CS 6250 Summer 2019

Project 4 - SDN Firewall

Goal

In this project, you will use SDN to create a configurable firewall using an OpenFlow enabled switch. This is beyond what is possible with traditional 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 Pyretic. That means that the firewall rules are provided in a configuration file, so they can be changed without altering the switch code.

This firewall is the type that allows all traffic that isn't expressly disallowed (a blacklist firewall). This is the type of firewall that you would find in an office environment where you wish to restrict access to certain resources from other devices. The alternative is to disallow all traffic that isn't allowed by the policy (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.

In the `Project4` 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 port 1080 on all 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
- IP Header Format with TCP and UDP Extensions https://en.wikipedia.org/wiki/IPv4#Header and https://en.wikipedia.org/wiki/Transmission_Control_Protocol

Instructions

DISCLAIMER: 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 Project4.zip to the VM and unzip.

- `unzip Project4.zip`

2. Next, move to the `Project4` 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:

    - 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 `Project4` directory:
        - `sudo python firewall-topo.py`
    - At the Mininet prompt, you need to open up a couple of new terminals:
        - `e1 xterm &`
        - `e2 xterm &`
    - In the e1 terminal, start the server:
        - `python test-tcp-server.py 10.0.0.1 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.
        - The UDP server works in the same manner.
    - In the e2 terminal, start the client:
        - `python test-tcp-client.py 10.0.0.1 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'll be using this quite a bit, so feel free to play around with it for a bit before moving onto creating the firewall.

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. This 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 PART 2 WITHOUT ERRORS**)

   - 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.** We are not using dot files with this project.

       If you accidentally run this 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
       ```

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

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

   - In the e1 terminal, start the server:
     - `python test-tcp-server.py 10.0.0.1 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 10.0.0.1 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 above connections 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 10.0.0.1 1080`
  - The UDP server can be used in the same manner.

- In the e2 terminal, start the client:
  - `python test-tcp-client.py 10.0.0.1 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. The `firewall-policies-good.pol` shows the necessary configuration to block TCP Port 1080.

A blocked connection will fail in many different methods depending on the way that you implemented the firewall. As long as the connection does not connect and pass traffic, it will be considered OK by the autograder as being blocked.

NOTE: A good practice is to run "sudo mn -c" and "killall python" to make sure all processes are shut down. If the error message includes the word "pox", also consider running "killall pox". A script has been added ("cleanup.sh") that will check the ownership of files and will cleanup mininet, python, and pox. This script must be run as superuser (i.e., use the sudo command)

If you see the following error, please kill mininet and all python processes 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
```

4. Now you will create the firewall implementation by editing `firewall_policy.py` and `firewall-config.pol` file. The next two steps need to completed in parallel. Using the example match block in the `firewall_policy.py` file, you will code an implementation that can implement match

and pyretic rules that will handle all of the possible rule permutations in the `firewall-config.pol` file.

- The format of the parsed configuration file is as follows:

  rulenum, source MAC, destination MAC, source IP, destination IP, source Port, destination Port, ProtocoL

  MAC Addresses are in the form of 00:11:22:33:44:55; IP Addresses are in the form of 10.0.0.1. You cannot use CIDR notation for IP Addresses; **for Protocol, use T for TCP, U for UDP, B for both TCP and UDP, and I for ICMP (see note below that "-" is a valid entry for Protocol)**. Hostnames are not allowed to be used in the configuration file. Your code will need to convert these to the appropriate protocol values. If you have questions to what is acceptable for the parsed configuration file, refer to the parse_config function in firewall.py. You may not use Hostnames in the configuration file.

  NOTE: Your implementation code DOES NOT NEED to validate that the format of the rules 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 violate the requirements for pyretic.

- **NOTE: DO NOT HARDCODE FIREWALL RULES IN THIS FILE!** The example was hard coded to demonstrate the desired behavior. The goal of this part of the assignment is to generalize and parse the configuration file to handle any particular firewall cofiguration file. After completing this part, only TCP Port 1080 should be blocked using the `firewall-policies-good.pol` file. You will not receive credit for hard-coding the firewall rules!

- The code you generate must support all items contained in the parsed 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.

- Treat "**-**" in a parsed configuration file field as if you were to ignore it. **Note that is acceptable to have a "-" in any field, including protocol.**

- 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***
  In general, 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.

- **You can assume that if a Port number is included, the protocol will be included in the configuration file.**

- Only IPV4 will be used in this project.

- 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 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 these operators. However, if you do use these, make sure you carefully test your firewall implementation. Most point deductions for this project is as a result of using those operators.

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

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

- 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), note that this rule will be ignored and not processed. There is an error in you you are parsing the data and passing it to pyretic. Read the **How to Use Match** pyretic document to help find the error.

Implementation Clarifications:

This section has been added to address edge case brought up from past semesters:

The rule:

1,-,-,-,-,-,-,- should be implemented to not block any traffic. This is consistent with the instructions to state that you need to ignore or not match "-" items.

1,-,-,-,-,-,1080,I is an invalid rule and will not be tested. Why?

Once you have coded your `firewall_policy.py` file, test it out by creating your own unique `firewall-test-config.pol` files and repeat the steps from Step 3 above (using `./run-firewall.sh firewall-test-config.pol` as the first command). Create this file using simple sets of parameters in order to test your firewall implementation. You are free to share example configuration files with other users. You are also allowed to create your own topologies and are allowed to share them with others. You are NOT allowed to share your firewall-policy.py implementation, or the `firewall-config.pol` you will be creating in subsequent steps.

Your firewall-policy.py 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 combinations of IP Addresses and MAC Addresses, different ports, and different protocols..

5. The next step of the project is to create a firewall policy configuration file in the format as specified in the previous step. 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). Also, make sure that you utilize the "-" operator to ignore a particular parameter. When you are writing your rules, you may use either IP Addresses or MAC Addresses. However your `firewall_policy.py` MUST be able to handle any combinations of IP Addresses or MAC Addresses.

Rules to be implemented:

1. **One common implementation for a virtual private network solution utilizes PPTP (Point-to-Point Tunnelling Protocol). It now has many issues related to the way it authenticates users. Write firewall policy rules to block PPTP that will prohibit all hosts from accessing a PPTP server running on server2. (TCP Port 1723)**

2. **SSH 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 connecting to a SSH server on the east hosts (e1-e3). (TCP and UDP Port 22) (https://en.wikipedia.org/wiki/Secure_Shell)**
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 server1 and server2 from being accessed all hosts. However, the DNS and NTP services on server3 should remain accessible to all hosts. (UDP Ports 123 and 53, respectively)**
4. **Write a series of firewall policy rules disallowing hosts w1 and w2 from pinging mobile1. This rule will be satisfied if the full ICMP response is not completed. Analyzing the interactions with Wireshark is helpful, but not required.**
5. **Write a series of firewall policies to disallow all traffic destined to TCP ports 9950-9952 on host e3 from host e1.**
6. **Write a series of firewall policies to restrict host mobile1 from communicating to any of the east hosts (e1-e3) on both TCP and UDP protocols.**

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-p4.zip).

Use the following command to zip the two files below using the class VM: **zip gburdell3-p4.zip firewall_policy.py firewall-config.pol**

(replace gburdell3-p4.zip with your GT ID name). 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

PLEASE MAKE SURE THAT YOU NAME THESE FILES WITH THE NAMES SPECIFIED. YOU WILL LOSE UP TO 15 POINTS PER EACH MANDATORY FILE IF IT IS MISNAMED.

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

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.

Questions to Ponder

To help you implement your firewall policy and configuration rules properly, think about the following topics. Feel free to discuss these on Piazza:

- When creating firewall rules, should you be using BOTH source and destination ports when creating a rule? Why or why not?
- What happens when you block one side of an ICMP request/response? What is the behavior if you block the sender vs the requester?
- Pyretic has specific requirements when port numbers are specified. Why is this so?
- If you do a wireshark traffic dump while testing your rules, why do you sometimes see one-sided responses?

- Given a rule to block access to a port 80 server on host E2, should you be able to access the port 80 service FROM host E2? Why?

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. Please remember to set it back to the original settings for Project 5 if you do so.

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 Project4GUI zip file from Canvas and follow the instructions for use found within the folder after you unzip it using the `unzip Project4GUI.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 a known good firewall_policy.py file. It will also be tested in conjunction with 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 additional partial credit.

30 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 15 points for each file. The penalty will have a variance based on the effort undertaken..
60 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.
60 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)