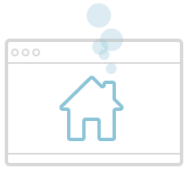
Nanodegree Catalog Sign In Sign Up



Can't find your way?
Visit udacity.com

Assignment 5 - SDN Firewall

In this assignment, we are going to use SDN to create a configurable firewall. 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.

We are going to create an *externally configurable* firewall in Pyretic. That means that the firewall rules are provided in a configuration file, so they can be changed without altering the switch code. We will be using some of the same techniques as were seen in the earlier assignments (in particular the learning switch assignment).

In the assignment-5 directory, there are a number of files, described below:

- firewall-policies-bad.cfg - This is an example firewall policy that is broken. When parsing, an error message will be thrown
- firewall-policies-good.cfg - This is an example firewall that blocks port 1080 from ever working in either direction.
- firewall_policy.py - This is the file where you will implement the firewall, based on the policy configuration that is passed in. It is the only file you need to modify and submit (although you may choose to edit the firewall configurations and topologies in order to test your code).
- firewall.py - This is the main file for the pyretic application. You do not need to modify it. A shell script is provided to help run it.
- firewall-topo.py - This is the 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.
- 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 applicaiton.) 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 file name on the command line.
- test-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-server.py - This acts as a TCP server: waits on a specified port, echos back whatever it hears. You can use this together with the test-client.py program.

Instructions

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 deducting points for not following directions, but you will not receive credit for your project working if our grader cannot tell whether or not it's working due your not following directions.

1. First, update your git repository to get the assignment 5 code:
   `git commit -a -m "Saving work"`
   `git pull --rebase`

2. First, you may wish to try out how test-client.py and test-server.py. To do this, you will need to start your topology, copy over the pyretic_switch.py file to the pyretic directory, and run the switch. Then you can try the client and server.

   - In one terminal, start the topology: `sudo python firewall-topo.py`
   - In the second terminal, copy over the pyretic_switch.py file: `cp pyretic_switch.py ~/pyretic/pyretic/modules`
   - In the second terminal, start the pyretic_switch.py file:
     `cd ~/pyretic; python pyretic.py pyretic.modules.pyretic_switch`
   - Back in the first terminal, at the mininet prompt, you need to open up a couple of new terminals by using the following: `e1 xterm &` `e2 xterm &`
   - In the e1 terminal, start the server: `python test-server.py 10.0.0.1 1234`
     - When you run the server, the IP you enter should be the server's own IP. 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.
   - In the e2 terminal, start the client: `python test-client.py 10.0.0.1 1234`
   - 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. Now, to create the firewall, you'll have to edit firewall_policy.py. The main part that you have to implement is taking the parsed configuration file (which is in a list-of-dictionaries) and create the appropriate match actions based on the configuration file. You only have to work with one entry at a time.

4. To run the firewall, it's very similar to what was done in step 2, with a couple of tweaks. For the step with the client and server, you'll want to select ports that are *supposed to be blocked* and ones that are not to make sure the correct traffic is passed through and the invalid traffic is blocked. Full steps are below:

   - In one terminal, start the topology: `sudo python firewall-topo.py`
   - In the second terminal, move over the appropriate files, start the firewall. You need to specify the config file you are using (replace config-file.cfg in the following command with the correct config file name, e.g., firewall-policy-good.cfg): `./run-firewall.sh config-file.cfg`
   - Back in the first terminal, at the mininet prompt, you need to open up a couple of new terminals by using the

following: `e1 xterm &` `e2 xterm &`
    - In the e1 terminal, start the server: `python test-server.py 10.0.0.1 1234`
    - In the e2 terminal, start the client: `python test-client.py 10.0.0.1 1234`
5. Once you have a working firewall, test it out by creating your own configuration files. Start small, like the provided files. Like in assignment 3, feel free to share configuration files that you create, with the exception of the one created for the step below (or analogs of below). Also share your test cases, i.e., what do you do to be sure your firewall is blocking everything the policy says it should, and nothing else. (If you create any custom topologies to test with, feel free to share those too, although the provided topology should allow testing a wide variety of policies already.)

6. Finally, we want you to create a configuration file. It should be named `my_config.cfg` _exactly_ (every character must match precisely, including case - all lower) The topology is pretty simple (you can find it using Mininet `net` and ifconfig commands), plus the source code for firewall-topo.py is provided. We'll be using the same topology for testing. Do the following:

    - Block all traffic in both directions between the East (e1, e2, e3) and West (w1, w2, w3) on port 1080. That is, e1 should not be able to connect to w2 on port 1080
    - Allow all traffic (that is, don't block) within the East or West sides to port 1080. That is e1 should be able to connect to e2, and w1 should be able to connect to w2
    - Block e1 from communicating with w1 completely in both directions
    - Block e2 from communicating with w2 over ports 2000-2004 in both directions
    - Block e3 from communicating with w3 over ports 3000-3002, but allow w3 to communicate with e3 over those same ports
    - There may be more than one way to accomplish these. We will be testing the functionality, not the exact text of your file (the exact text still needs to be formatted properly so that it can be parsed, of course), so any of the ways that work are fine.
7. Turn in `my_config.cfg` and `firewall_policy.py` to T-Square.

Notes

Installing Wireshark (`sudo apt-get install wireshark`) is a good idea: you can start it from any terminal (`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 blocked completely or if one half of the conversation gets through.

If you're blocking one node from talking to another, the ARPs may not be blocked (as it's destined for the broadcast), while the ARP response should.

The test programs only work with TCP. The configuration file is similar. We only care about TCP for this exercise (otherwise we'd need a protocol field to be added).

With your test program, stick to ports above 1024 - the ones below are reserved ports and shouldn't really be used. If your code works for 1080, it'll work for 80. (We will only test using ports above 1024).

For grading, your `my_config.cfg` will be tested for validity and functionality. Your `firewall_policy.py` will be tested against a number of configuration files, much like the Bellman Ford assignment was. Simple configuration where there is blocking of a single port will be tested, along with configurations more complex than what was seen in `my_config.cfg`. Use the configuration files and tests your classmates share to make sure your firewall is working well. Come up with some of your own, too, and share them with the class.

This page was last edited on 2015/07/06 02:42:45.

NANODEGREE PROGRAMS

Front-End Web Developer
Full Stack Web Developer
Data Analyst
iOS Developer
Android Developer
Intro to Programming

STUDENT RESOURCES

Blog
Help & FAQ
Catalog
Veteran Programs

PARTNERS & EMPLOYERS

Georgia Tech Program
Udacity for Business
Hire Nanodegree Graduates
Developer API

UDACITY

About
Jobs
News & Media
Legal
Service Status
Contact Us

FOLLOW US ON

MOBILE APPS

iOS
Android

Nanodegree is a trademark of Udacity
© 2011-2015 Udacity, Inc.