# 🔨Thesis Draft: Integration of Hardware and Software in Secure Embedded Systems

## 🎓Abstract

This thesis explores the integration of hardware and software systems with a focus on secure command execution, verification, and logging in embedded environments. It builds upon the concept of `igniteAuth` — a Python-based secure execution framework — and proposes architectural bridges to interface with embedded C systems. The thesis guides future expansion into real-world IoT security, microcontroller interactions, and battlefield-level digital trust models.

---

## 1. 🧱Introduction

### 1.1 Motivation

Security in embedded systems is no longer optional. As microcontrollers become the core of critical infrastructure, there's an urgent need for authenticated, intent-driven control layers that operate above firmware.

### 1.2 The igniteAuth Foundation

The `igniteAuth` system establishes a Python-based gateway that validates admin users, maps commands to intents, and audits each action to a log file. It aims to simulate a command-authorization pipeline that will eventually be mirrored in real embedded environments.

---

## 2. 🔐Software Control Layer (igniteAuth)

### 2.1 Functional Modules

- **Admin Token System:** Bcrypt-hashed tokens stored in `.env`
- **Intent-Purpose-Command Mapping:** Enforces task specificity
- **Target Node Abstraction:** Simulated with Python function pointers
- **Audit Trail Logging:** Every command with timestamp + result status

### 2.2 Innovations

- Stateless but secure verification logic
- Modular architecture for embedding future expansions
- CLI simulation + potential web-based UI for monitoring

---

# 3. 🛠️Embedded Layer (C / Microcontroller Logic)

## 3.1 Role of Embedded Devices

- Execute the validated commands
- Respond to controller (Python) with execution results or status
- Protect integrity of execution pipeline

## 3.2 Firmware Limitations

- Cannot self-verify integrity without cryptographic attestation
- Must rely on trusted bootloaders or signed firmware

## 3.3 Node Response Design

- Each node replies to command packets with ACK/NACK
- Responses can include health telemetry or sensor status
- Future enhancement: Remote attestation hashes at boot

---

# 4. 💸Bridging Python and C

## 4.1 Levels of Integration

| Level | Bridge Mechanism | Description |
|---|---|---|
| 1 | Serial (UART) / I2C / SPI | Python sends encoded commands to C firmware |
| 2 | API or Protocol | Custom command protocol with session token, checksum |
| 3 | Shared Libraries (`ctypes`) | C functions compiled into shared libs and imported in Python |
| 4 | MicroPython / CircuitPython | Python runs directly on the MCU with limited C integration |

## 4.2 Sample Packet Structure (for command stream)

```
<COMMAND>:<INTENT>:<SESSION_TOKEN>:<HASH>
```

- Command: `reboot_firmware`
- Intent: `emergency`
- Token: from verified user
- Hash: SHA-256 of all above parts

## 4.3 Recommended Tools

- `pyserial` (UART)
- `RPi.GPIO` / `gpiozero`

- `protobuf` or `MessagePack` for efficient binary communication

---

## 5. 🔬 Zones of Innovation (Freedom Points)

To future-proof the system, reserve **hooks** in both software and firmware:

### Software Hooks (Python)

```python
# === Future Hook: Streamlit dashboard log viewer
def render_log_dashboard():
    pass
```

### Firmware Hooks (C)

```c
// Hook for telemetry or watchdog ping
void send_node_acknowledgment() {
    // Respond to igniteAuth with OK/ERR
}
```

These zones allow controlled evolution over time without breaking architecture.

---

## 6. Trust Models & Verification

### 6.1 Problem Statement

> "How can we trust that the node executed the command even after software verification?"

### 6.2 Solutions

- Firmware checksums at boot (remote attestation)
- Token replay protection (nonces / time-based)
- Hardware watchdog acknowledgment
- Event telemetry sent back from firmware

---

## 7. 📊 Conclusion

This thesis outlines a complete vision for secure, future-ready embedded system control. Starting from a Python CLI system (`igniteAuth`), it lays the roadmap for real-world deployment across embedded C environments. By enforcing layered security, intent-based execution, and traceable logs, the system sets the foundation for battlefield-grade command integrity.

Next stages (v0.3+) will enable live web interfaces, multiple admin management, and hardware-level command confirmations.

---

## 📖References


- OWASP Embedded Security Guide
- NIST Secure Boot Guidelines
- MicroPython Docs
- pyserial Docs
- RFC 4086 (Randomness Requirements for Security)