Progress Report

Defensive Honeypots for IP IoT Devices: Quantitative Comparison between Vanilla and Sandboxed Honeypots

Franek Kruczynski

December 2025

Table of Contents

1. Introduction

1.1. Background

The IoT (Internet of Things) ecosystem is projected to exceed 27-billion devices by the end of 2025 (Jinesh, 2025). It has resulted in an increased attack surface exposure and accelerated malware sophistication. R et al. (2019) argues many consumer IP-based IoT devices are shipped with outdated firmware, inconsistent network hardening and have minimal patching strategies. As a result, adversaries leverage botnets, brute-force authentication attacks and automated propagation mechanisms (malware spread) to compromise IoT infrastructure at scale.

Traditional malware defences like VPNs, IDS/IPS systems and signature-based threat detection struggle mitigating polymorphic malware with complex obfuscation, and zero-day exploitation. Vanilla honeypots (controlled decoy systems) offer means to observe infection vectors, to analyse both payloads and post-compromise behaviour (Narendran, 2025). However, Kocaogullar et al. (2023) contends the degree of containment varies: low-interaction honeypots risk device takeover, whereas sandbox-enforced high-interaction honeypots may restrict full behavioural ability.

This motivates a controlled, practical quantitative comparison to evaluate the efficacy of sandboxed honeypots against vanilla environments and, determining its effect on malware propagation and behaviour.

1.2. Aims & Objectives

1.2.1 Aim

To evaluate how sandbox-containment affects malware propagation success within IoT honeypots, by quantitively comparing identical payload and sample execution across sandboxed and vanilla environments.

1.2.2. Objectives

- Investigating IoT-targeting malware behaviour and existing honeypot defence strategies.
- Design and deploy two honeypot environments: a vanilla model and a containerised, much more secure model.
- Execute various malicious samples in controlled-IoT environments, to observe payload behaviour, external propagation attempts and network traffic patterns.
- Collect and quantitively analyse network, system and process activity data.
- Evaluate whether sandboxing significantly improves containment, and security resilience.

1.3. Product Review

1.3.1. Scope

The scope of this project is focused around investigating malware behaviour, specifically targeting IP-IoT devices; evaluating whether sandboxed environments influence different malware behaviour, as opposed to vanilla honeypot environments. The primary focus is to analyse:

- Network activity,
- Execution patterns,
- Propagation attempts,
- Payload type.

This study aims to capture and quantify behavioural factors such as outbound network connections, filesystem modification events, attempts at both executing commands and lateral movement. Data will be collected via packet captures, custom logs, and system interface monitoring, permitting to a quantitative comparison between both environments. All sample executions will occur within an isolated virtual environment to eradicate compromise via propagation. Furthermore, it will ensure ethical compliance and allow safe reproducibility of results.

However, this study does not attempt to recreate real-world scale infrastructure or evaluate live malware performance. The work excludes testing on physical IoT hardware, large-scaled distributed honeypots, and long-term intelligence capture of botnet interactions. While results may inform and contain real-world defence techniques, the practical scope of this deployment remains confined to controlled lab execution for quantitative comparison, rather than a testing product deployment.

1.3.2. Audience

The primary audience concerns:

- Cybersecurity researchers investigating IoT malware behaviour, containment design methodologies, and mitigation strategies.
- Academic institutions requiring secure, reproducible frameworks for teaching malware analysis, ethical hacking and digital forensics.
- Network security analysts seeking empirical data on IoT threat landscapes, and comparative honeypot architectures.
- Graduate students and educators in cybersecurity programs, requiring practical implementations for malware research.
- IoT security practitioners, developing intrusion detection systems which require baseline data for threat modelling.

Secondary audiences are composed of:

- IoT manufacturers seeking data to understand attack vectors, implementing anti-propagation software within devices,

- Security Operation Centres (SOCs) evaluating malware mitigation strategies for threat intelligence.

2. Background Review

2.1. Existing Approaches

Research into IoT malware analysis is largely focused on static analysis, sandboxing samples and behavioural monitoring systems. Existing approaches demonstrate successful progress in malware classification however, limitations persist. Containment tends to be implemented poorly; scalability is not a feature in mind and, most tools lack realistic IP-IoT device emulation.

2.1.1. Static & Signature-based Analysis

Early work in malware classification commonly relied on static analysis via examination of program binaries, manual disassembly paired with decompiling; and signature-matching samples against known engines. Moser et al. (2008) introduced a Windows-based framework designed for containerised malware execution, capable of monitoring run-time traces and registry manipulation (similar to honeypots). However, such a system struggled with polymorphic analysis and realistic IoT-device emulation due to its limiting architecture and (as of present) outdated algorithms.

Moreover, Ngo et al. (2020) explored analysing IoT-targeting malware statically. Algorithms included Control Flow Graphs (CFGs) to map the flow of a binary, manual opcode analysis and binary string conversion to greyscale images, to be analysed via neural network models. Their machine-learning driven static models reported classification accuracy close to 99%, demonstrating the strength of static features in IoT-malware detection. Nevertheless, such techniques were significantly less effective against obfuscated, polymorphic malware and could not capture runtime-specific behaviours like propagation attempts, process procedures and resource usage.

Limitations identified by Moser et al. (2008) and Ngo et al. (2020) further justify the case of integrating dynamic malware analysis within honeypot environments for deeper behavioural learning.

2.1.2 Behavioural Analysis via Sandboxing

Dynamic analysis involves executing malware in containers, like honeypots and sandboxes, to observe runtime calls, payload execution and attempts in privilege escalation. Genç et al. (2019) observed malware behaviour within a Cuckoo sandbox, configured with twenty virtual machines all loaded with a custom BIOS mimicking IoT devices. They successfully mapped attacker methodologies. However, evasive malware families like *TeslaCrypt* delayed execution long enough to be unnoticed until post-payload deployment, demonstrating how intelligent malware can suppress activity when detecting virtual environments.

Furthermore, Singam et al. (2023) deployed malicious samples within an IoT-network contained in a multi-architecture, where data is processed via pyshark to obtain network activity like C&C (Command-and-Control) connectivity and DNS queries. Their findings deduced dynamic analysis alone fails in cases where an emulated environment corresponding to a sample is not supported. Therefore, a hybrid approach (static + dynamic + network analysis) is proposed to mitigate failover, which aids in handling obfuscated packets in cross-architectural samples.

Such confirms vanilla honeypots struggle to fully process malware and cannot reliably handle IP-IoT samples for cross-architectures; consequently, a specialised sandbox / hybrid system is superior.

2.2. Related Literature

Existing literature emphasises IoT malware significant differs from traditional PC malware due to:

- **System constraints**: IoT devices have significantly little memory, CPU cores and processing speeds,
- **Varying architectures**: General IoT networks may be IP-based; however, all devices differ in terms of architectural frameworks (i.e. ASM, ISO/IEC, i386),
- **Security Controls**: Manufacturers prioritise low costs and market delivery, rather than focusing on security policies.

Alasmary et al. (2019) proposed a graph-based analysis model for classification of obfuscated, polymorphic malware, through detecting unreachable dead-code segments and instruction-level entropy variance. While 97.9% accurate, the study was heavily limited to historical samples already analysed and publicly available and lacked real-time payload capture; a crucial requirements for IP-IoT systems which experience surges of botnets.

Genç (2019) further demonstrated malware supressing within virtual environments. This supports claim that sandboxing alone is not sufficient in providing defence within IP-IoT frameworks. Systems require deception-oriented strategies to extract meaningful behaviour, hence virtually replicating IoT devices is crucial for this project.

Recent work by Trajanovski & Zhang (2023) introduced the IoT-BDA framework, which integrated honeypots and custom sandbox environments capable of supporting varying architectures, hardware and software Over a seven-month deployment, 9306 unique botnet malware samples were collected, many of which exhibited anti-analysis and persistence mechanisms (Trajanovski & Zhang, 2023). The significance of the IoT-BDA lies in confirmation where malicious samples must be deployed on custom architectural frameworks, as opposed to generic x86 environments. Whilst findings illustrated most IoT botnet malware has anti-honeypot mechanisms (Trajanovski & Zhang, 2023) and runs obfuscated scripts with the aim to disrupt data collection; the authors note many existing sandboxes remain ineffective for IP-IoT analysis due to varying architectures, and anti-analysis techniques. This gap elucidates the importance of modernised, realistic and custom-configured sandbox-honeypot environments, precisely the aim of the project.

3. Technical Progress

3.1. Project Progress Update

With respect to the initial proposal, the project has underdone various chances crucial for processing data. Currently, the project has completed the initial implementation phase, with a testable prototype for data collection.

A Gantt chart was created to illustrate rough deliverables progress, with a full history on a public GitHub repository, and shared Google Drive folder (See Appendix A):

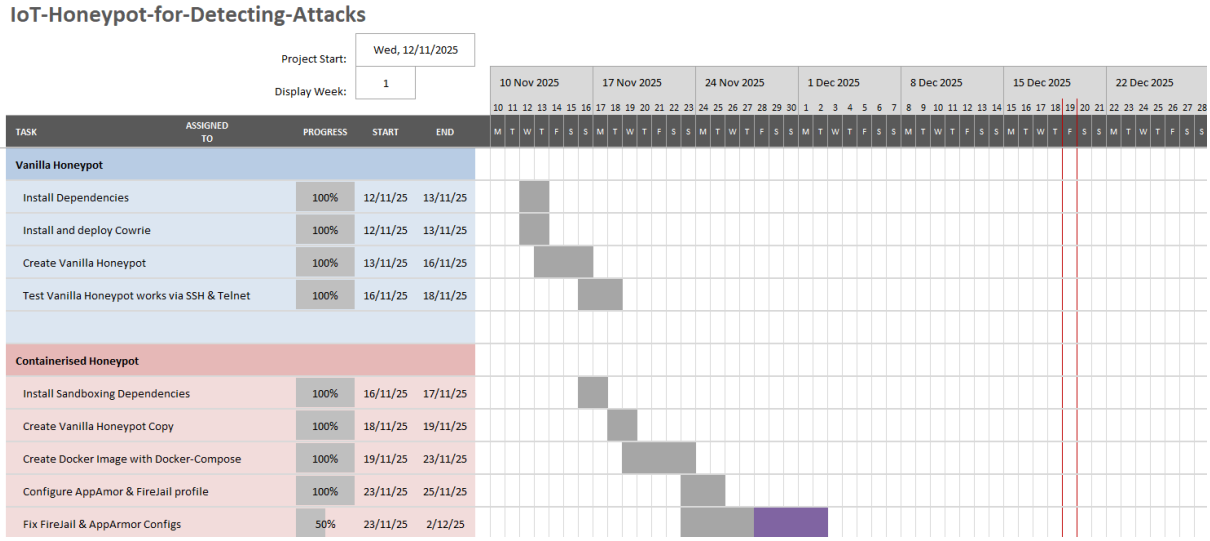

Figure 1: Gantt Chart

3.1.1 Completed Tasks

Progress includes:

- Visual environment established (Lubuntu 24.4.3 in Virtual Box), with all dependencies and tools installed,
- Vanilla Cowrie honeypot fully operational,
- Sandboxed honeypot architecture implemented, with multi-layer containment via: Docker and Docker-Compose; FireJail, AppArmor and Seccomp.

Current phase:

- Prototype testing, and a full configuration refinement to permit for various IoT device emulations.
    - Current implementation emulates each honeypot as a singular device, unlike a network composed of various IoT devices.
- Researching on visualisation techniques, to process sample data using Python. Such techniques include: Greyscale Imaging, Control Flow Graphs (CFGs) and, Entropy Imaging.
- Collecting a variety of malware samples, like Botnets and Rootkits.
- Researching IoT architectures, to develop a network mimicking a legitimate network.

Next Deliverables:

- Complete re-configuration of the system.
- Deploy various IoT architectures inside honeypots.

3.2. Requirements / Specification

3.2.1 Hardware & Virtual Environment

The host OS runs via a Oracle VirtualBox 7.2.x hypervisor. It was chosen due to:

- Being free of charge with cross-platform compatibility.
- It is open-source and is the most supported hypervisor on the market.
- Easy snapshot and backup restoration, with a simple host level kill switch.

The VM itself has the following configurations:

- OS: Lubuntu 24.0.3 LTS (lightweight, stable Linux distribution).
- Allocated 16GB of RAM with 4 CPU cores, for greater processing power.
- Storage: Isolated virtual disk with a host shared folder, removed when executing malicious samples.
- Network: Bridged adaptor with internal-only connectivity,
- Snapshots: Frequent snapshots at major implementation milestones.

The initial proposal specified Lubuntu would be the OS of choice but, Debian Bookworm/Trixie were evaluated as alternatives due to their resource efficiency. However, neither Debian distribution provided complete Cowrie compatibility (Python3 conflicts), resulting in Lubuntu 24.4.3 being chosen.

3.2.2 Network Architecture:

Containment policies:

- All honeypot traffic is restricted to only localhost (127.0.0.1).
- Future deployments will use internal-only virtual addressing schemes, with zero external routing,
- Kill-switch mechanisms at both host OS and kernel-level (nftables) to block all IP traffic if malware propagates outside its environment.
- Planned subnet isolation, separating honeypots from host OS utilising a demilitarised zone.

3.2.3 Security Techniques:

Vanilla Honeypot:

- Cowrie SSH/Telnet honeypot in default configuration,
- Minimal security modifications to baseline behaviour,
- Local-host only binding (SSH port 2222, Telnet port 2223)

Sandboxed Honeypot:

- Layer 1: Docker containerisation – Provides filesystem isolation, resource segmentation and a controller execution environment.
- Layer 2: FireJail Sandbox – Docker container itself runs resident in FireJail, acting like a tertiary boundary between the honeypot and host OS, with additional filesystem isolation and restricted network access.
- Layer 3: AppArmor – Kernel-level policy enforcement implements a least-privilege model, further securing filesystems, removing potentially exploitable system calls; and restricting kernel access.
- Layer 4: Seccomp filtering – System calls are all filtered through the Docker and FireJail profiles.

Automation & Management:

- Custom bash script (monitor.sh) providing a unified interface for all Docker and Docker-Compose operations: start, stop, build, restart, logs, status, shell access, reset, backup, export.
- docker-compose.yml for declarative imaged container orchestration,
- Various configuration files: vanilla-sandbox.cfg, sandboxed-honeypot's cowrie.aa (AppArmor profile for Cowrie), cowrie.profile (FireJail profile).

3.3. Test Plan

3.3.1 Validation Strategy

Testing will occur in three stages:

Phase 1 - Containment Validation:

- **Objective**: Verify all sandboxing layers to prevent propagation to host OS.
- **Method:** Execute malicious scripts attempting filesystem access outside containers.
- **Success Criteria:** All escape attempts are blocked, logged and contained.

Phase 2 - Behavioural Baseline:

- **Objective:** Establish baseline metrics for honeypots without malware.
- **Method:** Generate synthetic SSH/Telnet traffic, measure resource usage and verify logging mechanisms.
- **Metrics**: Verify response times, resource consumption and logs.

Phase 3 - Malware Execution:

- **Objective:** Execute identical malware samples in both environments under controlled conditions.
- **Method:** Sequential execution (vanilla first, sandboxed second) with VM snapshot rollback between runs.
- **Data Collection:** Packet captures (Wireshark), Cowrie logs, system-level activity.

3.3.2 Data Collection

Metrics to be captured:

- **Network activity:** Inbound/outbound connection attempts, with mapped IP addresses to ports; payload sizes and protocol types.
- **Execution patterns:** Command sequences, privilege escalation attempts and shell activity
- **Filesystem modifications:** Created/modified files, dropped payloads and persistence mechanisms.
- **Resource consumption:** CPU and memory usage, and process lifetimes
- **Propagation attempts:** Brute-force attempts, lateral movements and scanning behaviour.

3.4. Implementation progress

3.4.1 Prototype Phase

The current implementation represents a prototype validating all core functionalities and containment approaches (See Appendix A). Initial deployment configured each honeypot to emulate a specific IoT device and architecture however, review revealed this approach requires refinement for a comprehensive analysis. The prototype has successfully:

- Validated virtualisation environment and resource allocation stability.
- Confirmed Cowrie installation with basic SSH/Telnet connectivity.
- Tested a Docker image composed with FireJail, AppArmor and Seccomp.
- Developed automation tools and management interfaces.
- Identified technical challenges: OS compatibility and dependency conflicts.

Next objectives:

- Reconfigure all honeypots to emulate a honeynet of IP-IoT devices, inside a home network topology.
- Implement architecture-flexible configurations supporting ARM, MIPS, i386 and x86 samples; emulating the most used architectures.
- Establish subnet isolation mimicking SOHO (Small Office/Home Office) networks.

3.4.2 Vanilla Honeypot:

A baseline Cowie honeypot operates on the host OS with default configuration (vanilla-honeypot.cfg):

- **Services:** SSH and Telnet, with localhost binding and log storage.
- **Filesystem:** Default Debian-based virtual filesystem, emulating an IoT filesystem.
- **Session recording:** Captures and stores all executed commands.

3.4.3 Sandboxed Honeypot

The sandboxed honeypot uses an identical Cowrie configuration; however, it deploys a multi-layer architecture with all features illustrated as:

Feature	Purpose	Security Features
Docker	Cowrie honeypot executes inside a Docker container	• Filesystem isolation • Resource segmentation • Controlled execution environment
FireJail	Docker container itself is executed in a FireJail sandbox	• Provides a tertiary boundary between honeypot and host OS • Further filesystem isolation • Very limited network and process access
AppArmor (cowrie.aa profile)	Applied inside the FireJail sandbox at kernel level	• Mandatory access level to the kernel • Policy-based restrictions • Payload execution remains isolated from kernel access • Implements a least-privilege model
Docker-compose	Automates containerised sandbox	• .yml file defining Docker container for simplified control • Aids in snapshot restoration

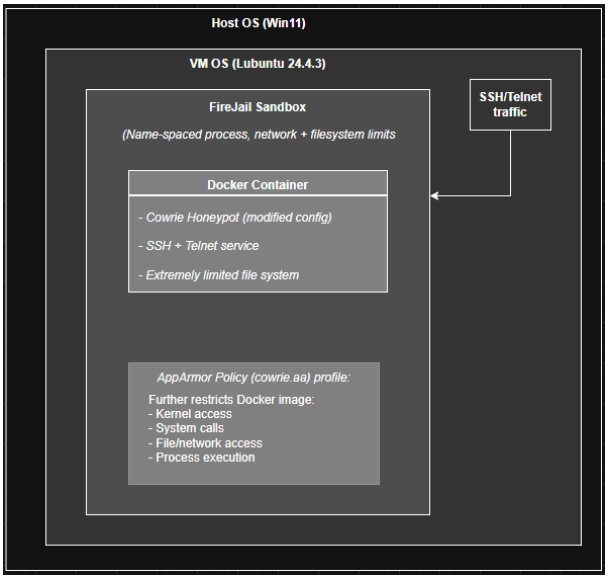

Figure 2: Current Containerised Sandbox Architecture

4. Professionalism & Risk

4.1. Risk

4.1.1 Technical Risks

Risk 1: Malware Containment Failure

- **Severity:** Critical with low likelihood.
- **Mitigation:** Multi-layer containment (Docker, FireJail, AppArmor and Seccomp). Host OS to VM shared folder to be removed prior malware execution. Kernel-level kill switches, frequent snapshot rollbacks and network isolation.
- **Status:** Mitigation implemented and tested.

Risk 2: Data Loss

- **Severity:** Medium with a medium likelihood.
- **Mitigation:** Automated backup scripts before sample execution, paired with snapshot rollbacks.
- **Status:** Backup procedures established.

Risk 3: Resource Exhaustion

- **Severity:** Medium with low likelihood.
- **Mitigation:** Docker and FireJail resource limits with real-time monitoring ensuring the 16GB of RAM and 4 CPU cores is not exhausted.
- **Status:** Resource limits configured.

Risk 4: Behavioural Suppression:

- **Severity:** Medium with a low likelihood.
- **Mitigation:** Dual-environment comparison detects suppression, with many data collection points. All samples are to be pilot-tested prior full execution.
- **Status:** Comparison methodology designed to identify such issue in real-time.

4.1.2 Project Timeline Risks

Risk 5: Implementation Delays

- **Severity:** Medium with medium likelihood.
- **Impact:** OS compatibility issues, causing an approximately two-week delay.
- **Mitigation:** Prototype-first approach; weekly supervisor meetings, and incremental development with frequent testing.
- **Status:** Fully mitigated, as demonstrated via the initial prototype.

Risk 6: Malware Sample Challenges

- **Severity:** Low with low likelihood.
- **Impact:** Insufficient sample diversity for a deep comparison.
- **Mitigation:** Utilising public malware repository (MalwareBazaar); focusing on well-documented IoT malware families and using historical samples sufficient for research objectives.
- **Status:** Samples have been identified.

4.2. Professionalism

4.2.1 Ethical Compliance

Handling Malware:

- **Principle:** All execution is in a completely isolated environment with zero external harm possibility.
- **Implementation:** No external connectivity and multi-layer containment.
- **Professional Standards:** Aligns with BCS Code of Conduct Principle 1 described by BCS, (n.d.) , and ACM Code of Edits 1.2 (ACM, n.d.).

4.2.2 Legal Considerations

Computer Misuse Act 1990:

- **Compliance**: All execution on personally owned and controlled systems.
- **No authorised access**: Isolated environment with no external connections.
- **Purpose:** Legitimate academic security research under supervision.

Data Protection (GDPR):

- **No personal data collection:** Only attacker activity is captured.
- **Anonymisation:** Attacker IPs remain anonymous in reports.
- Only necessary data is collected.

4.2.3 Social & Environmental Considerations

Social:

- Positive contribution to IoT security and malware research.
- Educational framework for future research.

Environmental:

- Virtual environment is arguable more energy and resource-efficient than a physical hardware implementation.
- Modest resource requirements minimise environment footprint.

Equality, Diversity & Inclusion:

- Open-source tools ensure research availability and accessibility.
- Universal applicability; IoT security affects users worldwide.

4.2.4 Professional Standards Alignment

BCS Code of Conduct:

- Code 1 – Public Interest: Improved cybersecurity for vulnerable IoT users and systems,
- Code 2 – Professional Competence: Appropriate supervision throughout development.
- Code 3 – Duty to Relevant Authority: University ethics approval.
- Code 4 – Duty to Progression: Maintains security research integrity.

ACM Code of Ethics:

- Code 1.1 - Contribute to Society: Advanced IoT security knowledge.
- Code 1.2 – Avoid Harm: Strict containment prevents malware escape and propagation.
- Code 1.3 – Be honest and trustworthy: Transparent methodology.

Bibliography

Jinesh (2025). How Many IoT Devices Are There in 2025? url: https://autobitslabs.com/how-many-iot-devices-are-there/

R, G. et al. (2019) An overview: Security issue in iot network | IEEE conference publication | IEEE Xplore, An Overview: Security Issue in IoT Network. Available at: https://ieeexplore.ieee.org/document/8653728/.

Narendran , V. (2025) Honeypots in Cybersecurity Explained, What is a honeypot in cybersecurity? Available at: https://www.crowdstrike.com/en-gb/cybersecurity-101/exposure-management/honeypots/.

Kocaogullar, Y. *et al.* (2023) *Hunting High or Low: Evaluating the Effectiveness of High-Interaction and Low-Interaction Honeypots*. Available at: https://kar.kent.ac.uk/102122/1/STAST2022.pdf.

Moser, A., Kruegel, C. and Kirda, E. (2008) *Limits of Static Analysis for Malware Detection*, *Limits of static analysis for malware detection | IEEE conference publication | IEEE Xplore*. Available at: https://ieeexplore.ieee.org/document/4413008/;jsessionid=KvYXh2re96Kfcyv3d_7Fd9b9zBLcX2vHQY8gy6FebsltepNFxdSl!-539104042.

Ngo, Q.-D. *et al.* (2020) *A survey of IoT malware and detection methods based on static features*. Available at: https://www.sciencedirect.com/science/article/pii/S2405844023026993.

Genç, Z.A., Lenzini, G. and Sgandurra, D. (2019) *Case Study: Analysis and Mitigation of a Novel Sandbox-Evasion Technique*. Available at: https://pure.royalholloway.ac.uk/ws/portalfiles/portal/34760373/cecc2019GLS.pdf.

Singam, S.B.S. *et al.* (2023) *A hybrid method for analysis and detection of malicious executables in IoT network*. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0167404823002493.

Alasmary, H. *et al.* (2019) *Analyzing, Comparing, and Detecting Emerging Malware: A Graph-based Approach*, *[1902.03955] Analyzing, Comparing, and Detecting Emerging Malware: A Graph-based Approach*. Available at: https://arxiv.org/abs/1902.03955.

Trajanovski, T. and Zhang, N. (2023) *An Automated and Comprehensive Framework for IoT Botnet Detection and Analysis (IoT-BDA)*. Available at: https://ieeexplore.ieee.org/document/9529169.

(No date) *BCS Code of Conduct*. Available at: https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/.

ACM Code of Ethics and Professional Conduct (no date) *Code of Ethics*. Available at: https://www.acm.org/code-of-ethics.

Appendix

Appendix A – Version Control:

The public GitHub repository storing both commits and the initial prototype: Link

Shared Google Drive folder with supervisor access, storing all relevant documentation: Link