Assignment 2: Firmware security and rehosting

Group 14: Sharaf E. Boukhezer — Wai Lee Boo — Kade Quakenbush

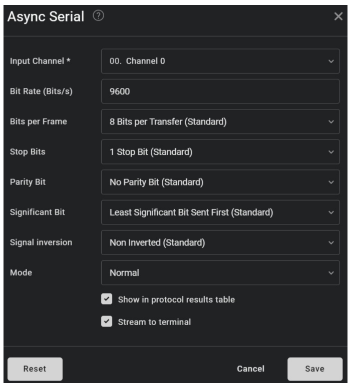
1 Introduction

This report investigates an embedded mechanism implemented on a Raspberry Pi Pico. The objective was to recover key information from a legacy, undocumented hardware token by unlocking the device using its PIN code and retrieving its associated flags. The analysis aimed to reverse engineer its communication interfaces: Part 1 by identifying and decoding Universal Asynchronous Receiver–Transmitter (UART) and Serial Peripheral Interface (SPI) outputs using a logic analyser, while Part 2 involved developing a function-level rehosting and PIN-brute-force automation script to emulate the firmware. This report summarises the applied procedure and reasoning behind it.

2 Protocol Reverse Engineering

2.1 UART Identification

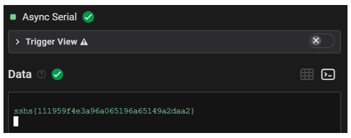
After completing the preliminary setup steps (firmware flashing, Logic 2 installation, and USB communication verification), the UART transmission was triggered via the firmware menu and the logic analyser was sequentially connected to candidate pins. Probing the candidate pins revealed a clear asynchronous signal on GPIO0 (Input Channel 0, Pin 1), with a common GND reference. Using Logic 2's serial decoder, the correct parameters were determined to be 9600 baud, 8N1, producing no decoding errors (Figure 1). The transmission resolved to the flag **sshs{111959f4e3a96a065196a65149a2daa2}** (Figure 2). The validated UART capture is shown in Figure 3.



Input Channel *	00. Channel 0
Bit Rate (Bits/s)	9600
Bits per Frame	8 Bits per Transfer (Standard)
Stop Bits	1 Stop Bit (Standard)
Parity Bit	No Parity Bit (Standard)
Significant Bit	Least Significant Bit Sent First (Standard)
Signal inversion	Non Inverted (Standard)
Mode	Normal

✓ Show in protocol results table
✓ Stream to terminal

Figure 1: UART Settings



```
sshs{111959f4e3a96a065196a65149a2daa2}
```

Figure 2: Decoded UART output.

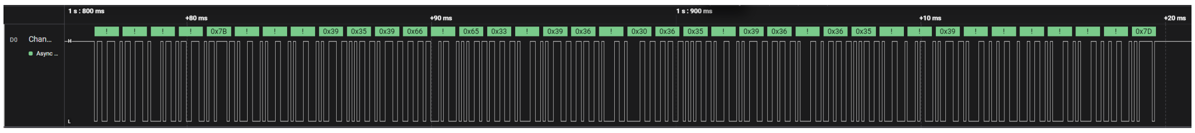
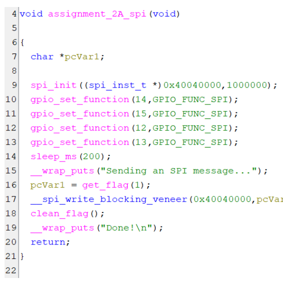


Figure 3: UART waveform showing serial transmission of flag bytes.

2.2 SPI Identification

SPI signalling was investigated by probing multiple GPIO pins with the logic analyser while triggering the SPI transmission from the firmware menu, revealing four active lines, consistent with the firmware configuration confirmed via the disassembly of `assignment_2A_spi` in Ghidra (Figure 4). The identified mapping was MOSI on GPIO15 (Channel 0, Pin 20), MISO on GPIO12 (Channel 2, Pin 16), Clock (SCK) on GPIO14 (Channel 1, Pin 19), and Enable (CS) on GPIO13 (Channel 3, Pin 17), with a shared GND reference. Using the captured waveforms (Figure 6), Logic 2 SPI decoder was configured to Mode 0, MSB-first (Figure 5), producing 16-bit values in which the lower byte corresponded to ASCII characters (Figure 8), which translates to the flag **sshs{68503db6c4be75ea99ff3c3af4d82897}** with no errors.



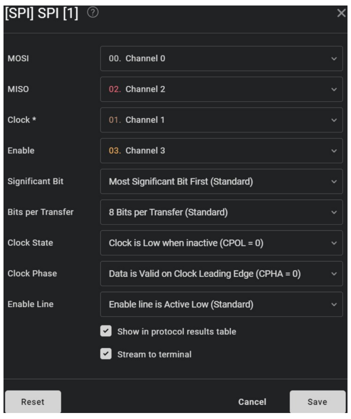
```c
void assignment_2A_spi(void)

{
  char *pcVar1;

  spi_init((spi_inst_t *)0x40040000,1000000);
  gpio_set_function(14,GPIO_FUNC_SPI);
  gpio_set_function(15,GPIO_FUNC_SPI);
  gpio_set_function(12,GPIO_FUNC_SPI);
  gpio_set_function(13,GPIO_FUNC_SPI);
  sleep_ms(200);
  __wrap_puts("Sending an SPI message...");
  pcVar1 = get_flag(1);
  __spi_write_blocking_veneer(0x40040000,pcVar1,0x26);
  clean_flag();
  __wrap_puts("Done!\n");
  return;
}

```

Figure 4: Ghidra view of assignment_2A_spi function.



[SPI] SPI [1]

MOSI	00. Channel 0
MISO	02. Channel 2
Clock *	01. Channel 1
Enable	03. Channel 3
Significant Bit	Most Significant Bit First (Standard)
Bits per Transfer	8 Bits per Transfer (Standard)
Clock State	Clock is Low when inactive (CPOL = 0)
Clock Phase	Data is Valid on Clock Leading Edge (CPHA = 0)
Enable Line	Enable line is Active Low (Standard)

✓ Show in protocol results table

✓ Stream to terminal

Figure 5: SPI Settings

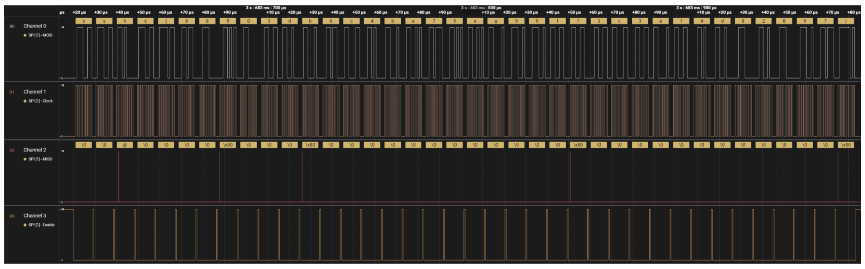


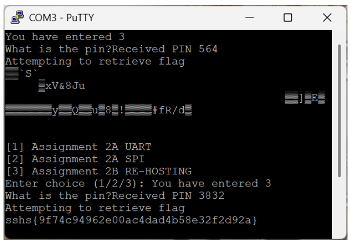
Figure 6: SPI waveform capture showing MOSI/MISO activity.

3 Rehosting

Rehosting was performed using Unicorn, which restored the saved register state, executing `assignment_2B_rehost` in isolation. This reconstructed the memory layout by mapping the ROM, flash, and SRAM, and loading the respective binaries. Static analysis of `sshs.elf` in Ghidra identified the function boundaries and determined which routines required emulation. CPU-local functions, such as memcpy, required no modification, as they execute natively under Unicorn. Hardware-dependent routines, however, were intercepted through targeted hooks:

- `hook_input`: Replaces the firmware's input routine by writing the current PIN attempt into the buffer pointed to by R1, bypassing the unavailable UART input.
- `hook_sleep`: Skips delay routines (e.g. `sleep_ms`), as they rely on hardware timers absent from the emulated environment. It also bypasses `printf`, as it outputs no information relevant to the PIN-checking logic.
- `hook_print`: Intercepts output routines (e.g. `puts`) by reading the string at the address stored in R0 and checking for "`sshs`", which indicates a flag. When detected, the flag and corresponding PIN are recorded and emulation is stopped.

Each hook redirected control flow via the link register, bypassing unimplemented firmware logic. The emulator iterated over the PIN-space [0,9999] until the flag was found, reinitialising registers from `regs.txt` on each and executing only the PIN-checking sequence for efficiency. When the injected value reached 3832, the print hook captured and output the flag **sshs{9f74c94962e00ac4dad4b58e32f2d92a}**, consistent with the Raspberry Pi Pico's behaviour (Figure 7).

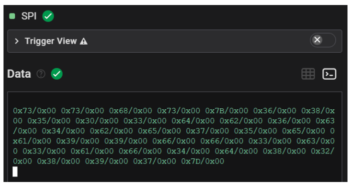


COM3 - PuTTY

```
You have entered 3
What is the pin?Received PIN 564
Attempting to retrieve flag
▒`S`
    ▒xV&8Ju
                                               ▒]▒E▒
▒▒▒▒▒▒▒y▒Q▒u▒8▒!▒▒▒▒#fR/d▒


[1] Assignment 2A UART
[2] Assignment 2A SPI
[3] Assignment 2B RE-HOSTING
Enter choice (1/2/3): You have entered 3
What is the pin?Received PIN 3832
Attempting to retrieve flag
sshs{9f74c94962e00ac4dad4b58e32f2d92a}
```

Figure 7: Rehosting output showing PIN entry and recovered flag.



```
0x73/0x00 0x73/0x00 0x68/0x00 0x73/0x00 0x7B/0x00 0x36/0x00 0x38/0x
00 0x35/0x00 0x30/0x00 0x33/0x00 0x64/0x00 0x62/0x00 0x36/0x00 0x63
/0x00 0x34/0x00 0x62/0x00 0x65/0x00 0x37/0x00 0x35/0x00 0x65/0x00 0
x61/0x00 0x39/0x00 0x39/0x00 0x66/0x00 0x66/0x00 0x33/0x00 0x63/0x0
0 0x33/0x00 0x61/0x00 0x66/0x00 0x34/0x00 0x64/0x00 0x38/0x00 0x32/
0x00 0x38/0x00 0x39/0x00 0x37/0x00 0x7D/0x00
```

Figure 8: Decoded SPI transmission displaying the flag bytes.