https://www.slideshare.net/null0x00/dos-threats-and-countermeasures-12990644

Note: Smurf attack is no longer effective because implementers of TCP/IP figured out that it doesn't make sense to reply to an ICMP which goes to the broadcast address.

SYN flood - an attacker uses a bot-net to initiate a bunch of SYN messages. The flood of SYN messages will fill up the connections buffer on the victim machine and it won't accept anymore connections, thus accomplishing a DDOS attack. Each bot-node will send a bunch of SYN requests but each request will have a different destination port.

What is a bot-net?

Cross Site Request Forgery - requirements

1. The victim host has a current session cookie to a high value webserver
2. The victim (person) can be enticed to click on a link which points to a webserver with a page that has an image tag <IMG> in the HTML which points to the high-value website and when the victim's browser tries to load what it thinks is an image, the bad code gets executed on the high-value website using the cached (still active) credentials.

What would a XSRF attack look like?

Defenses

HIDS, NIDS, host-NIDS, HIPS

Vulnerability Scanners - Qualys, OpenVAS. Authenticated and unauthenticated scans. Do an OpenVAS scan using kali.
www.sectools.org

Honeypots - systems which are deliberately designed to attract malicious traffic. They offer fake services and fake data. The attack attempts can be recorded and analysed by the white-hats to learn more about how the black-hats are attacking them.

Server hardening - **hardening** is usually the process of securing a system by reducing its surface of vulnerability, which is larger when a system performs more functions; in principle a single-function system is more secure than a multipurpose one. Reducing available ways of attack typically includes changing default passwords, the removal of unnecessary software, unnecessary usernames or logins, and the disabling or removal of unnecessary services
https://en.wikipedia.org/wiki/Hardening_(computing)

CVE database - https://cve.mitre.org/

Mapping your network on a regular basis will keep you informed of what is on your network and help to spot anomalies. Explore the different types of scans offered by nmap.
https://nmap.org/book/man.html

ARP requests can be used to map hosts in your broadcast domain. When you want to connect to an IP address on your subnet but your system doesn't have the MAC address in its ARP cache, your system will send out an ARP "Who has" message. If we can just send out "Who has" messages to every possible IP in the host ID range we can get responses back from each active node and thus see the active hosts on our subnet.

Firewalls

Firewalls are an important tool that can be configured to protect your servers and infrastructure. In the Linux ecosystem, iptables is a widely used firewall tool that interfaces with the kernel's netfilter packet filtering framework.

What are iptables and netfilter?

The basic firewall software most commonly used in Linux is called iptables. The iptables firewall works by interacting with the packet filtering hooks in the Linux kernel's networking stack. These kernel hooks are known as the netfilter framework.

Every packet that enters networking system (incoming or outgoing) will trigger these hooks as it progresses through the stack, allowing programs that register with these hooks to interact with the traffic at key points. The kernel modules associated with iptables register at these hooks in order to ensure that the traffic conforms to the conditions laid out by the firewall rules.

```
locate netfilter | grep iptable_
/lib/modules/4.8.0-53-generic/kernel/net/ipv4/netfilter/iptable_filter.ko
/lib/modules/4.8.0-53-generic/kernel/net/ipv4/netfilter/iptable_mangle.ko
/lib/modules/4.8.0-53-generic/kernel/net/ipv4/netfilter/iptable_nat.ko
/lib/modules/4.8.0-53-generic/kernel/net/ipv4/netfilter/iptable_raw.ko
/lib/modules/4.8.0-53-generic/kernel/net/ipv4/netfilter/iptable_security.ko
```

Netfilter Hooks

There are five netfilter hooks that programs can register with. As packets progress through the stack, they will trigger the kernel modules that have registered with these hooks. The hooks that a packet will trigger depends on whether the packet is incoming or outgoing, the packet's destination, and whether the packet was dropped or rejected at a previous point.

The following hooks represent various well-defined points in the networking stack:

- NF_IP_PRE_ROUTING: This hook will be triggered by any incoming traffic very soon after entering the network stack. This hook is processed before any routing decisions have been made regarding where to send the packet.
- NF_IP_LOCAL_IN: This hook is triggered after an incoming packet has been routed if the packet is destined for the local system.
- NF_IP_FORWARD: This hook is triggered after an incoming packet has been routed if the packet is to be forwarded to another host.
- NF_IP_LOCAL_OUT: This hook is triggered by any locally created outbound traffic as soon it hits the network stack.
- NF_IP_POST_ROUTING: This hook is triggered by any outgoing or forwarded traffic after routing has taken place and just before being put out on the wire.

Kernel modules that wish to register at these hooks must provide a priority number to help determine the order in which they will be called when the hook is triggered. This provides the means for multiple modules (or multiple instances of the same module) to be connected to each of the hooks with deterministic ordering. Each module will be called in turn and will return a decision to the netfilter framework after processing that indicates what should be done with the packet.

iptables tables and chains

The iptables firewall uses tables to organize its rules. These tables classify rules according to the type of decisions they are used to make. For instance, if a rule deals with network address translation, it will be put into the nat table. If the rule is used to decide whether to allow the packet to continue to its destination, it would probably be added to the filter table.

Within each iptables table, rules are further organized within separate "chains". While tables are defined by the general aim of the rules they hold, the built-in chains represent the netfilter hooks which trigger them. Chains basically determine *when* rules will be evaluated.

As you can see, the names of the built-in chains mirror the names of the netfilter hooks they are associated with:

- PREROUTING: Triggered by the NF_IP_PRE_ROUTING hook.
- INPUT: Triggered by the NF_IP_LOCAL_IN hook.
- FORWARD: Triggered by the NF_IP_FORWARD hook.
- OUTPUT: Triggered by the NF_IP_LOCAL_OUT hook.
- POSTROUTING: Triggered by the NF_IP_POST_ROUTING hook.

Chains allow the administrator to control where in a packet's delivery path a rule will be evaluated. Since each table has multiple chains, a table's influence can be exerted at multiple points in processing. Because certain types of decisions only make sense at certain points in the network stack, every table will not have a chain registered with each kernel hook.

There are only five netfilter kernel hooks, so chains from multiple tables are registered at each of the hooks. For instance, three tables have PREROUTING chains. When these chains register at the associated NF_IP_PRE_ROUTING hook, they specify a priority that dictates what order each table's PREROUTING chain is called. Each of the rules inside the highest priority PREROUTING chain is evaluated sequentially before moving onto the next PREROUTING chain. We will take a look at the specific order of each chain in a moment.

Which Tables are Available?

Let's step back for a moment and take a look at the different tables that iptables provides. These represent distinct sets of rules, organized by area of concern, for evaluating packets.

The Filter Table

The filter table is one of the most widely used tables in iptables. The filter table is used to make decisions about whether to let a packet continue to its intended destination or to deny its request. In firewall parlance, this is known as "filtering" packets. This table provides the bulk of functionality that people think of when discussing firewalls.

The NAT Table

The nat table is used to implement network address translation rules. As packets enter the network stack, rules in this table will determine whether and how to modify the packet's source or destination addresses in order to impact the way that the packet and any response traffic are routed. This is often used to route packets to networks when direct access is not possible.

The Mangle Table

The mangle table is used to alter the IP headers of the packet in various ways. For instance, you can adjust the TTL (Time to Live) value of a packet, either lengthening or shortening the number of valid network hops the packet can sustain. Other IP headers can be altered in similar ways.

This table can also place an internal kernel "mark" on the packet for further processing in other tables and by other networking tools. This mark does not touch the actual packet, but adds the mark to the kernel's representation of the packet.

The Raw Table

The iptables firewall is stateful, meaning that packets are evaluated in regards to their relation to previous packets. The connection tracking features built on top of the netfilter framework allow iptables to view packets as part of an ongoing connection or session instead of as a stream of discrete, unrelated packets. The connection tracking logic is usually applied very soon after the packet hits the network interface.

The raw table has a very narrowly defined function. Its only purpose is to provide a mechanism for marking packets in order to opt-out of connection tracking.

The Security Table

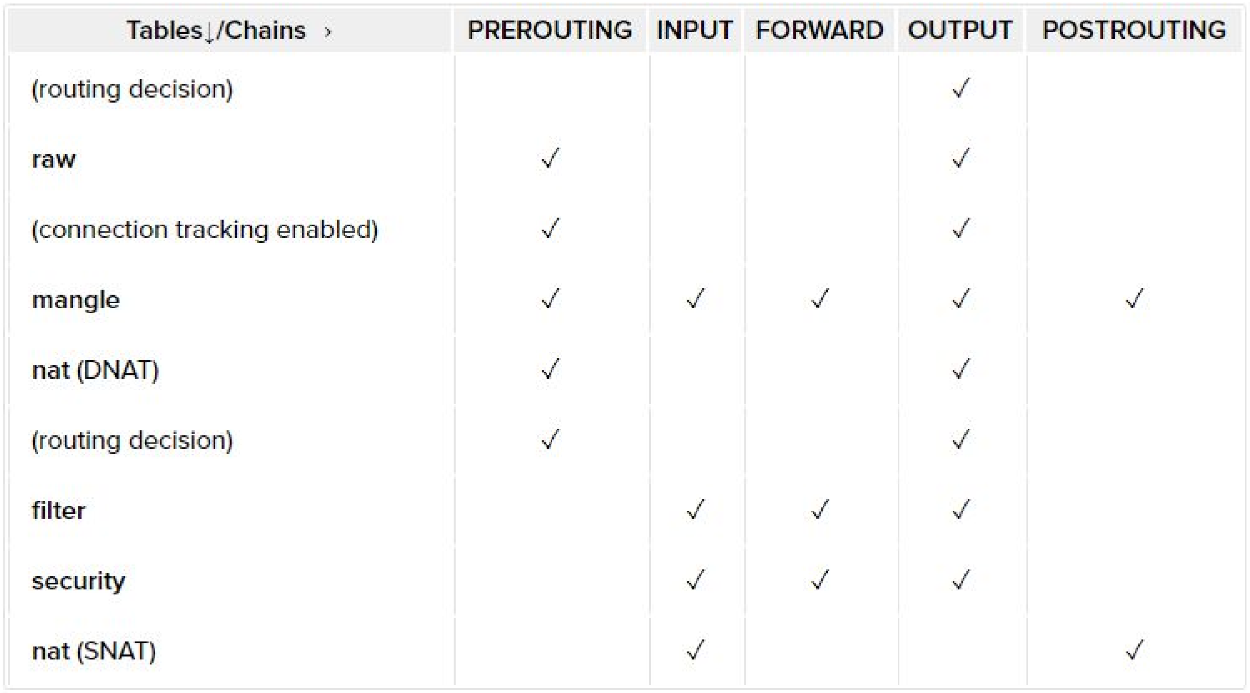
The security table is used to set internal SELinux security context marks on packets, which will affect how SELinux or other systems that can interpret SELinux security contexts handle the packets. These marks can be applied on a per-packet or per-connection basis.

Which Chains are Implemented in Each Table?

We have talked about tables and chains separately. Let's go over which chains are available in each table. Implied in this discussion is a further discussion about the evaluation order of chains registered to the same hook. If three tables have PREROUTING chains, in which order are they evaluated?

The following table indicates the chains that are available within each iptables table when read from left-to-right. For instance, we can tell that the raw table has both PREROUTING and OUTPUT chains. When read from top-to-bottom, it also displays the order in which each chain is called when the associated netfilter hook is triggered.

A few things should be noted. In the representation below, the nat table has been split between DNAT operations (those that alter the destination address of a packet) and SNAT operations (those that alter the source address) in order to display their ordering more clearly. We have also include rows that represent points where routing decisions are made and where connection tracking is enabled in order to give a more holistic view of the processes taking place:



Tables↓/Chains ›	PREROUTING	INPUT	FORWARD	OUTPUT	POSTROUTING
(routing decision)				✓	
raw	✓			✓	
(connection tracking enabled)	✓			✓	
mangle	✓	✓	✓	✓	✓
nat (DNAT)	✓			✓	
(routing decision)	✓			✓	
filter		✓	✓	✓	
security		✓	✓	✓	
nat (SNAT)		✓			✓

As a packet triggers a netfilter hook, the associated chains will be processed as they are listed in the table above from top-to-bottom. The hooks (columns) that a packet will trigger depend on whether it is an incoming or outgoing packet, the routing decisions that are made, and whether the packet passes filtering criteria.

Certain events will cause a table's chain to be skipped during processing. For instance, only the first packet in a connection will be evaluated against the NAT rules. Any nat decisions made for the first packet will be applied to all subsequent packets in the connection without additional evaluation. Responses to NAT'ed connections will automatically have the reverse NAT rules applied to route correctly.

Chain Traversal Order

Assuming that the server knows how to route a packet and that the firewall rules permit its transmission, the following flows represent the paths that will be traversed in different situations:

- Incoming packets destined for the local system: PREROUTING -> INPUT
- Incoming packets destined to another host: PREROUTING -> FORWARD -> POSTROUTING
- Locally generated packets: OUTPUT -> POSTROUTING

If we combine the above information with the ordering laid out in the previous table, we can see that an incoming packet destined for the local system will first be evaluated against the PREROUTING chains of the raw, mangle, and nat tables. It will then traverse the INPUT chains of the mangle, filter, security, and nat tables before finally being delivered to the local socket.

IPTables Rules

Rules are placed within a specific chain of a specific table. As each chain is called, the packet in question will be checked against each rule within the chain in order. Each rule has a matching component and an action component.

Matching

The matching portion of a rule specifies the criteria that a packet must meet in order for the associated action (or "target") to be executed.

The matching system is very flexible and can be expanded significantly with iptables extensions available on the system. Rules can be constructed to match by protocol type, destination or source address, destination or source port, destination or source network, input or output interface, headers, or connection state among other criteria. These can be combined to create fairly complex rule sets to distinguish between different traffic.

Targets

A target is the action that are triggered when a packet meets the matching criteria of a rule. Targets are generally divided into two categories:

- Terminating targets: Terminating targets perform an action which terminates evaluation within the chain and returns control to the netfilter hook. Depending on the return value provided, the hook might drop the packet or allow the packet to continue to the next stage of processing.
- Non-terminating targets: Non-terminating targets perform an action and continue evaluation within the chain. Although each chain must eventually pass back a final termination decision, any number of non-terminating targets can be executed beforehand.

The availability of each target within rules will depend on context. For instance, the table and chain type might dictate the targets available. The extensions activated in the rule and the matching clauses can also affect the availability of targets.

Jumping to User-Defined Chains

We should mention a special class of non-terminating target: the jump target. Jump targets are actions that result in evaluation moving to a different chain for additional processing. We've talked quite a bit about the built-in chains which are intimately tied to the netfilter hooks that call them. However, iptables also allows administrators to create their own chains for organizational purposes.

Rules can be placed in user-defined chains in the same way that they can be placed into built-in chains. The difference is that user-defined chains can only be reached by "jumping" to them from a rule (they are not registered with a netfilter hook themselves).

User-defined chains act as simple extensions of the chain which called them. For instance, in a user-defined chain, evaluation will pass back to the calling chain if the end of the rule list is reached or if a RETURN target is activated by a matching rule. Evaluation can also jump to additional user-defined chains.

This construct allows for greater organization and provides the framework necessary for more robust branching.

IPTables and Connection Tracking

We introduced the connection tracking system implemented on top of the netfilter framework when we discussed the raw table and connection state matching criteria. Connection tracking allows iptables to make decisions about packets viewed in the context of an ongoing connection. The connection tracking system provides iptables with the functionality it needs to perform "stateful" operations.

Connection tracking is applied very soon after packets enter the networking stack. The raw table chains and some basic sanity checks are the only logic that is performed on packets prior to associating the packets with a connection.

The system checks each packet against a set of existing connections. It will update the state of the connection in its store if needed and will add new connections to the system when necessary. Packets that have been marked with the NOTRACK target in one of the raw chains will bypass the connection tracking routines.

Available States

Connections tracked by the connection tracking system will be in one of the following states:

- NEW: When a packet arrives that is not associated with an existing connection, but is not invalid as a first packet, a new connection will be added to the system with this label. This happens for both connection-aware protocols like TCP and for connectionless protocols like UDP.
- ESTABLISHED: A connection is changed from NEW to ESTABLISHED when it receives a valid response in the opposite direction. For TCP connections, this means a SYN/ACK and for UDP and ICMP traffic, this means a response where source and destination of the original packet are switched.
- RELATED: Packets that are not part of an existing connection, but are associated with a connection already in the system are labeled RELATED. This could mean a helper connection, as is the case with FTP data transmission connections, or it could be ICMP responses to connection attempts by other protocols.
- INVALID: Packets can be marked INVALID if they are not associated with an existing connection and aren't appropriate for opening a new connection, if they cannot be identified, or if they aren't routable among other reasons.
- UNTRACKED: Packets can be marked as UNTRACKED if they've been targeted in a raw table chain to bypass tracking.
- SNAT: A virtual state set when the source address has been altered by NAT operations. This is used by the connection tracking system so that it knows to change the source addresses back in reply packets.
- DNAT: A virtual state set when the destination address has been altered by NAT operations. This is used by the connection tracking system so that it knows to change the destination address back when routing reply packets.

The states tracked in the connection tracking system allow administrators to craft rules that target specific points in a connection's lifetime. This provides the functionality needed for more thorough and secure rules.

Conclusion

The netfilter packet filtering framework and the iptables firewall are the basis for most firewall solutions on Linux servers. The netfilter kernel hooks are close enough to the networking stack to provide powerful control over packets as they are processed by the system. The iptables firewall leverages these capabilities to provide a flexible, extensible method of communicating policy requirements to the kernel. By learning about how these pieces fit together, you can better utilize them to control and secure your server environments.

https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture

iptables

iptables is a user-space utility program that allows a system administrator to configure the tables provided by the Linux kernel firewall (implemented as different Netfilter modules) and the chains and rules the tables stores. Different kernel modules and programs are currently used for different protocols; iptables applies to IPv4, ip6tables to IPv6, arptables to ARP, and ebtables to Ethernet frames.

https://en.wikipedia.org/wiki/Iptables

Iptables are used to set up, maintain, and inspect the tables of IPv4 packet filter rules in *Netfilter Framework* within the Linux kernel. Several different tables may be defined. Each table contains a number of built-in chains and may also contain user-defined chains.

Each chain is a list of rules which can match a set of packets. Each rule specifies what to do with a packet that matches. This is called a *target*, which may be a jump to a user-defined chain in the same table.

- firewall tables are defined in the kernel
    - tables contain built-in and user-defined chains
        - chains contains lists of rules which match packets
            - rules have actions for packets which match and this action is called the target

Tables → Chains → Rules

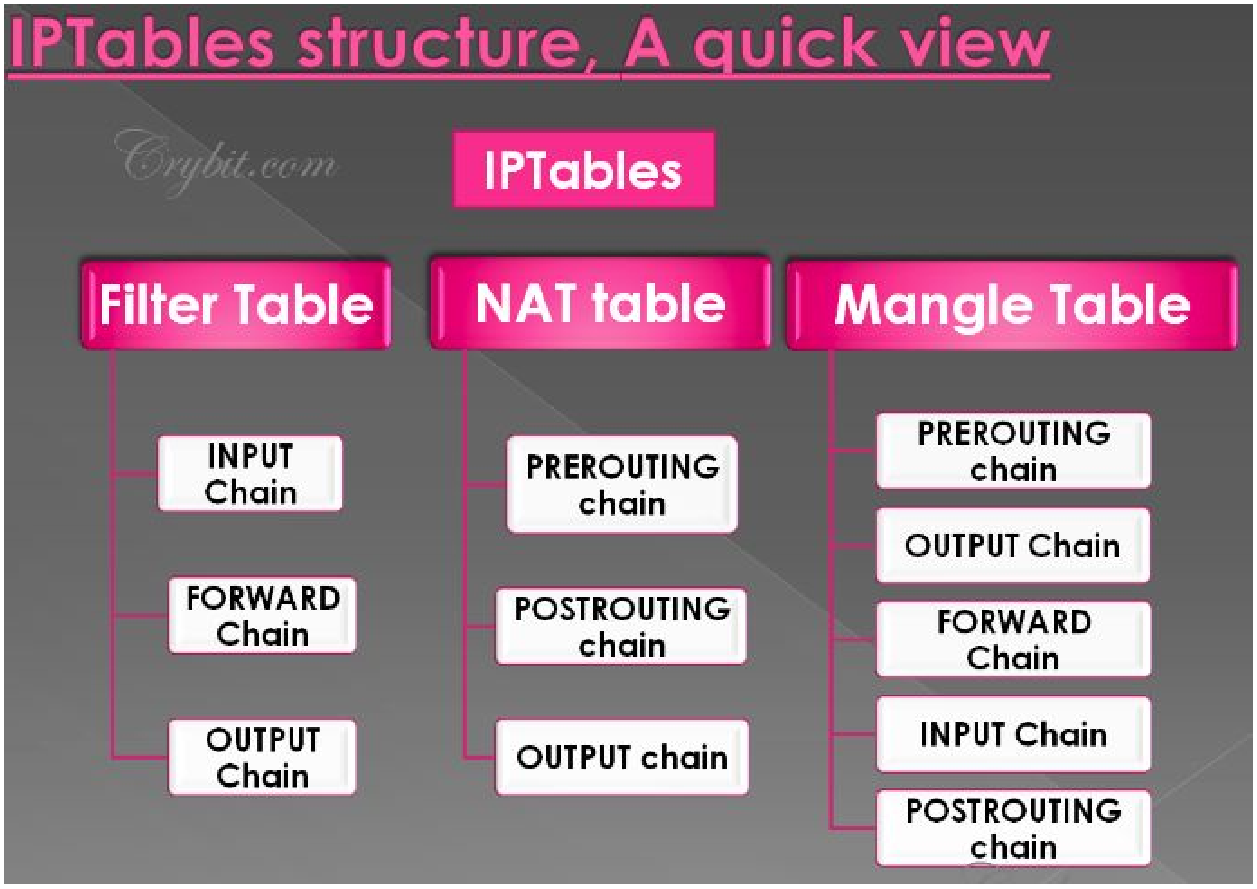
The rules are defined to control the packets for Input and Output

Crybit.com
Tables
Chains
Rules

Five default tables

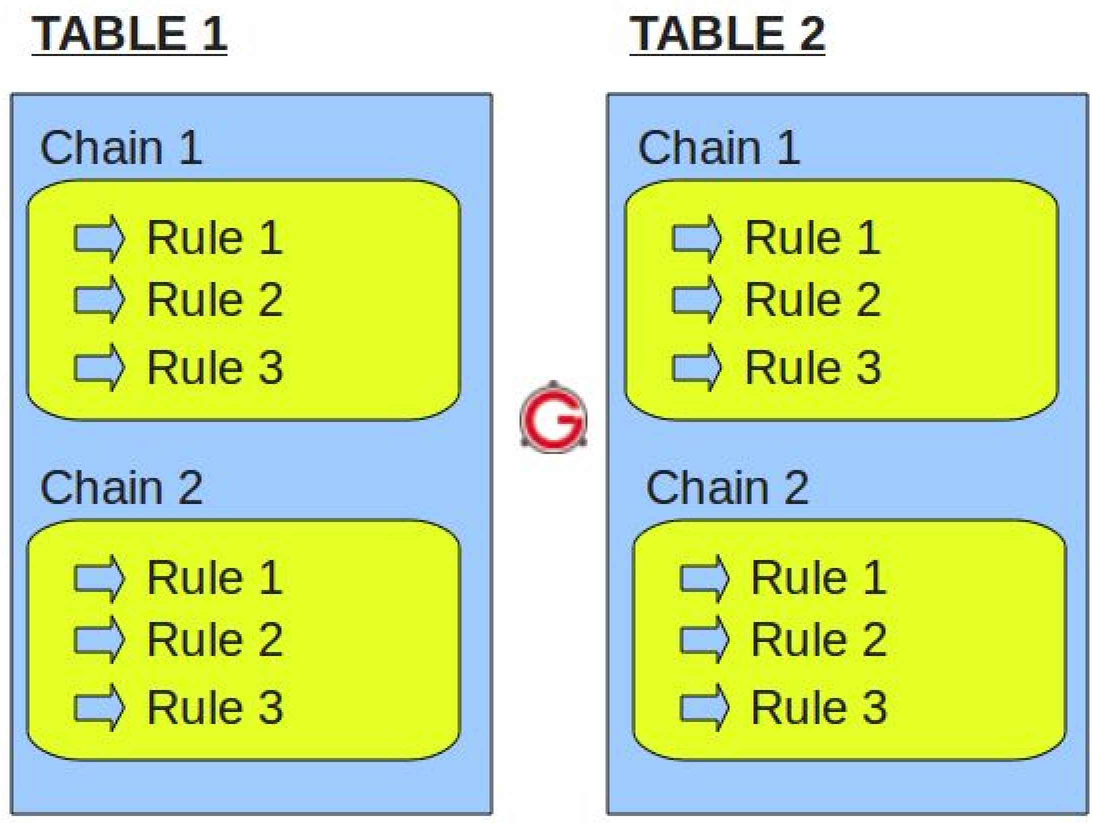
The three default tables are Filter, NAT, and Mangle.

- The Filter table is the default table
- The NAT table provides NAT and related functions
- The Mangle table is used when the packet will be modified by the firewall
- The Raw table is used to provide a mechanism for marking packets in order to opt-out of connection tracking.
- The Security table is used to set internal SELinux security context marks on packets, which will affect how SELinux or other systems that can interpret SELinux security contexts handle the packets.

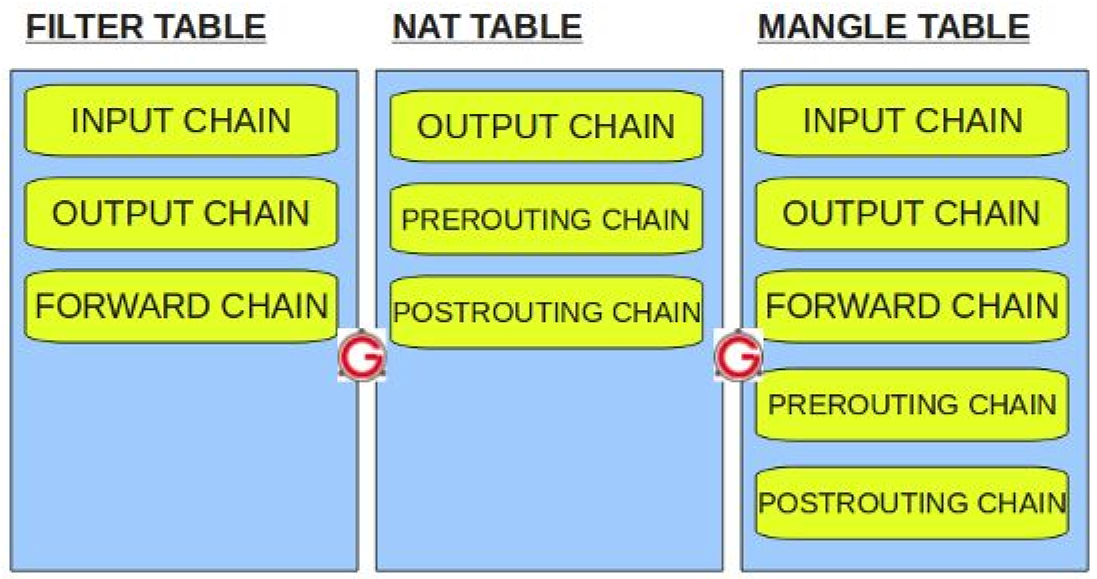


The basic syntax for an iptables command begins with the iptables command itself, followed by one or more options, a chain, a set of match criteria, and a target or disposition. The layout of the command largely depends on the action to be performed. Consider this syntax:

```
iptables <option> <chain> <matching criteria> <target>
```



The following diagram shows the three importants tables in iptables



The Filter table

This table is the default. If a rule does not use the -t <table name> option the filter table will be implied.

- *Chains* available are INPUT, OUTPUT, and FORWARD
- *Matching Criteria* include IP header protocol, source and destination addresses, input and output interfaces, fragment handling, MAC source and destination addresses, ICMP type, length of packet, and others
- *Targets* available are ACCEPT, DROP, REJECT, BALANCE, CLUSTERIP and others

The NAT table

The nat table contains the rules for Source and Destination Address and Port Translation. These rules are functionally distinct from the firewall filter rules. The built in *chains* include:

- PREROUTING (DNAT/REDIRECT)
- OUTPUT (DNAT/REDIRECT)
- POSTROUTING (SNAT/MASQUERADE)

The Mangle table

The mangle table contains rules for setting specialized packet-routing flags. These flags are then inspected later by rules in the filter table. The built-in *chains* include:

- PREROUTING (routed packets)
- INPUT (packets arriving at the firewall but after the PREROUTING chain)
- FORWARD (changes packets being routed through the firewall)
- POSTROUTING (changes packets just before they leave the firewall, after the OUTPUT chain)
- OUTPUT (locally generated packets)

In-class discussion 1

When would it be important to configure the NAT table rules? What kind of configuration would you apply the NAT table rules to?

In-class discussion 2

Set Default Chain Policies

The default chain policy is ACCEPT. Change this to DROP for all INPUT, FORWARD, and OUTPUT chains as shown below.

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
```

When you make both INPUT, and OUTPUT chain's default policy as DROP, for every firewall rule requirement you have, you should define two rules. i.e one for incoming and one for outgoing.
In all our examples below, we have two rules for each scenario, as we've set DROP as default policy for both INPUT and OUTPUT chain.
If you trust your internal users, you can omit the last line above. i.e Do not DROP all outgoing packets by default. In that case, for every firewall rule requirement you have, you just have to define only one rule. i.e define rule only for incoming, as the outgoing is ACCEPT for all packets.

Flush iptables rules

```
iptables -F
```

Block a specific IP address

```
BLOCK_THIS_IP="x.x.x.x"
iptables -A INPUT -s "$BLOCK_THIS_IP" -j DROP
iptables -A INPUT -i eth0 -s "$BLOCK_THIS_IP" -j DROP
iptables -A INPUT -i eth0 -p tcp -s "$BLOCK_THIS_IP" -j DROP
```

Allow ALL incoming SSH

```
iptables -t filter -A INPUT -i eth0 -p tcp --dport 22 -m state --state
NEW,ESTABLISHED -J ACCEPT
iptables -t filter -A OUTPUT -o eth0 -p tcp --sport 22 -m state --state
ESTABLISHED -j ACCEPT
```

Allow incoming SSH only from a specific network

```
iptables -A INPUT -i eth0 -p tcp -s 192.168.100.0/24 --dport 22 -m state
--state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 22 -m state --state ESTABLISHED -j
ACCEPT
```

Allow incoming HTTP and HTTPS

```
iptables -A INPUT -i eth0 -p tcp --dport 80 -m state --state NEW,ESTABLISHED
-j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 80 -m state --state ESTABLISHED -j
ACCEPT
iptables -A INPUT -i eth0 -p tcp --dport 443 -m state --state NEW,ESTABLISHED
-j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 443 -m state --state ESTABLISHED -j
ACCEPT
```

Combine multiple rules together using MultiPorts

When you are allowing incoming connections from outside world to multiple ports, instead of writing individual rules for each and every port, you can combine them together using the multiport extension as shown below.

The following example allows all incoming SSH, HTTP and HTTPS traffic.

```
iptables -A INPUT -i eth0 -p tcp -m multiport --dports 22,80,443 -m state
--state NEW,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp -m multiport --sports 22,80,443 -m state
--state ESTABLISHED -j ACCEPT
```

Allow Outgoing SSH

The following rules allow outgoing ssh connection. i.e When you ssh from inside to an outside server.

```
iptables -A OUTPUT -o eth0 -p tcp --dport 22 -m state --state NEW,ESTABLISHED
-j ACCEPT
iptables -A INPUT -i eth0 -p tcp --sport 22 -m state --state ESTABLISHED -j
ACCEPT
```

Prevent Denial of Service Attack

The following rule will help you prevent a DoS attack

```
iptables -A INPUT -p tcp --dport 80 -m limit --limit 25/minute --limit-burst
100 -j ACCEPT
```

- -m limit: This uses the limit iptables extension
- –limit 25/minute: This limits only maximum of 25 connection per minute. Change this value based on your specific requirement
- –limit-burst 100: This value indicates that the limit/minute will be enforced only after the total number of connection have reached the limit-burst level.

Port Forwarding

The following example routes all traffic that comes to the port 442 to 22. This means that the incoming ssh connection can come from both port 22 and 422.

```
iptables -t nat -A PREROUTING -p tcp -d 192.168.102.37 --dport 422 -j DNAT
--to 192.168.102.37:22
```

If you do the above, you also need to explicitly allow incoming connection on the port 422.

```
iptables -A INPUT -i eth0 -p tcp --dport 422 -m state --state NEW,ESTABLISHED
-j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 422 -m state --state ESTABLISHED -j
ACCEPT
```

http://www.thegeekstuff.com/2011/06/iptables-rules-examples/

References

https://en.wikipedia.org/wiki/Netcat
https://docs.oracle.com/cd/E64076_01/E64078/E64078.pdf
https://www.ostechnix.com/install-and-configure-dns-server-ubuntu-16-04-lts/