cafe chatter - a live packet profiler

```
a live packet capturing system using wireshark and a custom python script to consolidate ip and mac
addresses with their unencrypted traffic. captured data is then analyzed using gemini to generate
behavioural profiles, answering the question: "who is this?".

created as an investigation into real-world netowrk acitivity in public environments, exploring how
much can be inferred from passive observation.
```

Tools used:

- Wireshark / `tshark` / `airmon` suite for packet capture
- Kali Linux VM with VirtualBox to use monitor mode
- ALFA Wifi adapter to capture others' packets
- Custom Python scripts for analysis and clean up
- Gemini API for profiling

---

technical details

> **ethical note**
>
> The exact details of my process (commands and more volatile instructions) are not published publicly to avoid misuse. I am happy to discuss details privately or in interview contexts.

---

wifi types observed

There are 3 types of WIFI that I looked at for this project.

1. **Open** - unencrypted; any unencrypted protocols can be seen (DNS, TCP, HTTP, etc) in full.
2. **WPA-Personal** - password protected
3. **WPA-Enterprise** - user/password protected

WPA-Personal is password protected. When connect, your device and the router do the handshake and establish a secret key that your traffic will be encrypted with. By intercepting the handshake and having the WIFI password, it is able to be decrypted as if there were no password.
WPA-Enterprise is user and password protected. Even with the 4-way handshake, you need the unique credentials for each account to be able to do any decryption, which is basically impossible to acquire passively.

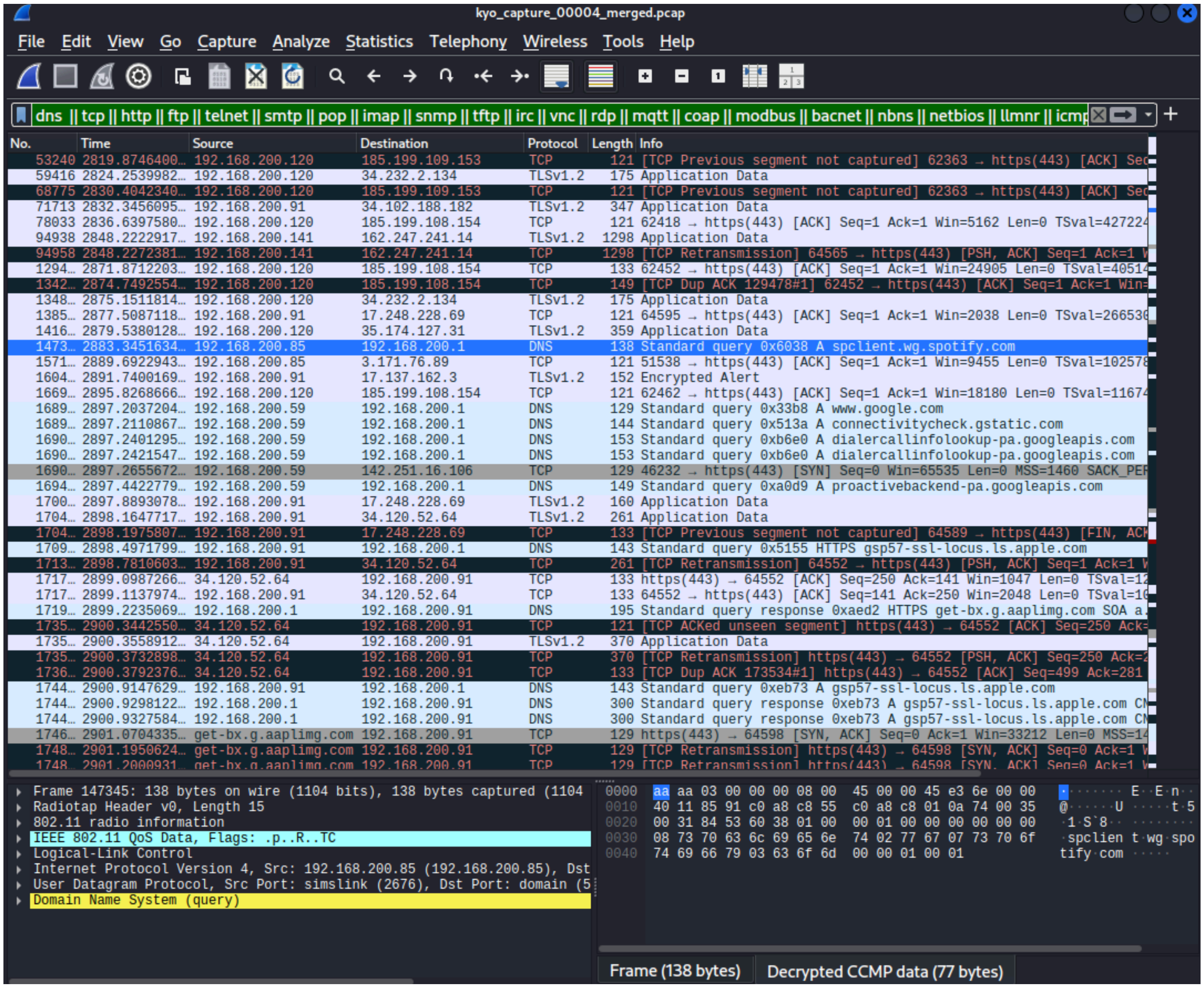
> for this project the cafe I frequented used WPA-Personal with a publicly available password for patrons.

---

capture and filtering

When you access a website and the IP address is not in your machine's cache, it will go to DNS server and ask for the IP. This request is typically unencrypted and can be quite frequent, so it was my primary profiling vector.

I used the aircrack-ng suite to locate active channels and collect information about the exact configuration of the WIFI and then started capture with `tshark`. Due to the high volume of traffic, I had a separate capture for handshakes, as not to miss them, and then stitched them onto the main pcaps after capture for decryption.


kyo_capture_00004_merged.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Frame (138 bytes)
Decrypted CCMP data (77 bytes)

```
dns || tcp || http || ftp || telnet || smtp || pop || imap || snmp || tftp || irc || vnc || rdp || mqtt || coap || modbus || bacnet || nbns || netbios || llmnr || icmp

No.   Time          Source              Destination      Protocol Length Info
53240 2819.8746400… 192.168.200.120     185.199.109.153  TCP       121 [TCP Previous segment not captured] 62363 → https(443) [ACK] Sec
59416 2824.2539982… 192.168.200.120     34.232.2.134     TLSv1.2   175 Application Data
68775 2830.4042340… 192.168.200.120     185.199.109.153  TCP       121 [TCP Previous segment not captured] 62363 → https(443) [ACK] Sec
71713 2832.3456095… 192.168.200.91      34.102.188.182   TLSv1.2   347 Application Data
78033 2836.6397580… 192.168.200.120     185.199.108.154  TCP       121 62418 → https(443) [ACK] Seq=1 Ack=1 Win=5162 Len=0 TSval=427224
94938 2848.2222917… 192.168.200.141     162.247.241.14   TLSv1.2  1298 Application Data
94958 2848.2272381… 192.168.200.141     162.247.241.14   TCP      1298 [TCP Retransmission] 64565 → https(443) [PSH, ACK] Seq=1 Ack=1 W
1294… 2871.8712203… 192.168.200.120     185.199.108.154  TCP       133 62452 → https(443) [ACK] Seq=1 Ack=1 Win=24905 Len=0 TSval=40514
1342… 2874.7492554… 192.168.200.120     185.199.108.154  TCP       149 [TCP Dup ACK 129478#1] 62452 → https(443) [ACK] Seq=1 Ack=1 Win=
1348… 2875.1511814… 192.168.200.120     34.232.2.134     TLSv1.2   175 Application Data
1385… 2877.5087118… 192.168.200.91      17.248.228.69    TCP       121 64595 → https(443) [ACK] Seq=1 Ack=1 Win=2038 Len=0 TSval=266530
1416… 2879.5380128… 192.168.200.120     35.174.127.31    TLSv1.2   359 Application Data
1473… 2883.3451634… 192.168.200.85      192.168.200.1    DNS       138 Standard query 0x6038 A spclient.wg.spotify.com
1571… 2889.6922943… 192.168.200.85      3.171.76.89      TCP       121 51538 → https(443) [ACK] Seq=1 Ack=1 Win=9455 Len=0 TSval=102578
1604… 2891.7400169… 192.168.200.91      17.137.162.3     TLSv1.2   152 Encrypted Alert
1669… 2895.8268666… 192.168.200.120     185.199.108.154  TCP       121 62462 → https(443) [ACK] Seq=1 Ack=1 Win=18180 Len=0 TSval=11674
1689… 2897.2037204… 192.168.200.59      192.168.200.1    DNS       129 Standard query 0x33b8 A www.google.com
1689… 2897.2110867… 192.168.200.59      192.168.200.1    DNS       144 Standard query 0x513a A connectivitycheck.gstatic.com
1690… 2897.2401295… 192.168.200.59      192.168.200.1    DNS       153 Standard query 0xb6e0 A dialercallinfolookup-pa.googleapis.com
1690… 2897.2421547… 192.168.200.59      192.168.200.1    DNS       153 Standard query 0xb6e0 A dialercallinfolookup-pa.googleapis.com
1690… 2897.2655672… 192.168.200.59      142.251.16.106   TCP       129 46232 → https(443) [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PER
1694… 2897.4422779… 192.168.200.59      192.168.200.1    DNS       149 Standard query 0xa0d9 A proactivebackend-pa.googleapis.com
1700… 2897.8893078… 192.168.200.91      17.248.228.69    TLSv1.2   160 Application Data
1704… 2898.1647717… 192.168.200.91      34.120.52.64     TLSv1.2   261 Application Data
1704… 2898.1975807… 192.168.200.91      17.248.228.69    TCP       133 [TCP Previous segment not captured] 64589 → https(443) [FIN, ACK
1709… 2898.4971799… 192.168.200.91      192.168.200.1    DNS       143 Standard query 0x5155 HTTPS gsp57-ssl-locus.ls.apple.com
1713… 2898.7810603… 192.168.200.91      34.120.52.64     TCP       261 [TCP Retransmission] 64552 → https(443) [PSH, ACK] Seq=1 Ack=1 W
1717… 2899.0987266… 34.120.52.64        192.168.200.91   TCP       133 https(443) → 64552 [ACK] Seq=250 Ack=141 Win=1047 Len=0 TSval=12
1717… 2899.1137974… 192.168.200.91      34.120.52.64     TCP       133 64552 → https(443) [ACK] Seq=141 Ack=250 Win=2048 Len=0 TSval=16
1719… 2899.2235069… 192.168.200.1       192.168.200.91   DNS       195 Standard query response 0xaed2 HTTPS get-bx.g.aaplimg.com SOA a.
1735… 2900.3442550… 34.120.52.64        192.168.200.91   TCP       121 [TCP ACKed unseen segment] https(443) → 64552 [ACK] Seq=250 Ack=
1735… 2900.3558912… 34.120.52.64        192.168.200.91   TLSv1.2   370 Application Data
1735… 2900.3732898… 34.120.52.64        192.168.200.91   TCP       370 [TCP Retransmission] https(443) → 64552 [PSH, ACK] Seq=250 Ack=2
1736… 2900.3792376… 34.120.52.64        192.168.200.91   TCP       133 [TCP Dup ACK 173534#1] https(443) → 64552 [ACK] Seq=499 Ack=281
1744… 2900.9147629… 192.168.200.91      192.168.200.1    DNS       143 Standard query 0xeb73 A gsp57-ssl-locus.ls.apple.com
1744… 2900.9298122… 192.168.200.1       192.168.200.91   DNS       300 Standard query response 0xeb73 A gsp57-ssl-locus.ls.apple.com CN
1744… 2900.9327584… 192.168.200.1       192.168.200.91   DNS       300 Standard query response 0xeb73 A gsp57-ssl-locus.ls.apple.com CN
1746… 2901.0704335… get-bx.g.aaplimg.com 192.168.200.91  TCP       129 https(443) → 64598 [SYN, ACK] Seq=0 Ack=1 Win=33212 Len=0 MSS=14
1748… 2901.1950624… get-bx.g.aaplimg.com 192.168.200.91  TCP       129 [TCP Retransmission] https(443) → 64598 [SYN, ACK] Seq=0 Ack=1 W
1748… 2901.2000931… get-bx.g.aaplimg.com 192.168.200.91  TCP       129 [TCP Retransmission] https(443) → 64598 [SYN, ACK] Seq=0 Ack=1 W
```

```
▸ Frame 147345: 138 bytes on wire (1104 bits), 138 bytes captured (1104
▸ Radiotap Header v0, Length 15
▸ 802.11 radio information
▸ IEEE 802.11 QoS Data, Flags: .p..R..TC
▸ Logical-Link Control
▸ Internet Protocol Version 4, Src: 192.168.200.85 (192.168.200.85), Dst
▸ User Datagram Protocol, Src Port: simslink (2676), Dst Port: domain (5
▸ Domain Name System (query)
```

```
0000  aa aa 03 00 00 00 08 00  45 00 00 45 e3 6e 00 00   ········ E··E·n··
0010  40 11 85 91 c0 a8 c8 55  c0 a8 c8 01 0a 74 00 35   @······U ·····t·5
0020  00 31 84 53 60 38 01 00  00 01 00 00 00 00 00 00   ·1·S`8·· ········
0030  08 73 70 63 6c 69 65 6e  74 02 77 67 07 73 70 6f   ·spclien t·wg·spo
0040  74 69 66 79 03 63 6f 6d  00 00 01 00 01            tify·com ·····
```

Sample Wireshark decrypted capture with non-encrypted filters

Working on-site, there were some complications with capture. Below are some issues that I ran into and my solutions.

Problem	Solution
ALFA adapter had low packet capture rate	Filtered with `eapol`, change capture times, or (future) upgraded hardware
`.pcap` files too large	Split files, used live filters, and rotated capture files
Missed handshakes	Forced handshakes to happen again and captured before peak hours
Poor frequency capture	Used aircrack-ng suite to lock to target channel

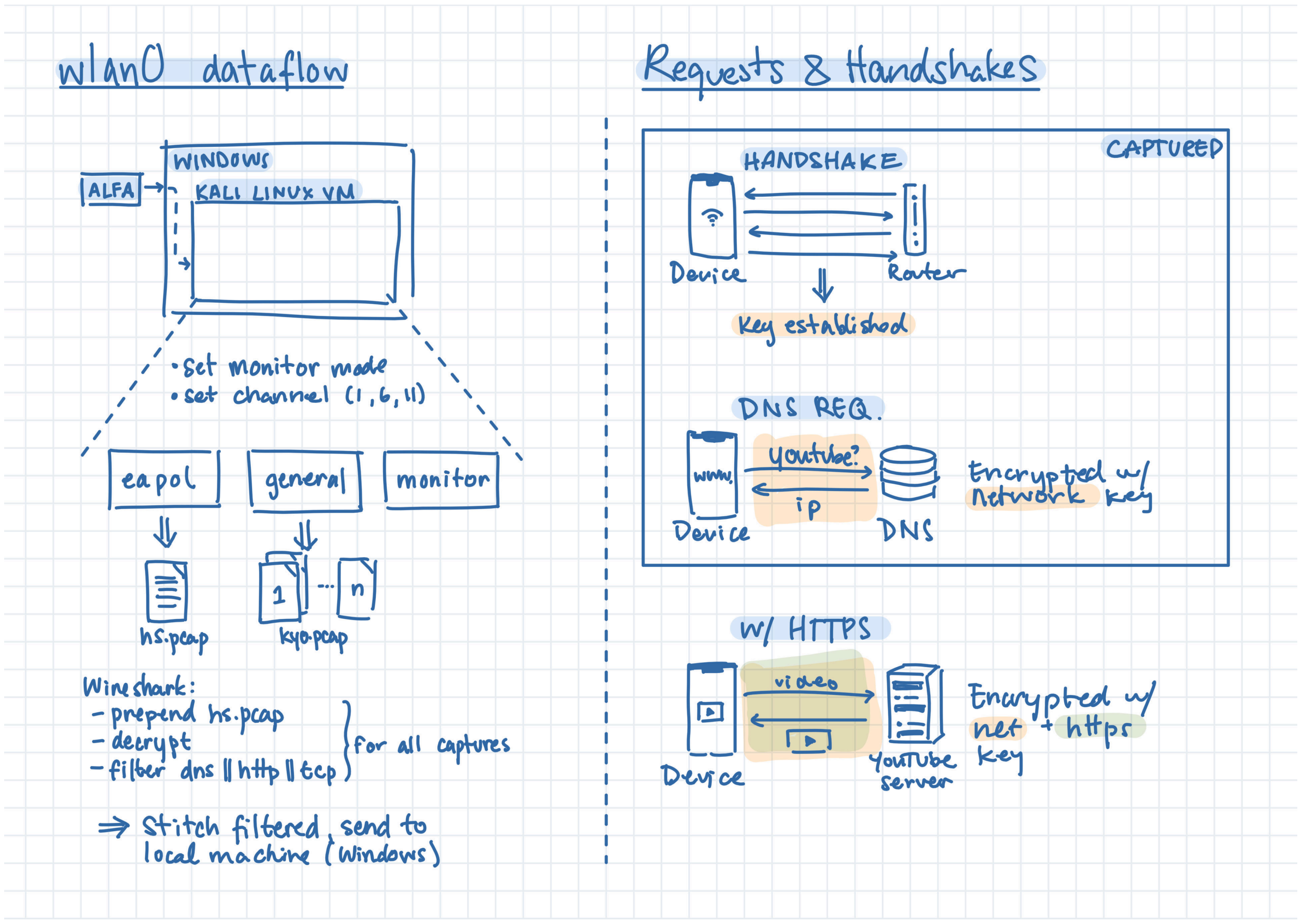


Diagram of general data flow (wlan0 was the previous project name)

---

data processing and analysis

After all the packets were captured, I decrypted and converted them to CSV files and sent them to my local machine to do further analysis. I wrote a Python script that consolidated all the non-encrypted packets to be sorted by IP (essentially by device). This also allowed me to reveal what IPs were given to what domains that they were searching for, just by matching request to response.
Sample output of grouping:

```
IP: 192.168.200.59
=================================
ENTRY 242134 -----------
dest: www.google.com @
time: 1936.409025507
ref : 0x5830

ENTRY 244970 -----------
dest: instantmessaging-pa-jms-us.googleapis.com @
time: 1937.962279076
ref : 0x0a7d

ENTRY 247821 -----------
dest: voilatile-pa.googleapis.com @
time: 1939.601636880
ref : 0x3b32

ENTRY 382168 -----------
dest: pixelweatherhub-pa.googleapis.com @
time: 2040.176508006
```

```
ref : 0x0e39

ENTRY 248796 -----------
dest: connectivitycheck.gstatic.com @
time: 2462.386269794
ref : 0x7609

...
```

For analysis, the above data was sent to Gemini with a guiding prompt to come up with a creative way to present a highly summarized manner. While this is a more playful approach, just by changing the prompt, more useful and accurate reports can be generated.
I can see this process being used to quickly condense data, and could pair well as an initial summary before further details can be gathered on key areas, anomalies, and points of interest.

> This project left me surprised at how much passive metadata can reveal about behavior. The profiling step felt simultaneously clever and unsettling — a small reminder of how fragile digital privacy can be in open environments.

---

links

- Python Source Code
- Python Sample Output
- Gemini Sample Output

thoughts regarding the use of AI

There were two ways I used AI in this project. One was in profiling the users with Gemini. This I think is a practical use for AI as it automates a lot of time-consuming parsing, as long as its used in moderation.
The other way, using DeepSeek to aid my learning while I built this entire system,
I was able to learn more about how networks worked by just applying myself and learning. There are small details that I picked up while putting everything together, and I think that was extremely productive.
However, it's scary how easily I was able to put this together. It is able to allow malicious actors to act faster and above their skill level. It brings into question how we can prevent the "hacking" of AI to build something malicious while still allowing it to be useful to the right people. I don't know if that will ever be possible and will always be something concerning.