IOT Lab 3: Enterprise IoT

DISCLAIMER: *This lab looks long - do not worry! Most of this document is more like reading material than a lab. Our goal is to provide you with extensive descriptions, to walk you through each step, so you will not get stuck, and to also teach you additional things. It is ok to skip over parts of the reading if you feel comfortable and don't feel like you need the extra help.*

Lab Overview

So what you've built so far is pretty cool - a self-driving car! And then in lab 2 you made it even more advanced - you made it talk over a network!

But there's a problem. When we added networking capabilities to the car, we created an interface where external parties can modify the internal state of the car. Which is fine if it's us just pushing on arrow buttons and reading battery data off of the car. But what if someone uses that interface maliciously - tries to get into our car and damage it, or crash it, or use it as a platform to launch attacks on other parties?

This might sound silly but the increasingly automated nature of vehicles combined with the fact that automobiles are involved in all sorts of critical infrastructures from trucking of goods to banking to transportation make them a tempting target of attack for adversaries. The fact that vehicles are increasingly connected to the cloud makes them prone regardless of location or distance from the adversary. And even if you don't care about cars - protecting your systems from external threat is something that is crucial to pretty much every networked IoT device in existence.

In this lab we will take some steps to secure and protect your car from threat by leveraging features in Raspberry Pi OS[1]. In particular, you will first set up firewalling capabilities to prevent external parties from connecting to your car on unauthorized ports. Next, you will leverage some packet inspection tools to study properties of the IoT protocols and see what data is getting sent out of your device. Finally, you will inspect and adjust the wireless settings on your car to constrain the region in which other devices can communicate with it. Along the way we will change your car into a network police officer - a vehicle that can drive around and look for wifi dead zones and attackers on your network! The experiences you gain in this lab will be

[1] These features are comparable in design to those that exist in other operating systems such as Windows and Linux, so your experience here will be broadly applicable to other environments.

applicable beyond cars - security is a big problem in IoT[2], and the techniques you will execute in this lab are examples of good practices you should use when building secure and resilient IoT devices.

[2] Some relevant statistics: (a) the average amount of time it takes for an IoT device to be compromised is 5 minutes (b) IoT malware attacks increased 215% in 2019 (c) 76% of risk professionals believe IoT leaves them at risk. This lab is an opportunity for you to learn about how to protect IoT devices that you build.

Step 1: Firewalling

When you connect a device that you design to the network, you can enable that device to do many wonderful things. But you also provide a way for bad people[3] on the Internet to cause harm to your device. In the early days of the Internet, attacks were relatively rare and often harmless. But today, criminals have learned that they can generate vast sums of money by infiltrating devices across the Internet. Modern Internet criminals run sophisticated business operations bent on locating, compromising, and using IoT devices as vectors for attack. An IoT device exposed to the Internet will receive a constant stream of probes and attack attempts, continually searching for weak points in your infrastructure.

Luckily, there are a collection of best practices for IoT that provide strong protection for your devices. We will investigate one here which is particularly good to know about, called a *firewall*. A firewall is a component which watches, and performs operations on, network traffic. In fact, you can think of a firewall of having two distinct pieces, something that decides which groups of traffic to pay attention to (what traffic you "match" on), and something that performs actions based on which group a particular piece of traffic falls into. Configuring a firewall, hence, involves telling the firewall a list of traffic groups, and for each telling the firewall what sort of action you want to perform. Selecting a traffic group usually involves specifying which specific headers in packets you care about, and what values you want them to be for the packet to be considered part of the group; common actions firewalls support include dropping the packet, giving it higher/lower priority, sending it to a particular next hop, logging it to a file, etc.

Firewalls are such a great idea they are built into all sorts of systems. When you build a network, you can purchase a firewall as a standalone device. If you purchase a router or switch, oftentimes a firewall will be built into it. But firewalls are also built into applications, protocol suites, and even operating systems. Raspberry Pi OS (like most operating systems) has a firewall built in to it, which can be activated on any network interface attached to the device. In this step, we would like you to get some experience configuring an OS-level firewall[4], and thinking critically about how you would apply firewalling concepts to improve security and robustness of IoT devices.

Please perform the following steps, and respond to the questions below in your report:

---

[3] It is worth noting that good security practice does not just protect you against "bad people" - accidental misconfigurations are the largest cause of failure across a broad spectrum of systems including IoT, and isolating your device can help improve robustness of it against error.
https://www.techrepublic.com/article/cloud-misconfigurations-cost-companies-nearly-5-trillion/

[4] In practice, if you have control of the network, a good practice is to establish a perimeter (one or more firewalls that isolate bad traffic outside your network), creating a protected network segment for your IoT devices. But good security practice provides protection at multiple layers, so even if you do have control of your network (which you don't always have), you should still lock down your IoT devices as much as possible - close unneeded ports, monitor and block bad traffic, etc.

1. Read about firewalls: read the wikipedia page on firewalls (Firewall (computing) - Wikipedia) [5]. Click around on the various links and get a sense of the different types of firewalls. What sorts of firewall technologies do you feel are helpful for IoT? What layer(s) of firewall do you feel are most appropriate to protecting your device?

2. Read about the Raspberry Pi OS firewall: Read about the following best security practices for Raspberry Pi OS: Raspberry Pi Documentation - Configuration . Follow the steps in the last section to set up the firewall (ufw). What are your thoughts on these security practices? Can you think of any possible attacks not covered by these configuration practices?

```
sudo apt install ufw
```

Warning: *After you turn on the ufw firewall, you will likely not be able to access VNC, as the firewall will be doing its job. However there are commands that will allow you to make exceptions to your firewall (like a good firewall should).*
Example:

```
$ sudo ufw allow from 15.15.15.0/24  to any port 22
```

See this link for more examples: UFW Essentials: Common Firewall Rules and Commands | DigitalOcean

3. Configure the firewall in Raspberry Pi OS: Leverage the commands above to install some basic security policies. In particular leave only ports open that are used by the protocols you used in Lab 2, and close all other ports.

[5] Here is a video introduction to firewalls that may be helpful: https://www.youtube.com/watch?v=9JQtyQEpQV8 .

Step 2: Packet Inspection

When you deploy a system of any type, you will likely encounter failures. Imagine a deployment of thousands of these cars - things will go wrong, they will break, customers will call you on the phone - you'll need to troubleshoot and figure out what is going on. Luckily, being attached to a computer network gives you the power to see inside your device and inspect all aspects of its software-level operation even when you are not attached to it.

There are a broad suite of diagnostic and monitoring procedures companies use to manage their IoT devices in the field. You'll create logging functionality, store checkpoints on crashes, enable watchdog timers to detect lockups, and push firmware updates with fixes.In this step, we will investigate a powerful mechanism to have in your arsenal when studying and debugging your IoT devices from an external perspective, namely, packet inspection.

I don't know if you've ever had a plumber come over to your home when your drain gets stopped up. It's tricky for the plumber, there's some sort of stoppage in your pipes but they don't know where it is. They can't see inside your pipes directly because they're embedded in walls and floors and they're not transparent anyway. So they'll do various sorts of inspections - they'll turn on water at various places in your house and see which drains start to back up. They'll watch how fast the water goes down. They'll look at access valves to see what flow is like at various parts of your home. They'll push a metal "snake" down and try to pull out blockages where they think they might be.

Debugging a networked system is a very similar procedure actually, but instead of water, we have packets. Sometimes when you deploy your IoT system things won't work right - packets won't get through, they'll get "stopped up", go slowly, go the wrong way, come out messed up, etc etc. To figure things out, you need to watch the packets going through your system. Like the plumber, you can't see packets everywhere in your system, rather there are certain places you'll be able to inspect them. You can inject "probe" packets of various types to see which packets go through, form hypotheses about what sorts of problems are happening, and modify configuration settings and software to localize and create a fix.

To do this effectively, you need some way to "see" packets at various locations in your system. Luckily, most operating systems provide locations (taps) which "mirror" the traffic so you can get a copy of packets going through. You also need a way to print out the packets in a human readable way - luckily again, there are a number of tools to do this as well. In this lab, you will gain experience with configuring an operating system to log packet data in a highly common packet storage format (pcap) and replaying and inspecting that output via Wireshark, a common packet visualization platform.

Please perform the following steps, and answer the following questions in your report:

1. Read the wikipedia article about pcap: https://en.wikipedia.org/wiki/Pcap (also see https://www.comparitech.com/net-admin/pcap-guide/) . Pay attention to what pcap is and what it can do. What is pcap? What sorts of information does pcap capture about packets? Can you think of some example problems you could diagnose with a pcap trace? Can you think of problems for which pcap would not be appropriate?

2. Learn how wireshark works: Next, skim over the wikipedia article for wireshark (https://en.wikipedia.org/wiki/Wireshark) . Focus on the "Features" section to get an overall sense of what it does. Then, watch this youtube video: https://www.youtube.com/watch?v=TkCSr30UojMr . What is wireshark? How is it related to or different from pcap? What are some situations you might want to use wireshark?

3. Set up wireshark[6] on your device and connect it to the Wifi interface. Verify that you are able to use wireshark to inspect packet contents. Send and receive some packets (however you like - you can run ping, wget, ssh, etc). Go through and read the traces. What do you notice? What do you see in the contents of packet headers, is it what you expected? Also do you see any packets you don't expect?

Some helpful hints:

a. You may need to install wireshark first. You can do that by running the following commands: `sudo apt-get update` and `sudo apt-get install wireshark.`

b. If you get any sorts of permission errors (e.g., "No interface can be used for capturing... couldn't run /user/bin/dumpcap..."), you may need to reconfigure permissions by running the commands `sudo dpkg-reconfigure wireshark-common` and `sudo chmod +x /usr/bin/dumpcap`. For more details, see: AskUbuntu.

4. Reflect on what you've learned: Think about how to use packet inspection to deal with various scenarios. If you believed you were under attack, and an adversary had infiltrated your system, how would you use wireshark to figure out what happened?

---

[6] If you've got an HDMI connection set up you can actually run wireshark right on your Raspberry Pi. If you're doing this over a network, you can actually still do that - read about "xhost" to learn how to tunner the graphics for windows over a network connection for display on your screen. If you prefer to do all this with raw text that is ok too - wireshark and its variants (eg tcpdump) provide the ability to print packet contents to the terminal. This may in fact be easier for certain sorts of debugging situation, eg if you want to perform various sorts of more complex searches (eg using python/awk/grep) over a long trace.

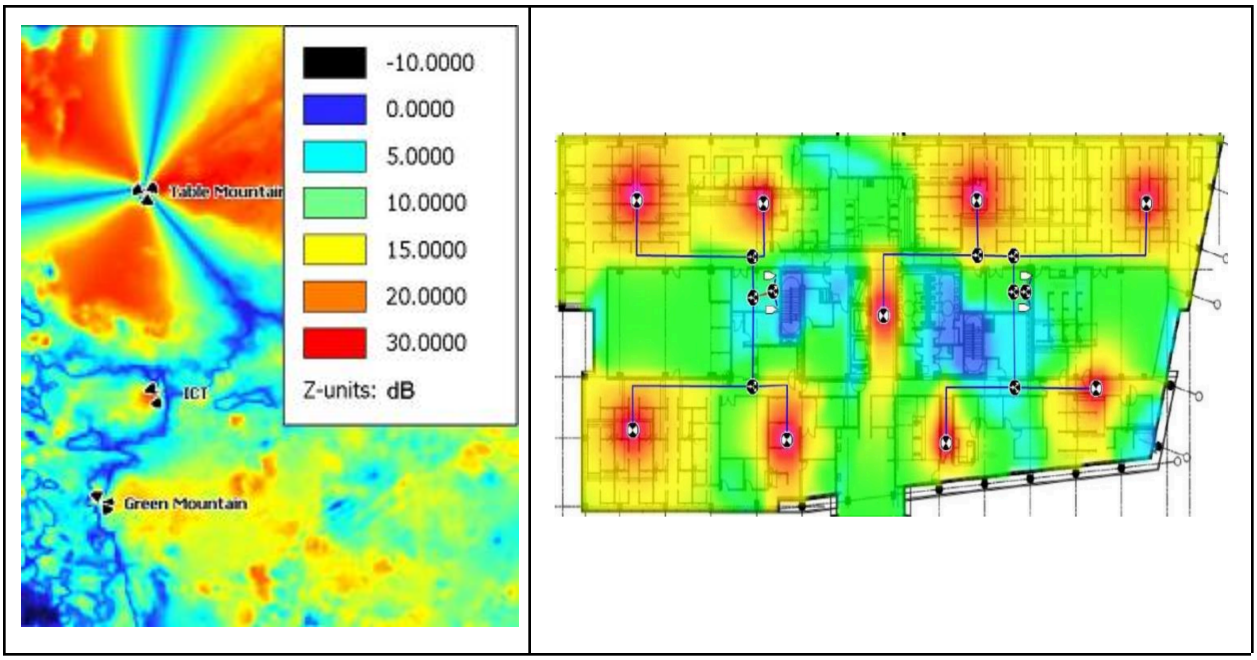


Figure 1: Wireless propagation examples. Left (Wide area): 700MHz basestation atop Table Mountain (Wyoming) using three directional panel antennas covering a range of 20+ miles. Coverage is highest closest to the towers as well as atop more distant elevation peaks. Right (Local area): Access point deployment for a typical office environment covering 100+ meters. Coverage is highest nearest access points and along infrastructural conduits which reflect and direct the signal (e.g., hallways, doorways, rooms).

Step 3: Wireless Configuration

Wireless is, in my opinion, the most fun way to send data. I mean, wired networks are fun too, but with wireless your packets will do such weird things, they will bounce off the walls, go through people, get inverted and conflict with themselves, and so much more. Wireless deployments do something new every day, it will start raining outside and there will be a new dead zone in the middle of your office, you'll have contractors come in and their walkie-talkies will interfere with your 2.4Ghz band and suddenly your devices all start roaming around between access points, you'll spend a week designing a nice point to point wifi system between two of your buildings and someone will park a moving truck on the street below and suddenly your throughput will increase by 50%.

Wireless networking is fun, but it can be kind of a pain too. Have you ever had problems with getting weak Wifi signals in parts of your house? Having to say "hello can you hear me now" when you drive through a dead spot with your cell phone? Now imagine a deployment where you have thousands of IoT devices, each with severe battery constraints, that wake up and have a few microseconds to send what they want to send, they are tired and want to go back to sleep and their data isn't getting through, and the data just isn't getting through, and you've got thousands of these deployed all over, and you need to figure out how to place them, how to place access points, which wireless protocols to choose and how to tune them, and how to design your wireless network around them?

In practice, it helps to have a few tools in your arsenal for manipulating properties of your wireless network. In this step, you'll gain experience with manipulating OS-level wireless network settings for the purposes of tuning performance and diagnostics. In this section, we will inspect and manipulate properties of the wireless interfaces using the `iwconfig` command. iwconfig is a program that lets you manipulate the various run-time settings within the operating system and network interface card that control properties of wireless communications.

To do this, please perform the following steps, and answer the following questions in your report:

1. First, read about iwconfig. There are lots of great sources online[7], but we suggest reading these: https://www.geeksforgeeks.org/iwconfig-command-in-linux-with-examples/ and https://linux.die.net/man/8/iwconfig . Look at each of the parameters of iwconfig and try to get a sense of what they mean.

[7] You can also search on Youtube and Google to find more examples. You can search for strings like "iwconfig" "iwconfig linux" or "iwconfig raspberry pi" on youtube for example. Here's an example video you can watch to get further examples of using iwconfig and supple

2. Try running ifconfig[8] (note the "f" - we're not doing iwconfig quite yet) on your Raspberry Pi - this command will give you a list of wireless interfaces on your device. Search for your wireless network interfaces - your wifi interface will likely start with a "w"). Try running ifconfig again, but this time give the name of your wifi interface after the ifconfig command - it will print information associated with your wireless interface, including addressing information, number of packets that have been sent and received through that interface, etc.

3. Now, let's try running iwconfig, by typing `iwconfig` at the prompt. Again you will get a list of interfaces. Try running iwconfig followed by your wifi interface name. This will give you wireless information associated with your wifi interface. Look through each of the pieces of information printed out and get a sense of what each is. A few things to note - you can see what versions of wifi the interface supports by the first field ("IEEE 802.11bgn" means that 802.11b is supported, as is 802.11g and 802.11n). You can also see the strength of the signal (how loud your wifi device is talking) is measured in terms of decibel-milliwatts (dBm).

4. Look around at the different wireless statistics associated with the interface. Suppose you noticed the wifi connection kept going down - what properties would you inspect? Take a look at the link quality using the Link Quality metric - walk around[9] with your car and watch that change. What happens when you get closer to the access point? Can you discover any "dead zones" in your house where you have poor coverage?

5. Turn your car into a network surveyor: Somehow everyone in your office has good wifi except you. You suspect your desk is in a dead zone. You can use the car you created to check this out! Use the `sudo iwlist` command (e.g., `sudo iwlist wlan0 scan`) to get a list of access points and their signal strength. Walk around with the car[10] and watch what access points can be seen, and see these values go up and down. What parts of your house get the best wifi signals? Walk outside your house. Could someone parked in front of your house get a good signal[11]? Walk down the street and study your neighbors' wireless signals. If you wanted free wifi at the park, which neighbor should you make friends with[12]?

---

[8] Note for some iwconfig commands you will need to run them as root, for example by appending "sudo" to the beginning of the command.

[9] If you wanted to, you could make your car autonomously drive around and search for dead spots at your work, by watching this metric. You could design an algorithm that kind of makes it drive around and look for areas with low coverage and kind of keep going in that direction (or if you want to be more formal about it, use https://en.wikipedia.org/wiki/Gradient_descent). Then you will always know the best places to sit at your work to get the best network connection!

[10] Note that if you wanted to, you could program your car to drive around autonomously to map out your network or discover the best wifi signal location (you don't have to do this, but note this would be fully within your capabilities after going through the labs so far).

[11] Note in practice, even if it appears they couldn't, they could simply get a directional antenna to boost gain and so the "attack radius" of your house's wifi is probably much bigger than you might think..

[12] To estimate out the source of a wifi signal you can just monitor signal strength as you approach/depart from a particular location. For more precise techniques, look up "foxhunting" (https://www.youtube.com/results?search_query=foxhunting+directional+antenna) - if you want to get into

6. Finally, let's modify some properties of the wireless connections. Suppose you wanted to reduce the power of your car's transmission capabilities (e.g., maybe you want it to be more covert, or use less battery). Use iwconfig to reduce the Tx-power default setting to do this.

---

this, make sure you use a directional antenna that's designed for the wifi signal you're tracking, e.g., 2.4 or 5Ghz.

Step 4: Router Configuration

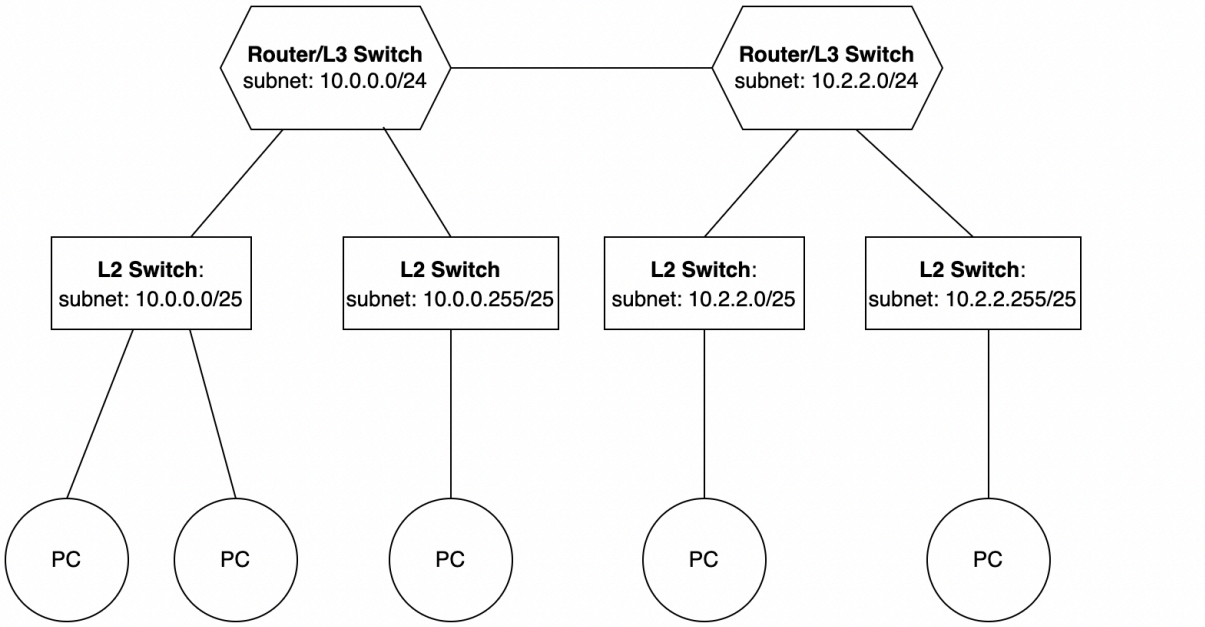
Internet-of-Things devices need some way to communicate with the Internet. To do this, some sort of infrastructure network must exist to provide coverage to these devices. Infrastructural networks often consist of a collection of infrastructural servers providing storage and backend services to the devices, and a wired network backhaul enabling communication between devices, services, and the cloud.

Today's IoT devices present significant security and management hurdles to infrastructural network operators. IoT devices are often highly constrained on memory and power and this limits the ability to run more powerful heavyweight security mechanisms on them. The software foundations of many IoT devices are also often immature, having only recently been introduced to the market, making them more prone to errors and vulnerabilities. At the same time, the rapid propagation of these devices on corporate networks means operators are needing to manage end systems on a scale unlike ever before.

In particular, in this lab, you will be a wireless carrier similar to AT&T or Verizon. You will be creating the network infrastructure necessary to support the networking capabilities of the self-driving car. To simplify your work and enable you to focus on learning the core concepts, you will construct this infrastructure in virtual space. In particular, you will leverage the Cisco Packet Tracer software package, a cost-effective and time-efficient alternative to physical network equipment. Cisco Packet Tracer is a cross-platform visual simulation tool enabling you to design, build, configure, and troubleshoot computer networks on your local computer. It Provides a realistic simulation of configurations of Cisco routers and switches using simulated command line interface and GUI, allowing you to design and configure a network topology in a simulated environment.

Download the Cisco Packet Tracer application, and follow the default installation process. Please refer to the Cisco Packet Tracer documentation for tutorials on getting started.

Create the following network in Cisco Packet Tracer:



The topology consists of two IOS routers and four switches. Two L2 switches hang off each router (L3). Each router has an upstream connection to a different ISP network. The network specifications are below:

- Router/L3 Switch Subnet: 10.0.0.0/24
    - L2 Switch Subnet: 10.0.0.0/25
    - L2 Switch Subnet: 10.0.0.255/25
- Router/L3 Switch Subnet: 10.2.2.0/24
    - L2 Switch Subnet: 10.2.2.0/25
    - L2 Switch Subnet: 10.2.2.255/25

Each PC should have an IP address within the subnet. Ensure each PC can route to each other across the network. You can configure the IP addresses of routers and switches using the CLI of each device. Use the simulation in Packet Tracer to view packets or connect the sniffer to switches to capture packets to debug.

IMPORTANT: Please make sure to understand the core principles before attempting Part 4, particularly with regards to IP addresses and subnetworks. Otherwise it will be extremely difficult. Part 4 is 90% comprehension and 10% execution. It should only take a couple minutes to set up the routing.

(Optional) ***If you want to explore more about router operating systems,*** then instead of using Cisco Packet Tracer, you can use GNS3 to construct the network. This platform will let you use actual router images -- which will provide a similar experience to

PacketTracer (which already emulates IOS images), but the advantage of GNS3 is you can load other images in there too, so you can try out other vendors such as Juniper and Avici, or open-source platforms such as OpenvSwitch and Quagga. Since GNS3 "really" runs images, you can connect the virtual GNS3 deployment inside your computer with routers outside your computer (for testing), in case you ever need to do that in the future.

GNS3 works by running real router images right on your local computer. It contains a hypervisor for routers, enabling you to construct realistic deployments within a simulated environment, giving you the chance to experiment with real routers within your own computer. Be forewarned that in our experience, GNS3 is a bit buggy on Macs, so if you use a Mac it may be better to just use Cisco Packet Tracer.

GNS3 works in two parts:

1. The GUI that allows you to virtualize and deploy various appliances
2. A Linux-based VM where the images are installed and executed (this will be just a regular linux VM, you won't connect to your Raspberry Pi or anything).

Download the GNS3 application (https://gns3.com/) and the GNS3 VM image. Import the GNS3 VM into your virtualization software (VirtualBox, VMWare). Note that you'll have to configure your own host networks and adapters for your host machine to be able to talk to the VM. You should be able to SSH into your VM from your host machine. Please refer to GNS3 documentation for tutorials on how to get started.
You can use Wireshark to capture packets within GNS3 for debugging purposes if something doesn't work. For more information about GNS3, please read:

- GNS3 Setup: Getting Started with GNS3, How To Run GNS3 VM on VirtualBox - TechViewLeo
- Switches: Switching and GNS3
- VPCS: VPCS | GNS3 Documentation

To conduct your work in this lab, we suggest you at least skim over the following documentation, which will give some useful background:

General Info:

- Routers vs Switches: Switch (L2/L3) Vs Router: Comparison and Differences in TCP/IP Networks
- IP Addresses & Subnets: IP Addressing and Subnetting for New Users - Cisco
- Cheatsheet: ip COMMAND CHEAT SHEET

Cisco Packet Tracer Docs:

- Download: https://skillsforall.com/resources/lab-downloads
- Cisco Packet Tracer Documentation: http://tutorials.ptnetacad.net/help/default/index.htm

- Free Tutorial course:
  https://skillsforall.com/course/getting-started-cisco-packet-tracer?utm_source=netacad.com&utm_medium=referral&utm_campaign=packet-tracer&userLang=en-US&userlogin=0&userlogin=0
  You need to register an account to get access to this tutorial. Videos 1.1.1, 1.1.2, and 1.1.3 in Module 1: Download and use Cisco Packet Tracer includes the basic information you need to know about the interface.

What to turn in

In steps one, two, and three, you ran a collection of commands. Please turn in screenshots and console output (showing both the commands you typed/executed and their output), to demonstrate you did every step and substep mentioned above. In addition, please write one paragraph for each step describing what you learned, what you found interesting, ideas for how you could apply what you learned, and answers to any questions we asked along the way.

For step four, please record a demo video clearly showing that the components in each task interact and operate as expected. Besides the demo video, you should also submit a report containing:

1) Cisco Packet Tracer screenshot with clear text notes specifying the connection and interfacing/addressing of all the appliances and PCs.
2) Detailed steps for each task, including configuration changes, setups, and commands used. Screenshots are also acceptable as long as they clearly show the configuration process. ***Note***: Despite Packet Tracer providing GUI for configuration, we ask you to configure the network with the CLI and command lines. **Therefore, you also need to submit screenshots of your commands in CLI.**

Submission and Grading

To submit your own work, you will create a **demo video** clearly showing the above steps working. Also, please give an overview of the code you wrote, show us your code and walk us through it a bit and how it works, so we can know your code is correctly written. Besides the demo video, you should also submit a **report** containing (briefly) your design thoughts and considerations for each part of the lab.

The rubric for grading is as follows:

1. **Was firewalling understood, and done correctly?** Does the report indicate that firewalling was done correctly, and does firewalling seem to be set up correctly on the Raspberry Pi? If so, give them 10 points for this item.

    a. If what they write seems correct and reasonable, and expresses some knowledge taught in the Background section given in this lab, and if firewalling seems to be set up correctly - give them 10 points.
    b. If what is written generally seems right but there is at least some clear technical incorrectness or lack of understanding, or if firewalling generally seems correct but has one or more problems - give them 7 points.
    c. If what is written has major problems, or if the firewalling setup seems substantially incorrect, give them 5 points. If what is written has major problems AND the firewalling setup seems substantially incorrect, give them 3 points.
    d. If nothing is written about this, and/or if what is configured with the firewall is largely incorrect - give them 0 points.
2. **Was packet inspection understood, and done correctly?** Was wireshark used and set up properly, and is the process of packet inspection generally understood? If so, give them 10 more points.
    a. If wireshark was set up and run properly, and the questions about packet inspection were generally answered correctly, give them 10 points.
    b. If wireshark was generally set up right but there are a few visible problems, or the questions about packet inspections were generally correct but have a few problems, give them 8 points. If both seem to be somewhat wrong, give them 6 points.
    c. If the wireshark setup seems to have major flaws, or the questions were answered in a way that seems majorly wrong, give them 5 points. If both seem majorly flawed, give them two points.
    d. If this step was not done, or there was no substantial attempt, for example if no answers were given or there was almost no work done to set up wireshark, give them 0 points for this item.
3. **Was wireless configuration understood, and done correctly?** Were the wireless configuration steps done properly, and does it seem to be working? Were any descriptions or answers provided sufficient and correct? If so, give them 10 more points.
    a. If the text provided in the report seems generally correct, and the wireless configuration steps seem to have been done correctly, give 10 points.
    b. If the provided text or wireless configuration seems somewhat wrong, give 7 points for this item. If both seem somewhat wrong, give 5 points.
    c. If the provided text or configuration seems substantially wrong, give 4 points for this item. If both seem substantially wrong, but not completely wrong, give 2 points.
    d. If this step was not done, or work was not described, or functionality was not demonstrated to any significant extent, give 0 points.
4. **Was the network implemented in Packet Tracer/GNS3 correctly?**
    a. If the network topology seems generally correct and PCs within the same or different subnets can communicate with each other, give 10 points.
    b. If the network topology seems generally correct and VMs within the same subnet can communicate with each other, give 5 points.

c. If the step is not done, or functionality is not demonstrated, give 0 points.

Submission/Points	**Points**
Step 1 - Firewalling	10
Step 2 - Packet Inspection	10
Step 3 - Wireless Configuration	10
Step 4 - Router Configuration	10
Video	10
Total	**50**