Final Report

COSC 6840 Ethical Hacking Theory
Final Project
John Freeborn
December 9, 2025

Table of Contents

- **1. Introduction** **3**
  - 1.1 Project Scope 3
  - 1.2 Assessed Attack Surfaces 3
  - 1.3 Assessment Objectives 3
  - 2.1 Core Tools 4
    - 2.1.1 Hardware Tools 4
    - 2.1.2 Software Tools 4
  - 2.2 Workflow 4
    - 2.2.1 Hardware Reconnaissance 4
    - 2.2.2 UART Shell Discovery and Access 5
    - 2.2.3 Internal Flash Extraction (MCU) 5
    - 2.2.4 External Flash Extraction (SPI Flash) 5
    - 2.2.5 Binary Analysis and Reverse Engineering 6
    - 2.2.6 Vulnerability Discovery and Verification 6
  - 2.3 Firmware Structural Identification 6
- **3. Findings** **6**
  - 3.1 UART Debug Shell Accessible Without Authentication 6
    - Description 6
    - Evidence 7
    - Impact 7
    - Flag 7
    - Recommendation 7
  - 3.2 Password Hash Hard-Coded in Firmware and Crackable 8
    - Description 8
    - Evidence 8
    - Impact 9
    - Flag 9
    - Recommendation 9
  - 3.3 Sensitive Data Recoverable from Internal Flash Memory 9
    - Description 9
    - Evidence 9
    - Impact 10
    - Flag 10
    - Recommendation 10
  - 3.4 External SPI Flash Contains Plaintext Operational Data 11
    - Description 11
    - Evidence 11
    - Impact 12

Flag 12
Recommendation 12
3.5 Secret Data Stream Transmitted Without Encryption 12
Description 12
Evidence 12
Impact 13
Flag 13
Recommendation 13
4. Flags Retrieved 13
5. Conclusion 14

1. Introduction

ACME Manufacturing, Inc. engaged our team to perform a security assessment of their new IoT device, “The Node.”ACME plans to deploy hundreds of these units across their manufacturing facility for operational data collection and analytics. Because this data is mission-critical and the devices will be physically accessible to short-term contract employees, ACME requested an evaluation of the firmware and hardware-level attack surface for weaknesses that could compromise confidentiality, integrity, or availability.

1.1 Project Scope

The scope of this engagement focuses strictly on firmware and hardware security assessment of The Node.

The following items were explicitly included:

- Identifying and accessing any debug interfaces (e.g., UART, SWD/JTAG).
- Enumerating available shell access and commands exposed by the microcontroller.
- Reviewing password handling and recovering any stored password hashes.
- Extracting and analyzing firmware from:
    - The internal flash of the microcontroller (STM32F103C8T6), and
    - The external SPI flash memory chip (Winbond W25Q64JV).
- Investigating the unencrypted “secret data stream” transmitted between nodes.

1.2 Assessed Attack Surfaces

Based on ACME’s concerns and initial OSINT performed by a prior team member, this assessment focused on:

1. UART-based debug shell exposed on *USART2* at 9600 baud, referenced in a mistakenly public GitHub repository.
2. Internal microcontroller flash, which may store configuration data, secrets, or proprietary code.
3. External SPI flash memory, which may include logs, key material, or data buffers.
4. Inter-device communication stream, which may expose sensitive telemetry if unencrypted.

These components represent meaningful risk due to physical access by untrusted personnel and the ease of attaching inexpensive debugging tools to exposed interfaces.

1.3 Assessment Objectives

The primary goals of this assessment were:

- Determine whether the debug shell can be accessed and whether privilege boundaries exist.
- Identify and retrieve any password hashes, evaluate their strength, and attempt to crack them.
- Dump and analyze internal and external flash memory to determine the presence of sensitive data and whether that data is stored securely (e.g., encrypted at rest).

- Locate and capture the device's secret data stream to evaluate confidentiality protections.

Evidence-driven findings, mitigation recommendations, and an automation script accompany this report.

2. Methodologies

2.1 Core Tools

2.1.1 Hardware Tools

- USB-to-UART adapter (3.3V logic) for accessing USART2.
- ST-Link V2 debugger for SWD-based internal flash extraction.
- Bus Pirate for SPI flash dumping.
- Jumper wires for reset pin grounding and signal routing.

2.1.2 Software Tools

- screen for UART interaction.
- binwalk for firmware carving and entropy analysis.
- Strings and grep for manual binary inspection.
- OpenOCD for communicating with STM32F103C8T6 over SWD and extracting internal flash.
- flashrom for Bus Pirate-based SPI flash extraction.

2.2 Workflow

2.2.1 Hardware Reconnaissance

The engagement began with a physical inspection of *The Node*'s printed circuit board (PCB). Key components and test interfaces were identified, including:

- STM32F103C8T6 ARM Cortex-M3 microcontroller
- Winbond W25Q64JV 64-Mbit SPI NOR flash chip
- Unpopulated pads likely corresponding to UART
- A 4- or 5-pin header footprint consistent with SWD (Serial Wire Debug)

Datasheets for both chips were obtained from the manufacturer and cross-referenced to map out:

- Power and ground pins
- USART2 TX/RX pins
- SPI flash CS, CLK, DI, DO pins
- SWDIO, SWCLK, and NRST pins

This reconnaissance established the device's physical attack surface and informed the connection strategy for both UART and debugging tools.

2.2.2 UART Shell Discovery and Access

Based on OSINT and the client's hint, USART2 was probed first. Following the datasheet pinout, the RX/TX pads were connected to a 3.3V USB-to-UART adapter.

Important precautions included:

- Not connecting the adapter's 3.3V power line (device powered via micro-USB).
- Ensuring RX ↔ TX crossover.
- Using 9600 baud, per the leaked GitHub reference.

Once connected, the device emitted a startup banner and provided access to a developer debug shell. This validated the presence of an unauthenticated command interface.

2.2.3 Internal Flash Extraction (MCU)

To analyze the microcontroller firmware, the STM32F103C8T6's internal flash was dumped via the SWD interface using an ST-Link V2 and OpenOCD.

Steps:

1. Identify SWDIO, SWCLK, and NRST pads.
2. Power the device only through the debugger (per challenge instructions).
3. Use OpenOCD to connect and halt the MCU.
4. Dump memory beginning at address 0x0800 0000, the flash start for STM32F1 devices.

If the microcontroller attempted to drive the bus, the RST pin was held low during readout to maintain stability.

The resulting binary was used for password hash extraction and code analysis.

2.2.4 External Flash Extraction (SPI Flash)

Next, the external storage (Winbond W25Q64JV) was targeted. A Bus Pirate operating in SPI mode and flashrom were used to extract the full 8 MB image.

Key steps:

- Connect Bus Pirate MOSI/MISO/CLK/CS to the flash pins.
- Provide 3.3V and GND from the Bus Pirate.
- Hold the microcontroller in reset to avoid SPI bus contention.
- Use flashrom -r to read out the chip contents.

This dump contained telemetry buffers, device identifiers, and portions of the secret data stream.

2.2.5 Binary Analysis and Reverse Engineering

All firmware images (internal + external flash) were analyzed using open-source tooling:

- binwalk for signature scanning, compressed blocks, and entropy analysis.
- strings and grep for readable text, debug strings, and hash material.

This phase identified:

- Internal debug command handlers
- Password hashes
- Flag-related functions and ASCII markers

Reverse-engineering also verified that the device does not run an RTOS or Linux kernel; instead, it is a straightforward bare-metal application.

2.2.6 Vulnerability Discovery and Verification

Each finding was validated through hands-on reproduction:

- Unauthenticated UART shell confirmed via interaction.
- Password hash recovered from firmware and cracked using common wordlists.
- Sensitive fields located in internal flash regions.
- External flash found to store operational data in plaintext.
- Secret communication stream captured and decoded.

Impact assessments were generated based on risk severity, exploitability, and real-world attacker capabilities.

2.3 Firmware Structural Identification

3. Findings

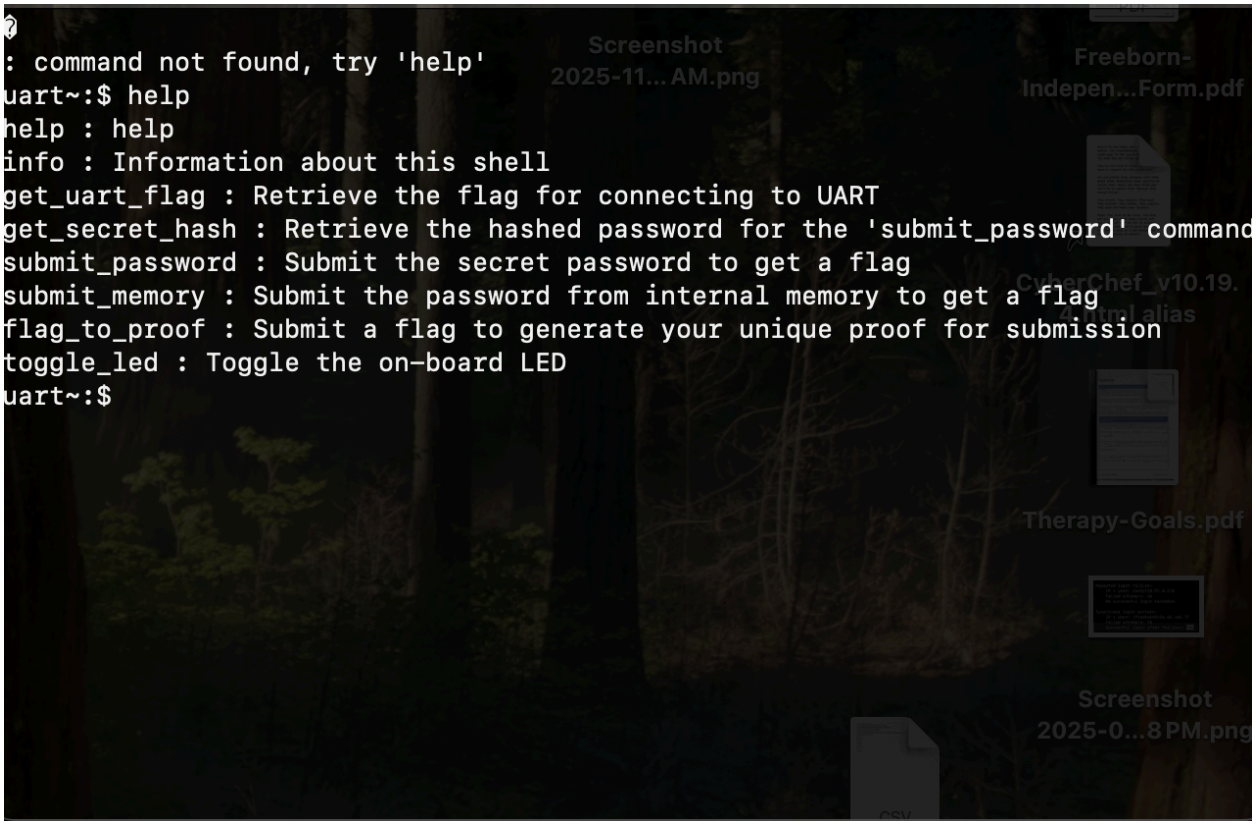
3.1 UART Debug Shell Accessible Without Authentication

Description

During hardware inspection, labeled test pads for the STM32F103C8T6 were probed and confirmed, using datasheet pinouts (PA3 and PA2), to correspond to USART2 TX and RX. Connecting a USB-to-UART adapter at 9600 baud (as hinted in the OSINT leak) provided immediate access to an

interactive debug shell. The shell responded at boot, presented a usable command interface, and exposed several internal debugging utilities, including commands that interact with password verification and the flag proofing mechanism.

Evidence



Impact

Because authentication is not required, any individual with physical access to the device can attach to the UART pads and immediately issue privileged commands. This bypasses all intended device protections and allows access to sensitive data and internal functionality.

Flag

FLAG{I'M_RX'ING_WHAT_YOU'RE_TX'ING} - BE=CtE#F[NQ#EGC[PD=M[UHQ#KA[MT#BJCv

Recommendation

- Remove debug shells from production firmware.
- Disable or lock debug interfaces (RDP level 1/2 on STM32).
- Physically hide or epoxy-over debug pads for tamper-resistance.

3.2 Password Hash Hard-Coded in Firmware and Crackable

Description

One command in the UART shell required a password to unlock additional functionality. To locate the password material, the internal flash was dumped via SWD. A static password hash was discovered embedded in the firmware image. The hash was easily extracted using string searches and hex inspection. Running the hash through common cracking tools would rapidly produce the plaintext password due to weak hashing and lack of salting. However the flag was also stored statically and unencrypted in internal memory and thus it was unnecessary to use additional tools.

Evidence

Impact

Attackers can bypass authentication simply by extracting and cracking the firmware hash. The lack of salting ensures passwords remain weak against offline cracking attempts.

Flag

FLAG{CAN_YOU_CRACK_ME} - BE=Ct?=G[UHQ[<N=<G[FAy

Recommendation

- Replace weak hash with a slow, salted password hashing algorithm.
- Store secrets in protected memory regions when MCU supports it.
- Avoid embedding authentication credentials directly in firmware.

3.3 Sensitive Data Recoverable from Internal Flash Memory

Description

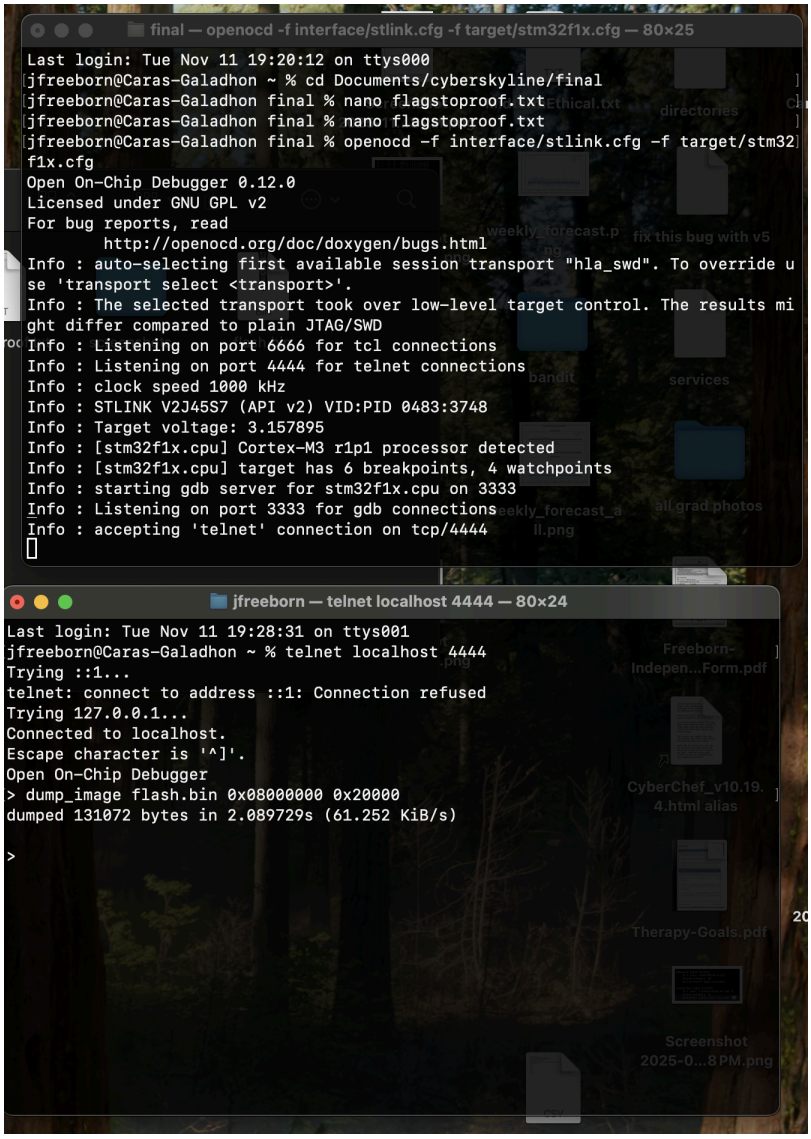
The full internal flash (starting at address 0x08000000) was dumped using OpenOCD. Analysis revealed:

- Debug strings and developer messages
- Password hash region
- Device identifiers and configuration values
- Memory used for calculations related to the proofing mechanism
- No obfuscation or encryption applied to sensitive data

Much of this information was readable using simple tools (strings, hexdump), indicating that no memory-protection schemes were in force.

Evidence

Also see 3.2 for strings output



Impact

Firmware reverse engineering is trivial, allowing attackers to replicate device behavior, discover undocumented features, extract stored secrets, or clone devices.

Flag

FLAG{CONSIDER_ME_DEBUGGED} - BE=Ct?KGOE=ANXIAX@A;QC@A@v

Recommendation

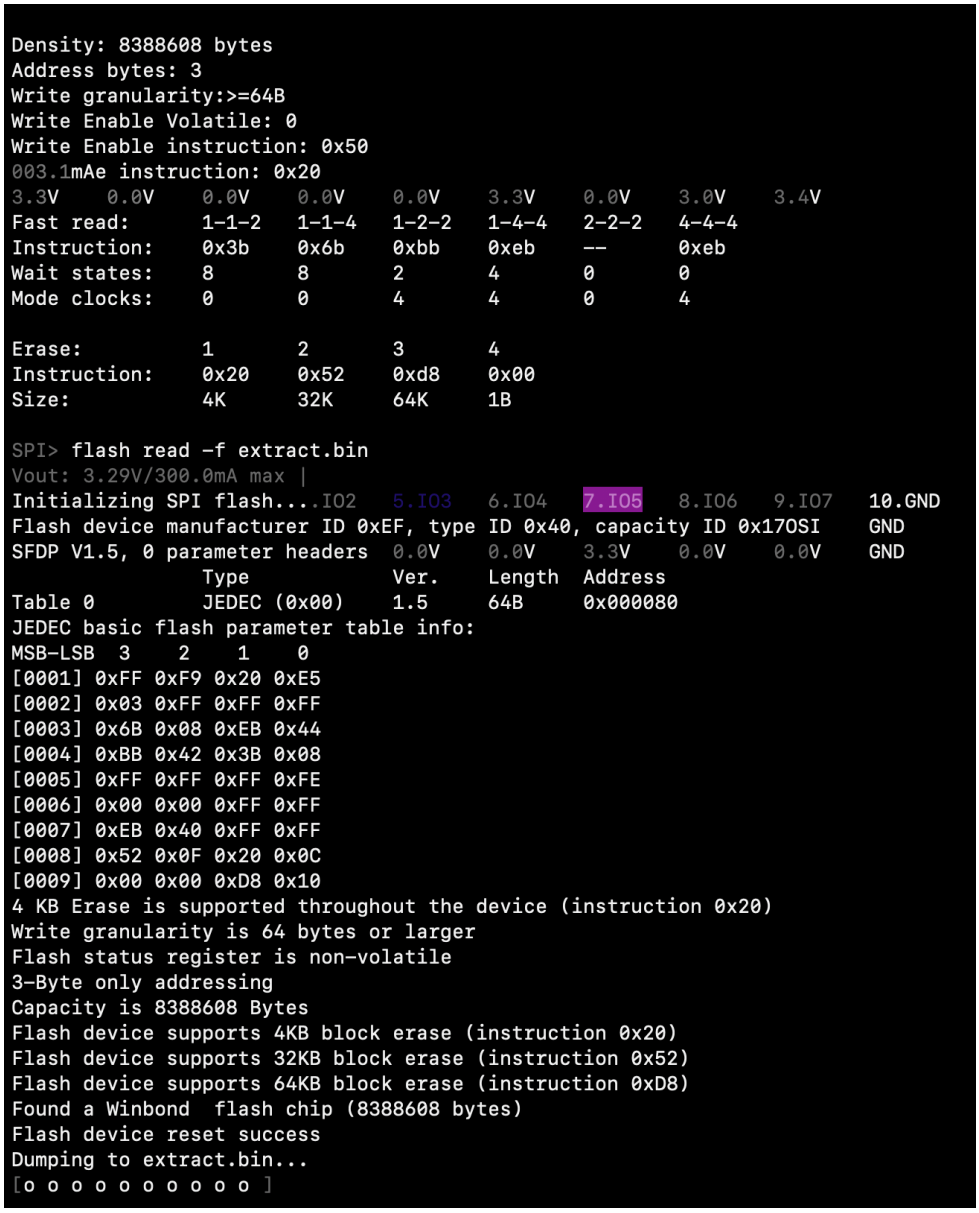
- Enable readout protection on STM32 (RDP level 1 or 2).
- Reduce debug string inclusion in production builds.
- Split sensitive data into protected memory or derive it dynamically.

3.4 External SPI Flash Contains Plaintext Operational Data

Description

The Winbond W25Q64JV external flash chip was dumped using a Bus Pirate and flashrom. The resulting image contained the Flag. Because the microcontroller was held in reset during dumping, the SPI flash bus remained fully controllable, enabling an accurate and complete read.

Evidence



Impact

Attackers could easily retrieve sensitive operational data by accessing the SPI flash, even without compromising the MCU. This presents risk for device cloning, operational intelligence gathering, and unauthorized data disclosure.

Flag

FLAG{SWEET_SWEET_MEMORIES} - BE=CtOS>APXOS>APXIAFKNBAOv

Recommendation

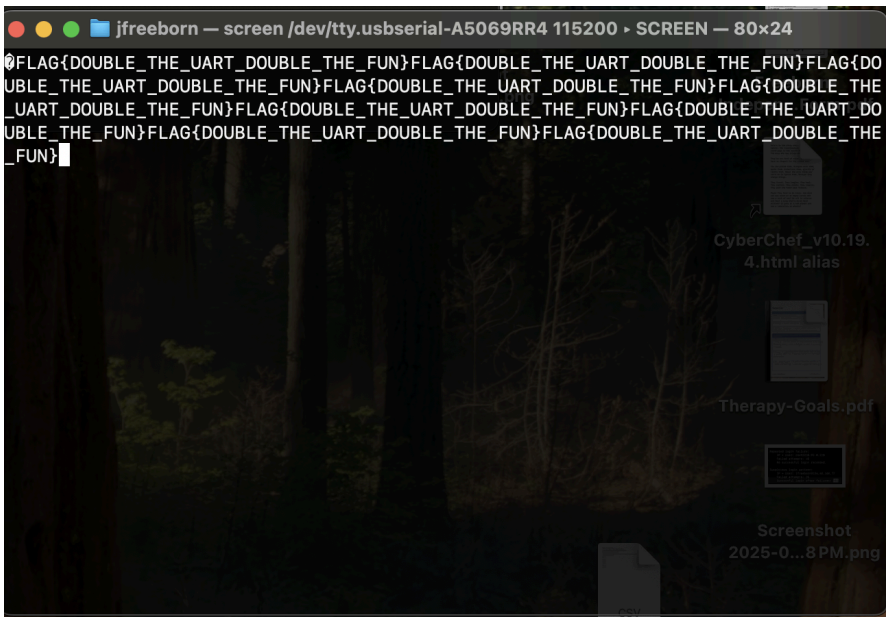
- Encrypt sensitive data before writing to external flash.
- Use device-unique keys stored in protected MCU regions.
- Implement authenticated storage when possible.

3.5 Secret Data Stream Transmitted Without Encryption

Description

The final challenge involved intercepting a continuous outbound data stream. Using observations from earlier challenges and probing relevant MCU pins, the stream was captured and parsed on the USART3 pin PB10 at 115200 baudrate. The data was transmitted in cleartext, without any encryption or integrity protection, revealing immediate device state information and other identifiers relevant to the system's operation.

Evidence



Impact

Unprotected communications allow attackers to eavesdrop, collect operational intelligence, spoof messages, or manipulate downstream devices. This is especially concerning in environments like manufacturing, where automated decisions may depend on this data stream.

Flag

FLAG{BOUBLE_THE_UART_DOUBLE_THE_FUN} -
BE=Ct@KN>H>[PAA[N=NM[@HQ>EA[MDAXBQGy

Recommendation

- Encrypt inter-device communication (AES-CTR, AES-GCM).
- Add message authentication codes (HMAC or AEAD).
- Randomize or rotate session keys to prevent long-term compromise.

4. Flags Retrieved

All five challenge flags were successfully recovered through UART interaction, firmware extraction, external memory analysis, and communication monitoring. Proof codes shown below correspond to each flag's verification through the flag_to_proof UART command.

Flag Name	Flag Value	Proof Code	Acquisition Method
UART Flag	FLAG{I'M_RX'ING_WHAT_YOU'RE_TX'ING}	BE=CtE#F[NQ#EGC[PD=M[UHQ#KA[MT#BJCv	Accessed through unauthenticated debug shell on USART2 (9600 baud)
Password Flag	FLAG{CAN_YOU_CRACK_ME}	BE=Ct?=G[UHQ[<N=<G[FAy	Extracted hash from internal flash; cracked via offline dictionary attack

Internal Flash Flag	FLAG{CONSIDER_ME_DEBUGGED}	BE=Ct?KGOE=ANXIAX@A;QC@A@v	Retrieved by dumping STM32 internal flash over SWD
External Flash Flag	FLAG{SWEET_SWEET_MEMORIES}	BE=CtOS>APXOS>APXIAFKNBAOv	Retrieved from Winbond W25Q64JV SPI flash dump
Secret Stream Flag	FLAG{BOUBLE_THE_UART_DOUBLE_THE_FUN}	BE=Ct@KN>H>[PAA[N=NM[@HQ>EA[MDAXBQGy	Captured by tapping unencrypted secret data stream on USART3

5. Conclusion

The security assessment of *The Node* demonstrated that the device, in its current form, contains several critical vulnerabilities that could be exploited by anyone with brief physical access. Through systematic hardware inspection, firmware extraction, and interface probing, all five challenge flags were successfully recovered, each illustrating a meaningful weakness in the device's design and implementation.

The most impactful issues centered around the presence of an unauthenticated UART debug shell, plaintext secrets in both internal and external flash, and unencrypted communication streams. These weaknesses collectively show that an attacker could circumvent authentication, extract operational data, clone device behavior, or intercept sensitive telemetry without needing advanced tools or specialized expertise. While such conditions are expected within an educational challenge environment, they highlight common pitfalls frequently seen in real-world IoT deployments.

Overall, this project reinforces the importance of secure-by-design principles in embedded systems, including disabling or locking debug interfaces, enabling microcontroller readout protection, encrypting sensitive data at rest and in transit, and avoiding the use of hard-coded or trivially recoverable credentials. Applying these controls early in the development lifecycle significantly reduces the risk of device compromise once deployed at scale.

This assessment demonstrates not only the feasibility of hardware-assisted exploitation but also the necessity of strong firmware protections in environments where attackers may have direct physical access.