Weaver
Fall 2017

CS 161
Computer Security

Project 3

Due: November 20, 2017, 11:59PM

Version 1: November 6, 2017

Background

"The Great Firewall of China" is notably misnamed. Rather than being a true firewall (an in-path device that can drop traffic), it is an on-path device that can only examine network traffic and respond by injecting either TCP RST packets or DNS replies. It also operates symmetrically: It can't tell the difference between traffic leaving China and traffic going into China. And finally because of Internet routing any instance of the Firewall may only see either requests or responses: it has to assume valid connections.

Which means we can directly play with it: triggering effects and observing responses. An easy test. `202.106.121.6` is the IPv4 address of the Chinese Ministry of Industry and Information Technology `www.miit.gov.cn`. So if we send a random DNS request to Tsinghua using "dig", such as `dig www.aoeuaoeuaoeu.com @202.106.121.6`, will fail with a timeout. But a request for a censored name, such a `dig www.facebook.com @202.106.121.6` will return an answer, as the Great Firewall sees "Oh, this is a DNS request, oh, and it is for a host we don't allow. So lets return a bad but seemingly valid answer".

For this project you will investigate multiple aspects of the Great Firewall by examining and crafting raw network traffic. We will generally provide you only a very little skeleton code in `common.py`, but you don't actually need to write that much code either. Yet just because you shouldn't need to produce much *code* doesn't mean this project won't take considerable time. It is *strongly* encouraged that you complete part one and two within the first week.

You will work either alone or in a team of two. We have provided a virtual machine that has all the necessary tools installed, and a source code archive at `http://www-inst.eecs.berkeley.edu/~cs161/fa17/proj3.tar` to start with.

The Virtual Machine

Every tool needed for this project is already installed on a virtual machine. You will be able to run and investigate the VM on your own computer. You will need the following on your computer:

1. VirtualBox

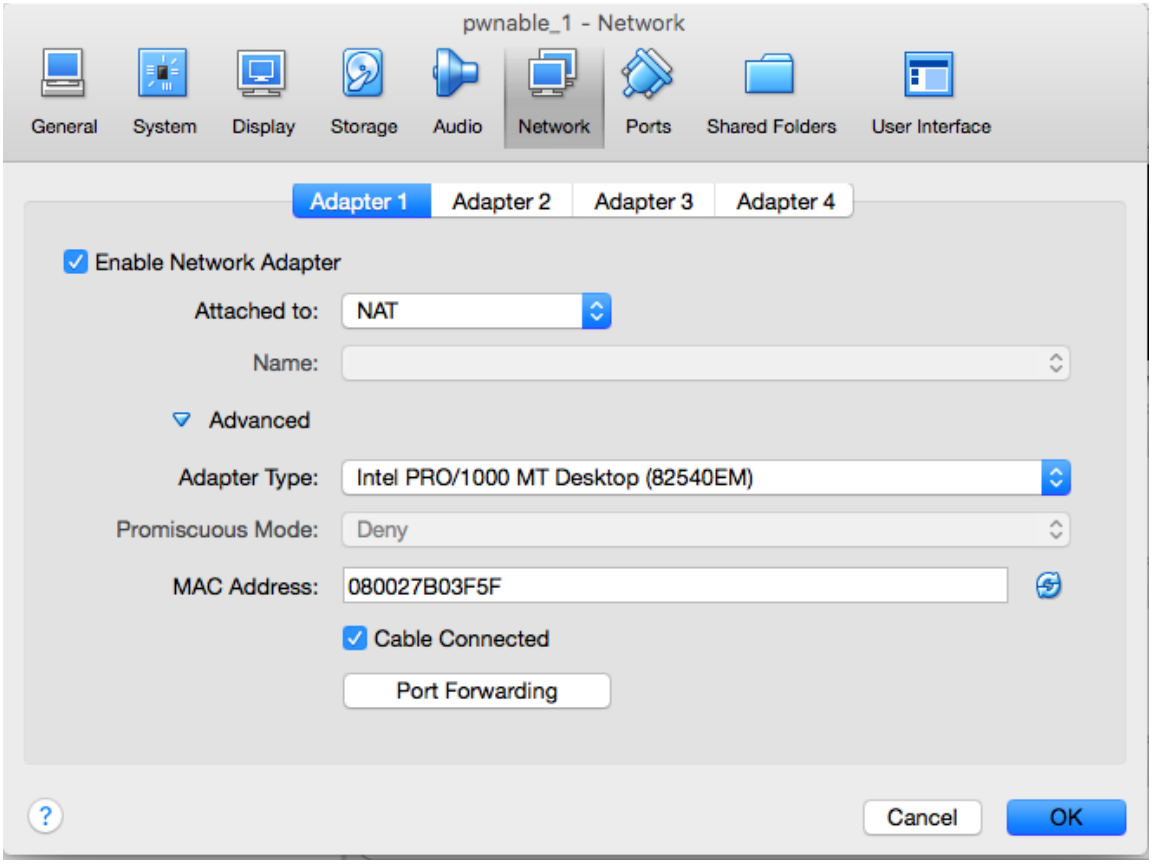
2. A text editor
3. An SSH client (on Windows, use Putty or the ssh that comes with Git)

On Linux and Mac, you can install these programs from your package manager (e.g., `apt` or `brew`).

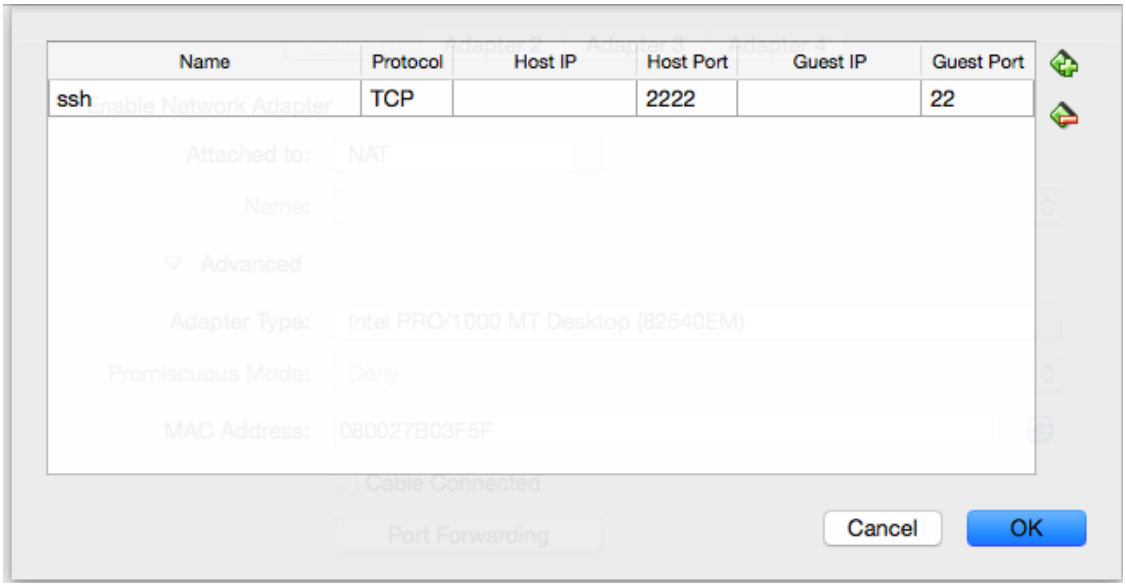
VM Setup

Open VirtualBox, and download and import the VM image (here) via `File -> Import Applicance`. We've also provided you a SHA-256 checksum file that you can use to verify the integrity of your downloaded `firewall.ova` VM file.

Make sure your network is configured correctly by clicking your VM's settings. Under `Network -> Adapter 1`, make sure the first NAT adapter is enabled and open the advanced settings.



Click the `Port Forwarding` button and ensure that you have a rule to forward port 22, for SSHing to the machine, to port 2222 on your host. Also make sure that `Cable Connected` checkbox is checked.



You should also make sure the second network interface is also active, and configured in "bridge" mode. This second interface is needed because the VM infrastructure's NAT will disrupt your probe attempts. (You may still have issues from the additional NAT on whatever network you use). The code we provide in `common.py` sends packets out of the second, bridged interface and sends as raw Ethernet packets in order to bypass the Linux system's routing infrastructure.

You can now start the VM, in which you will be able to run scapy-based python tools as you have root access. The image is a bare-bones Ubuntu Linux server installation on an Intel architecture. You can log into this VM using the username `neo` and password `caltopia`. This user has sudo access, so it can execute root commands within the virtual machine. You should IMMEDIATELY change the password on this account and, for the sake of your own sanity (so you don't need to keep typing the password), configure the `.ssh/authorized_keys` file to include your SSH public key.

Since we use a rule to forward to port 2222, use the command `ssh -p 2222 neo@127.0.0.1` to access the VM through the network. If you add in `-A`, this will also enable ssh agent forwarding so you can access a git archive on BitBucket or GitHub. We strongly recommend you create such an archive for your own use, but be sure to keep the archive private.

Question 1 ***PCAPs or it Didn't Happen*** **(40 points)**

A general theme of network security and censorship measurement is "PCAPs or it Didn't Happen". That is, a claim of something weird on the network is, just that, a claim that may result from any one of a number of causes. But by recording the network traffic which demonstrates the issue, someone else can verify the claim[1].

If you do a direct TCP connection to a webserver in China using `telnet` to port 80, you can manually perform a web fetch. If the Great Firewall thinks this is forbidden it

[1]Well, technically speaking they are verifying one of two things: either that your claim is correct or you spent a lot of effort willfully forging a PCAP file. For the purpose of this project, faking a PCAP file is considerably more work...

will respond by sending a series of `RST` packets which appear to be from the "server". Subsequently it will then block all communication with that server for a small period of time.

You are to capture a very limited PCAP using tcpdump that shows:

a) A TCP connection to a webserver in China where you successfully establish a connection and trigger an HTTP 404 (page not found) error before closing the TCP connection.

b) A second TCP connection to the same webserver where you do a fetch that the Great Firewall interprets as a Google Search for "Falun Gong" and responds with one or more RST packets.

Call this file `question1.pcap`. This pcap file should contain no other network traffic. You should also edit the `question1.txt` file to answer the specified questions. This file will be automatically graded and matched with the PCAP.

Question 2 ***GFC Ping*** **(40 points)**

We provide you with a common framework for building the rest of these utilities, `common.py` that is imported by `ping.py`. You should ONLY edit the `common.py` file and, if you aren't running on a VM, the `interfaces.py` file. You need to complete the `ping()` method which should use raw packets to create a TCP connection to the specified server.

You need to complete the ping() function. This should select a random source port between `2000` and `30000`, create a random TCP SYN, send it to the server, wait for the SYN/ACK or a 5 second timeout and, once the function receives the SYN/ACK, send an ACK and a single data packet that would trigger the Great Firewall and wait another 5 seconds to see if you get a RST packet back from the server.

Question 3 ***GFC Traceroute*** **(40 points)**

You also need to complete the `traceroute()` function in `common.py` to enable the `traceroute.py` utility. For each hop this should create a normal handshake with the server with a normal TTL, then try to send 3 copies of a packet which would trigger the Great Firewall with a short TTL, looking for ICMP time-expired replies and/or injected RST packets. Beware, any NAT you may be behind will correctly translate the RST packets but may not adjust the payload in the ICMP packets.

Question 4 ***Evasion*** **(40 points)**

You finally need to complete the `evade()` function in `common.py` to enable the `evade.py` utility. This function will take both an IP, a string which would normally trigger the Great Firewall, and the number of hops needed to reliably trigger the Great Firewall (from GFC Traceroute).

It will attempt to send a sequence of packets so that the Great Firewall does NOT censor the request but the server will reassemble the request uncensored. It will then return all the payload returned by the server over the next 5 seconds. You also need to complete the `question4.txt` file to describe the evasion mechanism you used. NOTE, you evasion mechanism doesn't need to be 100% reliable, but should succeed most of the time.

Question 5 ***Feedback (mandatory)*** **(10 points)**

You must also submit a text file, `feedback.txt`, with any feedback you may have about this project. What was the hardest part of this project in terms of understanding? In terms of effort? (We also, as always, welcome feedback about other aspects of the class.)

Your comments will not in any way affect your grade, apart from needing to be written in English, be about the project, and consist of at least 4 valid, roughly syntactically and semantically sort of correct and appropriate[2] English sentences, so you do need to provide at least some feedback to earn your 10 points!

Submission Summary

In summary, you must submit the following directory tree:

```
question1.pcap
question1.txt
question4.txt
common.py
feedback.txt
```

You will NOT be submitting the other python functions, so DO NOT put any code in those, only in common.py.

[2]Given Nick's approach to lectures, "appropriate" means it just has to be about the project. "FUCK THE GOD DAMNED VM NAT THAT REQUIRES ME TO USE A SECOND BRIDGE INTERFACE. GOD DAMN FUCKING SCAPY'S LAYER 3 SENDING API THAT FORCED ME TO USE LAYER 2 IN ORDER TO SELECT AN INTERFACE. FUCK LINUX AND ITS SENDING OF RST PACKETS THAT NEEDED THE IPTABLES RULES. SHIT I HATE WORKING WITH VMS BECAUSE I HAVE ROOT ON A GAZILLION LINUX BOXES ALREADY BUT I KNOW STUDENTS DON'T HAVE QUAD-CORE LINUX SERVERS SITTING ON THEIR DESKTOP SO I HAVE TO DO IT IN A WAY THAT WORKS FOR THEM. WHY IN GOD'S NAME AM I SO FUCKING STUPID THAT I THOUGHT IT WOULD BE "COOL" TO CREATE A NEW PROJECT INSTEAD OF REUSING THE OLD PROJECT?" is Nick's feedback during creating this project.