Firewalls

Chester Rebeiro

IIT Madras

Some of the slides borrowed from the book ‘Computer Security: A Hands on Approach’ by Wenliang Du

Firewall

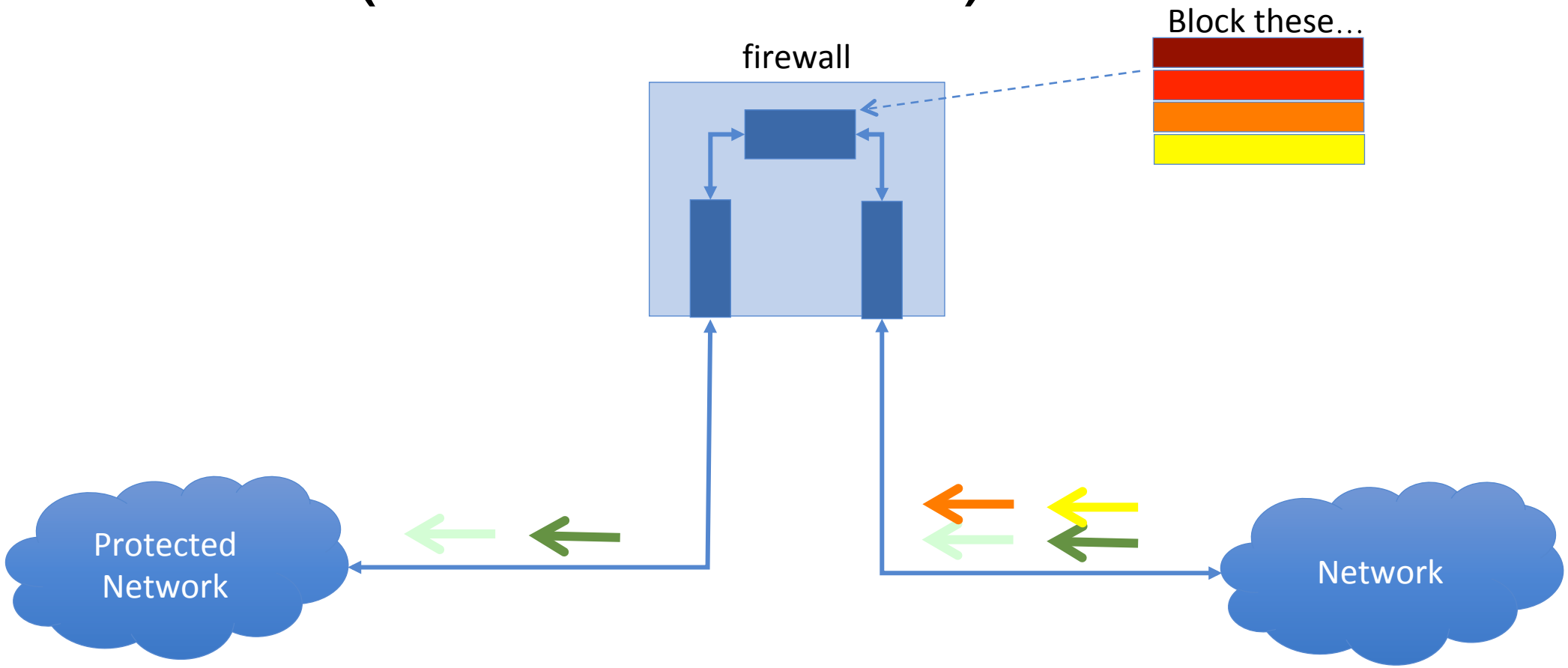
- Block unauthorized traffic flowing from one network to another
- Separate trusted and untrusted components of a network
- Main functionalities
  - Filtering data
  - Redirecting traffic
  - Protecting against network attacks

Two schools of thought

- That which is not expressly forbidden is permitted

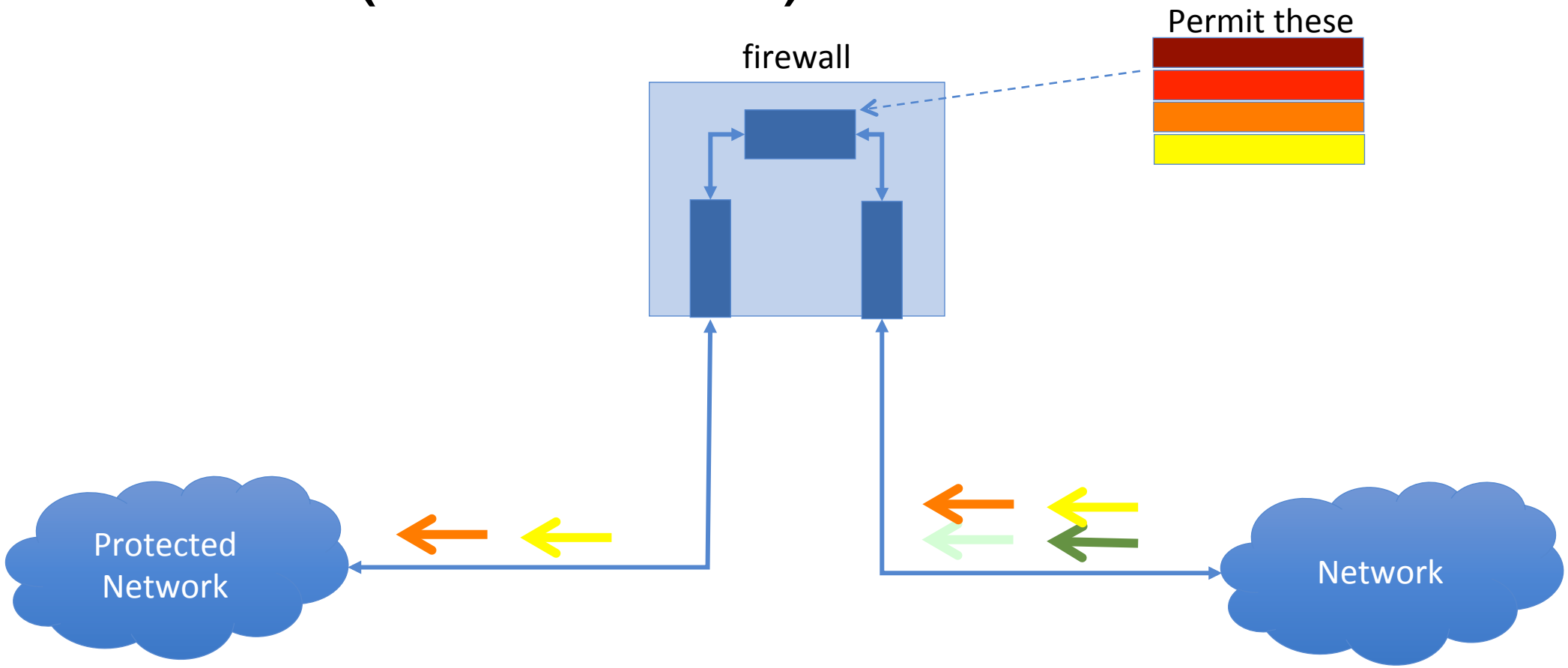
- That which is not expressly permitted is forbidden

Firewall (less strict rules)



That which is not expressly forbidden is permitted

Firewall (strict rules)



That which is not expressly permitted is forbidden

An Ideal Firewall

Requirements

- All traffic between two trust zones should pass through the firewall.
- Only authorized traffic, as designed by the security policy, should be allowed to pass through
- The firewall itself must be immune to penetration, which implies using a hardened system with secured OS

Actions

- Accepted: Allowed to enter the protected network
- Denied: Not permitted to enter the other side of the firewall
- Rejected: Similar to `denied', but tells the source about the decision through an ICMP packet

Firewall Policy

A firewall is as good as the rules that are being enforced by it.
Rules are defined to provide the following controls for the traffic on the network:

Examples:
“Prevent any access from outside but allow traffic to flow from inside to the outside”
“Allow traffic to enter from certain places, users, or for specific activities”

Firewall Controls

- Service Control
  - Determines which services on internal hosts are accessible from outside
  - Reject all other incoming services
  - Outgoing service requests and corresponding responses may also be controlled
  - Filtering is based on the contents of IP packets and the type of requests
  - Example: Reject all HTTP requests unless directed to an official web server

Firewall Control

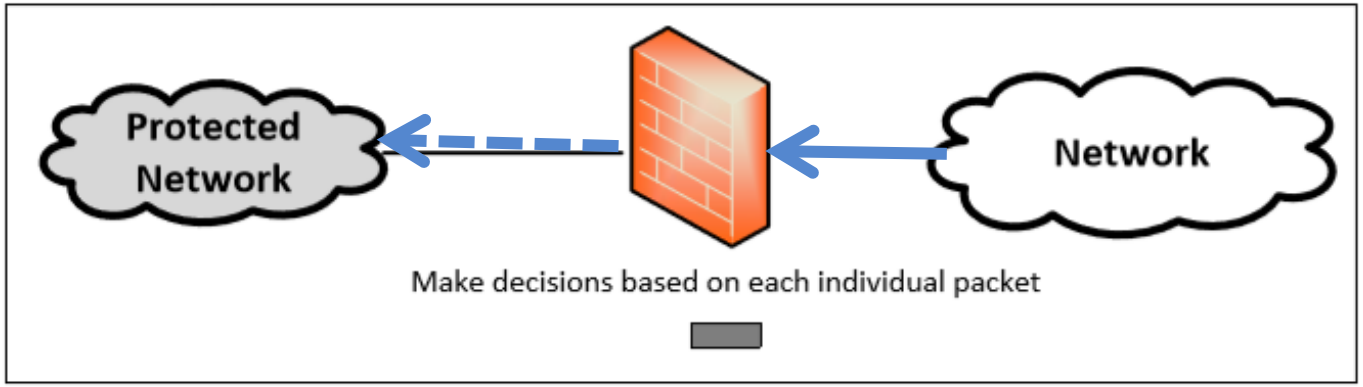
- Behavior Control
  - Infringing organizational policy
  - Anti-social activities on a network
  - Suspected attack
  - Filtering action:
    - May be applicable at IP or TCP level
    - May require further interpretation of messages at a higher level
  - Example: Filtering of spam emails. Would require sender's email address in message headers. May require to scan through the message contents.

Firewall Control

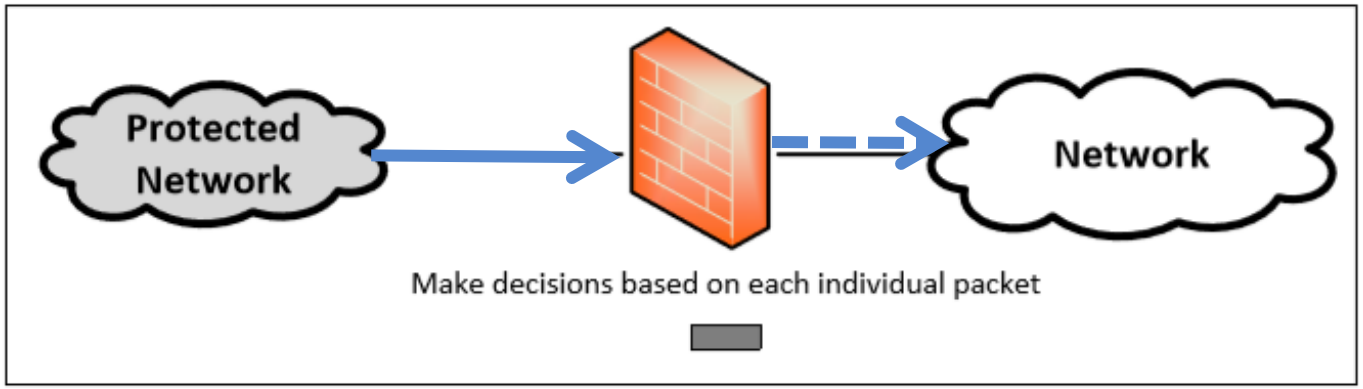
- User Control
  - Discriminate between users
    - Some users can access external services, others can’t
    - Inhibit some users from gaining access to services

Egress and Ingress Filtering

Ingress Filtering



Egress Filtering

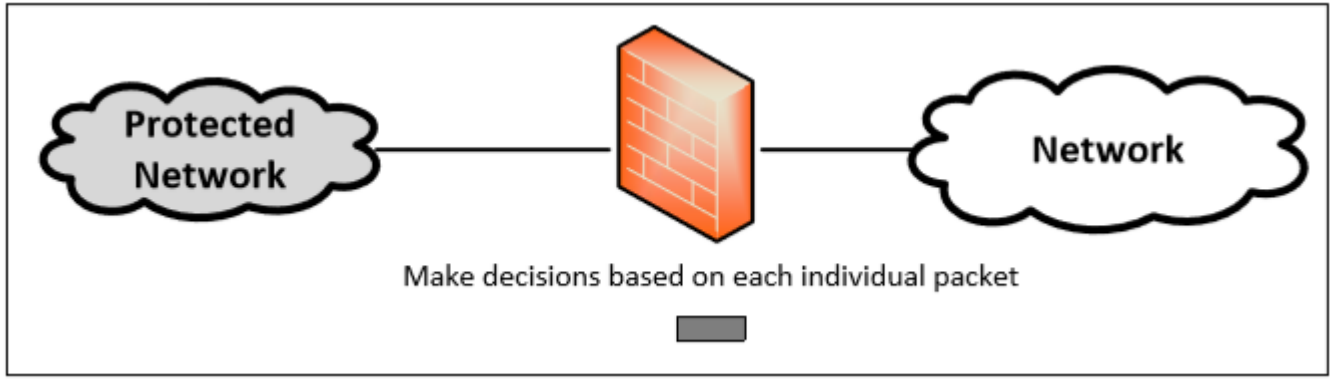


Types of Filters

- Packet Filter
- Stateful Filter
- Application / Proxy Firewall

Types of Filters

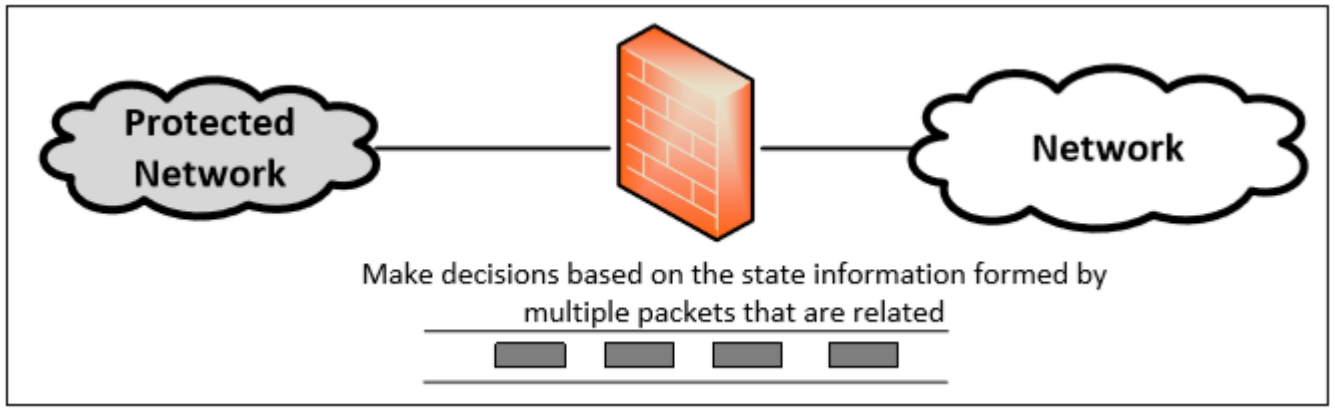
- Packet Filter (aka Stateless firewall)
- Stateful Filter
- Application / Proxy Firewall



- Controls traffic based on the information in packet headers, without looking into the payload that contains application data.
- Doesn’t pay attention if the packet is a part of existing stream or traffic.
- Doesn’t maintain the states about packets.
- Also called Stateless Firewall.

Types of Filters

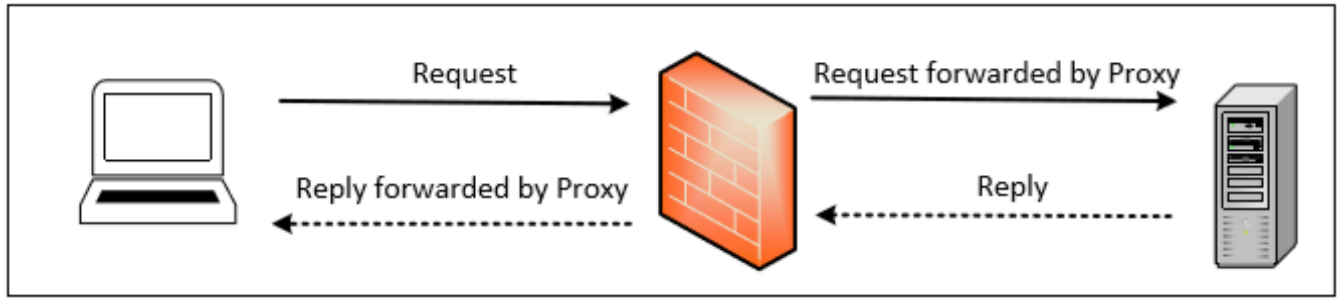
- Packet Filter (aka Stateless firewall)
- Stateful Filter
- Application / Proxy Firewall



- Tracks the state of traffic by monitoring all the connection interactions until closed.
- Connection state table is maintained to understand the context of packets.
- Example : Connections are only allowed through the ports that hold open connections.

Types of Filters

- Packet Filter (aka Stateless firewall)
- Stateful Filter
- Application / Proxy Firewall

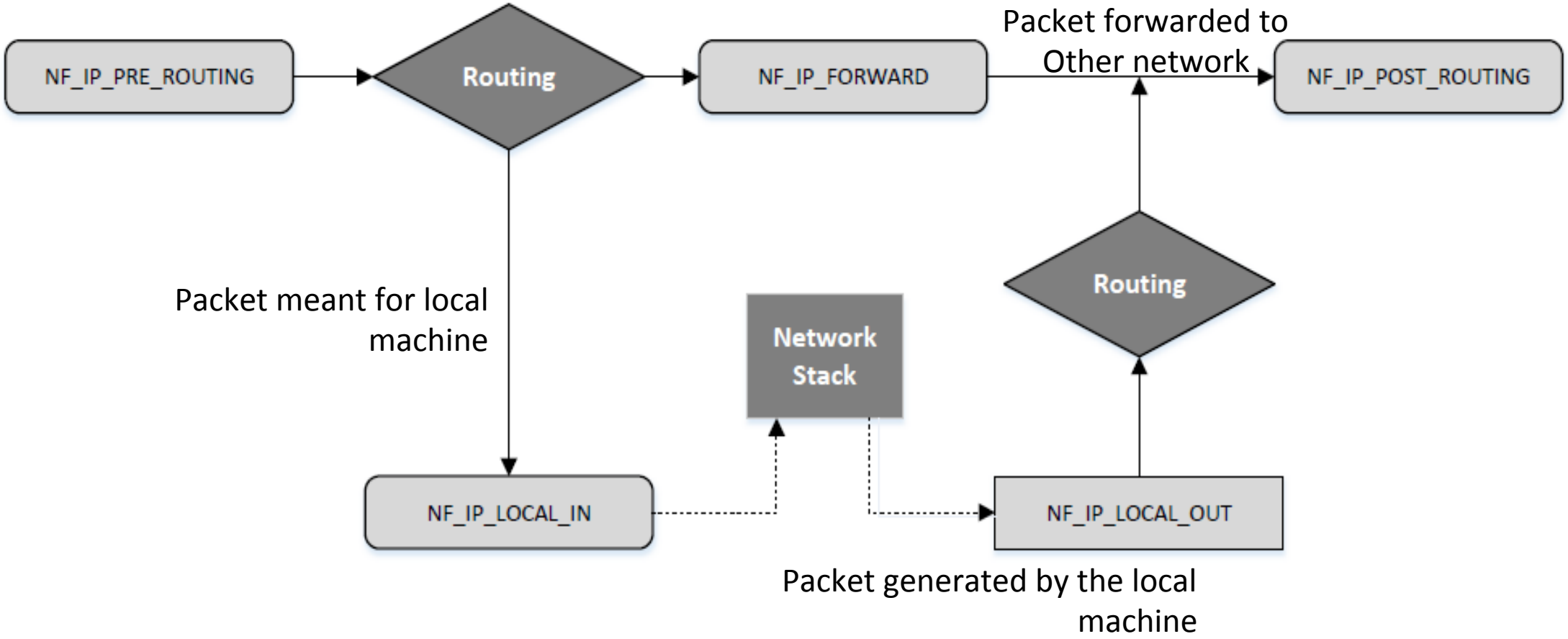


- Controls input, output and access from/to an application or service.
- The client's connection terminates at the proxy and a separate connection is initiated from the proxy to the destination host.
- Data on the connection is analyzed up to the application layer to determine if the packet should be allowed or rejected.
- Advantage : Ability to authenticate users directly rather than depending on network addresses of the system

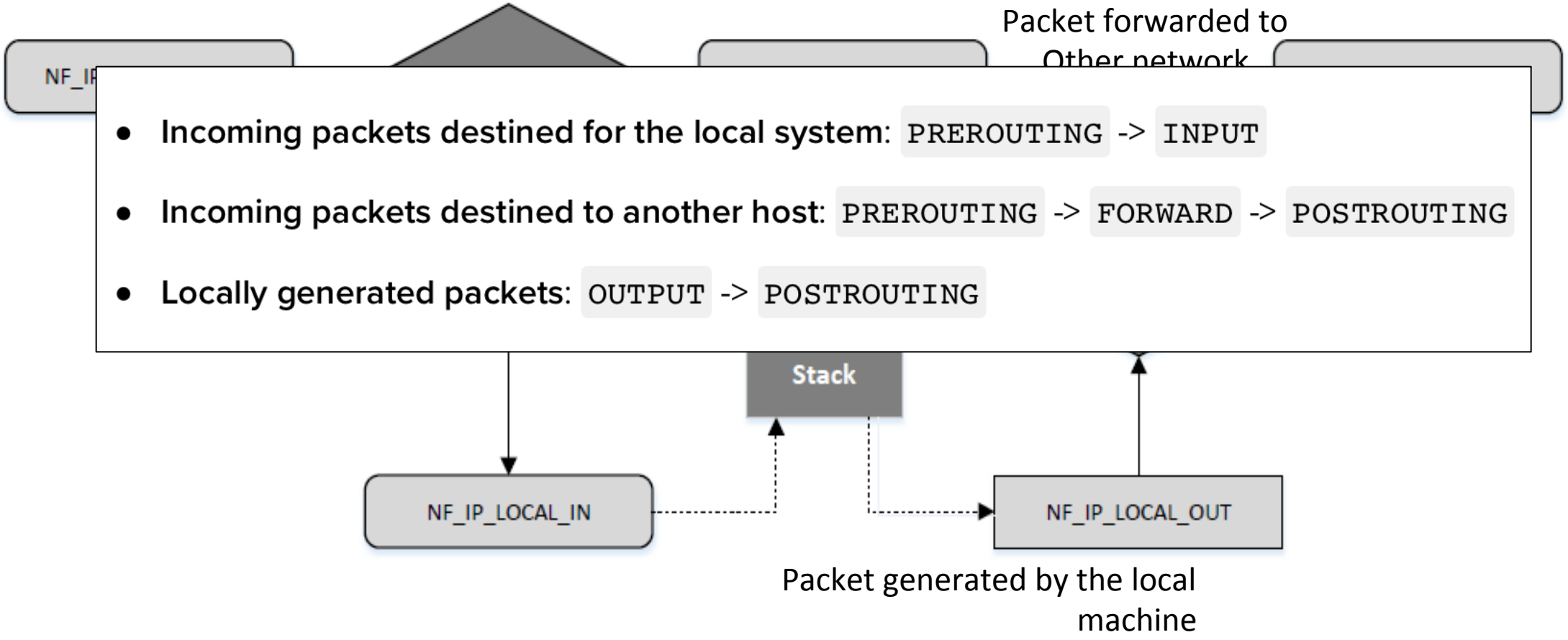
Netfilter: Linux Firewall Support

- Each protocol stack in the Kernel defines a series of hooks along the packet's traversal path in the stack
- Kernel modules can be used to register callback functions to these hooks
- Callbacks are appropriately invoked as a packet passes through the network stack
  - Callbacks take decisions to forward or drop packets

Netfilter Hooks for IPv4



Netfilter Hooks for IPv4



Netfilter: Verdict on Packets (Return Values)

NF_ACCEPT: Let the packet flow through the stack.

NF_DROP: Discard the packet.

NF_QUEUE: Pass the packet to the user space. Can be used to perform packet handling in user space.

NF_STOLEN: Inform the netfilter to forget about this packet, The packet is further processed by the module. Typically use for stateful filtering, the module can store the packet fragments and analyze in a single context.

NF_REPEAT: Request the netfilter to call this module again.

Implementing a Simple Packet Filter Firewall

```
unsigned int telnetFilter(unsigned int hooknum, struct sk_buff *skb,
      const struct net_device *in, const struct net_device *out,
      int (*okfn)(struct sk_buff *)) {
  struct iphdr *iph;
  struct tcphdr *tcph;

  iph = ip_hdr(skb);
  tcph = (void *)iph+iph->ihl*4;

  if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23)) {
    printk(KERN_INFO "Dropping telnet packet to %d.%d.%d.%d\n",
    ((unsigned char *)&iph->daddr)[0],
    ((unsigned char *)&iph->daddr)[1],
    ((unsigned char *)&iph->daddr)[2],
    ((unsigned char *)&iph->daddr)[3]);
    return NF_DROP;
  } else {
    return NF_ACCEPT;
  }
}
```

Decisions

The entire packet is provided here.

The filtering logic is hardcoded here. Drop the packet if the destination TCP port is 23 (telnet)

Implementing a Simple Packet Filter Firewall

```
int setUpFilter(void) {
   printk(KERN_INFO "Registering a Telnet filter.\n");
   telnetFilterHook.hook = telnetFilter;
   telnetFilterHook.hooknum = NF_INET_POST_ROUTING;
   telnetFilterHook.pf = PF_INET;
   telnetFilterHook.priority = NF_IP_PRI_FIRST;

   // Register the hook.
   nf_register_hook(&telnetFilterHook);
   return 0;
}

void removeFilter(void) {
   printk(KERN_INFO "Telnet filter is being removed.\n");
   nf_unregister_hook(&telnetFilterHook);
}

module_init(setUpFilter);
module_exit(removeFilter);
```

Hook this callback function

Use this Netfilter hook

Register the hook

Priority order for calling the hooks. Used For example, when there are multiple modules connected to the same NF hook.

Testing Our Firewall

```
$ sudo insmod telnetFilter.ko
$ telnet 10.0.2.5
Trying 10.0.2.5...
telnet: Unable to connect to remote host: ...    ← Blocked!
$ dmesg
......
[1166456.149046] Registering a Telnet filter.
[1166535.962316] Dropping telnet packet to 10.0.2.5
[1166536.958065] Dropping telnet packet to 10.0.2.5

// Now, let's remove the kernel module

$ sudo rmmod telnetFilter
$ telnet 10.0.2.5
telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 12.04.2 LTS
ubuntu login:                    ← Succeeded!
```

iptables Firewall in Linux

- iptables is a built-in firewall based on netfilter.
- Kernel part: Xtables; User-space program: iptables
- iptables use table to organize rules
  - Filters, nat, mangle, raw (stateful), security
- Chains are used to within a table and signify various hooks present in netfilter:

  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.

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

Chains control, where in the delivery path a rule will be evaluated.

Each table has multiple chains, therefore one table can influence multiple points in the processing stack.

Tables and Chains

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

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

Tables and Chains

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

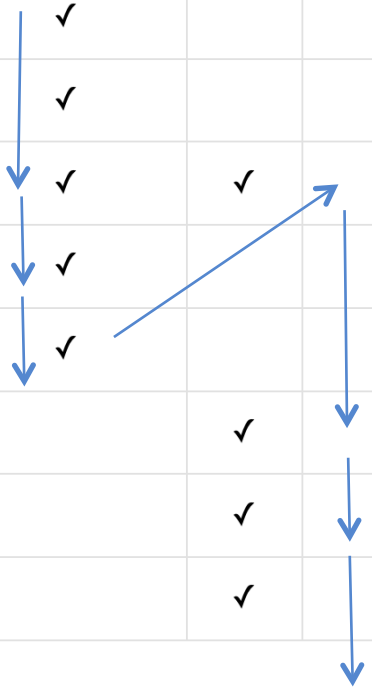
Local socket

Path taken for input packets Destined for the local machine

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

Tables and Chains

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



Protected network

Path taken for input packets
Destined for another machine

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

iptable rules

- Each chain will have rules

- Each rule comprises of two parts:
  - **Match:** criteria that a packet must meet in order for the associated action to be executed

  - **Target:** action to be taken once if match is successful.
    - Terminating target: eg. Drop the packet
    - Non-terminating target: perform an action then continue further in the chain

An example

Add a rule to block the IP address 59.45.175.62

```
iptables -t filter -A INPUT -s 59.45.175.62 -j REJECT
```

(-t is the table; filter is the default, therefore, here need not be specified)
(-A is ADD to INPUT chain)

Add a rule to drop packets going to IP 31.13.78.35

```
iptables -A OUTPUT -d 31.13.78.35 -j DROP
```

Example continued…

List the rules that are currently specified…

```
iptables -L --line-numbers
```

```
Chain INPUT (policy ACCEPT)
num target prot opt source destination
1 DROP all -- 59.45.175.0/24 anywhere
2 DROP all -- 221.194.47.0/24 anywhere
3 DROP all -- 91.197.232.104/29 anywhere
Chain FORWARD (policy ACCEPT)
num target prot opt source destination
Chain OUTPUT (policy ACCEPT)
num target prot opt source destination
1 DROP all -- anywhere 31.13.78.0/24
```


29

Traversing Chains and Rule Matching

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
```

Modules (-m options)

iptables −m option can be used to add specific modules, and there by creating user specific rules.

owner: To specify rules based on user ids. Ex: To prevent user Alice from sending out telnet packets. Owner module can match packets based on the user/group id of the process that created them.

Restricted only where uid/gid for a process can be determined.

Iptables modules: 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 12.04.2 LTS
ubuntu login:              ← telnet works!
```

Option specific to module owner

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

iptables modules

Block ssh from certain IP addresses

```
iptables -A INPUT -p tcp -m tcp --dport 22 -s 59.45.175.0/24 -j DROP
```

Block ssh and VNC (port 5901) from certain IP addresses

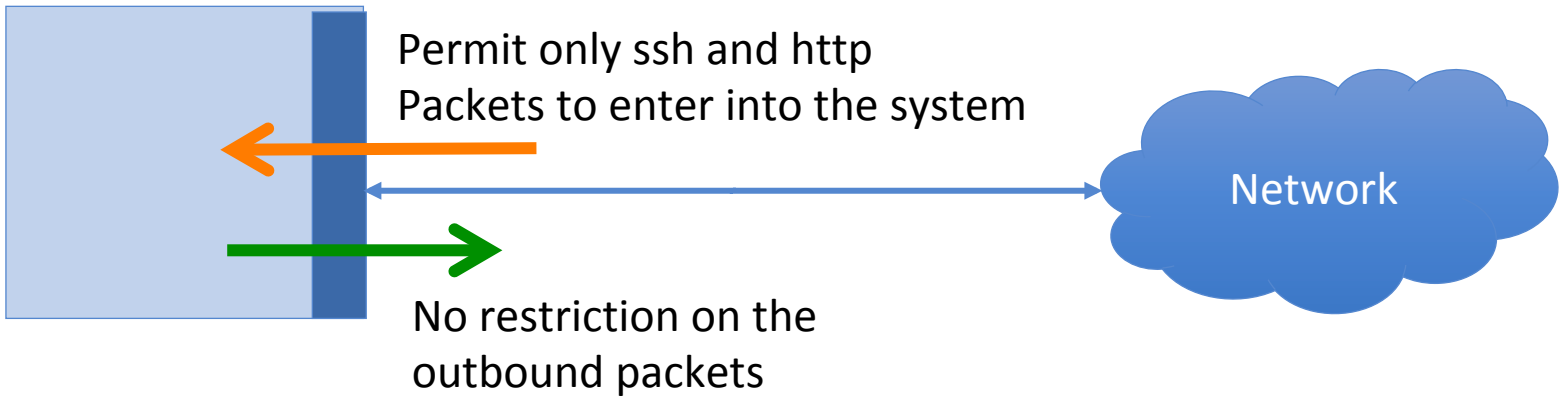
```
iptables -A INPUT -p tcp -m multiport --dports 22,5901 -s 59.45.175.0/24 -j DROP
```

https://www.booleanworld.com/depth-guide-iptables-linux-firewall/

Managing rules in a firewall

- In iptables, packets are sequentially compared against the rules until a match is found
  - Then appropriate target action is executed
- This does not scale with
  - Traffic speed
  - Number of rules

Building a Simple Firewall



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
```

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
```

- 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
```

Building a Simple Firewall

- 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
```

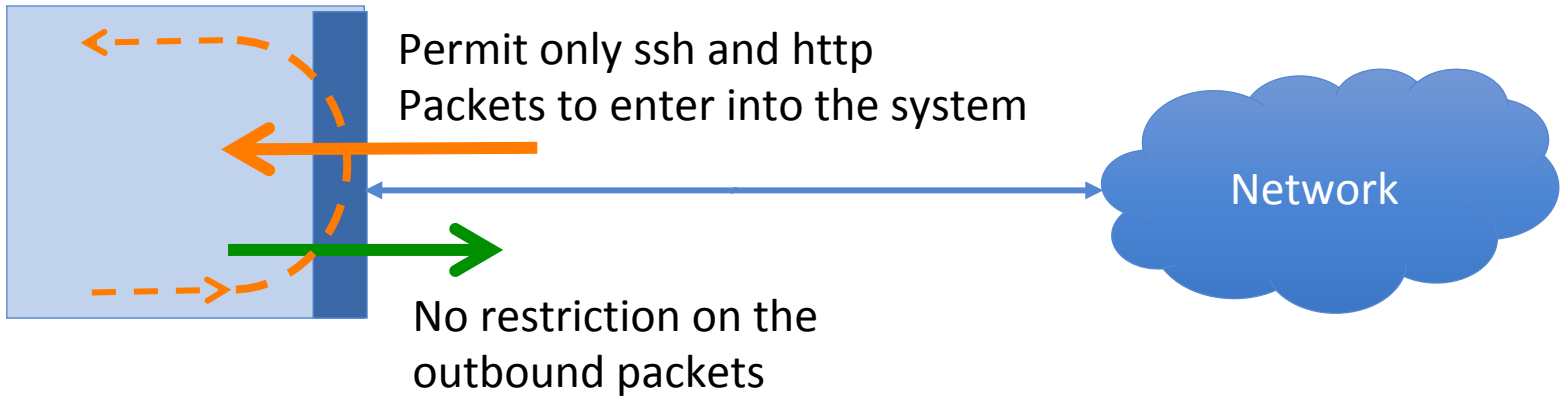
- Allow DNS queries and replies to pass through.

```
// 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

- 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
```



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
```

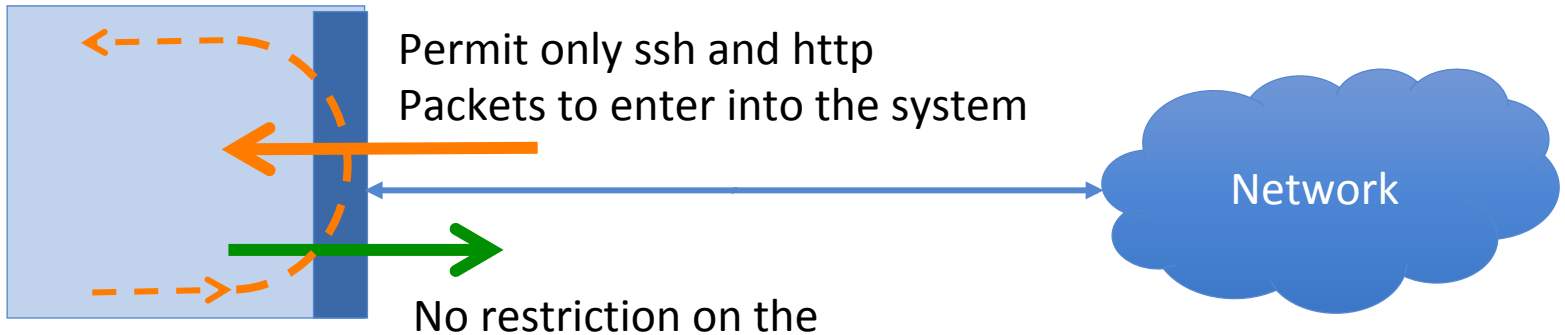
Change the default policy to DROP so that only our configurations on firewall work.

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
--2017-01-25 18:31:41--  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.

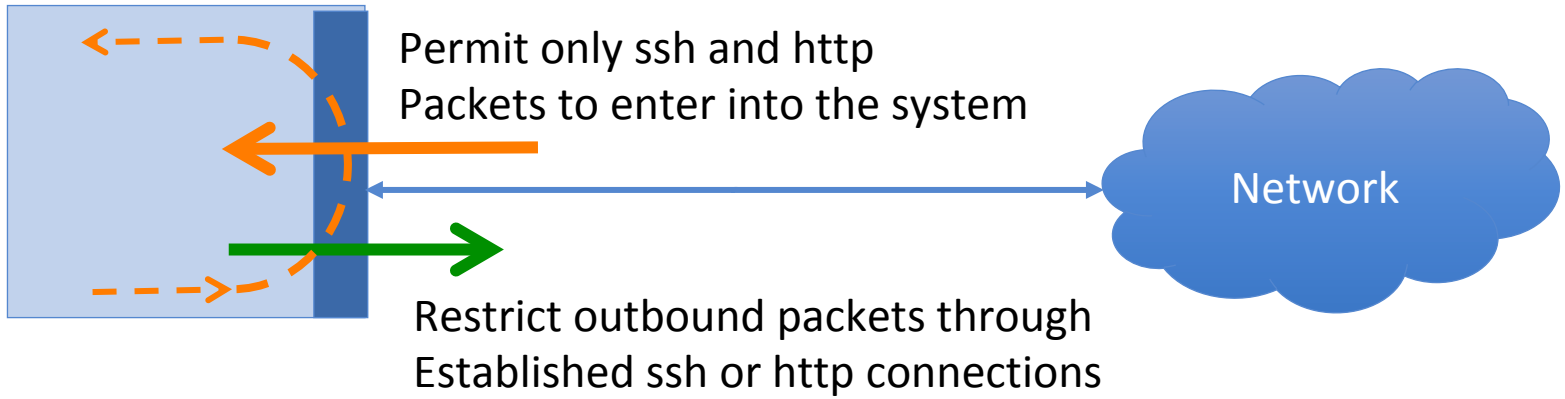
Limitation of the Simple Firewall



Insider can craft a packet and send it out of the network (note that no replies will be possible)

Stateful Firewall

- A stateful firewall monitors incoming and outgoing packets over a period of time.
- Records aspects about connection state (such as IP address, port numbers, sequence numbers).
- The state enables filtering decisions to be based on context of a packet (and not just headers)



Stateful Firewall

- Tracking TCP Connections
  - Stateful firewalls can monitor TCP handshake protocols between two machines to identify if there is a connection established between the two of them.
  - Example: monitoring 3-way handshake protocol or 4-way termination protocols.

Stateful Firewall

- Tracking TCP Connections
  - Stateful firewalls can monitor TCP handshake protocols between two machines to identify if there is a connection established between the two of them.
  - Example: monitoring 3-way handshake protocol or 4-way termination protocols.

- Tracking UDP connections
  - Connection less protocol.
  - Stateful firewalls monitor stream of packets between client and server. If no packets are exchanged for a certain period of time, the connection is considered to be terminated.

Stateful Firewall

- Tracking ICMP Connections
  - Not always possible, when only one ICMP packet is sent from client to server
  - If the ICMP packet has a request and response, then tracking of ICMP connections is possible

- Tracking Application connections
  - Some firewalls may be able to track certain application protocols such as HTTP, FTP, IRC etc.

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:
  - **NEW:** The connection is starting and packet is a part of a valid initialization sequence.
  - **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.

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.

Application/Proxy Firewall and Web Proxy

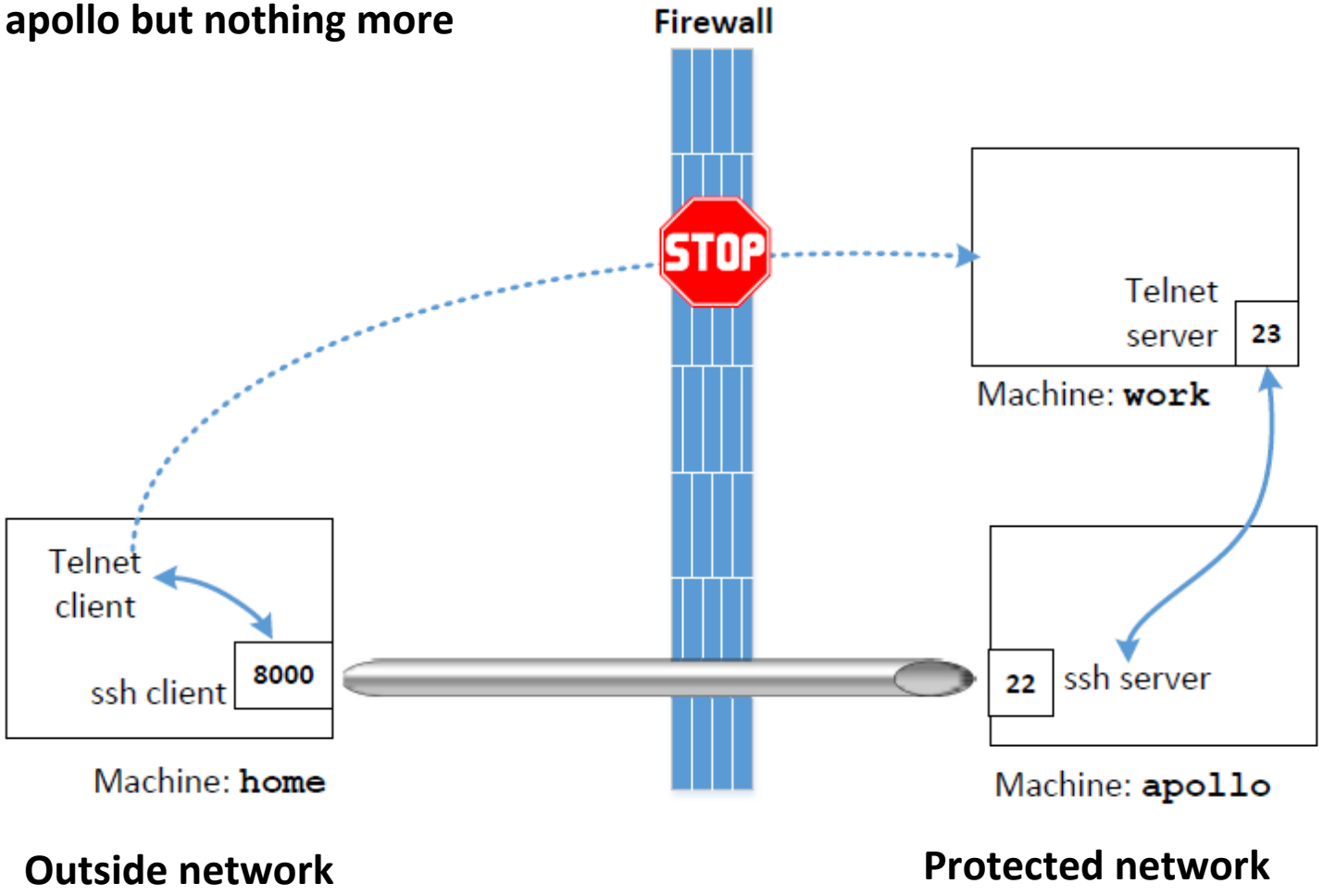
- Inspects network traffic up to the application layer.
- Typical implementation of an application firewall is an application proxy
- Web proxy: To control what browsers can access.
- To set up a web proxy in a network, we need to ensure that all the web traffic goes through the proxy server by:
  - Configuring each host computer to redirect all the web traffic to the proxy. (Browser's network settings or using iptables)
  - Place web proxies on a network bridge that connects internal and external networks.

Evading Firewalls

- SSH Tunneling
- Dynamic Port Forwarding
- Virtual Private Network

SSH Tunneling to Evade Firewalls

Firewall permits ssh connections
To apollo but nothing more

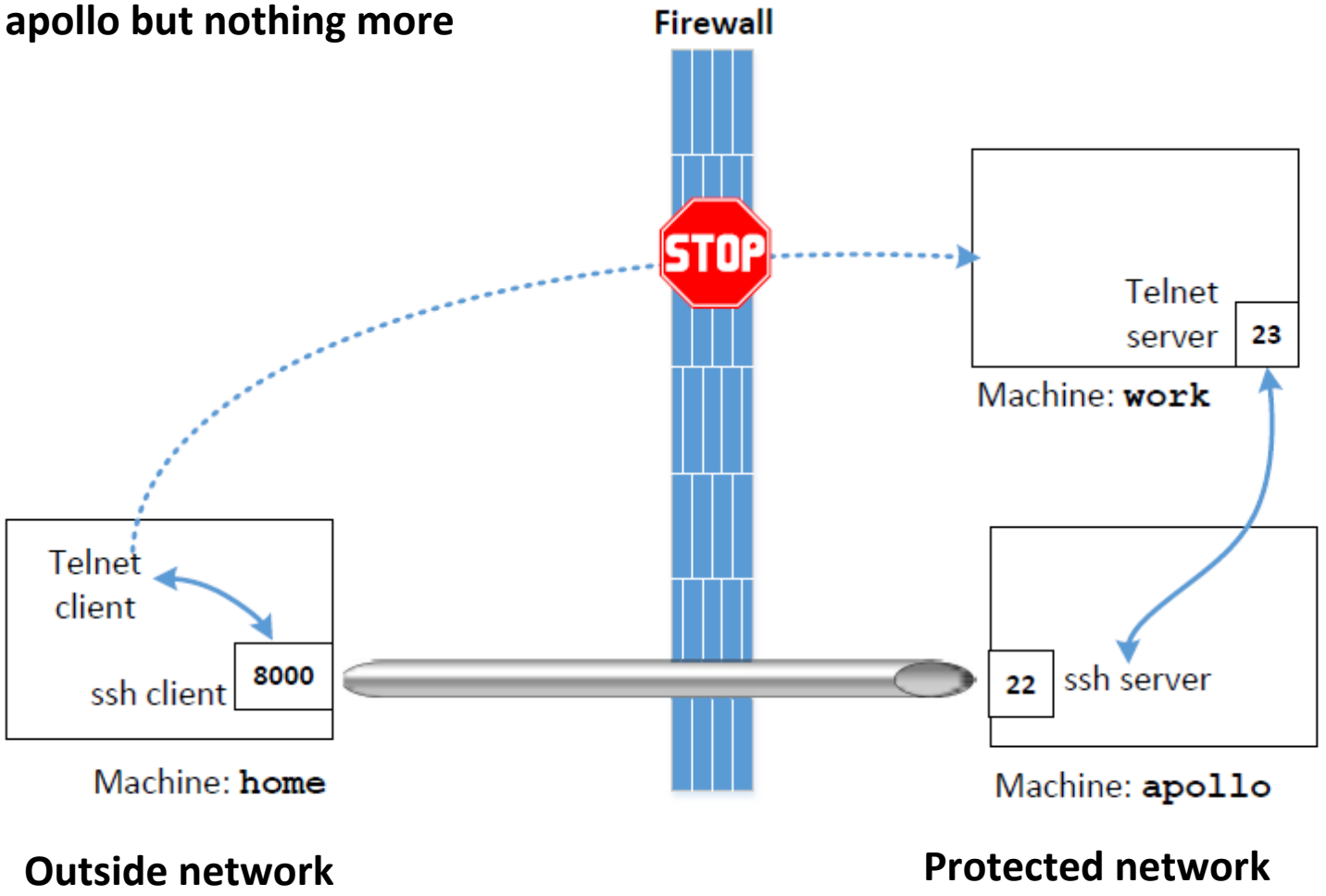



```
$ ssh -L 8000:work:23 apollo
```

Forward packets on the local port 8000 forwarded to work:23 on the remote side (apollo)

SSH Tunneling to Evade Firewalls

Firewall permits ssh connections
To apollo but nothing more




```
$ ssh -L 8000:work:23 apollo
```

Forward packets on the local port (home) 8000 forwarded to work:23 on the remote side (apollo)

```
$ telnet localhost 8000
```

telnet to localhost 8000 from home

SSH Tunneling to Evade Firewalls

Bypassing egress firewalls

```
$ ssh -L 8000:www.facebook.com:80  home
```

- Connect from protected network to a server outside, bypassing the firewall

Dynamic Port Forwarding

Used when we do not know the destination machine on the other side of the firewall

```
$ ssh -D 9000 -C home
```

Create a tunnel from local host port number 9000 to home (present on the other side of the firewall)
Any packet received on port 9000 is tunneled through to home.

Dynamic Port Forwarding

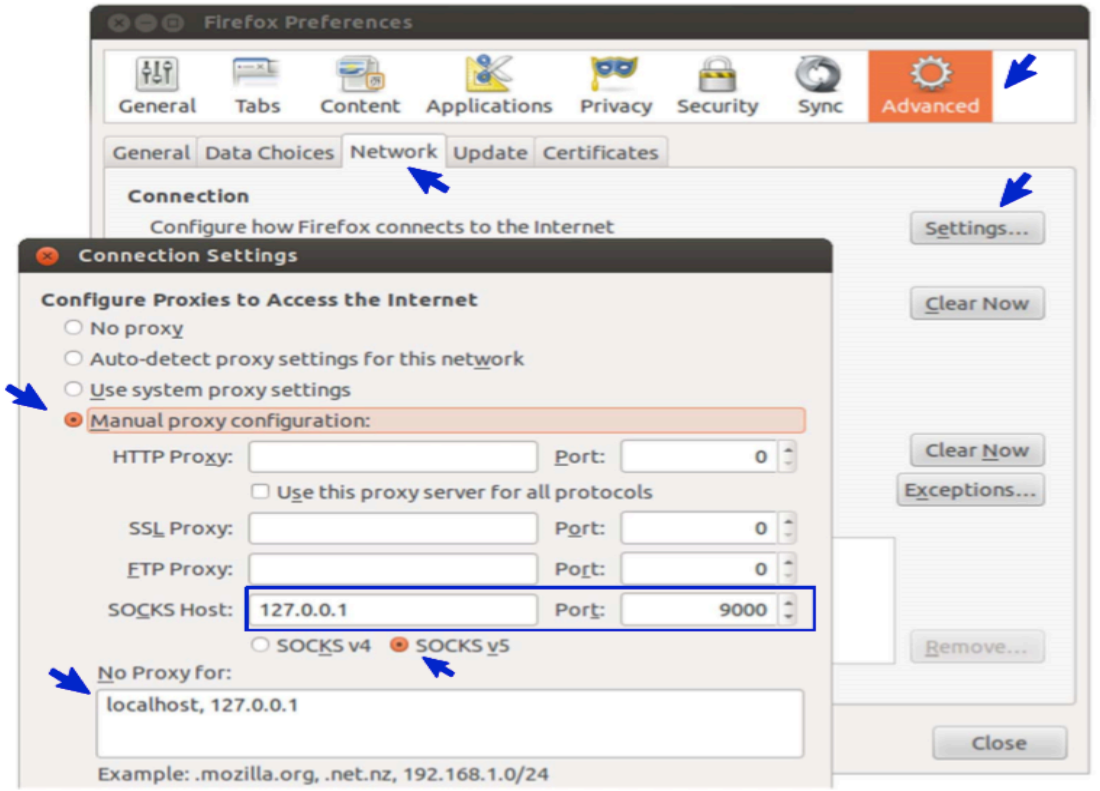
Used when we do not know the destination machine on the other side of the firewall

```
$ ssh -D 9000 -C home
```

Create a tunnel from local host port number 9000 to home (present on the other side of the firewall)

Any packet received on port 9000 is tunneled through to home.

Configure the browser to use proxy localhost:9000
Thus any transactions sent from the browser will be
Directed to localhost:9000
(SOCKS proxy)



Using VPN to Evade Firewall

Using VPN, one can create a tunnel between a computer inside the network and another one outside. IP packets can be sent using this tunnel. Since the tunnel traffic is encrypted, firewalls are not able to see what is inside this tunnel and cannot conduct filtering. This topic is covered in detail late in VPN topic.

Major Firewall Drawbacks / Vulnerabilities

- Insider Attacks
- Anomalies in Firewall configurations
- Firewall policy not updated (Missed Security Patches)
- Lack of deep packet inspection
- DDoS attacks

Next Gen Firewalls

- Not just tracking domains and port numbers of traffic
- Deep-packet inspection
  - Monitor content of messages, data exfiltration
  - Malware detection
- Can react in real-time to stop threats
- Behavioral analytics
- VPN support