National College of Ireland

Configuration Manual

MSc Research Project
MSc Cybersecurity

Elijah Dyeris
X23367415

School of Computing
National College of Ireland

Supervisor: Eugene Mclaughlin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Elijah Dyeris
Student ID:	X23367415
Programme:	MSc Cybersecurity **Year:** 2024
Module:	Practicum 2
Lecturer:	Eugene Mclaughlin
Submission Due Date:	11th August 2025
Project Title:	A Comparative Analysis of Traditional Rogue AP and Multi-Channel CSA-Spoofing Establishment Against WPA3 Protections In a Simulated Environment.
Word Count:	**Page Count:** ……..……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:

Date: 11th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Elijah Dyeris
X23367415
https://deepnote.com/workspace/project-0e93-7fb8f1e1-8326-4a27-990a-39af59cda096/project/0a574585-db8a-4b28-b5fe-0b1a8585eb55/Practicum%202/main.py

MitM WPA3 Simulation – Comprehensive Configuration & Reproduction Manual

Introduction

This manual provides step-by-step instructions for replicating the research study: "Evaluating the Technical Feasibility of Establishing Man-in-the-Middle Positions in Smart Home IoT Environments: A Comparative Analysis of Traditional Rogue AP and Multi-Channel CSA-Spoofing Establishment Against WPA3 Protections In a Simulated Environment."

What This Research Investigates

Modern smart homes are filled with IoT devices like cameras, smart plugs, thermostats, and voice assistants—all connected via Wi-Fi. While WPA3 security promises better protection than its predecessor WPA2, sophisticated attackers are developing new techniques that can potentially bypass these protections. This simulation evaluates two critical attack vectors:

- Traditional Rogue Access Point (Rogue AP) Attacks: Where attackers set up fake Wi-Fi hotspots to trick devices into connecting(Owusu Agyemang et al., 2019).
- Multi-Channel CSA-Spoofing Attacks: A more advanced technique that manipulates Channel Switch Announcement beacons to hijack device communications(Louca et al., 2021).

What I Intend to Achieve

By following this manual and running the simulation, you will:

Generate Quantitative Evidence showing that MC-MitM CSA-Spoofing achieves success rates of ~44% against WPA3 networks, while traditional Rogue AP attacks achieve ~25% success rates.

Demonstrate PMF Limitations by proving that Protected Management Frames (PMF)—a key WPA3 security feature—completely neutralizes traditional Rogue AP attacks (0% success when enabled) but provides minimal protection against CSA-Spoofing attacks.

Reveal Device Vulnerability Patterns across 50 heterogeneous IoT device profiles, showing how patch levels, device types, and security configurations impact susceptibility to MitM attacks.

Expose Critical Security Gaps in WPA3-Transition networks where legacy device support creates persistent vulnerabilities even in "secure" environments.

Provide Actionable Policy Recommendations including mandatory PMF enforcement, firmware update SLAs, network segmentation strategies, and WPA3-Transition mode sunset timelines.

2. System Requirements.

Requirement	Recommended Version	Notes
Operating System	Linux-based OS	Any modern Linux environment that supports Python 3.9
Python	3.9 – 3.11	Verify with Python3 version.
Disk Space	500 MB	CSV logs & PNG plots are 250 MB per million trials.
RAM	4 GB	The simulator streams to disk; memory spikes are modest.

2 Project Structure (Quick Reference)
Project component scripts

Attacks.py	Core attack logic
Device.py	IoT Device class & vulnerability helpers
Simulator.py	Simulator class (device gen + attack loop)
Visualize.py	Plotting & data-extraction utility
Main.py	Orchestrator – run simulation + plots
Requirements.txt	Python dependencies
Visualizations	Auto-generated PNG charts
Simulation log	Auto-generated raw results

3. Installation

1. Clone or download the project folder onto your machine.
2. (Strongly recommended) Create and activate a virtual environment:
   ‘‘’’’
   python3 -m venv venv
   source venv/bin/activate
   ‘‘’’’
3. Install dependencies:
   pip install -r requirements.txt

requirements.txt

```
numpy
pandas
matplotlib
scipy
seaborn
```

Fig1. The list currently includes pandas, numpy, matplotlib, seaborn, and scipy.

4 Running the Simulation

Execution

To execute the pipeline, run “python main.py”

```
(venv) root@deepnote:~/work # python main.py
--- Welcome to the MitM WPA3 Smart Home Network Attack Simulator ---

Phase 1: Starting MitM Attack Simulation...
  Parameters: Simulating 50 devices, with 5000 trials each per attack.
Simulator ready! Will generate 50 devices and run 5000 trials each.
Results will be saved to: simulation_log_20250810_125656.csv
No device profiles generated yet. Generating them now...
Generating 50 device profiles...
Successfully generated 50 unique device profiles. Let's see what we've got!

Buckle up! Starting the simulation...

--- SIMULATING FOR NETWORK MODE: WPA3-only ---
  Completed 500000 trials for WPA3-only mode.

--- SIMULATING FOR NETWORK MODE: WPA3-transition ---
  Completed 500000 trials for WPA3-transition mode.

--- Simulation Complete! ---
Total trials run across all modes: 1000000
Attempting to log 1000000 results to simulation_log_20250810_125656.csv...
Success! Simulation results have been logged to 'simulation_log_20250810_125656.csv'
  Simulation complete! All results have been logged to: 'simulation_log_20250810_125656.csv'
```

Fig 2.

Outcome:

- Creates simulation_log_.csv in the project root (1,000,000 rows).

simulation_log_20250810_125656.csv — Open as text

```
trial_id,network_mode,device_id,device_type,pmf_enabled,patch_level,accepts_csa_spoofing_config,attack_type,success
WPA3-only_rogue_t0000_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0000_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0001_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0001_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,False
WPA3-only_rogue_t0002_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0002_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0003_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0003_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0004_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0004_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0005_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0005_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0006_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0006_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0007_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0007_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0008_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0008_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0009_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0009_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0010_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0010_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,False
WPA3-only_rogue_t0011_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0011_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0012_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0012_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0013_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0013_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0014_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0014_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0015_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0015_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0016_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0016_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
WPA3-only_rogue_t0017_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,Rogue AP,False
WPA3-only_csa_t0017_diot_dev_000,WPA3-only,iot_dev_000,printer,True,1,True,CSA Spoofing,True
```

Fig3. - Simulation Log.

4.1 Results

```
*   For the **Traditional Rogue AP** attack:
    *   When PMF was **enabled** on the target device, the attack success rate plummeted to **0.00%**. This unequivocally confirms PMF's effec
    *   Conversely, when PMF was **disabled**, the Rogue AP attack succeeded in approximately **54.82%** of trials, highlighting the vulnerabi

*   For the **MC-MitM CSA-Spoofing** attack:
    *   When PMF was **enabled**, the attack success rate was approximately **40.94%**.
    *   When PMF was **disabled**, the success rate was approximately **48.05%**.
```

Fig4. Impact of PMF (Protected Management Frames) on the success rates of two wireless network attacks: Traditional Rogue AP and MC-MitM CSA-Spoofing.

```
*   **CSA-Spoofing Specifics:** The impact of patching was particularly pronounced for the CSA-Spoofing attack:
    *   Patch Level 1: **78.43%** success rate.
    *   Patch Level 2: **65.07%** success rate.
    *   Patch Level 3: **21.26%** success rate.
    *   Patch Level 4: **0.00%** success rate (in these trials).
    *   Patch Level 5: **0.00%** success rate (in these trials).
    This demonstrates a significant reduction in vulnerability to CSA-Spoofing as patch levels increase. The 0% success at levels 4 and 5
    *(Refer to Figure 4: `csa_vuln_by_patch_level.png` for a focused view of CSA-Spoofing vulnerability across patch levels.)*
```

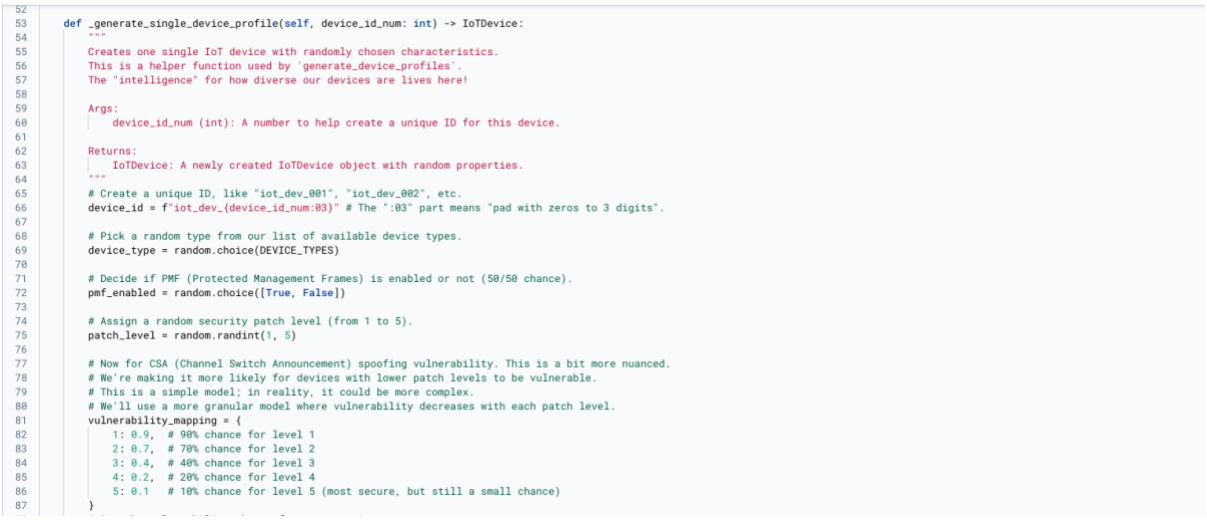
Fig 5. Results after Patch

5. Breakdown of functions

```python

# --- Configuration Constants ---
# These are like settings for our simulation. We can tweak them here.
NUM_DEVICE_PROFILES = 50 # How many unique types of devices do we want to test?
NUM_TRIALS_PER_DEVICE_ATTACK = 5000 # How many times should we try each attack on each device?
DEVICE_TYPES = ["camera", "plug", "bulb", "thermostat", "lock", "speaker", "sensor", "hub", "fridge", "tv", "printer"] # What kinds of IoT devices can we have?
LOG_FILE_TIMESTAMP_FORMAT = "%Y%m%d_%H%M%S" # How we format the date and time in our log file's name. e.g., "20240115_143055"

class Simulator:
    """
    This class is the main manager for our MitM attack simulation.
    It handles creating the devices, running all the attack trials,
    and then saving all the juicy data.
    """
    def __init__(self, num_devices: int = NUM_DEVICE_PROFILES, num_trials: int = NUM_TRIALS_PER_DEVICE_ATTACK):
        """
        Setting up our Simulator! This is called when we first create a Simulator object.

        Args:
            num_devices (int): The number of unique device profiles to generate.
                               Defaults to NUM_DEVICE_PROFILES if not specified.
            num_trials (int): The number of attack trials to run per device, per attack type.
                              Defaults to NUM_TRIALS_PER_DEVICE_ATTACK if not specified.
        """
        self.num_devices = num_devices # Store how many devices we're dealing with.
        self.num_trials = num_trials   # And how many trials for each.
        self.device_profiles = []      # This list will hold all our generated IoTDevice objects. Starts empty.
        self.results = []              # This list will store the outcome of every single attack trial. Starts empty.

        # Let's create a unique filename for our log file using the current date and time.
        # This way, if we run the simulation multiple times, we don't overwrite old results.
        current_time_str = datetime.datetime.now().strftime(LOG_FILE_TIMESTAMP_FORMAT)
        self.log_filename = f"simulation_log_{current_time_str}.csv"

        print(f"Simulator ready! Will generate {self.num_devices} devices and run {self.num_trials} trials each.")
        print(f"Results will be saved to: {self.log_filename}")

```

Fig6.

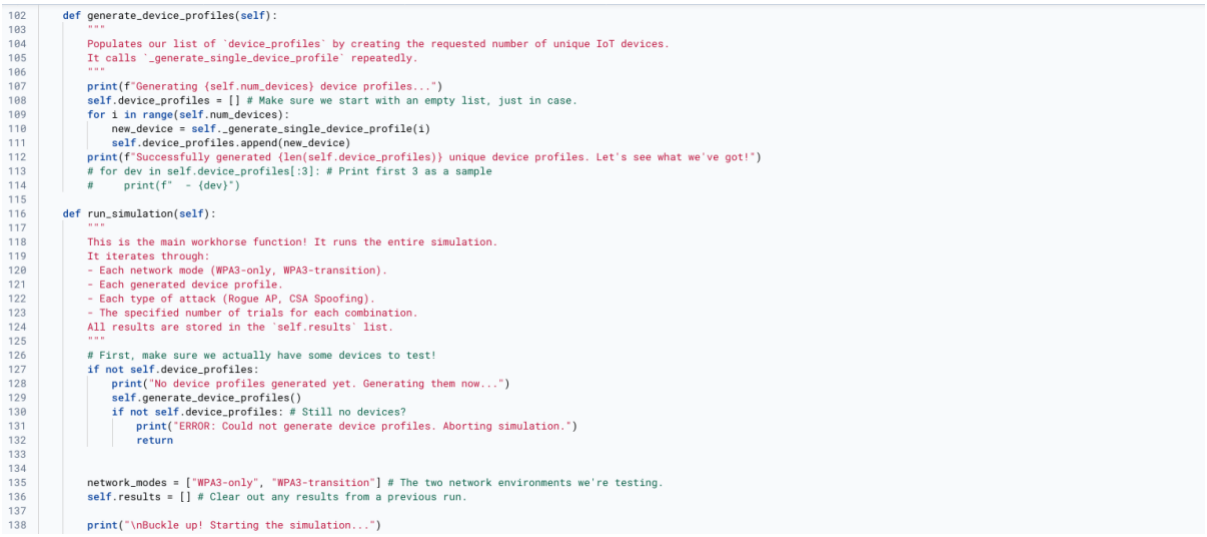
Here is the set up foundation for simulating Man-in-the-Middle (MitM) attacks on various IoT devices, It begins by defining configuration constants such as the number of device profiles, the number of attack trials per device.



```
    def _generate_single_device_profile(self, device_id_num: int) -> IoTDevice:
        """
        Creates one single IoT device with randomly chosen characteristics.
        This is a helper function used by `generate_device_profiles`.
        The "intelligence" for how diverse our devices are lives here!

        Args:
            device_id_num (int): A number to help create a unique ID for this device.

        Returns:
            IoTDevice: A newly created IoTDevice object with random properties.
        """
        # Create a unique ID, like "iot_dev_001", "iot_dev_002", etc.
        device_id = f"iot_dev_{device_id_num:03}" # The ":03" part means "pad with zeros to 3 digits".

        # Pick a random type from our list of available device types.
        device_type = random.choice(DEVICE_TYPES)

        # Decide if PMF (Protected Management Frames) is enabled or not (50/50 chance).
        pmf_enabled = random.choice([True, False])

        # Assign a random security patch level (from 1 to 5).
        patch_level = random.randint(1, 5)

        # Now for CSA (Channel Switch Announcement) spoofing vulnerability. This is a bit more nuanced.
        # We're making it more likely for devices with lower patch levels to be vulnerable.
        # This is a simple model; in reality, it could be more complex.
        # We'll use a more granular model where vulnerability decreases with each patch level.
        vulnerability_mapping = {
            1: 0.9,  # 90% chance for level 1
            2: 0.7,  # 70% chance for level 2
            3: 0.4,  # 40% chance for level 3
            4: 0.2,  # 20% chance for level 4
            5: 0.1   # 10% chance for level 5 (most secure, but still a small chance)
        }
```

Fig7.

Here, we created a single IoT device with randomized attributes. It takes a device ID number as input and generates a unique identifier. The function then randomly selects a device type from a predefined list and randomly assigns whether PMF (Protected Management Frames) is enabled, simulating variability in device security features.



```
    def generate_device_profiles(self):
        """
        Populates our list of `device_profiles` by creating the requested number of unique IoT devices.
        It calls `_generate_single_device_profile` repeatedly.
        """
        print(f"Generating {self.num_devices} device profiles...")
        self.device_profiles = [] # Make sure we start with an empty list, just in case.
        for i in range(self.num_devices):
            new_device = self._generate_single_device_profile(i)
            self.device_profiles.append(new_device)
        print(f"Successfully generated {len(self.device_profiles)} unique device profiles. Let's see what we've got!")
        # for dev in self.device_profiles[:3]: # Print first 3 as a sample
        #     print(f"  - {dev}")

    def run_simulation(self):
        """
        This is the main workhorse function! It runs the entire simulation.
        It iterates through:
        - Each network mode (WPA3-only, WPA3-transition).
        - Each generated device profile.
        - Each type of attack (Rogue AP, CSA Spoofing).
        - The specified number of trials for each combination.
        All results are stored in the `self.results` list.
        """
        # First, make sure we actually have some devices to test!
        if not self.device_profiles:
            print("No device profiles generated yet. Generating them now...")
            self.generate_device_profiles()
            if not self.device_profiles: # Still no devices?
                print("ERROR: Could not generate device profiles. Aborting simulation.")
                return


        network_modes = ["WPA3-only", "WPA3-transition"] # The two network environments we're testing.
        self.results = [] # Clear out any results from a previous run.

        print("\nBuckle up! Starting the simulation...")
```

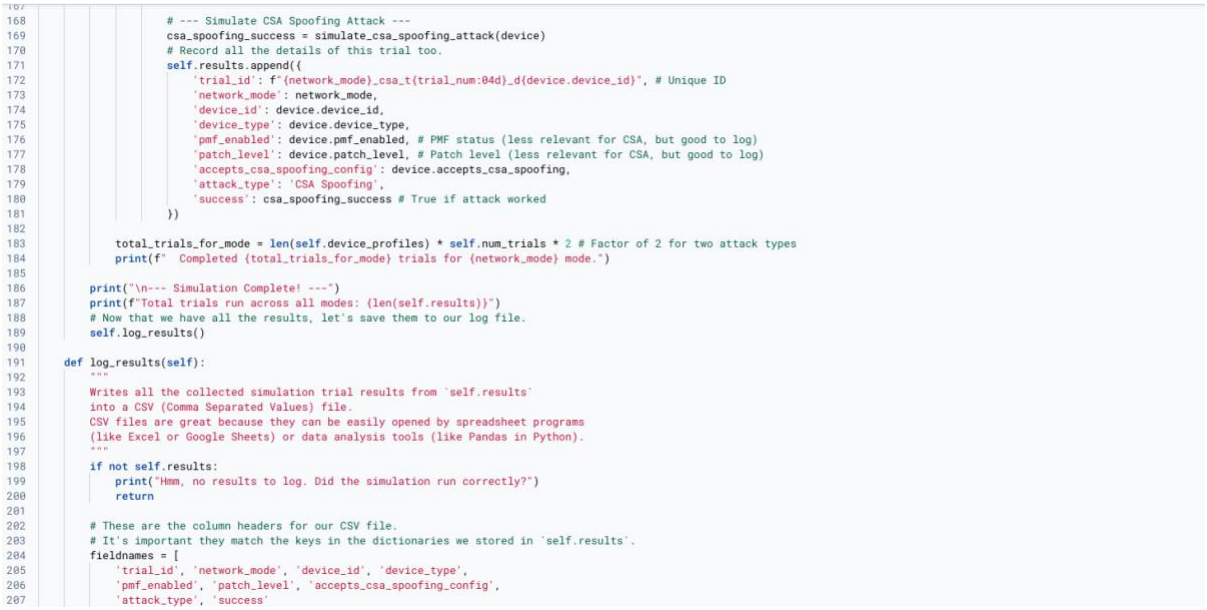
Fig8.

In this Figure it shows creating a collection of unique IoT device profiles within the simulation. For each iteration, it calls the helper function generate_single_device_profile, passing the current index to generate a uniquely identified device. Each new device is added to the device_profiles list. The function also prints progress messages to provide feedback during execution, helping users track the simulation setup.

```
    for network_mode in network_modes:
        print(f"\n--- SIMULATING FOR NETWORK MODE: {network_mode} ---")
        # A note on network modes:
        # Currently, our device generation (`_generate_single_device_profile`) doesn't change
        # based on `network_mode`. So, the same set of devices is used for both modes.
        # The main difference is that we *log* which mode the trial was run under.
        # In a more advanced simulation, "WPA3-transition" might mean devices are, on average,
        # less secure (e.g., more likely to have PMF disabled or accept CSA spoofing).
        # This is a potential area for future refinement!

        for device_index, device in enumerate(self.device_profiles):
            # print(f"  Testing Device {device_index + 1}/{len(self.device_profiles)}: {device.device_id} ({device.device_type})")
            for trial_num in range(self.num_trials):
                # --- Simulate Rogue AP Attack ---
                rogue_ap_success = simulate_rogue_ap_attack(device)
                # Record all the details of this trial.
                self.results.append({
                    'trial_id': f"{network_mode}_rogue_t{trial_num:04d}_d{device.device_id}", # Unique ID for this trial
                    'network_mode': network_mode,
                    'device_id': device.device_id,
                    'device_type': device.device_type,
                    'pmf_enabled': device.pmf_enabled,
                    'patch_level': device.patch_level,
                    'accepts_csa_spoofing_config': device.accepts_csa_spoofing, # Device's inherent vulnerability to CSA
                    'attack_type': 'Rogue AP',
                    'success': rogue_ap_success # True if attack worked, False otherwise
                })

                # --- Simulate CSA Spoofing Attack ---
                csa_spoofing_success = simulate_csa_spoofing_attack(device)
                # Record all the details of this trial too.
                self.results.append({
                    'trial_id': f"{network_mode}_csa_t{trial_num:04d}_d{device.device_id}", # Unique ID
                    'network_mode': network_mode,
                    'device_id': device.device_id,
                    'device_type': device.device_type,
                    'pmf_enabled': device.pmf_enabled, # PMF status (less relevant for CSA, but good to log)
```

Fig9.

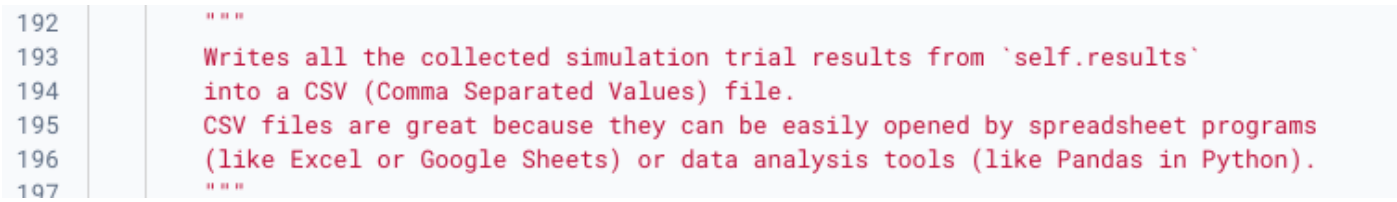
This figure shows the core of the attack simulation by testing IoT devices under two network environments: "WPA3-only" and "WPA3-transition". For each mode, it loops through all generated device profiles and runs a series of attack trials using the simulate_rogue_ap_attack function. This setup enables systematic testing of device vulnerabilities across different network configurations.



```
                # --- Simulate CSA Spoofing Attack ---
                csa_spoofing_success = simulate_csa_spoofing_attack(device)
                # Record all the details of this trial too.
                self.results.append({
                    'trial_id': f"{network_mode}_csa_t{trial_num:04d}_d{device.device_id}", # Unique ID
                    'network_mode': network_mode,
                    'device_id': device.device_id,
                    'device_type': device.device_type,
                    'pmf_enabled': device.pmf_enabled, # PMF status (less relevant for CSA, but good to log)
                    'patch_level': device.patch_level, # Patch level (less relevant for CSA, but good to log)
                    'accepts_csa_spoofing_config': device.accepts_csa_spoofing,
                    'attack_type': 'CSA Spoofing',
                    'success': csa_spoofing_success # True if attack worked
                })

        total_trials_for_mode = len(self.device_profiles) * self.num_trials * 2 # Factor of 2 for two attack types
        print(f"  Completed {total_trials_for_mode} trials for {network_mode} mode.")

    print("\n--- Simulation Complete! ---")
    print(f"Total trials run across all modes: {len(self.results)}")
    # Now that we have all the results, let's save them to our log file.
    self.log_results()

def log_results(self):
    """
    Writes all the collected simulation trial results from `self.results`
    into a CSV (Comma Separated Values) file.
    CSV files are great because they can be easily opened by spreadsheet programs
    (like Excel or Google Sheets) or data analysis tools (like Pandas in Python).
    """
    if not self.results:
        print("Hmm, no results to log. Did the simulation run correctly?")
        return

    # These are the column headers for our CSV file.
    # It's important they match the keys in the dictionaries we stored in `self.results`.
    fieldnames = [
        'trial_id', 'network_mode', 'device_id', 'device_type',
        'pmf_enabled', 'patch_level', 'accepts_csa_spoofing_config',
        'attack_type', 'success'
```

Fig10

Here is the second layer to the attack simulation by testing each device against a CSA (Channel Switch Announcement) spoofing attack. Each trial is uniquely identified and tagged with the current network mode. After all trials for a given mode are completed, the code calculates the total number of trials (accounting for both Rogue AP and CSA spoofing attacks) and prints a summary message, helping track simulation progress and completeness.



```
    """
    Writes all the collected simulation trial results from `self.results`
    into a CSV (Comma Separated Values) file.
    CSV files are great because they can be easily opened by spreadsheet programs
    (like Excel or Google Sheets) or data analysis tools (like Pandas in Python).
    """
```

5 Behind-the-Scenes Configuration
main.py sets:
random.seed(42)
np.random.seed(42)

```python
# --- Reproducibility ---
# By setting a 'seed' for the random number generators, we ensure that every time
# we run this simulation, the sequence of "random" events (like device generation
# and attack success rolls) will be exactly the same. This is crucial for
# getting consistent, reproducible results for our paper.
random.seed(42)
np.random.seed(42)
```

Fig11. Fixed Random Seed

5.2 Device Generation Logic

- Location: generate_single_device_profile () in simulator.py.
- Key Constants:
- DEVICE_TYPES – list of possible device categories.
- CSA vulnerability mapping (patch → probability) is hard-coded; edit to explore other threat models.

```python
def _generate_single_device_profile(self, device_id_num: int) -> IoTDevice:
    """
    Creates one single IoT device with randomly chosen characteristics.
    This is a helper function used by `generate_device_profiles`.
    The "intelligence" for how diverse our devices are lives here!

    Args:
        device_id_num (int): A number to help create a unique ID for this device.

    Returns:
        IoTDevice: A newly created IoTDevice object with random properties.
    """
    # Create a unique ID, like "iot_dev_001", "iot_dev_002", etc.
    device_id = f"iot_dev_{device_id_num:03}" # The ":03" part means "pad with zeros to 3 digits".

    # Pick a random type from our list of available device types.
    device_type = random.choice(DEVICE_TYPES)

    # Decide if PMF (Protected Management Frames) is enabled or not (50/50 chance).
    pmf_enabled = random.choice([True, False])

    # Assign a random security patch level (from 1 to 5).
    patch_level = random.randint(1, 5)

    # Now for CSA (Channel Switch Announcement) spoofing vulnerability. This is a bit more nuanced.
    # We're making it more likely for devices with lower patch levels to be vulnerable.
    # This is a simple model; in reality, it could be more complex.
    # We'll use a more granular model where vulnerability decreases with each patch level.
    vulnerability_mapping = {
        1: 0.9,  # 90% chance for level 1
        2: 0.7,  # 70% chance for level 2
        3: 0.4,  # 40% chance for level 3
        4: 0.2,  # 20% chance for level 4
        5: 0.1   # 10% chance for level 5 (most secure, but still a small chance)
    }
    # Get the vulnerability chance from our mapping.
    vulnerability_chance = vulnerability_mapping.get(patch_level, 0.0)
```

Fig12. Output Files Explained

Artefact	Location	Description
simulation_log_<timestamp>.csv	Project root	One row per trial; columns include device_id, device_type, patch_level, pmf_enabled, network_mode, attack_type, success.
visualizations/*.png	visualizations	Six figures referenced in the Results section.

```
# --- Configuration Constants ---
# These are like settings for our simulation. We can tweak them here.
NUM_DEVICE_PROFILES = 50 # How many unique types of devices do we want to test?
NUM_TRIALS_PER_DEVICE_ATTACK = 5000 # How many times should we try each attack on each device?
DEVICE_TYPES = ["camera", "plug", "bulb", "thermostat", "lock", "speaker", "sensor", "hub", "fridge", "tv", "printer"] # What kinds of IoT devic
LOG_FILE_TIMESTAMP_FORMAT = "%Y%m%d_%H%M%S" # How we format the date and time in our log file's name. e.g., "20240115_143055"
```

Fig13. The timestamp pattern %Y%m%d_%H%M%S ensures logs stay unique across runs.

7 Troubleshooting & FAQ.

Symptom	Likely Cause	Fix
ModuleNotFoundError	Virtualenv is not activated, or pip install was skipped.	Activate venv and reinstall requirements.
Very slow execution	Huge --trials_per_device.	Lower trials or run on faster hardware.
CSV grows but PNGs not generated	Error inside visualize.py.	Run it directly to see traceback.
Different results each run	Modified/removed random seed lines.	Restore random.seed(42) and np.random.seed(42).

To Extend the Study

- Network Conditions: Integrate basic RF noise or packet-loss models to refine attack success probabilities.
- IDS Evaluation: Add hooks to simulate intrusion-detection alarms and measure detection success.
- Additional Device Profiles: Import real-world firmware data to drive patch-level distributions.

Refrences

Louca, C., Peratikou, A., Stavrou, S., 2021. On the detection of Channel Switch Announcement Attack in 802.11 networks, in: 2021 IEEE International Conference on Cyber Security and Resilience (CSR). Presented at the 2021 IEEE International Conference on Cyber Security and Resilience (CSR), pp. 281–285. https://doi.org/10.1109/CSR51186.2021.9527971

Owusu Agyemang, J., Jerry, K., Klogo, G., 2019. A Lightweight Rogue Access Point Detection Algorithm for Embedded Internet of Things (IoT) Devices. Information Security and Computer Fraud 7. https://doi.org/10.12691/iscf-7-1-2