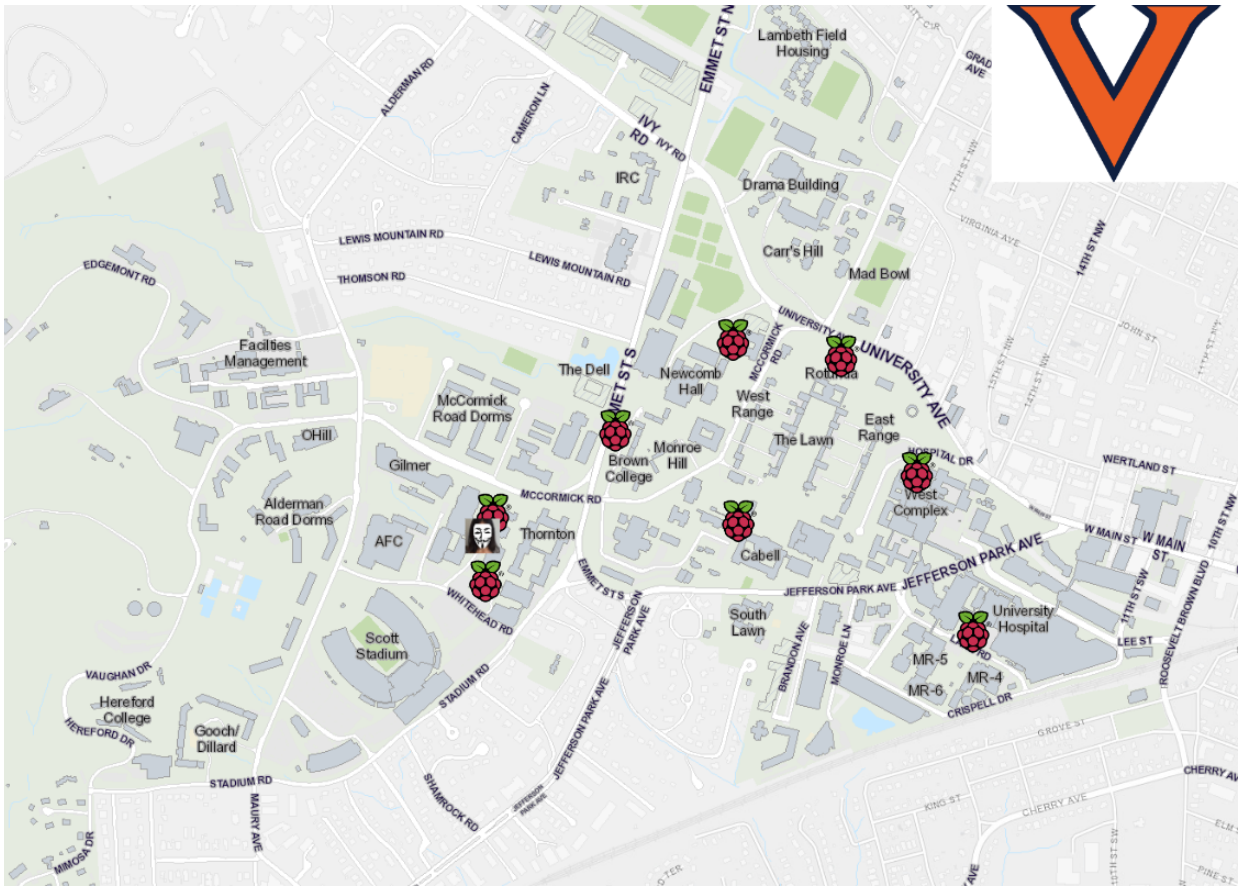
NERVE: Network Event Realtime Visualization Engine for Security Monitoring

Members: Kasra Lekan, Neha Bagalkot, Sneha Iyer, Nicki Choquette

Motivation

When a hacker performs malicious actions, such as broadcasting a rogue SSID, on a wide-scale network, it may not always be obvious. However, if WLAN traffic data is able to be collected at multiple locations of that network's reach, then patterns can be observed over time, which can potentially lead to capturing various bad actors. Additionally, we wanted to assess what type of analysis we could do on WLAN traffic collected around the University of Virginia.



Introduction to WLAN

What is WLAN

A wireless local-area network (WLAN) is composed of a group of computers or other devices that form a network. Specifically, this network is based on radio

transmissions instead of wired connections. The most common example of a WLAN is a Wi-Fi network like eduroam. Anyone connected to Wi-Fi while reading a webpage on the Internet, for example, is using a WLAN.

How WLAN Works

Similar to other broadcast media, a WLAN transmits information over radio waves. Over WLAN, data is sent in packets, which contain different layers with information and instructions. With this packet information and having MAC (Media Access Control) addresses assigned to endpoints, routing is enabled to intended locations.

Security of WLAN

Because a WLAN does not use physical wired connections, it is typically more vulnerable to being breached than a physical network. With a wired network, a bad actor must actually gain physical access to an internal network or breach an external firewall. On the other hand, to breach a WLAN a bad actor must simply be within range of the network. There are some well-known ways of securing a WLAN. One of the most basic methods is to use specific MAC addresses to forbid unauthorized stations.

WLAN and Our Project

We worked on designing a real-time dashboard tool to capture packet data on wireless networks and analyze it. Our tool is a type of wireless sniffer solution, which we built to capture wireless network traffic and analyze it to generate insights into what is going on in a network at any given time.

Approach

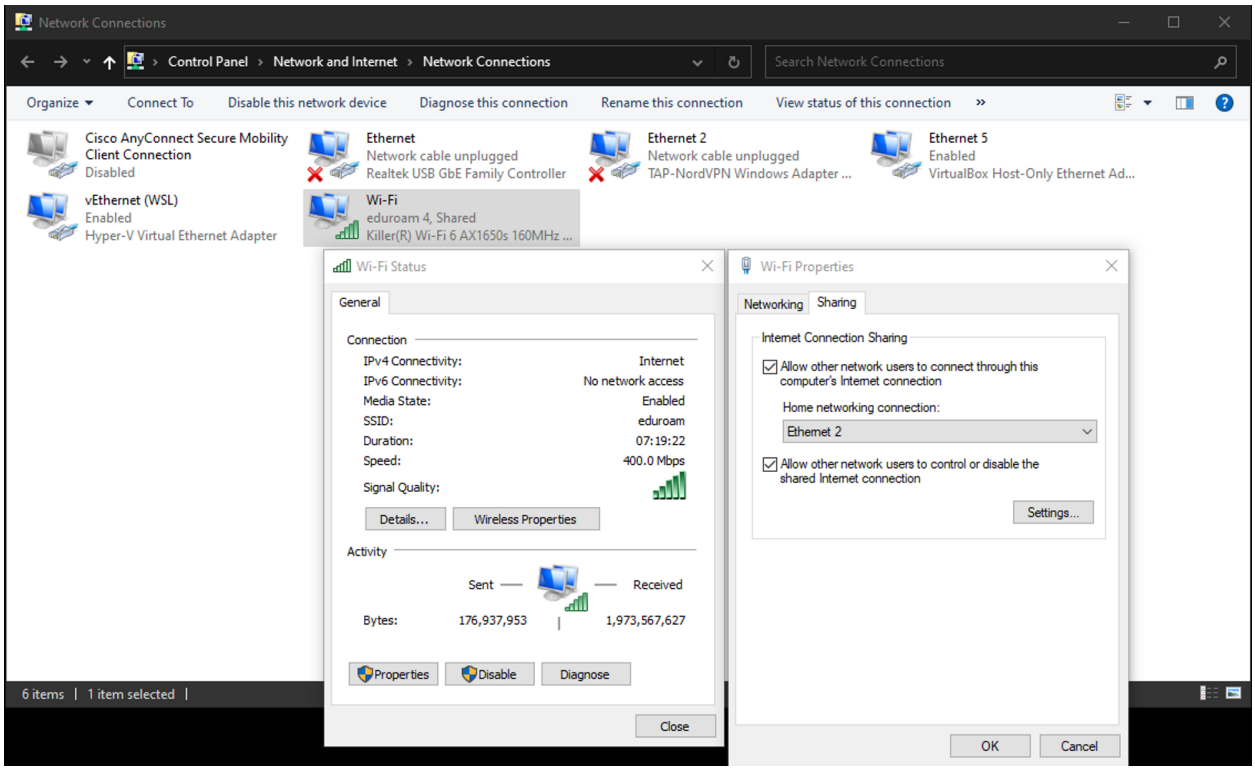
Collecting Data

For collecting data, at different times we set up a Raspberry Pi running Kismet connected to a laptop at Brown Residence Area and at The Flats at West Village. While it may have been interesting to see what data was collected at both locations simultaneously, we only had one Raspberry Pi to use for data collection. In order to set up the Wi-Fi sniffing, we needed a WLAN card that supported monitor mode and had a headless setup with the Raspberry Pi. We then needed to set up the tool to actually gather the data with Kismet, and also needed to set up port forwarding so that this device could be accessed by any other device on the University network with knowledge of the IP address.

Details on Work to get the Raspberry Pi Working

Having determined that using our personal laptops to perform Wi-Fi sniffing was infeasible, we decided to pair a USB WLAN card that was capable of monitor mode with a Raspberry Pi 4 (prebuilt for a different project). Although the hardware was already in place, setting up the Raspberry Pi for capture was not trivial. First, we had to perform a headless setup of the Pi since standard setup was impossible on eduroam. We originally installed Ubuntu on the device but were unable to connect to it. To determine that the Pi was starting properly when powered up, we used Wireshark to capture packets on the ethernet interface between the computer and the Pi. We saw some activity from the Pi in the capture once the device had successfully booted, but we were unable to SSH into it. Thus, Raspberry Pi OS was used with their setup tool to preload SSH credentials. Since we were planning to use the Pi primarily from Grounds where eduroam, which requires Netbadge authentication and a Wi-Fi certificate, is the only consistently available network, we could not preload standard Wi-Fi credentials onto the Pi. Therefore, we had to set up a network passthrough from a laptop logged into eduroam to the Pi. When researching how to set up network passthrough on Windows to a Raspberry Pi, we found

that it is uncommon so trial and error would be necessary to see which combination of proposed solutions online would work. This article was our first breakthrough. After applying the article's method, we tested the passthrough by attempting to SSH into the Pi and were successful.



Having SSHed into the Pi and assuming Network Passthrough worked, we attempted to install Kismet on the device. An error was returned about not being able to access the necessary servers. First, we tried pinging `google.com` which gave no response. We noted a possible DNS issue. Second, we tried pinging `8.8.8.8` which also gave no response. We noted that we were unable to access the outside internet. Third, we tried pinging the IP address of the interface with the laptop. This also failed. We surmised that this may be a Firewall issue. After more research, we determined that switching the interface from Public to Private would likely fix the Firewall issue. We found the necessary PowerShell commands (shown below) to perform this task.

```
Get-NetConnectionProfile
Set-NetConnectionProfile -InterfaceIndex {"index from command above"} -NetworkCategory P
rivate
```

When we attempted to ping the interface, it worked. We tried once again to ping `google.com` and `8.8.8.8`. We were unable to reach the former but the latter worked. Thus, we knew we were able to reach the internet but we had a DNS issue. We did some more research and found the necessary file in Raspberry Pi OS which controlled the default DNS server which we switched from the default IP of the interface to `8.8.8.8`. After restarting the Pi, we successfully pinged `google.com`.

Installing Kismet on a Debian-based distribution was not as easy as expected. Some of the dependencies failed to install so we had to manually find them on their source websites and perform a manual install (`wget` and then run install scripts). Having installed the failing dependencies, we were able to successfully install and run Kismet.

Our objective was to be able to run API calls against the device. Thus, we needed to set up port forwarding to allow others to access the API via my laptop's IP on eduroam. We determined how to open a hole in the Windows Firewall using PowerShell (shown below) to set up port forwarding to the Raspberry Pi. We determined the Pi's IP address was `192.168.137.74` (the interface was on `192.168.137.1` with a `/24` set of device IPs) and Kismet's port is `2501`.

```
# To establish new port forwarding rule
netsh interface portproxy add v4tov4 listenaddress=0.0.0.0 listenport=2501 connectaddres
s=192.168.137.74 connectport=2501

New-NetFirewallRule -DisplayName "forwarder_RDP_2501_rasbpi" -Direction Inbound -Protoco
l TCP -LocalPort 2501 -Action Allow
```

After performing these steps, we successfully accessed the API via the laptop's IP address. We then restarted the Raspberry Pi to make sure everything worked correctly from the beginning steps on startup. We were still able to access the API from the laptop locally but when we tried to access the Pi from outside the laptop, it no longer worked. We determined the Pi's IP address was now `192.168.137.38`. We realized that the interface was on `192.168.137.1` with a `/24` set of possible device IPs. Each time we restarted the connection another IP would be given and the port forwarding would no longer work, N.B. we chose not to do a blanket port forward for security reasons. We determined that we could set the Raspberry Pi to a static IP by editing `/etc/dhcpcd.conf`. We had to remove the rule we created before using the command shown below and set up port forwarding with the new static address. After we performed this patch, the capture setup was completely operational.

```
# To remove rule
netsh interface portproxy delete v4tov4 listenaddress=0.0.0.0 listenport=2501
```

Although the finalized steps are clearly laid out above, the process of setting up our capture device took a significant and prolonged effort due to a standard cycle of debugging (make limited progress, an issue occurs, blocker clearly identified, research, trial and error, solution) repeated many times. The entire process took somewhere from 20 - 30 hours over the course of ~10 days.

We also attempted to set up GPS capture which is supported by Kismet using a separate GPS USB module. We will not delve into the unsuccessful debugging details here; however, after expending considerable effort we were unable to make the GPS capture work since the GPS module was unable to get a position lock. This was likely due to the indoor location of the capture along with the relatively inexpensive GPS module. Due to our keen interest in the potential data with GPS tied into the capture pipeline, we spent time over several days attempting to set up GPS capture before abandoning it.

Our Dashboard

N.B. A SQL database was not used for this project since only one Raspberry Pi was active.

Building the Backend

The backend of this project was built with Flask. Flask is a web framework. Specifically, it is a Python module that allows users to develop web applications easily. According to MDN Web Docs, “Server-side web frameworks are software frameworks that make it easier to write, maintain, and scale web applications.” These types of frameworks (e.g. Flask) provide many tools and services to simplify common web development tasks. Flask applications must create an application instance. The web server passes every request it receives from clients to objects (the application instance is an object of class Flask). The Flask application instance has a “run” method that launches Flask’s integrated development web server. Once the script starts, it waits for requests. The client sends requests to the web server, and then those requests are sent to the Flask application instance. To process incoming data in Flask, you need to use the request object and then can use GET and POST methods to receive or send data. Flask invokes a

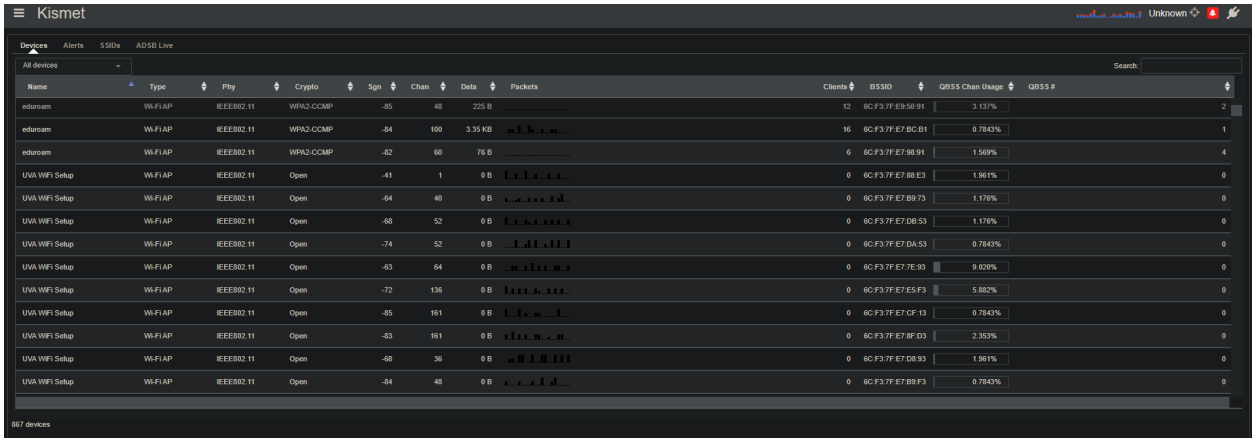
view function and returns a response value to the client. We also used pandas and Matplotlib libraries to create visuals of our analysis for display on the dashboard (creation of the visual is handled on the backend while the displaying is handled by the frontend).

Building the Frontend: (HTML/CSS)

The frontend of a web app, which for this project was built using HTML, CSS, and Javascript, handles how the application is displayed to the user (versus the backend which handles the functionality of the app). HTML is used to display the content of the app to the user (in the form of images, text, etc.) CSS describes how that content looks (styling, colors, etc.)

Facets of Our Dashboard

Our dashboard thoroughly explained our demo video; however, we wanted to go into more detail as to how our dashboard is different from Kismet. Firstly, Kismet does not provide much summary data for all of the devices. Instead, it focuses on summary information of a particular device, such as packets that have been transmitted over time for that one particular device. Additionally, since Kismet is primarily a data-gathering tool, it lacks user-friendly features to display gathered data. Kismet has few graphs for communicating data, and those graphs are often device-specific. In contrast, our dashboard focuses more on the broader picture. Our dashboard highlights wide-scale information about all the devices found. This includes types of devices, manufacturers of devices, names of devices, and more. In addition, these statistics are accessed on the dashboard by first choosing between information on summary (all), WLAN, or Bluetooth devices, and then by the location that the data was collected, with options being on grounds, The Flats, or real-time data. Another feature of our website is the SSID graph, which shows all the connections between devices and access points, etc. Finally, we have query features that allow users to retrieve information on a specific device by entering its MAC address.



Data

Data Retrieval

Using pre-set username, password, and API key, we were able to perform API calls in our Python code to Kismet endpoints to retrieve data. The endpoints we used included

… /devices.json (all device data, including WLAN, Bluetooth, and access points), … /channels.json, … /ssids.json, and … /tracked_fields.html (to observe the various fields present in the device data files). We obtained both .kismet files which we converted to pcaps, and .json files. We viewed the PCAP files for observational purposes but ultimately used the JSON files for data processing within our code. The JSON files were really large containing hundreds of data points. Some of the fields that were of interest to us were device.base.name, device.base.type, packets.total, manuf (manufacturer), macaddr, channel, first_time (the first time a packet was seen for this particular device), and last_time.

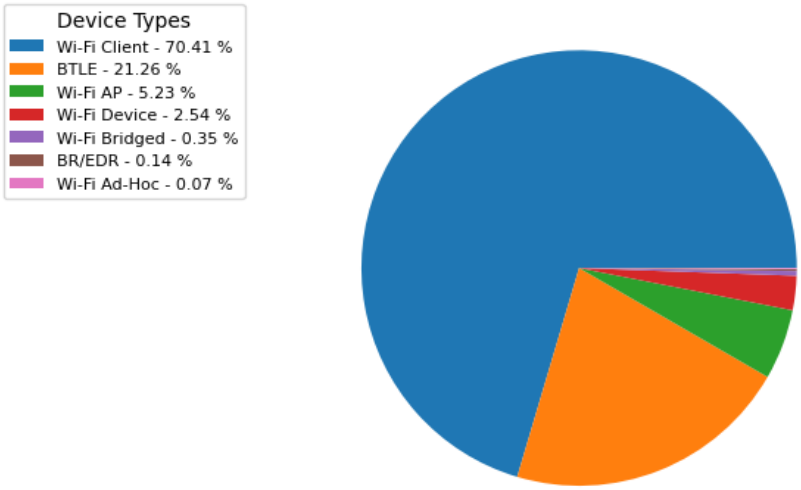
Data Processing

We did our initial data processing using Jupyter notebooks, converting the JSON files into Pandas dataframes. We did some of our own filtering, for example with the Bluetooth devices, many of them had device names listed as Unknown, so we compiled

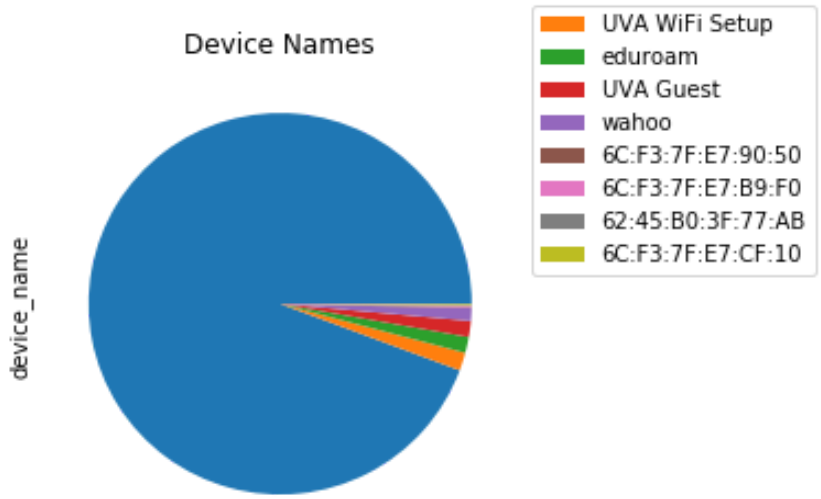
data on how many were known/unknown and only included known names in our final charts. We also used Matplotlib and Pyviz to generate plots.

Data Analysis

In this section are examples of some of the graphs that our dashboard produced.

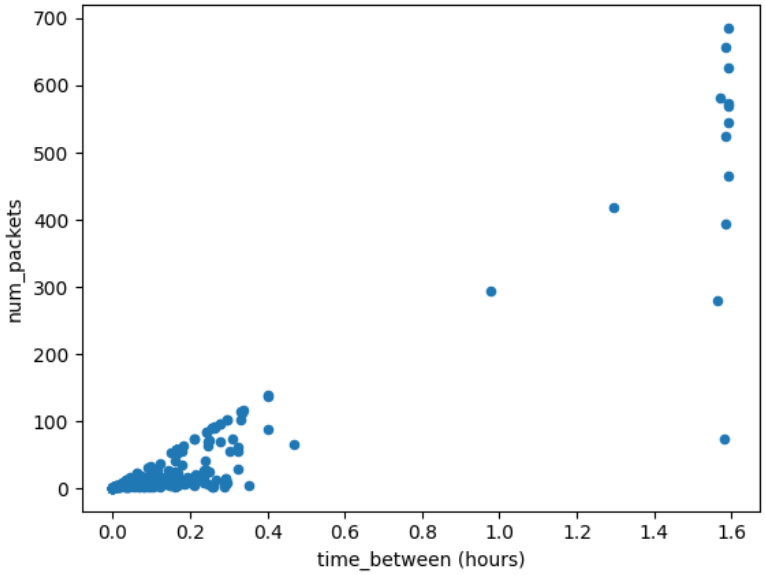


This first graph is taken from our summary statistics. Of note, the largest portion is made up of Wi-Fi clients. A Wi-Fi client is any device that transmits or receives Wi-Fi data but does not broadcast its own SSID, e.g. a laptop. Wi-Fi AP is a Wi-Fi access point or a device with broadcasts an SSID. Then there is BTLE, with only a couple being BR/EDR. Bluetooth LE allows for short bursts of long-range radio connection that prolong battery life. Classic Bluetooth (or Bluetooth BR/EDR) establishes a short-range, continuous wireless connection, which makes it ideal for use cases such as streaming audio. It makes sense that classic Bluetooth is not in use as much anymore

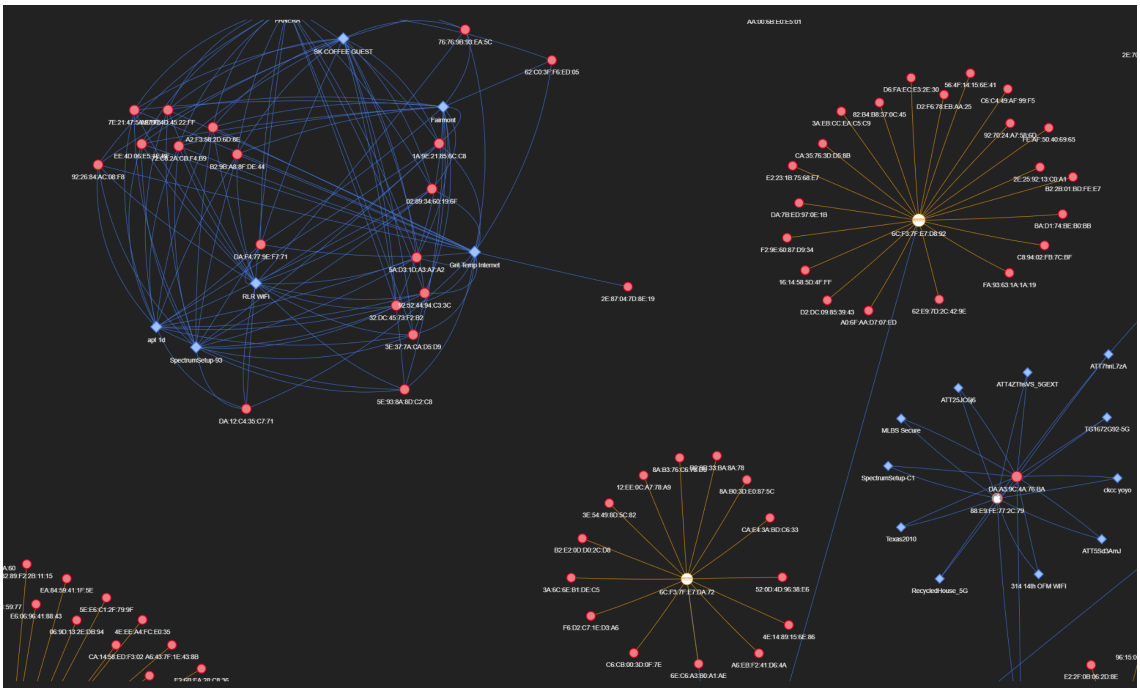


-

This second graph is taken from our WLAN statistics, and it details device names. The largest portion is made up of unknown names, and the last four names are Mac addresses.



This third graph is taken from our Bluetooth statistics. This graph concerns packet data. Most devices sent between 0 and 25 packets, and the number of packets sent was proportional to the time between the time of the last packet observed of that device minus the first observed packet time, with a few exceptions of a couple of devices sending 10, 20 thousand packets over long periods of time.



This fourth graph is taken from our SSID information. SSID is just an identifier for a network. Connections go from SSIDs to access points to devices. Here we sampled 1000 devices randomly. SSIDs are in blue, Access Points are in orange, and Wi-Fi clients are in Red. Smaller networks don't necessarily have access points.

Privacy Analysis

While the ability to collect all of this data allows for interesting data analysis, there is a privacy concern. In particular, in the SSID graph, devices probe for Wi-Fi connections, and as a result this can expose networks that devices have previously been connected to. Thus if a bad actor is aware of the MAC address of a particular person's device, then that bad actor can determine other places that person has been recently by checking what networks the device has probed for. This is a fairly significant privacy risk, and could likely be accounted for by only including networks that devices actually connected to, and leaving out networks that the device just probed, in order to display only active connections. This would erase the ability to use the dashboard tool to check a device's location history, thereby increasing privacy of the owners of all the devices.

Querying

In addition to doing data analysis and presenting a summary of analysis visuals for WLAN, Bluetooth, and All devices (summary), we also present a feature to query information and get a bit more of a privacy analysis for individual devices. Users can access a Query functionality on the frontend, which enables them to enter a specific MAC address and click a Search button. The Mac address they queried for will be sent to our backend script which will then look through our dataframe for that specific device. Information on device type, device name, manufacturer, number of packets, first/last seen timestamp, probed SSID's, and connected SSID is provided. Information like device type, manufacturer, and number of packets helps determine what the device is and what it is doing on the network. Timestamp information such as first/last seen is useful for showing how long a device is present. SSID information in particular is useful and something we added/looked into based on the professor's suggestion. Access points and wireless router networks broadcast their SSIDs to identify themselves. We chose to add information on SSIDs for a better privacy analysis because it enables us to see which network the device is connected to and the networks it searches for.

Note for the professor/TA's: if running our demo code, we currently only support campus data. Some example MAC addresses for testing the querying functionality are 4C:79:6E:1F:DA:A1, F8:4D:89:5D:34:36, 28:EA:2D:1C:B1:83, E8:5F:02:38:7F:DE.

Limitations and Challenges

Monitor Mode

When first beginning to try to collect data, we attempted to do so through our laptops. However, the only data that was able to be collected through our laptops was data that was directly related to our laptops. In order to collect other WLAN data, it was necessary to enable monitor mode on the device used to capture data; however, monitor mode is unsupported by most Windows laptops and some Macs. The way we resolved

this was by instead using a Raspberry Pi and a capture card that could enable monitor mode. This enabled us to capture nearby WLAN traffic data of all devices.

Kismet

Another challenge was that we wanted to use Kismet as our sniffer tool. However, Kismet doesn't work well with Windows, as it is mainly developed for Linux and macOS. Therefore, only one of our team members was able to use Kismet, but it was acceptable because the capture card could also only be in one place at a time.

Data Locations

Another limitation faced was where we could collect data. Since the aim was to collect a lot of data, this involved setting up the laptop with the Raspberry Pi and leaving it at the data collection location for many hours. Thus since it was infeasible to leave the devices unattended to collect data at other locations, data was only able to be collected at Brown (which is mainly on the eduroam network) and at The Flats, i.e. places where our team members lived.

Unavailable Information

Most of the data that we were able to collect was through eduroam. However, the limitation here is that because of how secure eduroam is, there was not as much data available on eduroam as there might have been on other networks, so there is potential to do deeper data analysis on networks that have more information available.

Another limitation faced was that device manufacturer information was unable to be collected from the vast majority of devices, and this was a statistic that was of interest to us.

Participation

Kasra

Kasra setup up the Raspberry Pi with Kismet, performed the associated debugging procedures, and wrote the section in this report which describes that setup process. Subsequently, he examined the Kismet API docs and wrote a script for exporting relevant data. He wrote the script for creating the SSID-graph visualization which involved sourcing a Python package designed to visualize graphs as well as cleaning the data from Kismet. While Kismet provided a solid foundation, Kasra had to develop scripts to match Device→AP→SSID. He also extended his script to gather probed SSIDs to enable the privacy analysis. Additionally, Kasra sourced the base theme for the dashboard and created the slides on his work and the threat model that NERVE seeks to address. Finally, he improved the performance of the Dashboard codebase by replacing many for-loops with vectorized operations.

Nicki

Nicki's main focus was coding the summary graphs for the real-time dashboard and they did the summary analysis. Additionally, they spent many hours working with the aim of making the graphs appear more presentable, as some of them did not turn out very well on the first attempt. They also worked to help debug the code of the real-time dashboard. The main part of the presentation they worked on was the limitations and challenges, although some of it was cut due to time constraints. Additionally, they put together the initial draft of this report.

Sneha

Sneha did the WLAN analysis. This involved writing a script to create the charts of the WLAN data. She also spent time formatting these and making the graphs appear more presentable. Sneha also did the initial setup of the Dashboard (front end and back end). This involved creating the main flask file and the HTML templates for the front end.

Later on, other templates were used/created. She also worked on debugging any issues with the real-time dashboard. The main part of the presentation she worked on was the introduction as well as explaining how the dashboard was created. Moreover, she and Neha worked on the querying/search functionality for getting more information and conducted a bit of privacy analysis on individual devices (based on a user querying a specific MAC address) based on the Professor's feedback after the presentation. Finally, she also contributed to editing/writing part of the report after the initial draft.

Neha

Neha primarily worked on the Bluetooth analysis. This involved parsing the Bluetooth data and initially analyzing the present features and many of the trends, as well as creating many graphs and working on making sure they looked good, within a Jupyter notebook. Then Neha chose some of the main graphs to present for Bluetooth and incorporated them into the Bluetooth page HTML, wrote the Bluetooth processing script, and added relevant functions into the main Python files. Neha also worked on debugging for both the real-time data as well as general issues. The main part of the presentation she worked on was explaining the data retrieval, processing, and analysis steps. Moreover, she and Sneha worked on the querying/search functionality for getting more information and conducted a bit of privacy analysis on individual devices (based on user querying a specific MAC address) based on the Professor's feedback after the presentation. Lastly, she also edited the draft of the final report.