Programming Assignment #5

Firewall

---

1. Introduction

Firewalls are common place in information security. They are the first defense against attackers. Firewalls are used to monitor network activity and to allow only specific network traffic while blocking unwanted network traffic.

In this programming assignment, you will implement a simple firewall using pre-defined APIs.

Writing a firewall requires access to subsystems within the operating system.

We'll provide the APIs for MS-Windows and Linux; if you want to do your project using a MAC, then you're on your own.

No matter what system you are running, make sure that when testing your code, all other firewall software is disabled, as this will interfere with the correct testing of your code. Also, you only need to worry about blocking incoming traffic.

2. Assignment

Since you are given the API, writing the actual firewall shouldn't be difficult. One possible implementation is to create a structure to store new rules that are added, then have a function that parses this structure and adds the rules to the firewall.

The details for each part of the program are below, along with examples of what the console input and output should look like.

IMPORTANT! Your program must perform, behave, and look *exactly* like the example input and output given below. This includes all user input, user output, and file I/O. This also includes specific messages, e.g. `Firewall Started!` is not the same as `Success!` or `Started!`. Points will be deducted for non-compliance.

2.1. Console Interface

You are to write a console program that acts as an interface to the API. The program *must* be named **consolefw**. It will have a menu that allows users to do the following:

1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit

Your menu should look exactly like the menu provided with the examples below.

2.2 Add Rule (10%)

Prompt the user for source IP address, netmask, and port, and destination IP address, netmask, and port. Also, prompt the user for the protocol, and whether the packets should be dropped or accepted. The added rule(s) should not take effect until the firewall is started. If the firewall is already started, added rules should not take effect until it is restarted. If rule is added correctly, respond `Rule added!`, or else respond `Error! Unable to add rule!`, then give an appropriate error message.

Example - add rule to the firewall to block incoming SSH traffic (the $ sign is the command line prompt): In all the examples below the $ sign is the command line prompt and lines beginning with the # sign are comments that will be ignored by consolefw.

```
$ ./consolefw
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example - add rule to the firewall to block incoming SSH traffic
1
Enter Source IP(0.0.0.0 if all):192.168.0.2
Enter Source Netmask:255.255.255.0
Enter Source Port(0 for all):0
Enter Destination IP(0.0.0.0 if all):0.0.0.0
Enter Destination Netmask:0.0.0.0
Enter Destination Port(0 for all):22
Enter Protocol(0-All, 1-ICMP, 6-TCP, 17-UDP):6
Enter Action(0-Accept, 1-Drop):1
Rule added!

1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example - add rule to block all ICMP traffic (including ping echo requests)
1
Enter Source IP(0.0.0.0 if all):0.0.0.0
Enter Source Netmask:0.0.0.0
Enter Source Port(0 for all):0
Enter Destination IP(0.0.0.0 if all):0.0.0.0
Enter Destination Netmask:0.0.0.0
Enter Destination Port(0 for all):0
Enter Protocol(0-All, 1-ICMP, 6-TCP, 17-UDP):1
Enter Action(0-Accept, 1-Drop):1
Rule added!

1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
```

```
# Example - add rule to block UDP traffic on port 138 from specific IP
1
Enter Source IP(0.0.0.0 if all):192.168.0.2
Enter Source Netmask:255.255.255.0
Enter Source Port(0 for all):0
Enter Destination IP(0.0.0.0 if all):0.0.0.0
Enter Destination Netmask:0.0.0.0
Enter Destination Port(0 for all):139
Enter Protocol(0-All, 1-ICMP, 6-TCP, 17-UDP):17
Enter Action(0-Accept, 1-Drop):1
Rule added!

1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example of format error
1
Enter Source IP(0.0.0.0 if all):abc.de.f.gh
Enter Source Netmask:255.255.255.0
Enter Source Port(0 for all):0
Enter Destination IP(0.0.0.0 if all):192.168.0.1
Enter Destination Netmask:255.255.255.0
Enter Destination Port(0 for all):139
Enter Protocol(0-All, 1-ICMP, 6-TCP, 17-UDP):17
Enter Action(0-Accept, 1-Drop):1
Error! Unable to add rule!
Source IP is invalid
```

2.3 View Rules (10%)

Print rules in the order they were entered. Make sure to number rules, so the user can tell what rule they want to delete if needed.

The output should be in this format:

```
<Rule#>-<SIP>/<SMask>:<SPort> <DIP>/<DMask>:<DPort> <Protocol> <Action>
```

Example

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example - print existing rules
3
1-192.168.0.2/255.255.255.0:0 0.0.0.0/0.0.0.0:22 6 1
2-0.0.0.0/0.0.0.0:0 0.0.0.0/0.0.0.0:0 1 1
3-192.168.0.2/255.255.255.0:0 0.0.0.0/0.0.0.0:139 17 1

1. Add Rule
2. Delete Rule
```

```
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
```

2.4 Save Rules to File (10%)

Save the rules to a file named rules.txt, if file exists, overwrite it. Your file must be in the following format:

```
<SIP>/<SMask>:<SPort> <DIP>/<DMask>:<DPort> <Protocol> <Action>
```

Example

```
$ cat rules.txt
192.168.0.2/255.255.255.0:0 0.0.0.0/0.0.0.0:22 6 1
0.0.0.0/0.0.0.0:0 0.0.0.0/0.0.0.0:0 1 1
192.168.0.2/255.255.255.0:0 0.0.0.0/0.0.0.0:139 17 1
```

If the file saves correctly, you should give the response `Rules Saved!`, if the file does not save correctly, give the response `Error! Unable To Save Rules!`, and an explanation why.

Example

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example - save rules to file
6
Rules Saved!

1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example - error message when rules cannot be saved
6
Error! Unable To Save Rules!
Cannot open file!
```

2.5 Delete Rule (10%)

"Delete Rule" expects the rule number of the rule to be deleted. The deleted rule(s) should not take effect until the firewall is started. If the firewall is already started, deleted rules should not take effect until it is restarted. If rule is deleted correctly, print `Rule deleted!`, otherwise print `Error! Unable to delete rule!`, together with a reason why.

Example

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example - print existing rules
3
1-192.168.0.2/255.255.255.0:0 0.0.0.0/0.0.0.0:22 6 1
2-0.0.0.0/0.0.0.0:0 0.0.0.0/0.0.0.0:0 1 1
3-192.168.0.2/255.255.255.0:0 0.0.0.0/0.0.0.0:139 17 1

1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example - delete rule
2 1
Rule deleted!

1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example - print existing rules
3
2-0.0.0.0/0.0.0.0:0 0.0.0.0/0.0.0.0:0 1 1
3-192.168.0.2/255.255.255.0:0 0.0.0.0/0.0.0.0:139 17 1

1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example - error when trying to delete rule
2 7
Error! Unable to delete rule!
Rule does not exist.
```

2.6 Load Rules From File (20%)

Load rules from the file named *rules.txt*. If the rules load successfully, then print `Rules Loaded!`, otherwise if there is an error, then print `Error! Unable to load rules!` and an explanation as to why. The rule(s) should not take effect until the firewall is started. If the

firewall is already started, rules should not take effect until the firewall is restarted.

Example

```
$ ./consolefw
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# In the beginning there are no rules
3
No rules to print!

1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Load rules from file
7
Rules Loaded!

1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Print existing rules
3
1-192.168.0.2/255.255.255.0:0 0.0.0.0/0.0.0.0:22 6 1
2-0.0.0.0/0.0.0.0:0 0.0.0.0/0.0.0.0:0 1 1
3-192.168.0.2/255.255.255.0:0 0.0.0.0/0.0.0.0:139 17 1
```

Here are some examples of errors when loading rules

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Error loading rules from file when rules file does not exist
7
Error! Unable to load rules!
File rules.txt not found.

1. Add Rule
2. Delete Rule
3. Print Rules
```

```
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Error loading rules due to bad format
7
Error! Unable to load rules!
Error parsing line 2.
```

2.7 Start Firewall (20%)

Activate all rules that are currently in the rules list. If the activation is successful, then print `Firewall Started!`, otherwise print `Error! Unable to start firewall!`, with an explanation why.

Example

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
4
Firewall Started!
```

Here is an example of this command failing

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example of error when trying to start the firewall
4
Error! Unable to start firewall!
DRV_ERROR_IO
```

2.8 Stop Firewall (20%)

Deactivate firewall and return network to normal state. If deactivation works, then print `Firewall Stopped` otherwise print `Error! Unable to stop firewall!`, with appropriate error message.

Example

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
```

```
8. Quit
5
Firewall Stopped!
```

Here is an example of this command failing

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Example of error when trying to stop the firewall
5
Error! Unable to stop the firewall!
DRV_ERROR_IO
```

2.9 Quit

Return the network to its original condition, and exit gracefully. Any rules that have been added but not saved will be quietly discarded.

Example

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
8
Bye now!
$
```

2.10 Required Document (5%)

In addition to your source code and your output you are required to submit a memo to your instructor that is flawlessly written. No spelling errors. No grammatical errors, etc. If the document is deemed unprofessional, e.g. a significant number of grammatical or spelling errors, then it will be assigned a grade of zero.

If you're not sure about the format for a memo, then just search for "memo format" on Google.

Your memo should state clearly the status of the assignment. Is it done or not? Does it meet all of the requirements? If anything is missing then state clearly what is missing from your work.

Failure to give a status for the assignment will result in a grade of zero for the entire assignment. Providing a status that's false or significantly misleading will also result in a zero for the entire assignment.

Make sure your document is well-written, succinct, and easy to read. If you encountered problems with the assignment, then provide a detailed description of those problems and the solution(s) you found.

2.11 Assignment Submission (10%)

You are required to submit your work online, using the Blackboard. Late work will be accepted, however it is subject to late penalties as described in the syllabus.

Here are the requirements for your submission:

- Submit your electronic copy using the Blackboard (attach the assignment as a compressed archive file (.zip, .tgz, .tbz2, .rar)
- The name of the compressed archive should be: *fistName-lastName*-PA-*assignmentNumber*.zip (e.g. Jane-Doe-PA-1.zip)
- Include (i) your memo, (ii) a completed handout with your results and your name on it, (iii) the source code, (iv) a README file that explains how to build and on how to run each program, (v) compiled executable(s) and (vi) a shell script or batch file that will compile your programs when executed; a Makefile (see GNU Make for details) would be great, however any form of script or building tool will do
- Include your e-mail address in the Comment field when submitting the assignment through the Digital Drop Box
- If for any reason you are submitting the assignment more than once, indicate this in the Comment field by including the word COMPLEMENT

3. API Details

3.1 Windows

To start with, you are going to need at minimum, the following headers:

```
#include <windows.h>
#include <winioctl.h>
#include "TDriver.h"
#include "DrvFltIp.h"
#include "sockutil.h"
```

Also, in your main program you are going to want the following global variables:

```
// The protocols MUST be defined as these values
const unsigned int ALL_PROTOCOL=0;
const unsigned int ICMP_PROTOCOL=1;
const unsigned int TCP_PROTOCOL=6;
const unsigned int UDP_PROTOCOL=17;

// These actions MUST be defined as these values
const unsigned int ACCEPT_ACTION=0;
const unsigned int DENY_ACTION=1;

// Low level driver
TDriver ipFltDrv;

// Interface to low level driver
TDriver filterDriver;
```

The firewall is controlled through the filterDriver, by using function member:

```
DWORD WriteIo(DWORD code, PVOID buffer, DWORD count);
```

- `code`: is a DWORD from one of the following(found in DrvFltIp.h):
  - START_IP_HOOK - starts the firewall
  - STOP_IP_HOOK - stops the firewall

    - ADD_FILTER - adds rule to firewall
    - CLEAR_FILTER - clears all rules from the firewall
- `buffer`: is a pointer to a structure; it is only needed when using the ADD_FILTER code, and is NULL otherwise.
- `count`: is the size of the buffer, and again is only needed when using the ADD_FILTER code, and is 0 otherwise.

If an error occurs during the operation, the WriteIo function will return error code DRV_ERROR_IO,

All code meanings should be straight forward, except for the ADD_FILTER. In order to add a filter (i.e. rule) to the firewall, you need to first create a compatible structure of the following form (found in DrvFltIp.h):

```
typedef struct filter
{
  USHORT protocol;
  ULONG sourceIp;
  ULONG destinationIp;
  ULONG sourceMask;
  ULONG destinationMask;
  USHORT sourcePort;
  USHORT destinationPort;
  BOOLEAN drop;
} IPFilter, *PIPFilter;
```

So you would simply create a new filter:

```
IPFilter testFilter;
```

Fill in the associated values, then use &testFilter as your buffer argument, and sizeof(testFilter) as your count argument. Below are snippets of code that will get you started.

```
//Initializing of driver

// Loads low level driver, dont worry about error
ipFltDrv.LoadDriver("IpFilterDriver", "System32\\Drivers\\IpFltDrv.sys", NULL,
TRUE);
ipFltDrv.SetRemovable(FALSE);

// Acts as interface to low level driver, error if != DRV_SUCCESS
filterDriver.LoadDriver("DrvFltIp", NULL, NULL, TRUE);

// Example of adding filter to firewall
bool AddFilterToFirewall(string srcIp,
          string srcMask,
          unsigned short srcPort,
          string dstIp,
          string dstMask,
          unsigned short dstPort,
          unsigned int protocol,
          int action)
{
  // Filter structure
  IPFilter pf;

  // Protocol, UDP, TCP, ICMP, or all
  pf.protocol = protocol;
```

```
  // Destination IP and Mask, must be converted to long
  // If inet_addrt returns -1, then the address could not be
  // converted, your code MUST check for this!
  inet_addrt(dstIp.c_str(), &pf.destinationIp);
  inet_addrt(dstMask.c_str(), &pf.destinationMask);

  // Same for src IP and Mask
  inet_addrt(srcIp.c_str(), &pf.sourceIp);
  inet_addrt(srcMask.c_str(), &pf.sourceMask);

  // Convert to destination and source port to short
  pf.destinationPort = htons(dstPort);
  pf.sourcePort = htons(srcPort);

  // Action, either 1-Drop, or 0-Accept
  pf.drop = action;

  // Add the rule to firewall, error if == DRV_ERROR_IO
  filterDriver.WriteIo(ADD_FILTER, &pf, sizeof(pf));
}

// Activate the firewall, error if == DRV_ERROR_ERROR_IO
filterDriver.WriteIo(START_IP_HOOK, NULL, 0);

// Deactivate the firewall, error if == DRV_ERROR_IO
filterDriver.WriteIo(STOP_IP_HOOK, NULL, 0);

// Clears all firewall rules, error if == DRV_ERROR_IO
filterDriver.WriteIo(CLEAR_FILTER, NULL, 0) == DRV_ERROR_IO)
```

If you are using the Windows API, you need to download consolefw_student.zip, which contains the API, default source files, and MSVC++ workspace files. This should get you started in the right general direction.

3.2 Linux

If you are using Linux, you have two options, you can either use iptables directly by using system() calls, or you can use the associated kernel API with the netfilter modules. I strongly recommend you simply use iptables.

In order to use iptables, you will first need to compile in the correct module support; read the iptables HOWTO in order to get started.

Below are some examples of using iptables taken from a firewall script.

```
# IPTables Location - adjust if needed
IPT="/sbin/iptables"

# Stop firewall by flushing any existing rules or chains,
# this is probably really more than you will have to do, but just to be safe
# Reset Default Policies
$IPT -P INPUT ACCEPT
$IPT -P FORWARD ACCEPT
$IPT -P OUTPUT ACCEPT
$IPT -t nat -P PREROUTING ACCEPT
$IPT -t nat -P POSTROUTING ACCEPT
$IPT -t nat -P OUTPUT ACCEPT
$IPT -t mangle -P PREROUTING ACCEPT
$IPT -t mangle -P OUTPUT ACCEPT
```

```
# Flush all rules
$IPT -F
$IPT -t nat -F
$IPT -t mangle -F

# Erase all non-default chains
$IPT -X
$IPT -t nat -X
$IPT -t mangle -X

# Start firewall, by default we just ACCEPT everything
$IPT -P INPUT ACCEPT
$IPT -P OUTPUT ACCEPT

# Create user chains to reduce the number of rules each packet
# must traverse.
# Create separate chains for icmp, tcp (incoming and outgoing),
# and incoming udp packets.
$IPT -N icmp_packets
# Used for UDP packets inbound from the Internet
$IPT -N udp_inbound
# Used to allow inbound services if desired
# Default fail except for established sessions
$IPT -N tcp_inbound

# Example of blocking all UDP traffic to port 137 and 138 to stop NETBIOS calls
from all
$IPT -A udp_inbound -p UDP -s 0/0 --destination-port 137 -j DROP
$IPT -A udp_inbound -p UDP -s 0/0 --destination-port 138 -j DROP

# Accept DHCP traffic from all
$IPT -A udp_inbound -p UDP -s 0/0 --source-port 67 --destination-port 68 -j
ACCEPT

# Accept HTTP and HTTPS TCP traffic from all
$IPT -A tcp_inbound -p TCP -s 0/0 --destination-port 80 -j ACCEPT
$IPT -A tcp_inbound -p TCP -s 0/0 --destination-port 443 -j ACCEPT
```

For more information on how to set specific destination IPs, source ports, and blocking outbound traffic please read the manual pages for iptables.

Also, you could use http://easyfwgen.morizot.net/gen/ to generate iptables firewall scripts.

3.3 Interfacing With Java

Some of you may want to write **consolefw** in Java and may have been wondering how to interface your Java application with something like iptables. The follwoing examples show you how to execute a system command from within a Java application.

Here is an example for how to execute a command, read the output of that command and printed to the console.

```
import java.io.*;

public class ReadOutputFromCommand {
  static public void main(String[] args) {
    try {
      String command = "ls -l /tmp";
// exec(command) will start a child process to execute the command
```

```
      Process child = Runtime.getRuntime().exec(command);
      InputStreamReader isr = new InputStreamReader(child.getInputStream());
      BufferedReader in = new BufferedReader(isr);
      String line = null;
      while ((line=in.readLine()) != null) {
        System.out.println(line);
      }
      in.close();

// wait for child process to exit and then print exit status
      int exitVal = child.waitFor();
      System.out.println("Exited with error code "+exitVal);
    }
    catch (Exception e) {
      System.out.println(e.toString());
      e.printStackTrace();
    }
  }
}
```

The following example shows how to pass data to a command being executed with exec().

```
import java.io.*;
public class WriteDataToCommand {
  static public void main(String[] args) {
    try {
      int c;

      String command = "cat";
      Process child = Runtime.getRuntime().exec(command);
      OutputStream out = child.getOutputStream();
      out.write("Hello world!".getBytes());
      out.close();
      int exitVal = child.waitFor();
      System.out.println("Exited with error code "+exitVal);
    }
    catch (Exception e) {
      System.out.println(e.toString());
      e.printStackTrace();
    }
  }
}
```

4. Protocol Definitions

4.1 TCP: Transmission Control Protocol

TCP is a transport layer protocol defined in IETF RFC 793. It used for connections where data integrity is required.

TCP has three phases:

- connection establishment
- data transfer
- connection termination

For connection establishment TCP uses a 3-way handshake. This works by the client sending the server a SYN packet, to which the server replies with a SYN/ACK packet, and finally the client will send a ACK packet to the client to finish the transaction. SYN and ACK are two types

of flags that can be set within TCP packet.

A TCP packet can be defined as follows:

```
typedef struct _TCPHeader
{
  USHORT sourcePort;
  USHORT destinationPort;
  ULONG sequenceNumber;
  ULONG acknowledgeNumber;
  UCHAR dataoffset;
  UCHAR flags;
  USHORT windows;
  USHORT checksum;
  USHORT urgentPointer;
} TCPHeader, *PTCPHeader;
```

TCP takes advantage of many things to ensure data transmission, including: sequence numbering for ordering of packets, acknowledgement numbers for detecting lost traffic, and checksums for checking integrity of the data.

4.2 UDP: User Datagram Packet

UDP is a transport layer protocol defined in IETF RFC 768. Its main uses are for applications when data integrity is not important, such as video and audio streaming. UDP contains no error checking like TCP, therefore UDP packets are smaller, which allows more data to be sent.

UDP can be defined by the following structure:

```
typedef struct _UDPHeader
{
  USHORT sourcePort;
  USHORT destinationPort;
  USHORT len;
  USHORT checksum;
} UDPHeader, *PUDPHeader;
```

The source port does not need to be sent, since for some applications, like streaming, the server will not expect a reply from the client. The checksum may also be skipped, but typically is always used. UDP packets also contain no ability to transmit flags, so they may not be used for connection negotiating.

4.3 ICMP: Internet Control Message Protocol

ICMP is completely different from TCP and UDP. Unlike the TCP/UDP, ICMP messages are constructed before being encapsulated at the IP level. They are used for diagnostic and routing purposes. Tools such as ping and traceroute use them in order to discover details about a network.

Each ICMP packet contains an 8-bit control message, which gives the meaning of the packet. Examples of the first few control messages are as follows:

```
0 - Echo Reply
1 - Reserved
2 - Reserved
3 - Destination Unreachable
4 - Source Quench
5 - Redirect Message
```

```
6 - Alternate Host Address
7 - Reserved
8 - Echo Request
9 - Router Advertisement
10 - Router Solicitation
11 - Time Exceeded
12 - Parameter Problem
```

Since the ICMP is encapsulated within the IP datagram, it too can not guarantee delivery.

5. Testing

The following is a guide on how your program will be graded, and a good way of testing your program. The first thing your program will need to do is be able to block ICMP pings, from all hosts, and from a specific host. The only difference is that for all hosts, you would use 0.0.0.0 and for for a specific host, you would just use whatever their IP is. So lets add a rule for blocking all traffic:

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Add rule to block all ICMP traffic (including ping echo requests)
1
Enter Source IP(0.0.0.0 if all):0.0.0.0
Enter Source Netmask:0.0.0.0
Enter Source Port(0 for all):0
Enter Destination IP(0.0.0.0 if all):0.0.0.0
Enter Destination Netmask:0.0.0.0
Enter Destination Port(0 for all):0
Enter Protocol(0-All, 1-ICMP, 6-TCP, 17-UDP):1
Enter Action(0-Accept, 1-Drop):1
Rule added!
```

Don't forget to (re)start the firewall after you add new rules!

The best way to test to see if ICMP blocking works, is to ping your machine from a separate IP. If no results are returned, it works!

The next metric you will be tested on is blocking TCP traffic. You will need to be able to block single ports, and all ports from one host, or all hosts. For our example, lets block all FTP traffic from all hosts.

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Add rule to block all FTP traffic from all hosts
1
Enter Source IP(0.0.0.0 if all):0.0.0.0
Enter Source Netmask:0.0.0.0
```

```
Enter Source Port(0 for all):0
Enter Destination IP(0.0.0.0 if all):0.0.0.0
Enter Destination Netmask:0.0.0.0
Enter Destination Port(0 for all):21
Enter Protocol(0-All, 1-ICMP, 6-TCP, 17-UDP):6
Enter Action(0-Accept, 1-Drop):1
Rule added!
```

To test TCP traffic, the best tool to use is GNU Netcat. Once netcat is installed you can have it listen on your local machine by using the following command:

```
nc -l -p 21
```

Now from a remote machine, try connecting into your localmachine using netcat:

```
nc <IP address> 21
```

Now type a few characters and hit Enter. If nothing is seen on the local machine, then the traffic was blocked! You can also do the same procedure with UDP. So lets add a rule block traffic on port 50.

```
1. Add Rule
2. Delete Rule
3. Print Rules
4. Start Firewall
5. Stop Firewall
6. Save Rules To File
7. Load Rules From File
8. Quit
# Add rule to block all FTP traffic from all hosts
1
Enter Source IP(0.0.0.0 if all):0.0.0.0
Enter Source Netmask:0.0.0.0
Enter Source Port(0 for all):0
Enter Destination IP(0.0.0.0 if all):0.0.0.0
Enter Destination Netmask:0.0.0.0
Enter Destination Port(0 for all):50
Enter Protocol(0-All, 1-ICMP, 6-TCP, 17-UDP):17
Enter Action(0-Accept, 1-Drop):1
Rule added!
```

To test UDP traffic, you can use netcat again, just add the u switch, so the command on the local machine would be:

```
nc -l -u -p 50
```

Now from a remote machine, try connecting into your local machine using netcat:

```
nc -u 50
```

Now type a few characters and hit Enter. If nothing is seen on the local machine, then the traffic was blocked!

Examples like these, as well as testing with blocking of specific IPs will be the metrics used to grade your assignment.
