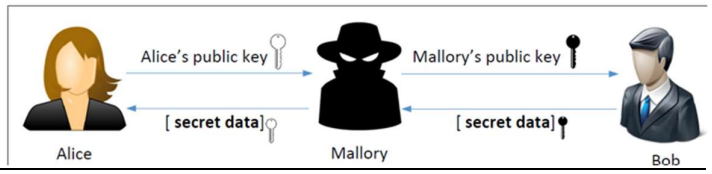
Public Key Cryptography

Man-in-the-Middle (MITM) Attack



2 -> Giving Alice's public key to Bob allows Bob to send secret data to Alice, because Bob can encrypt it using Alice's public-key. However, is this really secure?

3 -> Problem: how can Bob be sure that the public-key is coming from Alice? The Internet can be sniffed and changed by attacker easily. This is called the man-in-the-middle attack. Attacker Mallory can pretend to be Bob when talking to Alice and pretend to be Alice when talking to Bob.

What Is the Fundamental Problem?

Fundamental Problem: Bob has no way to tell whether the public key he has received belongs to Alice or not.

Solution:

- Find a trusted party to verify the identity of Alice (e.g., check Driver's license of Alice).
- Issue a certificate: this public-key belongs to Alice.
- the certificate should be unforgeable (a good digital signature works)

Defeating MITM Attacks using Digital Signature

- Alice needs to go to a trusted party to get a certificate.
- After verifying Alice's identity, the trusted party issues a certificate with Alice's name and her public key.
- Alice sends the entire certificate to Bob.
- Bob verifies the certificate using the trusted party's public key.
- Bob now confirms the true owner of a public key is Alice.
- If an attacker replaces the public-key with his own and claims it is Alice's public-key, Bob will not accept because attacker can not provide a certificate binding Alice and this **modified** public-key.

4 -> The fundamental issue here is that Bob has no way to verify whether the received public-key belongs to Alice or not.

If some well-known person can generate a proof that this public-key belongs to Alice, then Bob will be convinced. This is the idea of public-key certificate.

5 -> Return to the man-in-the-middle attack problem between Alice and Bob. The above procedure shows how this attack is no longer working, if Alice sends her public-key certificate to Bob.

Verify a public-key certificate

- **Certificate Authority (CA):** a trusted party who verifies the public key is owned by an ID (such as **www. uwindsor.ca**), provide a proof about this (i.e., sign on (PK_uw, www.uwindsor.ca)).
- **Digital Certificate:** The proof described above.
  - **X.509 standard**: the standard to generate the certificate

- Certificate is a signature by CA.
- We need to know CA's public-key in order to verify this certificate.

- CA's public-key is presented in its **own** certificate.
- This certificate **could** be provided by **this** CA itself.
- That is, CA's certificate is a signature by itself.
- This type of CA is called **root CA**.

6 -> Certificate Authority (CA) is the entity providing the certificate service.

The standard certificate format is to use x.509. In our certificate experiment, you will practice how this can be done.

7 -> To verify a certificate, we need the issuer (i.e., CA) public-key. This CA's public-key is presented in its own certificate.

8 -> In our experiment, you will play as a root CA to issue certificates.

The best-known root certificates are distributed in operating systems by their manufacturers. Microsoft distributes root certificates belonging to members of the Microsoft Root Certificate Program to Windows desktops and Windows Phone 8. Apple distributes root certificates belonging to members of its own root program.

9 -> TLS evolved from the older version SSL and is gradually replacing SSL

When SSL was standardized by IETF, it was renamed to TLS

SSL version 3.0 was deprecated and replaced by TLS in June 2015

TLS is designed to run on top of TCP but can be implemented with other protocols such as UDP

Root CA and Digital Certificate

- Why do we trust root CA's certificate is authentic (not fake)?
- They are famous and this type of certificate has already be saved in your computer at the time of your computer installation.
- To see an example of such certificate, go to **/etc/ssl/certs**
- view one of the file (ex: WoSign.pem) using x509:
- openssl x509 –in WoSign.pem –text -noout

Verify a certificate not provided by root CA

- Sometimes, a certificate is not issued by a root CA.
- In this case the server can provide a list of certificates such as cert_pk, cert_CA1, cert_CA2, .., cert_CA5, cert_rootCA
- **cert_pk**: proof that pk belongs to its owner (e.g., Alice), provided by CA1
- **cert_CA1**: proof that pubkey in this cert belongs to CA1, provided by CA2
- **cert_CA2**: proof that pubkey in this cert belongs to CA2, provided by CA3

...

- **cert_CA5**: proof that pubkey in this cert belongs to CA5, provided by rootCA
- **cert_rootCA**: proof that pubkey in this cert belongs to rootCA, provided by itself
- **cert_rootCA** is in your computer (can be removed from the above cert list).

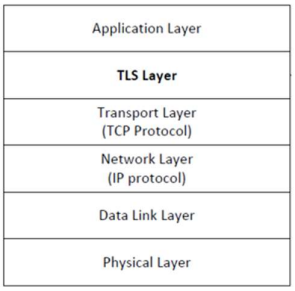
Transport Layer Security

Overview of TLS

- Transport Layer Security (TLS) is a protocol that provides a secure channel between two communicating **applications**. The secure channel has 3 properties:
  - Confidentiality: Attacker should not be able to understand the message sent over the channel.
  - Integrity: Channel can detect any changes to the data in the transmission
  - Authentication: This is to confirm the identity of one communicant. If Alice is authenticated, then no one else can pretend to be Alice.

TLS Layer

- TLS sits between the Transport and Application layer
  - Unprotected data is given to TLS by Application layer
  - TLS handles encryption, decryption and integrity checks
  - TLS gives protected data to Transport layer



12 -> Application will use TLS_socket.send(msg) to send message.

- TLS will take msg from application and do encryption and integrity operations.
- After this, a ciphertext of msg is computed. TLS then sends using TCP_socket.send(ciphertext)
- At receiver side, TLS will verifies the integrity of ciphertext and decrypt it to msg. Finally, it gives to application which will be received via TLS_socket.recv().

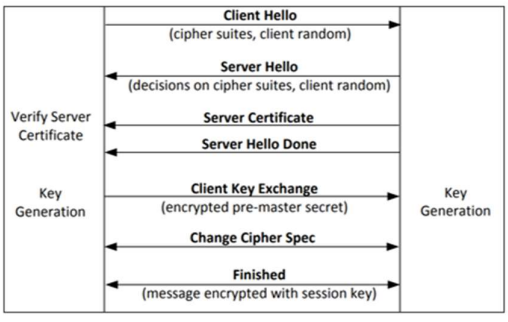
13 -> In the previous slide, we did not mention how the encryption key is shared between client and server, what algorithm for encryption and integrity will be used

- All these will be done in the TLS Handshake protocol. Imagine that TCP connection will run 3-way handshake protocol to set up the connection as well as its parameters such as receive window. TLS handshake protocol has the similar purpose.

TLS Handshake

- Before a client and server can communicate securely, several things need to be set up first:
  - Encryption algorithm and key
  - MAC algorithm
  - Algorithm for key exchange
- These cryptographic parameters need to be agreed upon by the client and server
- This is the primary purpose of the handshake protocol

TLS Handshake Protocol



14 -> Ciphert suites: the possible encryption algorithms (such as RSA, AES, MD5, SHA2). Client-server will need to agree on what algorithms will be used. Client hello asks this to server and server makes decision on server hello.

- **Client random:** a random number chosen by client. This random will always be put into the encryption and integrity operation so that the ciphertext is different from old one (so attacker can not use older ciphertext to cheat the receiver).
- **Server certificate:** server will provide his public-key certificate to client. This will be verified as mentioned before.
- **Client key exchange:** this will be key exchange between client and server so that they will share a secret key.

Network Traffics During TLS Handshake

Since TLS runs top of TCP, a TCP connection needs to be established before the handshake protocol. This is how the packet exchange looks between a client and server during a TLS handshake protocol captured using Wireshark:

No.	Source	Destination	Protocol	Info
1	10.0.2.45	10.0.2.35	TCP	59930 –> 11110 [SYN] Seq=0 Win=14600 Len=0 MSS=1460...
2	10.0.2.35	10.0.2.45	TCP	11110 –> 59930 [SYN, ACK] Seq=0 Ack=1 Win=14480...
3	10.0.2.45	10.0.2.35	TCP	59930 –> 11110 [ACK] Seq=1 Ack=1 Win=14720 Len=0...
4	10.0.2.45	10.0.2.35	TLSv1.2	Client Hello
6	10.0.2.35	10.0.2.45	TLSv1.2	Server Hello, Certificate, Server Hello Done
8	10.0.2.45	10.0.2.35	TLSv1.2	Client Key Exchange, Change Cipher Spec, Finished
9	10.0.2.35	10.0.2.45	TLSv1.2	New Session Ticket, Change Cipher Spec, Finished

Certificate Verification

- Certificate is a signature from an authority (e.g., VeriSign): the public key (e.g., RSA public key) belongs to the hostname or ip address.
- The client first does a validation check of the certificate
  - Check expiration date, signature validity, etc.
  - Hostname and certificate's common name match
- Cert verification is discussed before. Especially, the server could send a list of certificates to client with one of them signed by root CA.

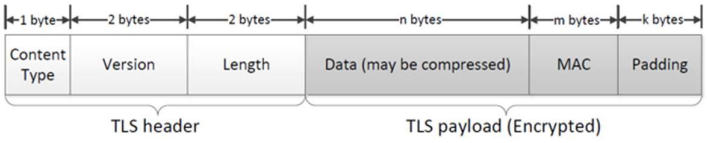
15 -> TLS runs on top of TCP. So before TLS, TCP connection should be established.

16 -> When client receives the certificate, it will verify it as mentioned before.

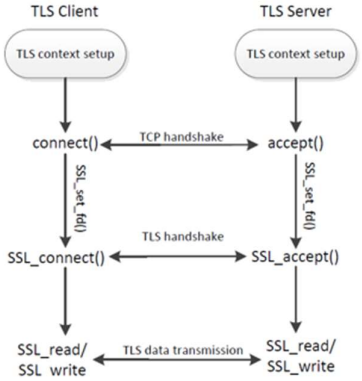
- In addition, client will verify that the hostname it wants to talk with is the same as that carried in the certificate.

TLS Data Transmission

- Once the handshake protocol is finished, client and server can start exchanging data.
- Data is transferred using records.
- Each record contains a header and a payload



TLS Programming : Overall Picture



17 -> In this section, we focus on the application record type, which is used to transfer application data between a client and a server

Content Type: Indicates the type of protocol data carried by current record (handshake, alert, application, heartbeat or ChangeCipherSpec)

Version: This field identifies major and minor version of TLS being used

Length: The length of the payload field.

Python TLS Client Program

```python
#!/usr/bin/python3

import socket, ssl, sys, pprint

hostname = sys.argv[1]
port = 443
cadir = '/etc/ssl/certs'

# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
context.load_verify_locations(capath=cadir)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True

# TCP handshake
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((hostname, port))

# TLS handshake
ssock = context.wrap_socket(sock, server_hostname=hostname, do_handshake_on_connect=False)
ssock.do_handshake()    # Start the handshake

# Send HTTP Request to Server
request = b"GET / HTTP/1.0\r\nHost: " + hostname.encode() + b"\r\n\r\n"
ssock.sendall(request)
# Read HTTP Response from Server
response = ssock.recv(2048)
while response:
      pprint.pprint(response.split(b"\r\n"))
      response = ssock.recv(2048)

# Close the TLS Connection
ssock.shutdown(socket.SHUT_RDWR)
ssock.close()
```

Test: $ python3 client.py www.google.com You will received the initial get response.

19 **-> cadir:** it is the location you store CA certificates (recall CA's public-key will be used to verify the server public-key certificate).

- **ssl.CERT_REQUIRED:** need to verify server certificate
- **check_hostname:** need to check sys.argv[1] is the same as the owner carried in server certificate.

Python TLS Server Program

- Follow reference file to create and use certificate and key.
- **$ python3 server.py**
- run client.py (with port **4433**) on another VM to test client and server communication

$python3 client.py 10.0.2.4

- 10.0.2.4 is th server ip.

```python
#!/usr/bin/python3

import socket, ssl, pprint

html = """
HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
<!DOCTYPE html><html><body><h1>This is Bank32.com!</h1></body></html>
"""

SERVER_CERT    = './cert/Test.crt'
SERVER_PRIVATE = './cert/Test.key'

#TLS Context Setup
context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
context.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)

#Run TCP server
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
sock.bind(('0.0.0.0', 4433))
sock.listen(5)

while True:
   newsock, fromaddr = sock.accept()
   try :
     ssock = context.wrap_socket(newsock, server_side=True)       #TLS handshake
     print("TLS connection established")
     data = ssock.recv(1024)                                      # Read data over TLS
     pprint.pprint("Request: {}".format(data))
     ssock.sendall(html.encode('utf-8'))                          # Send data over TLS

     ssock.shutdown(socket.SHUT_RDWR)                             # Close TLS connection
     ssock.close()

   except Exception:
     print("TLS connection fails")
     continue
```

Outline

- What are firewalls?
- Types of Firewalls
- Iptables firewall in Linux
- Stateful Firewall
- Application Firewall

Firewalls

- It is a part in a computer system or in a network, designed to stop unauthorized traffic flowing from one network to another.
- Separate trusted and untrusted components of a network.
- Main functionalities are filtering data, redirecting traffic and protecting against network attacks.

Requirements of a firewall

- In general, authorized traffic is defined by the security policy and is allowed to pass through the firewall.
- The firewall itself must be immune to penetration.

Firewall Policy

- User control: Control user's access right to the data. This usually is applied to users inside the firewall perimeter.
- Service control: Control the access to the service requested in the packet such as Telnet. Applied on the basis of ip address, protocol and port numbers in the packet.
- Direction control: Determines the direction in which requests may be initiated and are allowed to flow through the firewall. It tells whether the traffic is "inbound" (From the network to firewall) or vice-versa "outbound"

Firewall actions

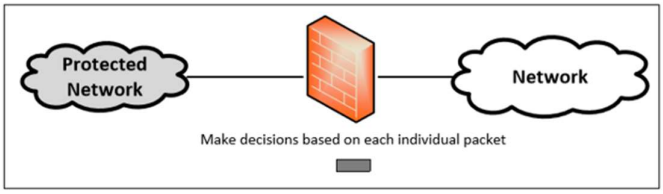
Accepted: Allowed to enter the connected network/host through the firewall.
Denied: Not permitted to enter the other side of firewall.
Rejected: Similar to "Denied", but tells the source about this decision, through ICMP packet.

Types of filters

Depending on the mode of operation, there are three types of firewalls :

- Packet Filter Firewall
- Stateful Firewall
- Application/Proxy Firewall

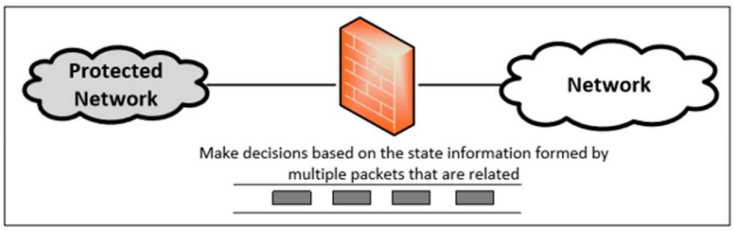
Packet Filter Firewall



- Controls traffic based on the information in packet headers, without looking into the payload that contains application data.

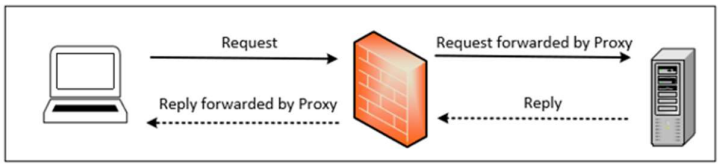
- Doesn't care whether the packet is a part of existing stream or traffic.
- Doesn't maintain the states about packets. Also called Stateless Firewall.

Stateful Firewall



- If multiple packets are "**related**", classify them as a class of the traffic. Filtering policy is defined for the traffic class.
- Later, if a packet belongs to this class of traffic, then apply the class decision to the packet.

Application/Proxy Firewall



- The client's connection terminates at the proxy and a separate connection is initiated from the proxy to the destination host.
- Data on the connection is analyzed up to the application layer to determine if the packet should be allowed or rejected.

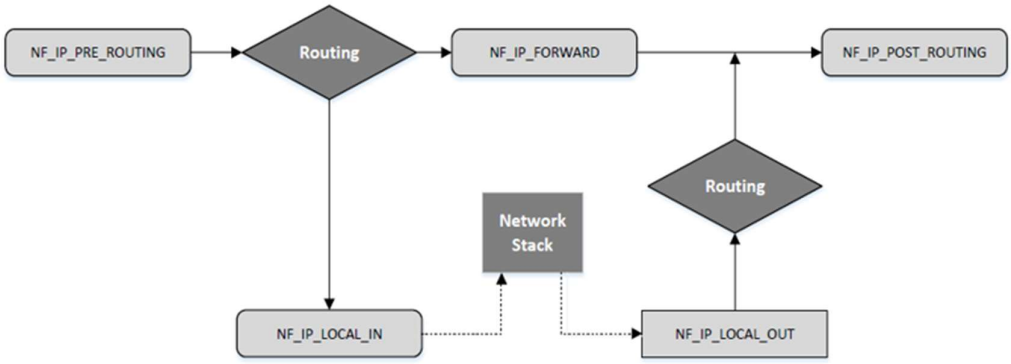
- Controls input, output and access from/to an application or service.
- Acts an intermediary by impersonating the intended recipient.

Limitation : Need to implement new proxies to handle new protocols. Slower compared to other firewalls
Advantage : Ability to authenticate users directly rather than depending on network addresses of the system. Reduces the risk of IP spoofing attacks that are easy to launch against a network.

Netfilter

- Netfilter is a packet processing and filtering framework.
- For each protocol (such IPv4), it defines a sequence of processing stages (called **hooks).** The packet will go through these hooks to get processed in the Netfilter.
- In each hook, developers can define their own callback functions to specify how to check the captured packet and what to do with the packet.

Netfilter Hooks for IPv4



- **NF_IP_LOCAL_OUT:** The packets generated by the current host start its way out to Internet from here.
- **NF_IP_POST_ROUTING:** the packets are about to be out of the host and entering a different network.
- **NF_IP_PRE_ROUTING:** All packets will go through this hook
- **NF_IP_LOCAL_IN:** This hook is called when a packet is destined to the machine itself
- **NF_IP_FORWARD:** This hook is called if the packet is destined to other hosts (not the current host).
- **Routing:** This is routing processing, which either lets in the current host or lets it continue the traverse.
- **Network Stack:** This is where the packet goes through the network layer, transport layer and application layer. Here it **ends** its traverse at application program socket.

Iptables Firewall in Linux

- Iptables is a built-in firewall based on netfilter.
- Three tables: filter, nat and mangle
- Each table is an independent instance of netfilter. E.g., in filter,
  - NF_IP_PRE_ROUTING: do nothing
  - NF_IP_LOCAL_IN: using rules in INPUT
  - NF_IP_FORWARD: using rules in FORWARD
  - NF_IP_LOCAL_OUT: using rules in OUTPUT
- All three tables filter, nat and mangle form the firewall of Iptables.

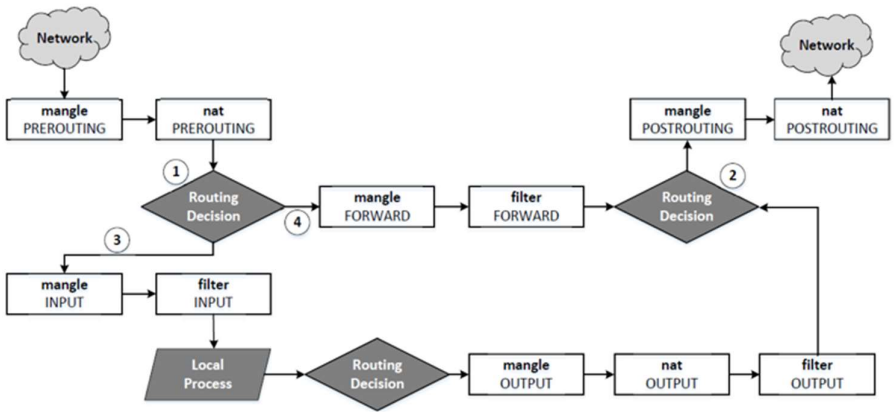
Table	Chain	Functionality
filter	INPUT FORWARD OUTPUT	Packet filtering
nat	PREROUTING INPUT OUTPUT POSTROUTING	Modifying source or destination network addresses
mangle	PREROUTING INPUT FORWARD OUTPUT POSTROUTING	Packet content modification

PREROUTING: for altering incoming packets before routing;
OUTPUT: for altering locally-generated packets before routing;
INPUT: for packets coming into the box itself,
FORWARD: for altering packets being routed through the box;
POSTROUTING: for altering packets as they are about to go out.

Iptables Firewall - Structure

- Each table contains several chains, each of which corresponds to a netfilter hook.
- Each chain indicates which hook its rules are enforced at.
  - Example : Rules on FORWARD chain are enforced at NF_IP_FORWARD hook and rules on INPUT chain are enforced at NF_IP_LOCAL_IN hook.
- Each chain contains a set of firewall rules that will be enforced.
- User can add rules to the chains.
  - Example : To block all incoming telnet traffic, add a rule to the INPUT chain of the filter table

Netfilter structure: merging Filter, NAT and Mangle



In the figure, option 1, 2, 3, 4 are explained as follows.

1. Do the routing decision: the current packet is going to the current host or going to other hosts.
2. Decide which interface to send out the packet.
3. Take the packet in as destined to the current host
4. Let the packet continue traverse to other hosts.

Since filter, NAT and mangles are independents firewall, they all go through Netfilter basic structure. It turns out, a Netfilter hook could be processed by more than one times.

For example, NF_IP_PRE_ROUTING hook is implemented by mangle and nat both. However, the processings by NAT and MANGLE are certainly different.

Rule Matching at a chain

As a packet traverses through each chain, rules on the chain are examined to see whether there is a match or not. If there is a match, the corresponding target action is executed: ACCEPT, DROP, REJECT, or jump to user-defined chain.

17 -> -j: it executes the **target** rule TTL –ttl-inc 5, which will increase TTL field of a packet by 5.
In general, this will have format:
-j **target** *target-opt*, where **target** is the program (or module) name and *tart-opt* is its option.
The possible **target** modules are described in man iptables-extensions. You can see in the

Rule Matching at a chain: example

Example: Increase the TTL field of all packets by 5.
Solution: Add a rule to the mangle table and choose a chain provided by netfilter hooks. We choose PREROUTING chain so the changes can be applied to all packets, regardless they are for the current host or for others.

```
// -t mangle = Add this to 'mangle' table
// -A PREROUTING = Append this rule to PREROUTING chain

iptables -t mangle -A PREROUTING -j TTL --ttl-inc 5
```

description that TTL module is only available for **mangle** table.
Also, **target** can be ACCEPT, DROP or RETURN. Here ACCEPT means to let the packet through. DROP means to drop the packet on the floor. RETURN means stop traversing this chain and resume at the next rule in the previous (calling) chain.

Iptables Extension (example): owner module

Iptables functions can be extended using modules also called as extensions.
Owner: To specify rules based on user ids.
format: -m owner --uid-owner **seed**
Application: How to prevent user Alice from sending out telnet packets?
Analysis: Owner module can match packets based on the user/group id of the process that created them. This works only for **OUTPUT** chain (outgoing packets) as it is impossible to find the user ids for INPUT chain (incoming packets).

Iptables Extension: Block a Specific User

```
seed$ sudo iptables -A OUTPUT -m owner --uid-owner seed -j DROP
seed$ telnet 10.0.2.5
Trying 10.0.2.5...
telnet: Unable to connect to remote host: ...   ← telnet is blocked!

seed$ su bob
Password:
bob$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
ubuntu login:            ← telnet works!
```

This rule drops the packets generated by any program owned by user seed. Other users are not affected.

Building a Simple Firewall

- Flush all existing firewall configurations
- Default policy is set to ACCEPT before all the rules.

```
// Set up all the default policies to ACCEPT packets.
$ sudo iptables -P INPUT ACCEPT
$ sudo iptables -P OUTPUT ACCEPT
$ sudo iptables -P FORWARD ACCEPT

// Flush all existing configurations.
$ sudo iptables -F
```

All commands here omit the default table: -t filter
So actually, the above flush should be sudo iptables –t filter -F

20 -> -P define the default policy. The general format is

-P **chain_name** **ACCEPT**_or_**DROP**

-t TABLE_NAME is ignored in the above slides. Then, it applies to the default table **filter**. If we want to apply this default policy to other tables, you need to specify the table name: e.g., -t nat

21 -> -p ***protocol*** *option. Ex.* –p tcp --sport 23*; see iptables-extensions for description.*
The above rule can be read as
Append a rule to **input** chain (of default table filter). This rule is: for protocol **tcp** with dport 22, apply the target decision **accept**.

Building a Simple Firewall

- Rule on INPUT chain to allow TCP traffic to ports 22 and 80

```
// Allow all incoming TCP packets bound to destination port 22.
// -A INPUT: Append to existing INPUT chain rules.
// -p tcp: Select TCP packets
// -dport 22: Select packets with destination port 22.
// -j ACCEPT: Accept all the packets that are selected.
$ sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

// Similarly, accept all packets bound to destination port 80.
$ sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

-p stands for protocol

j for jump to tartget. Here the tartget is ACCEPT

In iptables-extensions

- Rule on OUTPUT chain to allow all outgoing TCP traffic

```
// Allow all outgoing TCP traffic.
// -A OUTPUT: Append to existing OUTPUT chain rules.
// -p tcp: Apply on TCP protocol packets
// -m tcp: Further apply matching rules defined in 'tcp' module.
// -j ACCEPT: Let the selected packets through.

$ sudo iptables -A OUTPUT -p tcp -m tcp -j ACCEPT
```

-m tcp can be omitted

Building a Simple Firewall

-i: for interface

- Allow the use of the loopback interface.

```
// -I INPUT 1 : Insert a rule in the 1st position of the INPUT chain.
// -i lo : Select packets bound for the loopback (lo) interface.
// -j ACCEPT: Accept all the packets that are selected.

$ sudo iptables -I INPUT 1 -i lo -j ACCEPT
```

- Allow DNS queries and replies to pass through.

```
// Allow DNS queries and replies to pass through.

$ sudo iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
$ sudo iptables -A INPUT -p udp --sport 53 -j ACCEPT
```

Building a Simple Firewall

```
seed@ubuntu:$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source      destination
ACCEPT     tcp  --  anywhere    anywhere      tcp dpt:ssh
ACCEPT     tcp  --  anywhere    anywhere      tcp dpt:http
ACCEPT     udp  --  anywhere    anywhere      udp spt:domain

Chain FORWARD (policy DROP)
target     prot opt source      destination

Chain OUTPUT (policy DROP)
target     prot opt source      destination
ACCEPT     tcp  --  anywhere    anywhere     tcp
ACCEPT     udp  --  anywhere    anywhere     udp dpt:domain
```

These are all the rules we have added

```
// Setting default filter policy to DROP.
$ sudo iptables -P INPUT DROP
$ sudo iptables -P OUTPUT DROP
$ sudo iptables -P FORWARD DROP
```

Change the default policy to DROP so that only our configurations on firewall work.

24 -> After the experiment, remove all the rules by allowing traffic on all the chains and flushing out the existing configurations

27-> The earlier firewall allowed all the outgoing TCP traffic. An attacker who compromises an internal host can exfiltrate data over TCP. The connection cannot be made as incoming traffic is blocked. But this is sufficient for exfiltrating data. To avoid such attacks, we need to setup stateful firewalls

Building a Simple Firewall: Testing

```
$ telnet 10.0.2.6       ← Our firewall is running on 10.0.2.6.
Trying 10.0.2.6...
telnet: Unable to connect to remote host: ...      ← Blocked!
$ wget 10.0.2.6
--2018-11-10 11:26:28--  http://10.0.2.6/
Connecting to 10.0.2.6:80... connected.
HTTP request sent, awaiting response... 200 OK     ← Succeeded!
```

- To test our firewall, make connection attempts from a different machine.
- Firewall drops all packets except the ones on ports 80(http) and 22(ssh).
- Telnet connection made on port 23 failed to connect, but wget connection on port 80 succeeded.

Stateful Firewall using Connection Tracking

- A stateful firewall monitors incoming and outgoing packets over a period of time.
- Records attributes like IP address, port numbers, sequence numbers. Collectively known as connection states.
- A connection state, in context of a firewall signifies whether a given packet is a part of an existing flow or not.
- Hence, it is applied to both connection-oriented (TCP) and connectionless protocols (UDP and ICMP).

Connection Tracking Framework in Linux

- nf_conntrack is a connection tracking framework in Linux kernel built on the top of netfilter.
- Each incoming packet is marked with a connection state as described:
  - NEW: The connection is starting and packet is a part of a valid sequence. It only exists for a connection if the firewall has only seen traffic in one direction.
  - ESTABLISHED: The connection has been established and is a two-way communication.
  - RELATED: Special state that helps to establish relationships among different connections. E.g., FTP Control traffic and FTP Data traffic are related.
  - INVALID : This state is used for packets that do not follow the expected behavior of a connection.
- iptables can use nf_conntrack to build stateful firewall rules.

Example: Set up a Stateful Firewall

```
// -A OUTPUT: Append to existing OUTPUT chain rules.
// -p tcp: Apply on TCP protocol packets.
// -m conntrack: Apply the rules from conntrack module.
// --ctsate ESTABLISHED,RELATED: Look for traffic in ESTABLISHED or
    RELATED states.
// -j ACCEPT: Let the selected packets through.

$ sudo iptables -A OUTPUT -p tcp -m conntrack --ctstate
    ESTABLISHED,RELATED -j ACCEPT
```

- To set up a firewall rule to only allow outgoing TCP packets if they belong to an established TCP connection.
- We only allow ssh and http connection and block all the outgoing TCP traffic if they are not part of an ongoing ssh or http connection.
- We will replace the earlier rule with this one based on the connection state.