SECURITY ASSESSMENT: THE NODE

Samuel LaTourette

ACME Manufacturing

Table of Contents

Introduction

ACME Manufacturing, Inc. has asked our team to perform an IoT security assessment of their new data collection device, "The Node." The company plans to deploy hundreds of these devices across their manufacturing facility and expressed concerns about the security of sensitive data, particularly given that short-term contract employees have physical access to the devices.

The primary attack surface identified by the client includes the STM32F103C8T6 microcontroller and an external Winbond W25Q64JV SPI flash memory chip. The client specifically requested we investigate a suspected debug shell, password-related functionality, sensitive data storage in internal and external memory, and any data streams transmitted to connected devices.

This report documents our security assessment methodology, findings, and recommendations for each vulnerability discovered. Using common hardware hacking tools including a USB-to-serial adapter and an ST-Link V2 debugger, we successfully retrieved four flags representing critical security weaknesses in the device. Our assessment reveals that The Node in its current state lacks fundamental security controls and should not be deployed in a production environment without significant mitigation.

UART Vulnerability

An exposed debug shell was discovered on the UART interface, accessible to anyone with physical access to the device.

Reproducible steps to find vulnerability:

The datasheet for the STM32F103C8T6 Microcontroller to identify the correct pins to access the USART2 interface. These corresponded to the PA2 (TX) and PA3 (RX) pins on the board visible with the silkscreen labels.

The FT232RL USB-to-serial adapter was then connected to analyze the USART2 interface. The connections were: RX→PA2, TX→PA3, GND→GND, and VCC→3.3V (*Figure 1*).

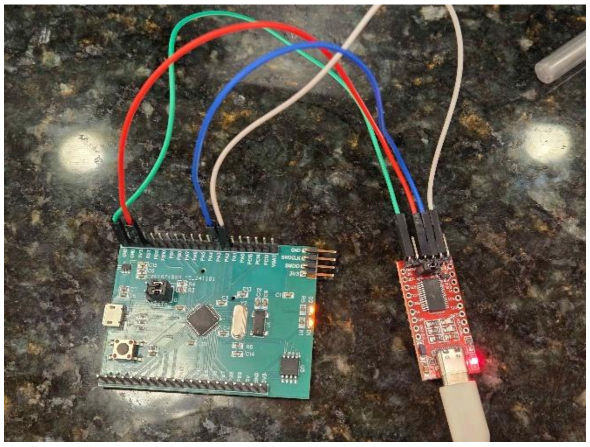
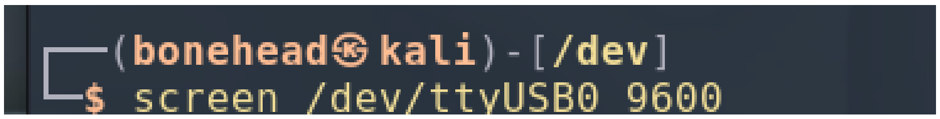


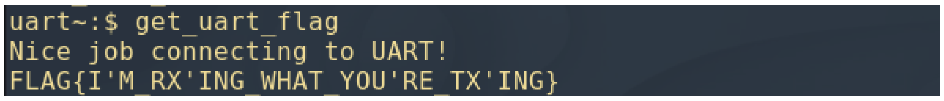
Figure 1

After establishing the connections, a screen command was executed to connect at the documented baud rate (*Figure 2*). I was then able to run a help command in the UART interface, which revealed a command titled **get_uart_flag**. Upon running the command, the confidential data was revealed (*Figure 3*).



```
┌──(bonehead㉿kali)-[/dev]
└─$ screen /dev/ttyUSB0 9600
```

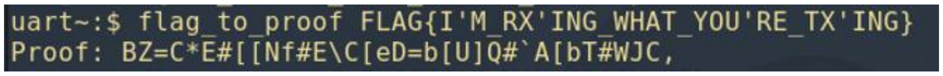
Figure 2



```
uart~:$ get_uart_flag
Nice job connecting to UART!
FLAG{I'M_RX'ING_WHAT_YOU'RE_TX'ING}
```

Figure 3

Flag to proof: BZ=C*E#[[Nf#E\C[eD=b[U]Q#`A[bT#WJC, (*Figure 4*)



```
uart~:$ flag_to_proof FLAG{I'M_RX'ING_WHAT_YOU'RE_TX'ING}
Proof: BZ=C*E#[[Nf#E\C[eD=b[U]Q#`A[bT#WJC,
```

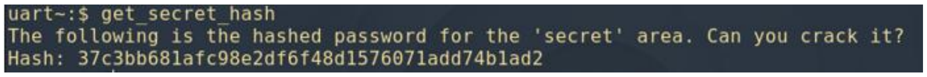
Figure 4

Exposed Password Hash Vulnerability

An exposed password hash was discovered within the debug shell. After cracking this hash, another confidential flag was obtained.

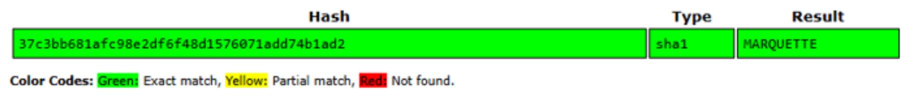
Reproducible steps to find vulnerability:

After accessing the UART shell, there was another command listed in the help menu – **get_secret_hash**. Upon running this command (*Figure 5*), a hashed password was provided. I identified this password as SHA-1, as there are 40 hex characters.
Using CrackStation, an online rainbow table, I was able to crack the hash to reveal the plaintext password **MARQUETTE** (*Figure 6*). I then used the **submit_password** command, which provided the next confidential flag (*Figure 7*).



```
uart~:$ get_secret_hash
The following is the hashed password for the 'secret' area. Can you crack it?
Hash: 37c3bb681afc98e2df6f48d1576071add74b1ad2
```

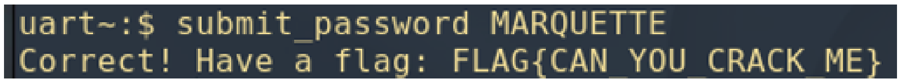
Figure 5



Hash	Type	Result
37c3bb681afc98e2df6f48d1576071add74b1ad2	sha1	MARQUETTE

Color Codes: 🟩 Green: Exact match, 🟨 Yellow: Partial match, 🟥 Red: Not found.

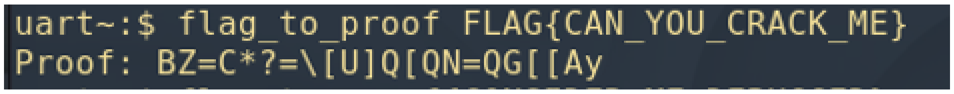
Figure 6



```
uart~:$ submit_password MARQUETTE
Correct! Have a flag: FLAG{CAN_YOU_CRACK_ME}
```

Figure 7

Flag to proof: BZ=C*?=\[U]Q[QN=QG[[Ay (*Figure 8*)



```
uart~:$ flag_to_proof FLAG{CAN_YOU_CRACK_ME}
Proof: BZ=C*?=\[U]Q[QN=QG[[Ay
```

Figure 8

Internal Flash Vulnerability

A confidential flag was discovered stored in plaintext within The Node's internal flash memory.

Reproducible steps to find vulnerability:

The SWD debug header pins (GND, SWDCLK, SWDIO, 3V3) were located on The Node's PCB. The ST-Link V2 debugger was then connected to the corresponding pins (*Figure 9*).

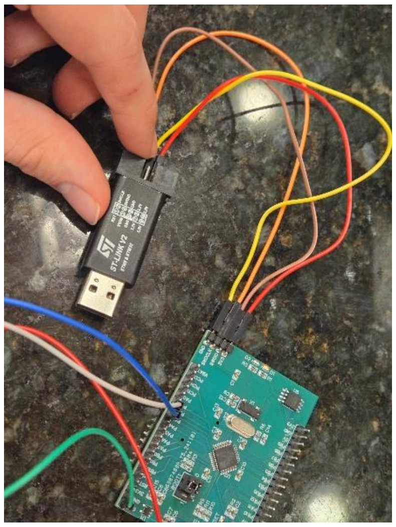
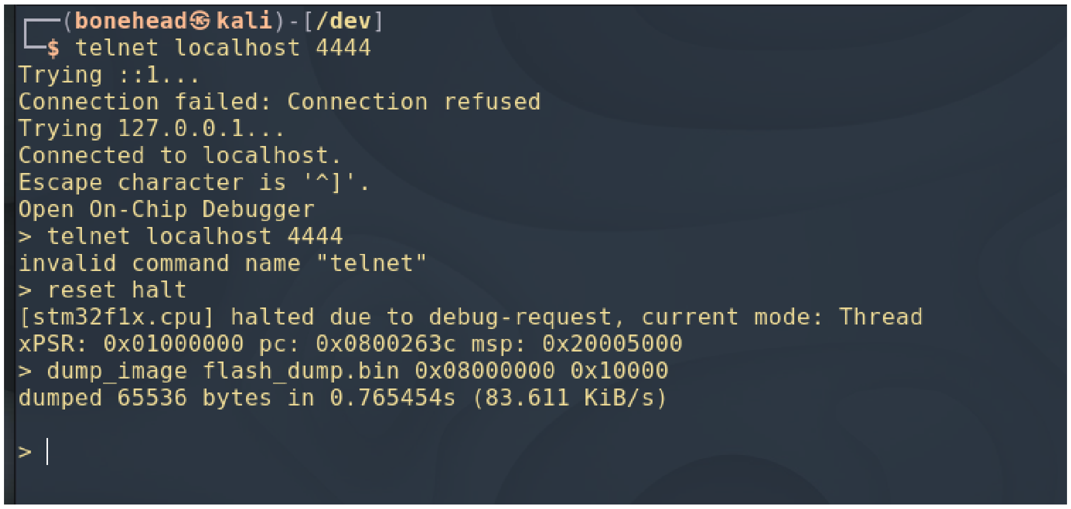


Figure 9

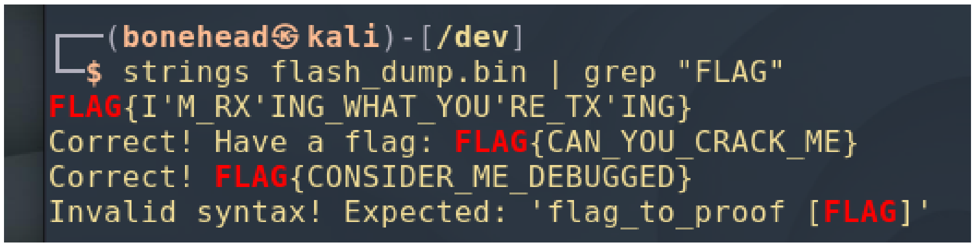
Following the connection to the debug pins, I ran OpenOCD with the STM32F1x configuration (sudo openocd -f interface/stlink.cfg -f target/stm32f1x.cfg). This allowed me to connect via telnet, halt the CPU, and then dump the internal flash image. The datasheet revealed the flash memory address (0x08000000) and size (0x10000) (*Figure 10*). Then I used strings and grep to find the confidential flag (*Figure 11*).



```
┌──(bonehead㉿kali)-[/dev]
└─$ telnet localhost 4444
Trying ::1...
Connection failed: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> telnet localhost 4444
invalid command name "telnet"
> reset halt
[stm32f1x.cpu] halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x0800263c msp: 0x20005000
> dump_image flash_dump.bin 0x08000000 0x10000
dumped 65536 bytes in 0.765454s (83.611 KiB/s)

> |
```

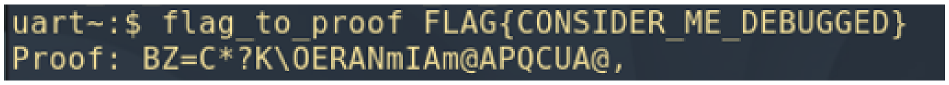
Figure 10



```
┌──(bonehead㉿kali)-[/dev]
└─$ strings flash_dump.bin | grep "FLAG"
FLAG{I'M_RX'ING_WHAT_YOU'RE_TX'ING}
Correct! Have a flag: FLAG{CAN_YOU_CRACK_ME}
Correct! FLAG{CONSIDER_ME_DEBUGGED}
Invalid syntax! Expected: 'flag_to_proof [FLAG]'
```

Figure 11

Flag to proof: BZ=C*?K\OERANmIAm@APQCUA@, (*Figure 12*)



```
uart~:$ flag_to_proof FLAG{CONSIDER_ME_DEBUGGED}
Proof: BZ=C*?K\OERANmIAm@APQCUA@,
```

Figure 12

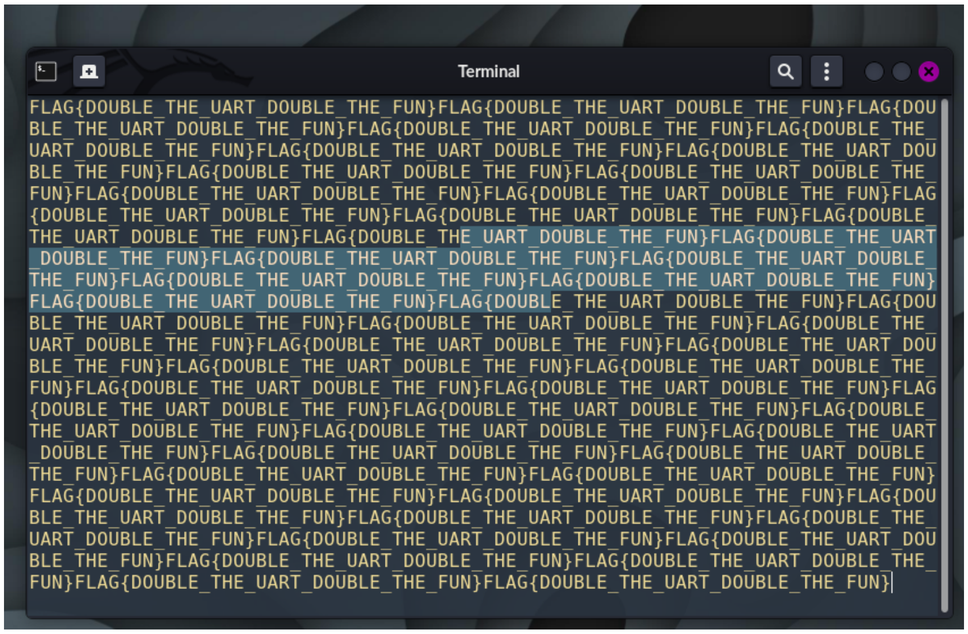
Secret Stream Vulnerability

A confidential flag was discovered being continuously broadcast on the USART3 interface.

Reproducible steps to find vulnerability:

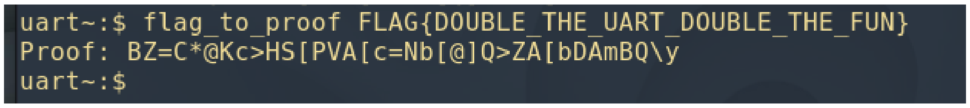
The STM32F103C8T6 datasheet was reviewed for additional communication interfaces. The USART3 interface on pins PB10 (TX) and PB11 (RX) was identified as an unexplored attack vector. The FT232 RXD was connected to PB10 to listen for transmissions. Using the screen command at a baud rate of 9600, a continuous unencrypted data transmission containing the confidential flag was observed (*Figure 13*).

Figure 13



```
FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOU
BLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_
UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOU
BLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_
FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG
{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_
THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART
_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_
THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}
FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOU
BLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_
UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOU
BLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_
FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG
{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_
THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART
_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_
THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}
FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOU
BLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_
UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOU
BLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_
FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}
```

Flag to proof: BZ=C*@Kc>HS[PVA[c=Nb[@]Q>ZA[bDAmBQ\y (*Figure 14*)



```
uart~:$ flag_to_proof FLAG{DOUBLE_THE_UART_DOUBLE_THE_FUN}
Proof: BZ=C*@Kc>HS[PVA[c=Nb[@]Q>ZA[bDAmBQ\y
uart~:$
```

Figure 14

Mitigation Recommendations

UART Vulnerability Mitigation Recommendation:

There is an exposed debug shell on the UART interface accessible to anyone with physical access. We would recommend the following:

1: Disable debug interfaces in production firmware.

2: Require authentication (password) before granting shell access.

3: Implement secure boot to prevent unauthorized firmware modifications.

Exposed Password Hash Vulnerability:

There is an exposed password hash that can be easily cracked to access confidential information. Our recommendations are:

1. Use a stronger password hashing algorithm. SHA-1 is deprecated and uses no salt.

2. Enforce stronger password policies (special characters, length, etc.)

3. Do not leave exposed password hashes through any interface.

Internal Flash Memory Vulnerability:

There is a confidential flag stored in plaintext located in the Internal Flash Memory that can be accessed by anyone physically in contact with the device, with no read-out protection. We would recommend the following:

1. Enable read-out protection to prevent flash dumping.

2. Disable SWD/JTAG interfaces in production.

3. Encrypt sensitive data in flash memory (strong encryption algorithm).

4. Use secure boot to verify firmware integrity.

Secret Stream Vulnerability:

The USART 3 interface is constantly broadcasting confidential flag information in plaintext. To mitigate:

1. Encrypt all inter-device communication

2. Don't continuously broadcast; use request-response patterns instead.

Conclusion

To conclude, our security assessment of The Node IoT device revealed significant vulnerabilities across multiple attack vectors. Through physical access alone, we were able to access an unauthenticated debug shell via UART, crack a weakly-hashed password, dump internal flash memory, and intercept sensitive data being continuously broadcast. We strongly recommend implementing the mitigation steps provided to secure critical data before deploying these devices in a production environment.