Firewall Final Documentation
CS 5434 - Defending Computer Networks

Fall 14 - Cornell University

Student: Eduardo Felipe Gama Ferreira
NetID: efg28

Features:

- Stateful Firewall: TCP, UDP and ICMP.
- Firewall text rules/text rules parser.
- Packets can be exchanged between different interfaces and pcap files.
- Faster flow lookup using hash tables.
- NAT Translator.
- ARP handling system.
- Generic hash tables for lookup and store states.
- Flexible setup: file storage and program call with arguments.
- Debugging mode (Compile with –DDEBUGGING)
- IP, TCP and UDP checksum.

Firewall Rules

Rules are loaded from default file: “default.rules”.
Each line of this file should have exactly one rule.

Rule syntax:
```
rule direction service ip_src port_src -> ip_dst port_src
```

Possible attributions:
```
block/allow in/out tcp/udp/icmp/any IP/any Port/any -> IP/any Port/any
```

The ‘/’ means that you have multiple choices.
Reserved words:
“any” – means arbitrary value.
“HOME” – will be translated to the machine IP protected by firewall.

Rule examples:
```
block in tcp 178.79.172.241 80 -> HOME any
```
(blocks incoming packets of protocol TCP from IP 178.79.172.241 with source port 80 to HOME IP and with any destination port)

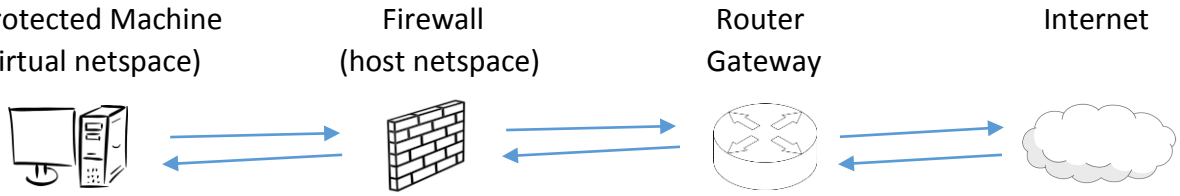
```
allow out any HOME any -> 8.8.8.8 any
```
(allow packets to go out of HOME IP of any service and from any port to address 8.8.8.8 at any destination port)

Default Policy:
Those policies are applied without any explicit text rule.
Explicit firewall rules can modify this behavior.
● By default every packet is allowed to go out from protected machine.
● By default every packet is blocked to get into protected machine.
● By default every packet going out from machine will open up a hole in the firewall to allow reply packets, even if the firewall rule is to block incoming packets. This 'hole' is only open for the duration of a valid flow lookup hash table entry, which is by default 1 minute.
● However packets that are allowed to come in but their reply packets are blocked to go out, will remain blocked (you can enforce that the protected machine will not talk to strangers!)

Stateful Firewall

Firewall will keep track of TCP, UDP and ICMP 'connection' states.

Overview:



The firewall will be exchanging packets between the protected machine and router. Packets are forwarded based on the firewall rules and the 'connection' states.
Protected machine network interface is connected to the Firewall network interface. All packets from the machine are directed to the firewall interface.
The firewall network interface will use the ''switch interface'' in the host netspace to send/receive packets from the router.
The router is connected to the internet.

Setup:
The firewall program needs to know some information about the network before it starts working.
A file called "settings.config" is loaded and it must contain the following addresses, one by line:

```
protected machine MAC address
firewall MAC address
switch MAC address
router MAC address
protected machine IP address
firewall IP address
switch IP address
```

The MAC addresses are from the network interfaces.

To run the firewall it is needed to inform which mode it will execute: real interface sniffing or pcap files.
Call the executable with two parameters:

```
./firewall [interface in/pcap in] [interface out/pcap out]
```

If no parameters or invalid ones are provided, a list of available interfaces will be displayed.

With the network information and interface names or pcap files, the firewall can be setup.
A struct with all the network information and pcap files will be created and defined as the firewall context.
After this step, the firewall rules are loaded from the "default.rules" file and parsed.
The Rules list is created, state tables and port mappings are all initialized to "empty state".

Execution:
The firewall executes by getting one packet and analyzing it.
Each step the firewall will pick a packet.
First it will come from the inside interface (protected space), and then the next will be from the unprotected space.

1) Deserialization
Packets will be deserialized and casted to some header struct depending on the protocol inside the packet.
We start with the Ethernet – every packet must be transmitted using the Ethernet protocol as the link layer.
By examining the ethertype we can identify how to interpret the remaining of the packet.
The same process follows for the next layers.

Supported packets for the link layer: Ethernet.
Supported packets for the network layer: ARP, IPv4, ICMP.
Supported packets for the transport layer: TCP, UDP.
Supported packets for the application layer: not analyzed/all.

2) Flow state lookup
If the packet is part of some current ''connection' flow then it will be pre-approved by the flow lookup step.
The lookup is done by using a flow hash table. It is a faster way to approve packets, than looking through all of the firewall rules in a linear fashion.
Depending on the type of the packet (protocol), the flow state is determined by a hash of a tuple of IPs and/or ports.
For ICMP, <IP Source, IP Dest>
For UDP, <IP Source, Port Source, IP Dest, Port Dest>
For TCP, <IP Source, Port Source, IP Dest, Port Dest>

The IPs and Ports numbers are inverted for verification of connection initiated by the other side.
Flow states are kept fresh by using a timestamp. Only recent flows are considered.
By default the "time to live" of a flow is 60 seconds. Whenever a packet is approved, because of a previous flow, the flow timestamp is updated to the timestamp of this last packet.
Outdated flow entries in the hash table are only removed when they are visited by some find operation. This is a lazy removal operation/garbage collector.

The first packet of a new flow will be checked against the rules of the firewall, and then create a new entry in the flow hash table.
TCP connection state (SYN/SYNACK/etc) will be stored in the flow table.

3) Firewall rule checking
If there are no current flow for the packet we can apply the firewall rules.
If a packet is going out, the default policy is that it is allowed to pass, otherwise if it is trying to get in, it is blocked by default.
The list of rules is traversed and only the rules that do affect the current packet are applied. The rule priority is given to the last ones. That means that multiple rules might conflict, allowing or blocking the same packet, however the last applied rule is the one that will determine the firewall action.
Based on the resulting action, packets might be dropped or forwarded.
New packets create/update flow states.

ARP Handling

ARP packets are handled by the firewall.
This is essential for the discovery of the firewall MAC address by the protected machine.
The protected machine will issue a MAC request for the IP address of the firewall. It is needed to get this ARP packet and reply correctly with the MAC address of the firewall to establish a way for communication.
The reply ARP packet is built upon the request packet. The operation code is modified to ''reply'', the target field is set to the protected machine IP/MAC addresses and the sender field is set to the firewall IP/MAC addresses.

NAT Translator

TCP, UDP and ICMP packets that are going to be forwarded by the firewall will need to be reconstructed.
This is the task of the NAT Translator.
For packets going to the outside world, the source IP of the packet (which is of the protected machine) will be modified to the IP of the firewall. The source MAC address will be modified to be the firewall MAC address and the destination MAC address will be the router MAC address.
The same happens for incoming packets. Source MAC will be the firewall MAC, destination MAC will be the protected machine MAC. Destination IP will be the protected machine IP.
This is done so that packets that are gone to the internet will reply to the firewall, and then the firewall forward them to the protected machine.
IP Checksum is recomputed for every translated IP packet.

TCP and UDP packets will need also to translate their port numbers.
The approach used is to translate unseen ports by the firewall to available high ports. A variable is maintained to point to the next available port to be used.
This works to translate new ports from the inside or outside world.
An array will map ports of each side of the connection (in/out) and the firewall will know which ports it needs to replace inside the handcrafted network packets.
TCP and UDP checksum is recomputed by applying the same algorithm of the IP checksum and using pseudo headers.

Generic Hash Tables

Generic hash tables are used as the data structure for the flow lookup table.
The hash table is generic because the same struct is used:

A hash node with length and a head pointer to a node element.
Node elements with a pointer to the next element and a void pointer to a struct called “stats”.

The same hash functions calls of insert, find, update and hash-tuples are applied for TCP, UDP and ICMP flow states.
The trick is to pass the appropriate hash table (a generic one, but one for each protocol), the protocol type and a “stats” void pointer pointing to appropriate tuple of IP/Port numbers that defines a “connection”.
With the protocol type the matching and hashing function can be abstracted. The same function is called to check if a “stats” from the current packet matches a hash table “stats” of a node element, then based on the protocol type, the “stats” pointer casting and checking routines are determined on the fly.

Example:
A void* stats A is passed with a TCP tuple <IP Source, Port Source, IP Dest, Port Dest> computed from the incoming packet to the hash_find function. The function will hash the tuple pointed by the void* and determine the hash entry. Then this entry will have a list of possible collisions and for each element of this list we will check the void* stats A with the void* stats pointer of each element. If they match, this means that the tuple was already in the hash table. The match checking is done by comparing each tuple element <IP Source, Port Source, IP Dest, Port Dest>, 4 elements in this case because of TCP. If it were ICMP protocol the algorithm would behave exactly the same but the final checking would be done over <IP Source, IP Dest> and the void* stats pointer would be casted to an ICMP stats struct.

Debugging Mode

If compiled with –DDEBUGGING the DEBUGGING macro is set and then every Ethernet and IP packet seen by the firewall will be displayed on the screen showing:

```
Packet Direction
Protocol Type
Source MAC Address
Destination MAC Address
Source IP Address
Destination IP Address
```

This is helpful to debug and few auxiliary functions were created to support this mode.

Files Description

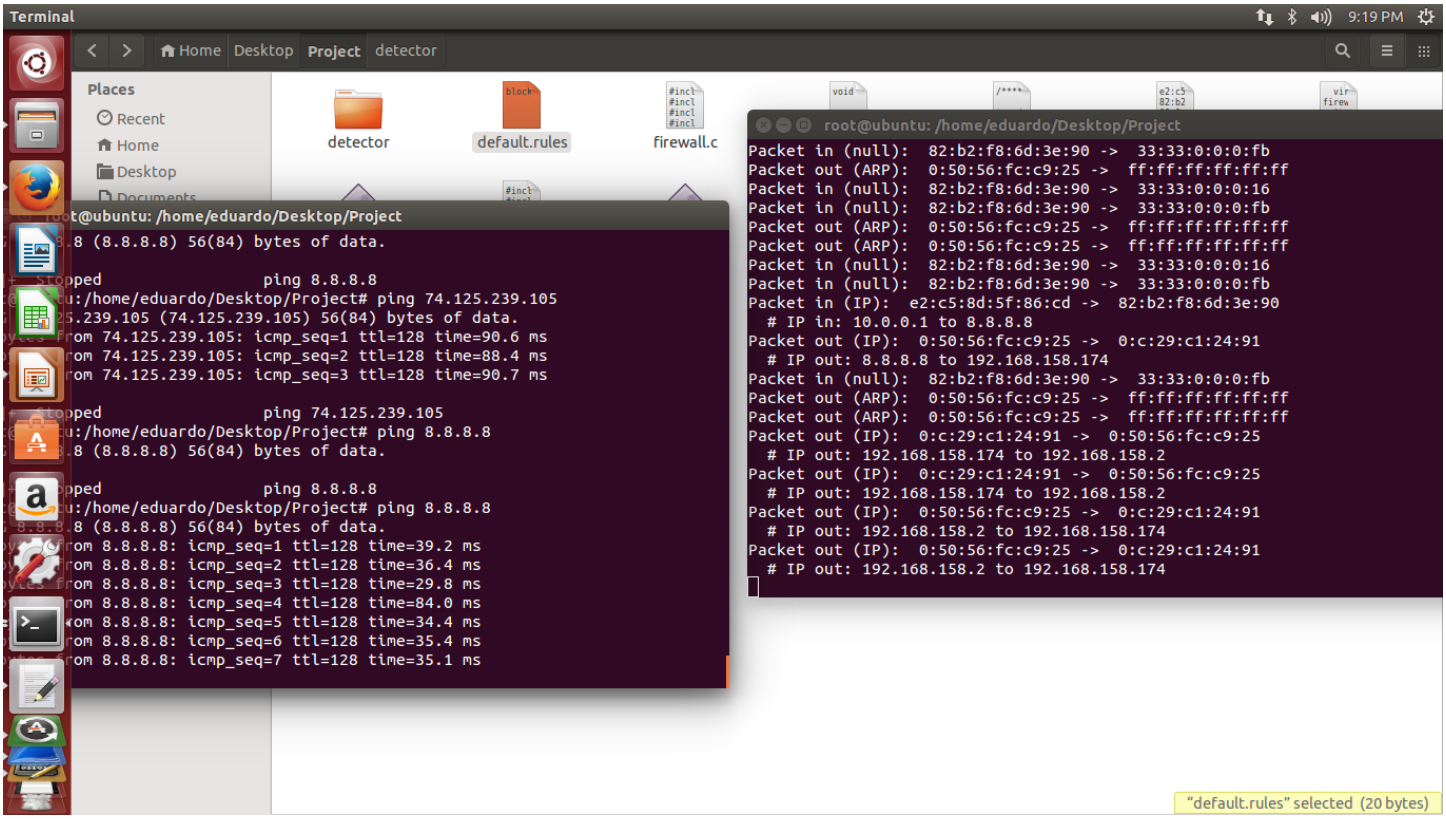
File called “headers.h” contains auxiliary headers and definitions.
File called “network.h” contains auxiliary functions.
File called “firewall.c” is the main program, which executes the firewall.
File called “default.rules” is where a user could write blocking rules for IPs.
File called “settings.config” is where information about the network (interface mac/ip addresses) are saved.

Screenshots (Debugging mode)

Allowed 8.8.8.8: Issuing pings from virtual space.

We can see a packet from protected machine to firewall (by MAC addresses). (10.0.0.1 ask for 8.8.8.8).

Then we see a packet from router to switch interface (by MAC addresses). (8.8.8.8 reply 192.168.158.174).



Firewall rule inserted, screenshot in the next page.

“block in icmp 8.8.8.8 any -> HOME any” – added rule.
We can see a packet from protected machine to firewall (by MAC addresses). (10.0.0.1 ask for 8.8.8.8).
But this time, packets are blocked by firewall, and so we never get a reply from the internet (they are dropped by the firewall).