SDN Firewall
(Fall 2020)

Goal

In this project, you will use Software Defined Networking (SDN) to create a configurable firewall using an OpenFlow enabled switch and controller. This is beyond what is possible with traditional Layer 2 (L2) switches, and shows how the simplest of SDN switches are more capable than even the fanciest of L2 switches.

You are going to create an *externally configurable* firewall using a SDN platform named Pyretic and the POX OpenFlow Controller. That means that the firewall rules are provided in a configuration file which can be changed without altering the switch code.

This firewall is of the type that allows all traffic to pass that isn't expressly disallowed (a blacklist firewall). The alternative is to disallow all traffic that isn't expressly allowed by the policy (a whitelist firewall). A whitelist firewall is typically used in edge situations protecting internal resources from the outside/Internet (an example is a home internet router). The code we have provided for this project could be modified to implement this second type of firewall, and would be a worthwhile project for those who are interested in learning further.

Before You Start

This project assumes basic knowledge about IP and TCP/UDP Protocols. It is **highly** encouraged that you review the following items before starting. This will help you in understanding IP packets and what you need to match.

1. What is IP (Internet Protocol)? What are the different types of Network Layer protocols?
2. Review TCP and UDP. How does TCP or UDP differ from IP?
3. Examine the packet header for a generic IP protocol entry. Contrast that with the packet header for a TCP packet, and for a UDP packet. What are the differences? What does each field mean? (Reviewing the material available at the following document may assist you in this project:

https://www.airuniversity.af.edu/Portals/10/CyberCollege/documents/Portal/PDF/Hands-on%20Keyboard%205%20-%20Wireshark.pdf

4. This is a review from earlier material, but examine what constitutes a TCP connection versus a UDP connection. What happens if you block either side of a connection and what would you expect to see (for example, timeout, connection drop, or connection failed)?
5. A special IP protocol is ICMP. Why is ICMP important? What behavior happens when you do an ICMP Ping? If you block an ICMP response, what would you expect to see?
6. If you block a host from ICMP, will you be able to send TCP/UDP traffic to it?
7. Can you explain what happens if you get an ICMP Destination Unreachable response?

Project 4 Files

In the `SDN-Firewall` directory, there are many files, described below:

- `firewall-policies-bad.pol` - This is an example firewall configuration policy that is broken. When parsing, an error message will be thrown. Each line has an error of some type. Try removing lines to see the different possible error messages. This shows how the policy file is parsed by the *parse_config* function in **firewall.py**
- `firewall-policies-good.pol` - This is an example firewall configuration policy that disallows all devices to connect to TCP port 1080 on all other devices. You can use this as the base for the `firewall-config.pol` file you will generate later in the instructions. The code to implement this firewall policy is included in the `Firewall_policy.py` file.
- `Firewall_policy.py` - This is the file where you will implement the firewall using python and pyretic code, based on the policy configuration that is passed in via the configuration files.
- `firewall.py` - This is the file that sets up pyretic application and reads the firewall config policy into a data object. **DO NOT MODIFY THIS FILE**. A shell script is provided to help run it. This file contains the code

that is used to parse your firewall-config.pol file, so please look here for the import format for your firewall-config.pol.

- `firewall-topo.py` - This is a mininet program to start your topology. It consists of one switch and two groups of hosts. Modifying this file isn't necessary, but you may choose to try different topologies for testing your code (make sure that your firewall-config.pol works with the original firewall-topo.py, however).
- `pyretic_switch.py` - This implements a learning switch. You do not need to modify this file.
- `run-firewall.sh` - This script runs the firewall using pyretic (it starts the firewall.py application.) The files need to be in the pyretic directory trees, and this script makes that happen. Also, it allows for different configuration files to be used by giving the filename on the command line.
- `test-tcp-client.py` - This acts as a TCP client: opens a connection, sends a string, then waits to hear it echoed back. You can use this to test your firewall policies.
- `test-tcp-server.py` - This acts as a TCP server: listens on a specified port, echoes back whatever it hears. You can use this together with the `test-tcp-client.py` program. Note that the IP Address used for this command MUST match the IP address of the machine you run this command on.
- `test-udp-client.py` - This acts as a UDP client to test your firewall policies.
- `test-udp-server.py` - This acts as a UDP server which echoes back whatever it hears. You can use this together with the `test-udp-client.py` program. Note that the IP Address used for this command MUST match the IP address of the machine you run this command on.

References

The following documents and papers will help with completing and understanding this project:

- Pyretic Manual (especially How To Use Match and Language Basics) - https://github.com/frenetic-lang/pyretic/wiki
- Wireshark: https://www.wireshark.org/docs/wsug_html/
- Analyzing TCP Packet Blog post: https://medium.com/@eranda/analyze-tcp-dumps-a089c2644f19

- https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml for TCP and UDP Services
- http://www.firewall.cx/networking-topics/protocols/icmp-protocol/152-icmp-echo-ping.html (ICMP Ping)

Instructions

IMPORTANT: Read all the instructions carefully! Then read them again after you finish but before you submit, so you can verify that what you did matches what the assignment says ***exactly****. Even seemingly small details in the instructions can be very important! Some of those details exist to allow our grading code to interface with your project properly.* ***You will lose points if your project does not work with our grader because you didn't follow the instructions.*** *We do not arbitrarily deduct points for not following directions, but you will not receive credit for your project if our grader cannot tell whether or not it's working.*

1. Before getting started, you will need to download SDN-Firewall.zip to the VM and unzip.
    - `unzip SDN-Firewall.zip`
2. Next, move to the `SDN-Firewall` directory and try out the `test-tcp-client.py`, `test-tcp-server.py`, `test-udp-client.py`, and `test-udp-server.py` tools to learn how to use them by following the following steps (you will use these techniques in future steps to test your firewall implementation):
    - Open a terminal and copy the `pyretic_switch.py` file to the pyretic modules folder:
        - `cp pyretic_switch.py ~/pyretic/pyretic/modules`
    - Start the `pyretic_switch.py` file:
        - `cd ~/pyretic`
        - `python pyretic.py pyretic.modules.pyretic_switch`
    - In a second terminal, start the topology in the `SDN-Firewall` directory:
        - `sudo python firewall-topo.py`

      After running this step (or in Step 3 at the same point), you may see the following error in the first terminal window:

      ```
      ERROR:core:Exception while handling OpenFlowNexus!PortStatus...
      Traceback (most recent call last):
       File "/home/mininet/pox/pox/lib/revent/revent.py", line 231, in raiseEventNoErrors
         return self.raiseEvent(event, *args, **kw)
       File "/home/mininet/pox/pox/lib/revent/revent.py", line 278, in raiseEvent
         rv = event._invoke(handler, *args, **kw)
       File "/home/mininet/pox/pox/lib/revent/revent.py", line 156, in _invoke
         return handler(self, *args, **kw)
       File "/home/mininet/pyretic/of_client/pox_client.py", line 577, in _handle_PortStatus
         self.switches[event.dpid]['ports'][event.port] = event.ofp.desc.hw_addr
      KeyError: 1
      ```

      Exit Mininet by typing in `quit` and then run `sudo ./cleanup.sh` and repeat this step.

- At the Mininet prompt, you need to open up a couple of new terminals:
  - `xterm e1`
  - `xterm e2`

- **HINT: All IP and MAC Addresses are assigned in order based on the alphabetical order of the host names. So client1 is 10.0.0.1 (MAC 00:00:00:00:00:01) and e1 is 10.0.0.2 (MAC 00:00:00:00:00:02). The test-*-* commands below MUST use the IP Address and NOT the hostname.**

- In the e1 terminal, start the server:
  - `python test-tcp-server.py 10.0.0.2 1234`
  - The parameters are the **server's IP and port**. You can run `ifconfig` in the xterm window to check the IP.
  - The server simply uses an infinite loop around the accept function, so you can use `Ctrl-C` to kill it when you're done running all the steps including running `test-tcp-client.py`.
  - The UDP server works in the same manner.

- In the e2 terminal, start the client:
  - `python test-tcp-client.py 10.0.0.2 1234`
  - The parameters here are the **server's IP and port**, and should be the same as specified above.
  - Likewise, the UDP client functions in a similar manner to the TCP client.

  You can stop the test-*-*.py scripts by pressing Control-C.

  If you get the following error, you may ignore it - it is caused by the MPTCP kernel installed on the VM:

  ```
  INFO:packet:(tcp parse_options) warning, unknown option 1e
  ```

- **HINT - You must use the server IP address and Port for both the test-*-client.sh and test-*-server.sh commands. You need to make sure that you run the test-*-server.sh script on the xterm/host that you designate as the "server" and test-*-client.sh on the xterm/host that you designate as the "client". If you get this backwards, you will get an error on your "client" that an IP address could not be assigned/used. Why is this?**

- You'll be using this quite a bit, so feel free to play around with it for a bit before moving onto creating the firewall.

- After you have finished this step, exit out of mininet and then run the `sudo ./cleanup.sh` command to kill pyretic. The cleanup script will kill mininet and any orphaned python/pox/pyretic processes that may have been left running after a potential crash.

3. The following set of steps shows how to start and run the firewall. This step will use the one included match rule in `firewall_policy.py` file and the `firewall-policies-good.pol` file. The code in firewall_policy.py will implement a minimal firewall that will block TCP port 1080 for all hosts. (**NOTE THAT ONCE YOU START THIS STEP, YOU WILL NOT BE ABLE TO RE-RUN STEP 2 WITHOUT ERRORS**)

- Make sure that you return to the SDN-Firewall directory in the first terminal window.

- In the first terminal, start the firewall as specified below. Please provide the firewall-policies-*.pol file that you want to test as the second parameter. **DO NOT RUN THIS AS ROOT (i.e., sudo).**
  - `./run-firewall.sh firewall-policies-good.pol`

    Note: You may get the following error message when you run this command that you can safely ignore: **Couldn't import dot_parser, loading of dot files will not be possible.**

- In a second terminal, start the topology:
  - `sudo python firewall-topo.py`

- At the mininet prompt, open client and server terminals:
  - `xterm e1`
  - `xterm e2`

- In the e1 terminal, start the **server**:
  - `python test-tcp-server.py 10.0.0.2 1234`
  - The UDP server can be used in the same manner.
  - Note that the IP address used in the server command MUST match the IP of the machine you are connected to (in this case, e1).

- In the e2 terminal, start the client:
  - `python test-tcp-client.py 10.0.0.2 1234`
  - The UDP server can be used in the same manner. Note that the IP and port here should be that of the server.

  Using the provided good firewall-policies-good.pol files, the connections above (port 1234) should connect and pass traffic.

- Stop the test client/server from the last two steps.

- In the e1 terminal, start the server:
  - `python test-tcp-server.py 10.0.0.2 1080`
  - The UDP server can be used in the same manner.

- In the e2 terminal, start the client:
  - `python test-tcp-client.py 10.0.0.2 1080`
  - The UDP server can be used in the same manner.

By default, the firewall_policy.py file has been hardcoded to block TCP connections to TCP Port 1080 on all servers. If you test the UDP server/client, you should be able to connect (test this functionality by replacing tcp with udp in the `test-tcp-server.py` and `test-tcp-client.py` filename). The `firewall-policies-good.pol` shows the necessary configuration to block TCP Port 1080.

- Again, like in Step 2, exit out of mininet and then run `sudo cleanup.sh` command to kill pyretic. It is a good practice to do this after every attempt to run the firewall.

A blocked connection will fail in many different methods depending on the way that you implemented the firewall (See #4, #5, and #7 from the Before You Start section. As long as the connection does not connect and pass traffic, it will be considered OK by the autograder as being blocked.

4. Now you will create the firewall implementation by editing `firewall_policy.py` and `firewall-config.pol` file. The `firewall_policy.py` file should implement code that will match all the permutations of entries in a possible configuration file (as specified below) without hardcoding a solution (i.e., do not specify specific addresses, and ports in your python implementation field - use the data passed to it from your firewall configuration file - if you do hardcode the configuration, you will lose many points when your implementation code is exercised with other topologies and rulesets). The `firewall-config.pol` file will contain the ruleset specified later in this document. It is suggested that you build this file in a stepwise approach to help you code your solution for the implementation code.

`firewall-config.pol` file Specifications:

The format of the parsed configuration file (i.e., `firewall-config.pol`) is as follows:

rulenum, source MAC, destination MAC, source IP, destination IP, source Port, destination Port, Protocol, IPProtocol

Each field is described as follows:

- **rulenum** exists to help you debug your code. If a rule is "invalid", the parser will give you the number you specify.
- **MAC Addresses** are specified in the form of 00:00:00:00:00:01 or "**-**" if you are not specifying it.
- **IP Addresses** are specified in the form of 10.0.0.1/24 in CIDR notation. "**-**" if you are not specifying an IP address.
- **Port numbers** are integers with a valid range of 0 to 65535. If a port is specified, you must also specify the protocol for TCP or UDP. If no port is specified, use a "**-**" to indicate that you are not matching this field
- For **Protocol**, use **T** for TCP IP protocol, **U** for UDP IP protocol, **I** for ICMP, and **O** for other IP Protocols that aren't TCP/UDP/ICMP (i.e., IP Protocols for GRE, IPSEC, etc).

Remember, you must specify a protocol if you specify a Port number. If you are not specifying a protocol, supply a "-" for this field.

- ○ The **IPProtocol** field is ONLY used if you specify **O** in the protocol field. This field should contain the integer value of the particular IP Protocol that you are matching. If you do not use a value of **O** for the protocol field, this field should have a value of "-".

General Notes

- ○ Hostnames are not allowed to be used in the configuration file (i.e, you must specify an IP address or MAC address in the appropriate field). **All IP and MAC Addresses are assigned in numerical order based on the alphabetical order of the host names.**
- ○ Remember that every field can have a "-" as it's value. "-" indicates that you are not matching traffic with that particular element. For instance, if you want to match only a destination IP address, your rule would be **1,-,-,-,destip,-,-,-,-**.
- ○ You can get the IP Address or MAC address for any host by typing in hostname ifconfig in the mininet CLI. For example, for host w1, use the command inside your mininet terminal: **`w1 ifconfig`**
  The IP Addresses and MAC addresses are defined in order alphabetically based on the host name. You can also get a summary by issuing the following command in your mininet terminal: **`dump`**
- ○ IP addresses do not change from machine to machine. The address for e1 will always be the same on every machine unless you change the firewall-topo.py file.

`firewall_policy.py` file Specifications:

General Notes for Your Implementation:

- **NOTE: Your implementation code DOES NOT NEED to validate that the format of the rules from the firewall-config.pol file is correct. Validation logic has been provided in firewall.py. Your code will NOT be tested with an invalid configuration file or with any rules that are considered invalid based on this documentation**
- Translating the protocol entries to their numerical value or constant is not considered hardcoding.

- The code you generate **must support all possible permutation of items** in the firewall configuration file. For example, you should be able to handle a source IP address and a destination MAC address. Also note that some parameters may need additional information. Refer to the Pyretic How to Match (https://github.com/frenetic-lang/pyretic/wiki/How-to-use-match) and Language Basics (https://github.com/frenetic-lang/pyretic/wiki/Language-Basics) for the information needed to fill in the code in firewall-policy.py file.
- This project does not include IPv6 support. You may assume that all traffic is IPv4 only.
- Your code may use any particular library installed on the VM, but it should not require any third party libraries to complete.
- Comments or debug lines will not adversely impact the autograder. The "print config" code also will not adversely impact the autograder.
- You may make any changes you need to the *make_firewall_policy* function, but it is recommended to drop any code you need in the #TODO areas. You may or may not need both areas. You may also make auxiliary functions in this file. DO NOT ADD CODE TO OTHER FILES.
- In writing your implementation code, you may use any Pyretic functions. **However, beware that using the >>, +, and | operators may cause unexpected behavior. It is fairly trivial to implement the project without using any of these operators. However, if you do use these, make sure you carefully and fully test your firewall implementation (especially in maintaining pyretic prerequisites for different items. Most point deductions for this project is as a result of using those operators.**
- To test the **"O"** protocol, consider simulating an ICMP block by specifying a rule with **O,1** for the last two entries for that rule and see if you can complete a PING request (ICMP is IP Protocol 1). If your code can handle the **O,1** case, it can handle any other **O,#** cases. You could also use a TCP or UDP rule to likewise test to make sure your implementation of "**O**" is correct. There are also possible testing solutions by using the netcat tool or modifying the test-*-*.py scripts to support other protocols. ALSO,additional methods to test certain IP Protocols will be posted in Piazza.
- Note that you are allowed to add items into the "config" list object using your code in `firewall_policy.py` if you would like.
- You may handle the empty case in any way you would like. It is not a rule that will be tested (i.e., '-' in all fields except for rulenum).

The best approach to step 4 is to start coding by picking an individual field (say, destination MAC address), write a `firewall-test-config.pol` entry to model only the destination MAC address, and then code your `firewall_policy.py` file to handle the destination MAC address case. Test out your rule by repeating the steps used in step 3 above (using `./run-firewall.sh firewall-test-config.pol` as the first command) to test your implementation for each field. Repeat for other fields. The most difficult to implement will be your protocol handling.

You are free to create additional rules and topologies that you can use to test your firewall implementations and you can share these on Piazza. You are NOT allowed to share your `firewall_policy.py` implementation or the `firewall-config.pol` that you will be creating in subsequent steps.

Your firewall-policy.py implementation file will be tested against alternate configuration files and topologies, so please make sure that your code is robust to handle different combinations of situations including using <u>all possible permutations of IP Addresses and MAC Addresses, different ports, and different protocols.</u>

5. The final step of the project is to create a firewall policy configuration file that will implement the following ruleset. This file MUST be named `firewall-config.pol`. The rules you need to implement are detailed below. Please know that you need to craft the rules in such a way that you don't over-block items (i.e., if you are asked to block TCP port 50, make sure that you do not block UDP port 50). Please refer to the `firewall-config.pol` file specifications above.

Rules to be implemented:

1. **One common implementation for a virtual private network (VPN) solution utilizes PPTP (Point-to-Point Tunnelling Protocol). It now has many issues related to the way it authenticates users. Write a series of firewall policy rules to block PPTP that will prohibit all hosts from sending traffic to a PPTP server running on server2. (One TCP port and the GRE IP Protocol)**

2. SSH (Secure SHell) is used to provide a remote shell which can be used to forward other ports or to bypass firewalls. Write firewall policy rule(s) that will prohibit all hosts from sending traffic to a SSH server on the east hosts (e1-e3) (Block only one TCP and UDP Ports).

3. One common way to perform a distributed denial of service (DDOS) attack is to use an amplification attack using the Network Time Protocol (NTP) and Domain Name Services (DNS) taking advantage of the UDP protocol to saturate the links. Write firewall policy rule(s) to protect the DNS and NTP services on both server1 and server2 from receiving traffic by all hosts. However, the DNS and NTP services on server3 should be accessible to all hosts. (Only block the UDP Ports for NTP and DNS)

4. Write a series of firewall policy rules disallowing hosts w1 and w2 from pinging client1. This rule will be satisfied if the full ping request/response is not completed. Analyzing the connections with Wireshark or tcpdump is helpful, but not required.

5. Write a series of firewall policies to disallow host e1 from sending traffic destined to TCP ports 9950-9952 on host e3.

6. Write a series of firewall policies to restrict host client1 from sending traffic to any of the east hosts (e1-e3) on both TCP and UDP protocols.

7. Another common implementation for a VPN uses L2TP/IPSEC (Layer 2 Tunneling Protocol and IPSEC). Write a series of firewall policy rules that will prohibit all hosts from sending traffic to a L2TP/IPSEC server running on server3. L2TP uses a variety of ports. The Key Exchange for IPSEC takes place over UDP Port 500, the Encapsulation Security Payload (ESP) IP Protocol is also used by IPSEC, and the actual tunnel is set up on UDP Port 1701. (2 UDP Ports and 1 ESP IP Protocol)

Notes:
All hosts mean any other possible hosts on the network.
When a protocol is specified, use the IANA Protocol Lists to find the number for the specified IP or TCP/UDP Protocol.

Common Error Conditions and How to Handle Them

If you see the following error, please run the cleanup.sh script as a sudo and try again:

```
ERROR:core:Exception while handling OpenFlowNexus!PortStatus...
Traceback (most recent call last):
 File "/home/mininet/pox/pox/lib/revent/revent.py", line 231, in raiseEventNoErrors
   return self.raiseEvent(event, *args, **kw)
 File "/home/mininet/pox/pox/lib/revent/revent.py", line 278, in raiseEvent
   rv = event._invoke(handler, *args, **kw)
 File "/home/mininet/pox/pox/lib/revent/revent.py", line 156, in _invoke
   return handler(self, *args, **kw)
 File "/home/mininet/pyretic/of_client/pox_client.py", line 577, in _handle_PortStatus
   self.switches[event.dpid]['ports'][event.port] = event.ofp.desc.hw_addr
KeyError: 1
```

If you see the following warning, you may safely ignore it:

```
WARNING:openflow.of_01:<class 'pox.openflow.PortStatus'> raised on dummy OpenFlow nexus
```

If you accidentally run the run-firewall.sh script as root, you will need to run the following commands to avoid the "access denied" error.

```
sudo chown mininet:mininet /home/mininet/pyretic/pyretic/modules/firewall_policy.py
sudo chown mininet:mininet /home/mininet/pyretic/pyretic/modules/firewall-policies.cfg
sudo chown mininet:mininet /home/mininet/pyretic/pyretic/modules/firewall.py
sudo chown mininet:mininet /home/mininet/pyretic/pyretic/modules/pyretic_switch.py
```

If you get the following error, make sure that the pyretic (run-firewall.sh) has not ended with an error message :

```
Unable to contact the remote controller at 127.0.0.1:6633
```

If you get this error, you will want to run the cleanup.sh as a superuser (sudo). If you still get this error after running cleanup, then the error is in your code - something in your code is causing pyretic to crash.

When running the firewall, you may get errors like:

```
Warning: libopenflow_01:Fields ignored due to unspecified prerequisites: nw_proto
```

If you get this error (or similar error messages like tp_dst), please note that this rule will be ignored and not processed by pyretic. This indicates that there is an error in your implementation code. Any error that indicates unspecified prerequisites indicates this. Read the **How to Use Match** pyretic document to help find the error.

When running the firewall, you may get a warning like:

```
INFO:packet:(tcp parse_options) warning, unknown option 1e
```

This error is caused by the kernel that is running on the VM (MPTCP). It can be safely ignored.

If you get the following error: `Error: pox not found in PYTHONPATH`

Double check your PYTHONPATH environment variable to see if it is as follows:

```
mininet@mininet-VirtualBox:~$ echo $PYTHONPATH
/home/mininet/pyretic:/home/mininet/mininet:/home/mininet/pox
```

If Wireshark is installed but you cannot run it (e.g. you get an error like $DISPLAY not set, please consult the FAQ: https://github.com/mininet/mininet/wiki/FAQ#wiki-x11-forwarding.)

What to turn in

For this project you need to turn in two files to Canvas in a ZIP file. Please name the zip file based on your GaTech username (i.e., gburdell3_p6.zip).

Use the following command to zip the two files below using the class VM:

zip gburdell3_p6.zip firewall_policy.py firewall-config.pol

(replace "gburdell3" in "gburdell3_p6.zip" with your actual GT username). This command will zip the two files without including the path or extraneous directories. Please test the ZIP file before submitting to Canvas to ensure that you have the proper versions of the files listed below.

- `firewall_policy.py` - The SDN firewall you created in Step 4. MANDATORY
- `firewall-config.pol` - The configuration file you created in Step 5. MANDATORY

AFTER YOU SUBMIT THE ZIP FILE TO CANVAS, PLEASE DOUBLE CHECK AND REDOWNLOAD YOUR SUBMISSION FROM CANVAS TO ENSURE THAT YOU DID NOT INADVERTENTLY SUBMIT A PREVIOUS PROJECT ACCIDENTALLY.

Please refer to Piazza Post @35 for more information on how to ZIP and submit projects.

What you can and cannot share

For this project, you can and are encouraged to share testing techniques and frameworks, your testing firewall policies, and testing topologies. What you cannot share is code for `firewall_policy.py` or examples of configuration policies that address the ruleset in Step 6 that belong in `firewall-config.pol`. See Part 1 Step 4 for more examples of what can and cannot be shared. If you have any doubt please ask the Instructors privately on Piazza before sharing.

Notes

Wireshark may be helpful for this project. You can start it from any terminal with the following command: `sudo wireshark &` (it needs to run as root to sniff traffic) and use it to look at traffic on specific ports. You may wish to start *two* instances - one either side of the switch - to see if traffic is being if actually being completed and received.

Note that you can increase the RAM or processor usage for the VM for this project to improve performance. You can also run multiple cpus in the VM for the next project, BGP Hijacking.

You are being provided a Project 4 GUI tool developed by Sam Paulissian, a former TAs for this course. You may find it helpful for testing, but you are not required to use it if you are more comfortable using the command line interface. Download the SDN-FirewallGUI zip file from Canvas and follow the instructions for use found within the folder after you unzip it using the `unzip SDN-FirewallGUI.zip` command.

Grading Policy and Rubric

Your `firewall_policy.py` will be tested with a set of known good configuration files and different topologies that are not provided. These configurations will range from simple port blocking to others that are more complex than what was defined for this project. Also, both simple and complex topologies will be used to evaluate your policy.

Your `firewall-config.pol` file will be tested for validity and functionality by using your `firewall_policy.py` file.

If you have trouble coding the `firewall_policy.py` file, at least attempt to create the `firewall-config.pol` to get partial credit.

10 pts	Correct Submission	for turning in all the correct files with the correct names, and significant effort has been made in each file towards completing the project. This will be up to 5 points for each file. The penalty will have a variance based on the effort undertaken.
45 pts	Firewall Policy	the policy in `firewall-config.pol` passes a variety of tests to ensure correct blocking of traffic per the rules, and allows all other traffic. If there are significant issues with your firewall_policy.py file, your work may be verified against a known good firewall_policy.py file.
45 pts	Firewall Implementation	the firewall implementation in `firewall_policy.py` passes a variety of tests to ensure it works properly with several different firewall configurations (i.e., different firewall rules and topologies)