Drone Networking

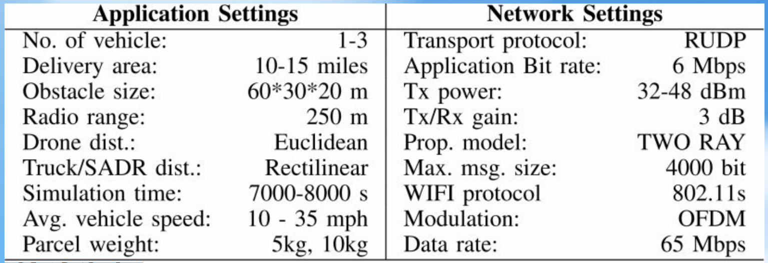
Firewall Force

Connor Plonka, Nicholas Paceski, Andrew Destacamento, Lazaro Loureiro, Samson Silver, Richardson Jacques

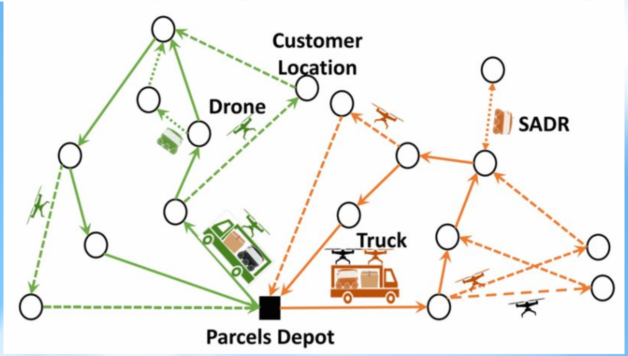
Mentor: Nicholas Taylor

Context

- Network of delivery trucks, sidewalk autonomous delivery robots (SADRs), and drones.
  - Trucks have a mesh WiFi system
  - SADRs and drones connect to truck WiFi
  - Truck WiFi connect to Ground Control Station (GCS)
- Based on “Environmentally-Aware Robotic Vehicle Networks Routing Computation for Last-mile Deliveries” by Qu et al. (2023)
  - 2023 32nd International Conference on Computer Communications and Networks (ICCCN)
  - Department of Electrical Engineering and Computer Science, University of Missouri - Columbia, USA
    - No actual drones, SADRs, or routers available



Application Settings		Network Settings	
No. of vehicle:	1-3	Transport protocol:	RUDP
Delivery area:	10-15 miles	Application Bit rate:	6 Mbps
Obstacle size:	60*30*20 m	Tx power:	32-48 dBm
Radio range:	250 m	Tx/Rx gain:	3 dB
Drone dist.:	Euclidean	Prop. model:	TWO RAY
Truck/SADR dist.:	Rectilinear	Max. msg. size:	4000 bit
Simulation time:	7000-8000 s	WIFI protocol	802.11s
Avg. vehicle speed:	10 - 35 mph	Modulation:	OFDM
Parcel weight:	5kg, 10kg	Data rate:	65 Mbps



Overview

We've chosen to work on the security aspects of drone networking, creating a system in which the drones themselves and the network they run on are protected from attacks.

We plan to protect the network through various means, such as:

- IP/MAC Address Rotation
- WPA3 Authentication
- Rate Limiting
- Flooding Prevention
- And others

Implementation Details

01 IP and MAC rotation

- Randomized, but valid, IP and MAC address renewal

02 Rate Limiting & Authentication Flooding Prevention

- Secure secret password between truck and drones.
- Encrypted tokens for authenticated drones.

03 Secure P2P Networking

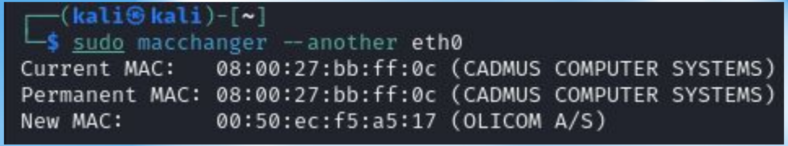
- VXLAN over IPSec

04 WPA3 Authentication

- RADIUS server w/o a central authentication point
- Hacking prevention measures

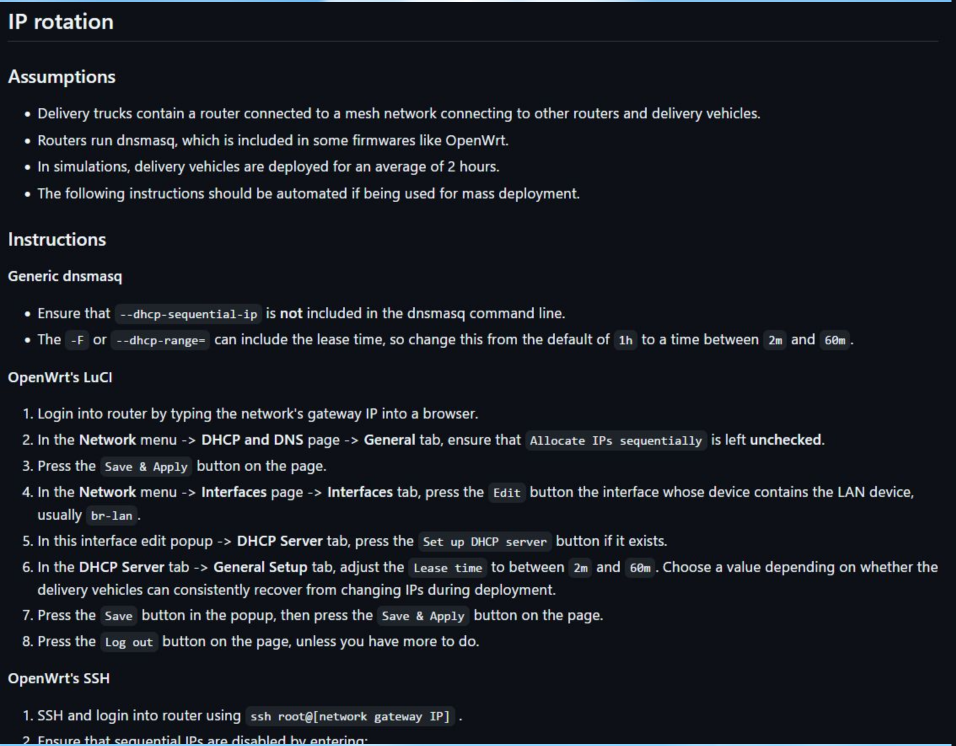
05 Firewall Defense

IP and MAC rotation



```
┌──(kali㉿kali)-[~]
└─$ sudo macchanger --another eth0
Current MAC:   08:00:27:bb:ff:0c (CADMUS COMPUTER SYSTEMS)
Permanent MAC: 08:00:27:bb:ff:0c (CADMUS COMPUTER SYSTEMS)
New MAC:       00:50:ec:f5:a5:17 (OLICOM A/S)
```

- IP address randomization (e.g. `192.168.100.100 → 192.168.100.112`)
  - Handled by truck WiFi, depending on router firmware
  - Solution assumes dnsmasq is used, such as in OpenWrt
  - Proposed solution: Instructions on how to set up dnsmasq for randomization
- MAC address randomization (e.g. `08:00:27:bb:ff:0c → 00:50:ec:f5:a5:17`)
  - Handled by device firmware
  - Solution assumes a Debian-based firmware
  - Proposed solution: **`systemd`** service, uses GNU **`macchanger`** package
    - **`macchanger`** does not disconnect from network
- Rotation:
  - Average simulated delivery drone deploy time is about 2 hours
  - Allow user to edit rotation timings
    - IP: DHCP lease time (2 minutes to 60 minutes, unspecified)
    - MAC: service script **`sleep`** command (3 minutes to 13 minutes, random)



IP rotation

Assumptions

- Delivery trucks contain a router connected to a mesh network connecting to other routers and delivery vehicles.
- Routers run dnsmasq, which is included in some firmwares like OpenWrt.
- In simulations, delivery vehicles are deployed for an average of 2 hours.
- The following instructions should be automated if being used for mass deployment.

Instructions

Generic dnsmasq

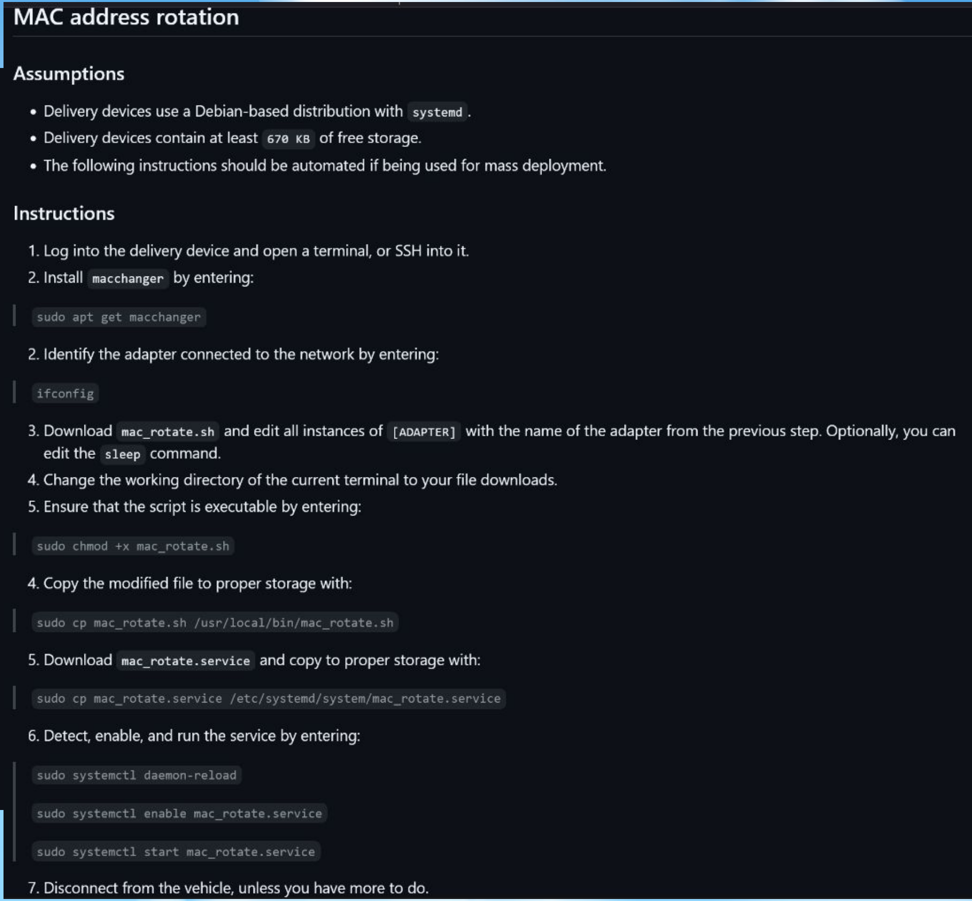
- Ensure that `--dhcp-sequential-ip` is **not** included in the dnsmasq command line.
- The `-F` or `--dhcp-range=` can include the lease time, so change this from the default of `1h` to a time between `2m` and `60m` .

OpenWrt's LuCI

1. Login into router by typing the network's gateway IP into a browser.
2. In the **Network** menu -> **DHCP and DNS** page -> **General** tab, ensure that `Allocate IPs sequentially` is left **unchecked**.
3. Press the `Save & Apply` button on the page.
4. In the **Network** menu -> **Interfaces** page -> **Interfaces** tab, press the `Edit` button the interface whose device contains the LAN device, usually `br-lan` .
5. In this interface edit popup -> **DHCP Server** tab, press the `Set up DHCP server` button if it exists.
6. In the **DHCP Server** tab -> **General Setup** tab, adjust the `Lease time` to between `2m` and `60m` . Choose a value depending on whether the delivery vehicles can consistently recover from changing IPs during deployment.
7. Press the `Save` button in the popup, then press the `Save & Apply` button on the page.
8. Press the `Log out` button on the page, unless you have more to do.

OpenWrt's SSH

1. SSH and login into router using `ssh root@[network gateway IP]` .
2. Ensure that sequential IPs are disabled by entering:



MAC address rotation

Assumptions

- Delivery devices use a Debian-based distribution with `systemd` .
- Delivery devices contain at least `670 KB` of free storage.
- The following instructions should be automated if being used for mass deployment.

Instructions

1. Log into the delivery device and open a terminal, or SSH into it.
2. Install `macchanger` by entering:

```
sudo apt get macchanger
```

2. Identify the adapter connected to the network by entering:

```
ifconfig
```

3. Download `mac_rotate.sh` and edit all instances of `[ADAPTER]` with the name of the adapter from the previous step. Optionally, you can edit the `sleep` command.
4. Change the working directory of the current terminal to your file downloads.
5. Ensure that the script is executable by entering:

```
sudo chmod +x mac_rotate.sh
```

4. Copy the modified file to proper storage with:

```
sudo cp mac_rotate.sh /usr/local/bin/mac_rotate.sh
```

5. Download `mac_rotate.service` and copy to proper storage with:

```
sudo cp mac_rotate.service /etc/systemd/system/mac_rotate.service
```

6. Detect, enable, and run the service by entering:

```
sudo systemctl daemon-reload
```

```
sudo systemctl enable mac_rotate.service
```

```
sudo systemctl start mac_rotate.service
```

7. Disconnect from the vehicle, unless you have more to do.

Rate Limiting & Flooding Prevention

Static drone IP's, Dynamic tokens

Truck

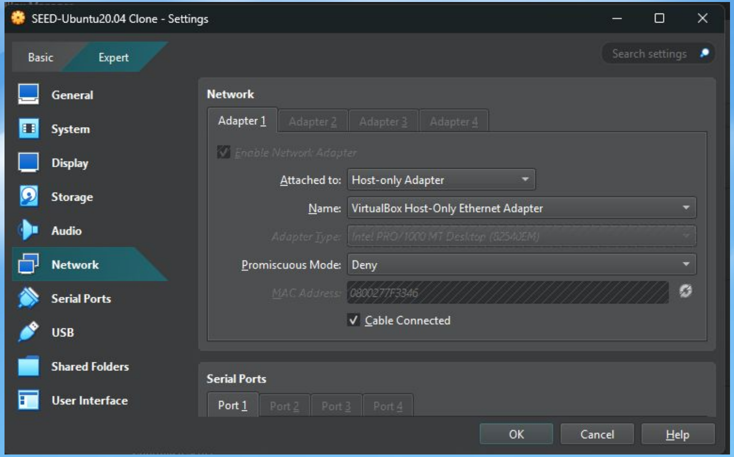
- Secure key only shared with the truck and the drones.
- Store drone IP to to track attempts and prevent flooding.
- Generate hashed token using the secure key (HMAC).
- Authenticate with drone by sending POST requests using token generated (send data to a server).

Drone

- Verify that the token generated by the truck is the same as the drone.
- If token matches, authentication succeeds and any command sent from truck is ran.
- Authentication request to the truck takes drone IP and validates the token.

Rate Limiting & Flooding Prevention

Static drone IP's, Dynamic tokens **(SIMULATION)**



Network

Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

✓ Enable Network Adapter

Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

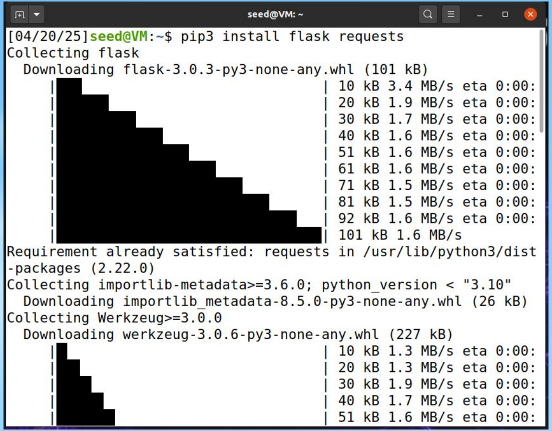
Promiscuous Mode: Deny

MAC Address: 0800277F3346

✓ Cable Connected

Serial Ports

Port 1 | Port 2 | Port 3 | Port 4



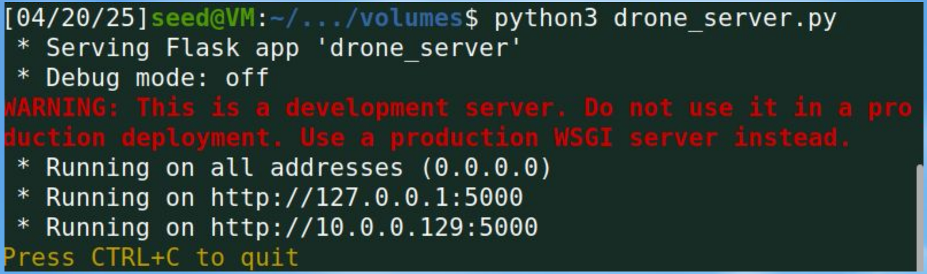
```
[04/20/25]seed@VM:~$ pip3 install flask requests
Collecting flask
  Downloading flask-3.0.3-py3-none-any.whl (101 kB)
     |███                             | 10 kB 3.4 MB/s eta 0:00:
     |██████                          | 20 kB 1.9 MB/s eta 0:00:
     |█████████                       | 30 kB 1.7 MB/s eta 0:00:
     |█████████████                   | 40 kB 1.6 MB/s eta 0:00:
     |████████████████                | 51 kB 1.6 MB/s eta 0:00:
     |███████████████████             | 61 kB 1.6 MB/s eta 0:00:
     |██████████████████████          | 71 kB 1.5 MB/s eta 0:00:
     |██████████████████████████      | 81 kB 1.5 MB/s eta 0:00:
     |█████████████████████████████   | 92 kB 1.6 MB/s eta 0:00:
     |████████████████████████████████| 101 kB 1.6 MB/s
Requirement already satisfied: requests in /usr/lib/python3/dist
-packages (2.22.0)
Collecting importlib-metadata>=3.6.0; python_version < "3.10"
  Downloading importlib_metadata-8.5.0-py3-none-any.whl (26 kB)
Collecting Werkzeug>=3.0.0
  Downloading werkzeug-3.0.6-py3-none-any.whl (227 kB)
     |█                               | 10 kB 1.3 MB/s eta 0:00:
     |██                              | 20 kB 1.3 MB/s eta 0:00:
     |████                            | 30 kB 1.9 MB/s eta 0:00:
     |█████                           | 40 kB 1.7 MB/s eta 0:00:
     |███████                         | 51 kB 1.6 MB/s eta 0:00:
```

The truck and drone simulations take place in Ubuntu VM.

Using 2 Network adapters for the VM's, one bridged to physical LAN and second as a host-only adapter.

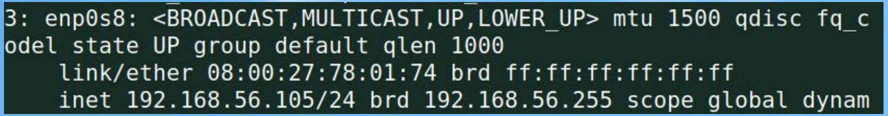
This simulates the connection of drone to truck.

Installed flask requests on both VMs to simulate server requests.



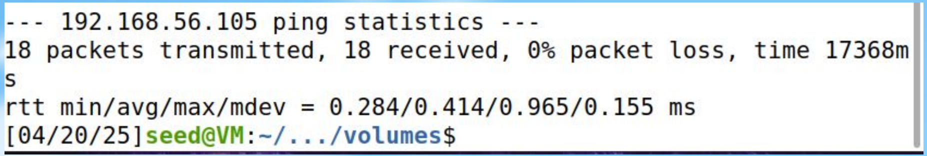
```
[04/20/25]seed@VM:~/.../volumes$ python3 drone_server.py
 * Serving Flask app 'drone_server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a pro
duction deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.0.0.129:5000
Press CTRL+C to quit
```

Ran drone_server.py on the drone VM.



```
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_c
odel state UP group default qlen 1000
    link/ether 08:00:27:78:01:74 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.105/24 brd 192.168.56.255 scope global dynam
```

Ran “ip a” terminal command to get drone IP to add to truck_controller.py.



```
--- 192.168.56.105 ping statistics ---
18 packets transmitted, 18 received, 0% packet loss, time 17368m
s
rtt min/avg/max/mdev = 0.284/0.414/0.965/0.155 ms
[04/20/25]seed@VM:~/.../volumes$
```

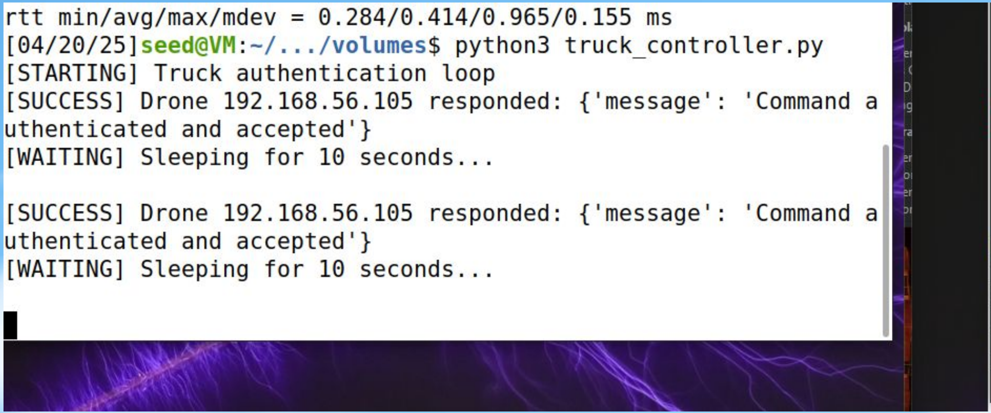
Pinged drone on truck to show connection.

I ran the respective scripts on both VMs (truck and drone).

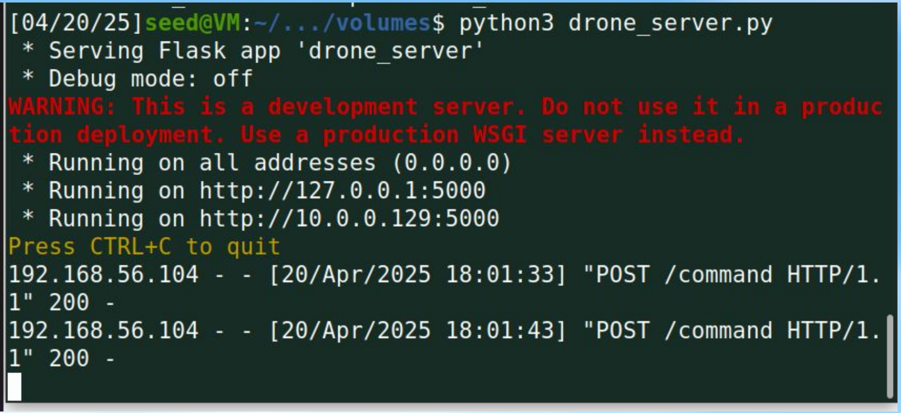
The drone server is receiving flask requests to authenticate with the truck controller.

This allows the truck to send whatever commands to an authenticated drone.

After authenticating, the truck will wait a number of seconds before trying to authenticate with the drone again.

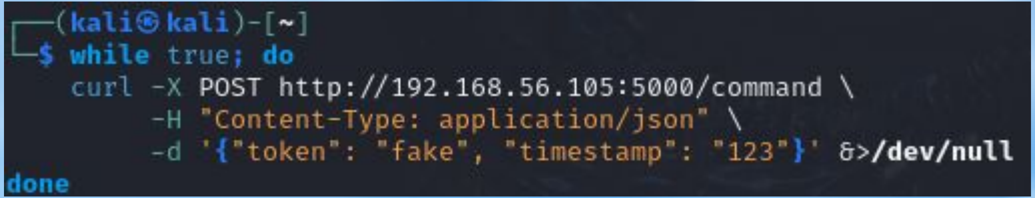


```
rtt min/avg/max/mdev = 0.284/0.414/0.965/0.155 ms
[04/20/25]seed@VM:~/.../volumes$ python3 truck_controller.py
[STARTING] Truck authentication loop
[SUCCESS] Drone 192.168.56.105 responded: {'message': 'Command a
uthenticated and accepted'}
[WAITING] Sleeping for 10 seconds...

[SUCCESS] Drone 192.168.56.105 responded: {'message': 'Command a
uthenticated and accepted'}
[WAITING] Sleeping for 10 seconds...
```



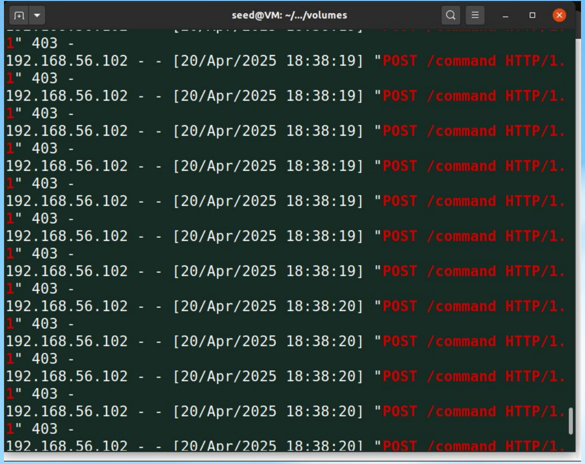
```
[04/20/25]seed@VM:~/.../volumes$ python3 drone_server.py
 * Serving Flask app 'drone_server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a produc
tion deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.0.0.129:5000
Press CTRL+C to quit
192.168.56.104 - - [20/Apr/2025 18:01:33] "POST /command HTTP/1.
1" 200 -
192.168.56.104 - - [20/Apr/2025 18:01:43] "POST /command HTTP/1.
1" 200 -
```

Here I attempted flooding the drone_server by repeatedly sending fake requests to the drone.



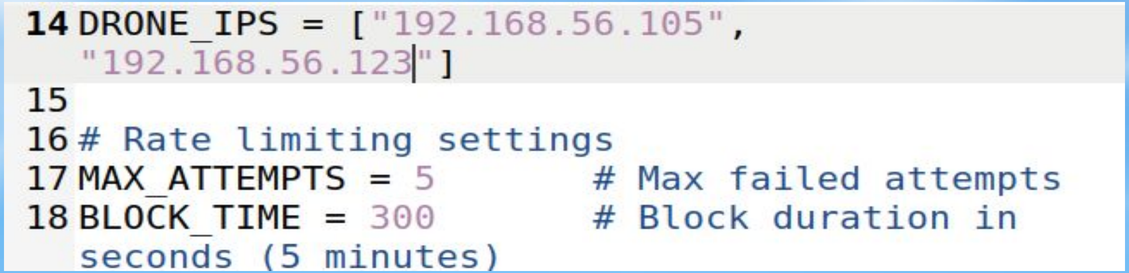
```
┌──(kali㉿kali)-[~]
└─$ while true; do
    curl -X POST http://192.168.56.105:5000/command \
         -H "Content-Type: application/json" \
         -d '{"token": "fake", "timestamp": "123"}' &>/dev/null
done
```

These attempts were unsuccessful in authenticating with the drone. This shows how the truck is the only one allowed to send commands to the drones.



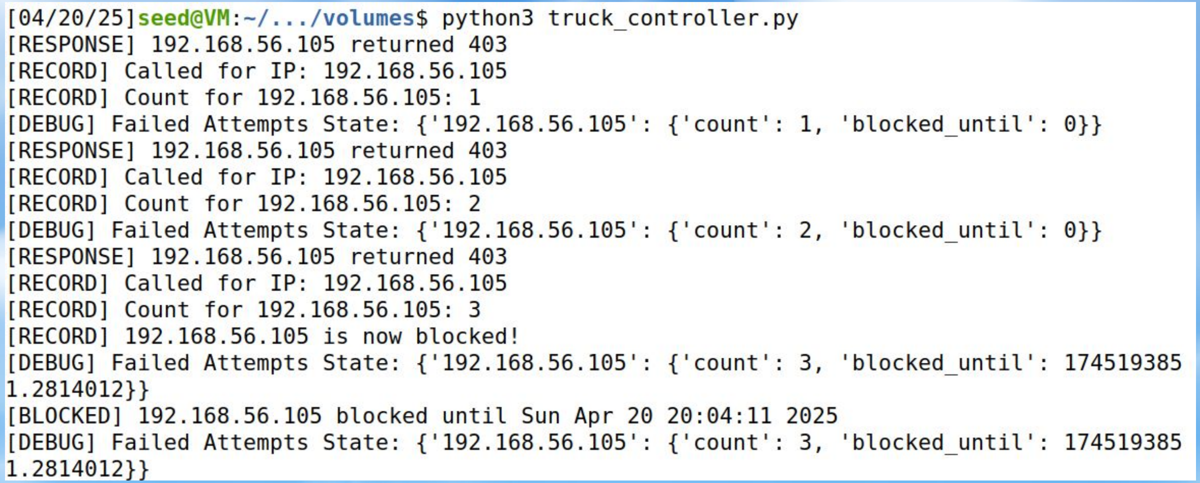
```
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:19] "POST /command HTTP/1.
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:19] "POST /command HTTP/1.
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:19] "POST /command HTTP/1.
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:19] "POST /command HTTP/1.
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:19] "POST /command HTTP/1.
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:19] "POST /command HTTP/1.
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:19] "POST /command HTTP/1.
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:20] "POST /command HTTP/1.
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:20] "POST /command HTTP/1.
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:20] "POST /command HTTP/1.
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:20] "POST /command HTTP/1.
1" 403 -
192.168.56.102 - - [20/Apr/2025 18:38:20] "POST /command HTTP/1.
```

I will add an unused IP address to the truck controller list of trusted drone IPs. This IP will simulate a ‘misbehaving’ drone that is not authenticating properly.



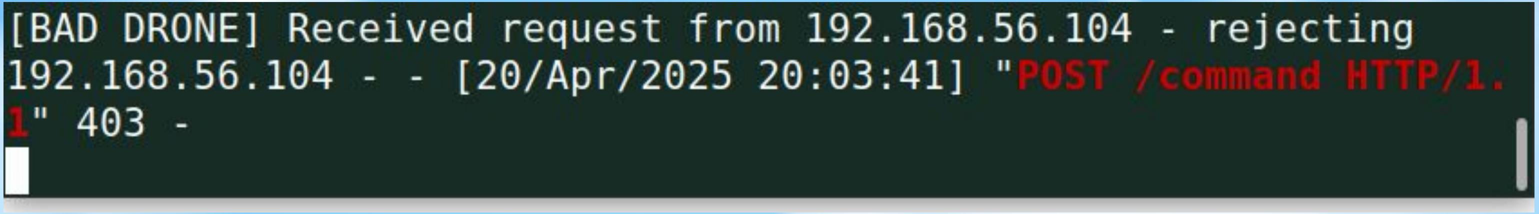
```
DRONE_IPS = ["192.168.56.105",
"192.168.56.123"]

# Rate limiting settings
MAX_ATTEMPTS = 5          # Max failed attempts
BLOCK_TIME = 300          # Block duration in
seconds (5 minutes)
```

Once running the truck_controller and it starts authenticating with the drone IPs, it will track and block for any failed attempts.



```
[04/20/25]seed@VM:~/.../volumes$ python3 truck_controller.py
[RESPONSE] 192.168.56.105 returned 403
[RECORD] Called for IP: 192.168.56.105
[RECORD] Count for 192.168.56.105: 1
[DEBUG] Failed Attempts State: {'192.168.56.105': {'count': 1, 'blocked_until': 0}}
[RESPONSE] 192.168.56.105 returned 403
[RECORD] Called for IP: 192.168.56.105
[RECORD] Count for 192.168.56.105: 2
[DEBUG] Failed Attempts State: {'192.168.56.105': {'count': 2, 'blocked_until': 0}}
[RESPONSE] 192.168.56.105 returned 403
[RECORD] Called for IP: 192.168.56.105
[RECORD] Count for 192.168.56.105: 3
[RECORD] 192.168.56.105 is now blocked!
[DEBUG] Failed Attempts State: {'192.168.56.105': {'count': 3, 'blocked_until': 174519385
1.2814012}}
[BLOCKED] 192.168.56.105 blocked until Sun Apr 20 20:04:11 2025
[DEBUG] Failed Attempts State: {'192.168.56.105': {'count': 3, 'blocked_until': 174519385
1.2814012}}
```

Ran a “bad drone” script to try to flood the truck controller. The attempts were tracked and blocked after so many tries.



```
[BAD DRONE] Received request from 192.168.56.104 - rejecting
192.168.56.104 - - [20/Apr/2025 20:03:41] "POST /command HTTP/1.
1" 403 -
```

Rate Limiting & Flooding Prevention

Dynamic drone IP’s, static tokens

Truck

- Manually give truck drone(s) starting IP’s
- Store drone IP’s with their respective token (stored in a JSON file).
- When drone sends an authentication request with truck, if the token is correct but the IP is different, it will authenticate successfully and update the drone’s IP.
- If the IP is the same and token is different, it will fail authentication.

Drone

- Each drone will generate their own secure token (stored in a JSON file).
- Drone sends IP and token at run time to truck to store.
- IP changes within a given timeframe from MAC rotation.
- Authentication with truck will run and new IP will replace old IP if token is correct.

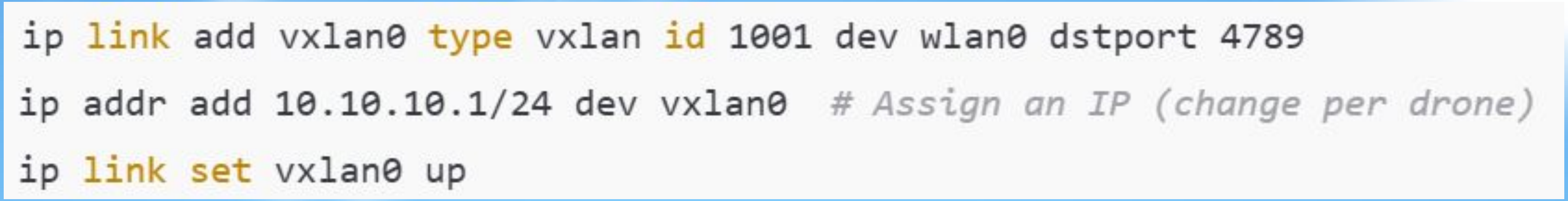
Due to time constraints and team members being at different checkpoints in the project, I was unable to test the dynamic components of authentication and flooding. I was able to get the scripts written but they have not been tested.

Secure P2P Networking

For secure P2P networking, the drones will use VXLAN over IPSec

- Ensures unencrypted traffic does not enter or leave the tunnel

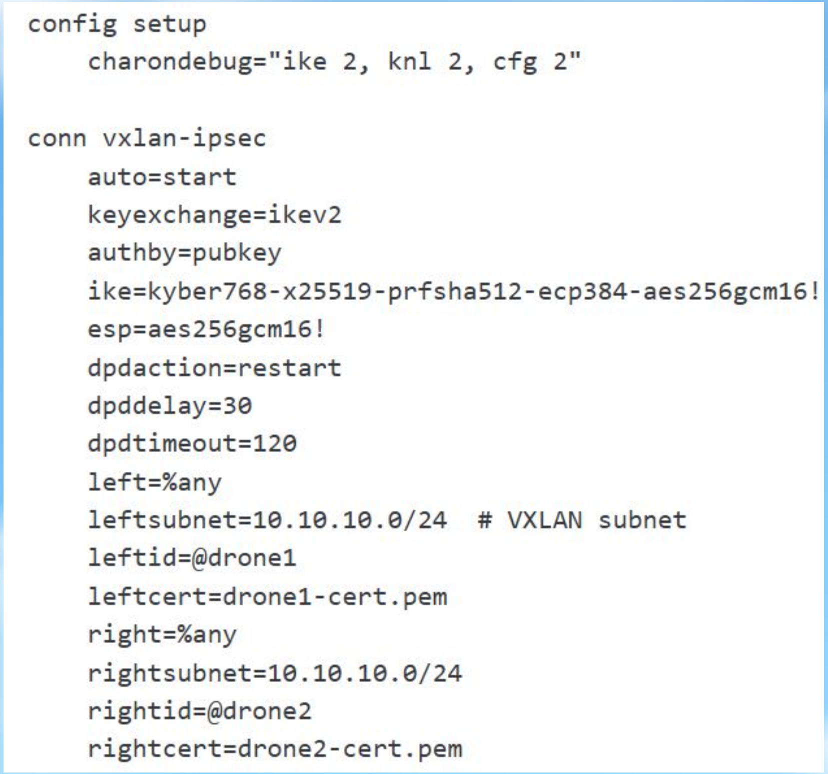
Setting up VXLAN tunnel:



```
ip link add vxlan0 type vxlan id 1001 dev wlan0 dstport 4789
ip addr add 10.10.10.1/24 dev vxlan0  # Assign an IP (change per drone)
ip link set vxlan0 up
```

strongSwan will be used to configure the IPSec connection

IPSec Configuration



```
config setup
    charondebug="ike 2, knl 2, cfg 2"

conn vxlan-ipsec
    auto=start
    keyexchange=ikev2
    authby=pubkey
    ike=kyber768-x25519-prfsha512-ecp384-aes256gcm16!
    esp=aes256gcm16!
    dpdaction=restart
    dpddelay=30
    dpdtimeout=120
    left=%any
    leftsubnet=10.10.10.0/24  # VXLAN subnet
    leftid=@drone1
    leftcert=drone1-cert.pem
    right=%any
    rightsubnet=10.10.10.0/24
    rightid=@drone2
    rightcert=drone2-cert.pem
```

- IKEv2 is used for a secure exchange of keys
- Uses certificates
- Kyber768 + X25519 key exchange
- AES-GCM-256 cipher
- Penetration testing will be used to make sure the VXLAN tunnel is indeed secure
    - Sniffing attack- attacker should only receive unreadable esp ciphertext
    - Man in the middle spoofing attack- drones should drop any and all packets that are not encrypted and authenticated

WPA3 Authentication

- WPA3 (WiFi Protected Access) authentication is a security standard from the Wi-Fi Alliance
- This system in particular uses a RADIUS server (Remote Authentication Dial-In User Service) without any central authentication point
  - W/O a central authentication point the server would instead rely on a local authentication database or other methods if needed.
- The system simulates a WPA3-Enterprise using TLS+X.509 between the drone and control server
- Authentication is locally enforced with Zero Trust principles, instead of a traditional RADIUS server
- Each drone would be verified by certificate signatures, expiration, issuer, and the assigned role

Firewall Implementation

- Created firewall settings for the drone and truck so that a truck can only accept a connection from an IP address of a valid drone, and only to a specific port. The drone’s firewall is also set to only accept traffic from the IP address of the truck, and only from a verified port.

- Was not able to implement the zero-policy encryption, due to not finding good materials teaching how to implement

Demonstration

Demonstration 1 Synopsis

The first demonstration will present our drone project performing authentication flood prevention and secure tunneling in real time.

- The truck-drone server will prevent a attacker server from accessing the truck-drone server in this demonstration.

Shown on Laptop

Demonstration 2 Synopsis

The second demonstration will present our drone project performing WPA3 authentication in real-time.

- In the first runthrough of this code, the drone with proper authentication would be able to be certified and connect to the server.
- In the second runthrough, the drone trying to connect to the server would have authentication certifications that do not match the ones required by the server. The drone will not be able to connect to the server

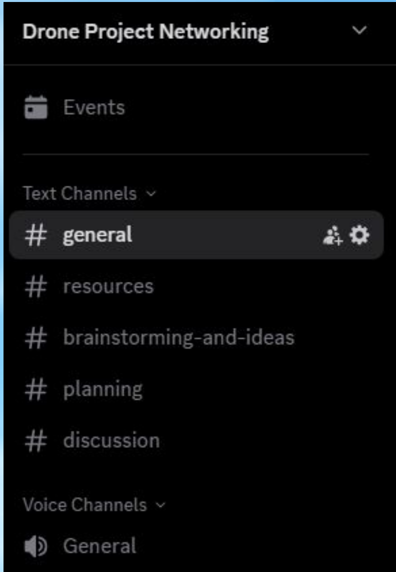
Shown on Laptop

Reflection

Lessons Learned	• Experience in interacting and programming with Debian-based Linux distributions • Handling security across different security measures and systems
Areas for Improvement	• Interfacing with actual drones and routers for delivery trucks • Penetration testing • Mass deployment / Automation

Team Collaboration

- Interactions over Discord
  - Mentor created the server
  - Project plan development
- Team meetup at campus library
  - Testing code across different machines with oversight
  - Final presentation creation
  - Resolving solution conflicts



Drone Project Networking

Events

Text Channels

general

resources

brainstorming-and-ideas

planning

discussion

Voice Channels

General