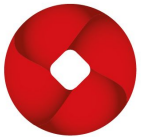
The Art & Craft of writing ARM shellcode

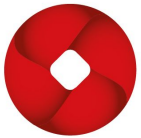
Munawwar Hussain Shelia (IoT Security Researcher @ Payatu)

Who Am I?

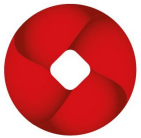


- Working with Payatu as IoT Security Research
- Speaker & Trainer
  - Nullcon, CPX 360, c0c0n, Beside
- Interest
  - Binary Analysis
  - Reverse Engineering
  - Malware Analysis
  - Exploitation Dev
- Blog - taintedbits.com
- Twitter - @0xd3xt3r
- Github - github.com/0xd3xt3r

ARM Instruction Set Overview

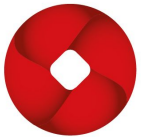


1. Architecture Overview
2. Register
3. Data processing instruction
4. Load-Store Instructions
5. Branching Instructions



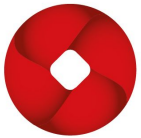
ARM Architecture Overview

- RISC instruction set architecture
- Load/Store architecture
- Fixed instruction size ~ 32 bits
- Fairly large register bank of 32 bit registers
- 3 address instruction format
- Pipelining – fetch-decode-execute
- Instruction Set - ARM, THUMB, THUMBEE, JAZELLE
- In ARM mode instruction size are 32-bit, 4-bytes and in THUMB MODE 16-bit, (2-bytes)
- Multiple Privilege level.



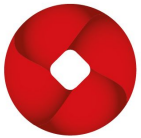
Register

- Core registers : R0, R1, … R15
- CPSR status register
- R15 ~ PC (Program Counter)
- R14 ~ LR (Link Register) – Hold the return link address (function calls)
- R13 ~ SP (Stack Pointer) – Points to the process stack
- R11 ~ FP (Frame Pointer)



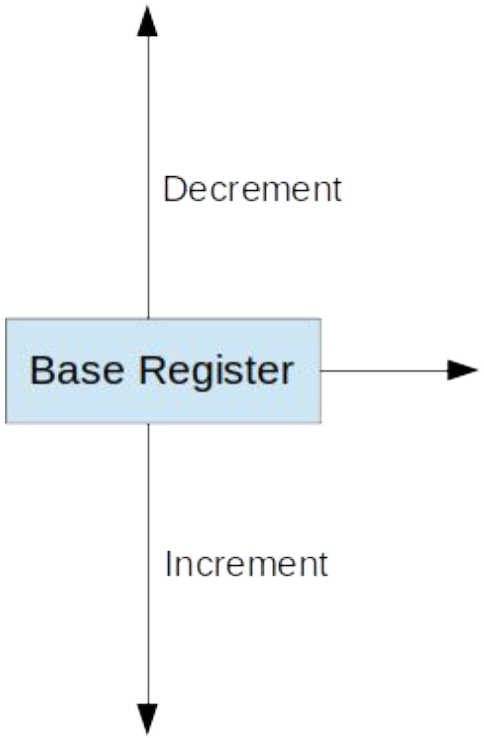
Data Processing Instruction

- Have a destination register, first operand register and second operand(register or immediate) and optionally a shifter
- Arithmetic operations
  - ADD, ADC, SUB, SBC, RSB, RSC, MUL, MLA, MLS…
  - Example : ADD{S}<condition> dst, op1, op2 <shift_type> → ADD r0, r1, r2, LSR #2
- Bitwise Logical operations
  - AND, ORR, EOR, BIC, MVN, ORN, LSL, LSR, ASR
- Register movement operations – move values from one register to the other
  - MOV r0, r1
- Comparison operations – Just set the condition code flags, do not produce a result
  - CMP, CMN, TST, TEQ



Load-Store Instruction

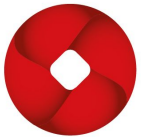
- Single register load and store
  - Load - LDR, LDRB, LDRH, LDRD…
  - Store - STR
  - Example : LDR r0, [r1]
- Multiple register load and store
  - Load - ldmia, ldmib, ldmda, ldmdb
  - Store - stmia, stmib, stmda, stmdb
  - Example: strdb r0, {r4 - r6}, ldmia r0, {r4 - r6}

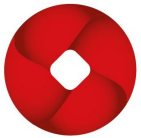


Memory	
Address	**Data**
0x00010054	1000
0x00010058	2000
0x0001005c	3000
0x00010060	4000
0x00010064	5000
0x00010068	6000
0x0001006c	7000
0x00010070	8000

```
int add_sub(int a, int b){
      int c ;
      c = a + b;
      printf ("a + b = %d\n", c);
      c = a - b ;
      printf ("a - b = %d\n", c);
      c = a * b;
      printf ("a * b = %d\n", c);
      return c;
}
```

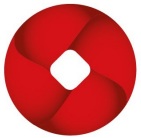
```
0x000102dc   push {fp, lr}
....
0x000102f0   ldr r2, [fp, -0x10]
0x000102f4   ldr r3, [fp, -0x14]
0x000102f8   add r3, r2, r3
...
0x0001030c   ldr r2, [fp, -0x10]
0x00010310   ldr r3, [fp, -0x14]
0x00010314   sub r3, r2, r3
....
0x00010328   ldr r3, [fp, -0x10]
0x0001032c   ldr r2, [fp, -0x14]
0x00010330   mul r1, r2, r3
....
0x00010348   mov r0, r3
0x0001034c   sub sp, fp, 4
0x00010350   pop {fp, pc}
```





Addressing Modes

- Addressing modes for memory access instructions
- Offset addressing - **[Rb, <offset>]**
  - Offset is applied to the base register value and the result is used as memory address. Register value is not changed
- Pre-indexed addressing - **[Rb, <offset>] !**
- Offset is applied to the base register value and the result is used as memory address and written back to base register
- Post-indexed addressing - **[Rb], <offset>**
  - Base register value is used as memory address. Offset is then applied to the base register and written back to it.
- <offset> here can be an immediate constant, register or shifted register



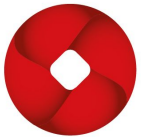
Branching Instruction

- Change the Status register
  - CMP, CMP, TST, TEQ
- Unconditional and conditional Branches
  - B, BL, BLX, BEQ, BLE, BGE....
- Taking an exception
  - Supervisor call – svc

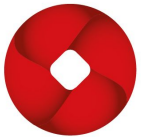
Condition Code	Meaning	Status Flag	Example
EQ	Equal	Z==1	BEQ <label> / Rd
NE	Not Equal	Z==0	BNE <label> / Rd
GT	Signed Greater Than	(Z==0) && (N==V)	BGT <label> / Rd
LT	Signed Less Than	N!=V	BLT <label> / Rd
GE	Signed Greater Than Equal	N==V	BGE <label> / Rd
LE	Signed Less Equal	(Z==1) || (N!=V)	BLE <label> / Rd
HS	Unsigned Higher	C == 1	BHS <label> / Rd
LO	Unsigned Lower	C == 0	BLO <label> / Rd

```
void brch(int a, int b){
    if (a>b)
        printf ("a>b\n");
    if (a==b)
        printf ("a==b\n");
    if (a<b)
        printf ("a<b\n");
}
```

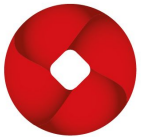
```
0x000103e4      push {fp, lr}
....
0x000103f8      ldr r2, [fp, -8]
0x000103fc      ldr r3, [fp, -0xc]
0x00010400      cmp r2, r3
0x00010404      ble 0x10410
.... [ Branch Body ]
0x00010410      ldr r2, [fp, -8]
0x00010414      ldr r3, [fp, -0xc]
0x00010418      cmp r2, r3
0x0001041c      bne 0x10428
.... [ Branch Body ]
0x00010428      ldr r2, [fp, -8]
0x0001042c      ldr r3, [fp, -0xc]
0x00010430      cmp r2, r3
0x00010434      bge 0x10440
..... [ Branch Body ]
0x00010440      mov r0, r0
0x00010448      pop {fp, pc}
```



Linux Shellcode

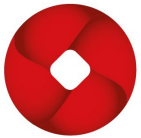


1. What is Shellcode?
2. System Calls
3. Tools of the Trade
4. Shellcoding Steps
5. Lab : Spawn a shell
6. Shellcoding Techniques
7. Lab : Remote shell
8. Lab : Reverse TCP Shell



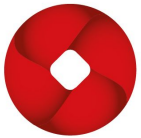
What is Shellcode?

- Machine instructions injected into a program
- Originally used to just spawn shell, but now it has more sophisticated functionality.
- Process independent machine code
- Shellcode being machine code depends on the processor
- Change the manner in which the original program executes, this is done by making system calls



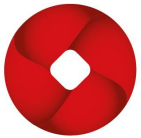
System Calls

- Syscall allows you to access and manipulate OS resources
- Syscall is the interface between kernel and user mode.
- Load the appropriate argument in the register and raise software interrupt
- Calling Convention for ARM
  - Registers R0-R6 used to store the arguments
  - Register R7 is used to store the syscall no.
  - Svc (or Swi) instruction causes the processor to switch to supervisor mode
    - svc #0
    - Legacy ABI used swi (#0x900000 + syscall no.) instead of R7
  - Return value of system call is stored in r0



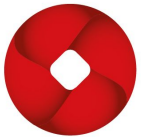
Tools of the Trade

- **GDB** - debugger
- **pwndbg** - let make GDB great again
- **pwntools** - exploit development framework
- **as** - assembler
- **ld** - linker
- **strace** - system call tracing
- **objdump** - disassemble binary
- **objcopy** - extract shellcode



Shellcoding Steps

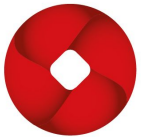
1. Write your shellcode in high-level language
2. Identify the system call number and parameter by disassembling the program
3. Understanding the input constraints
    a. Payload size
    b. Input filters - null, alphanumeric, etc
4. Organizing the parameter data in the code such that the payload becomes self-contained
5. Extract the bytes!



Lab : Spawn a shell

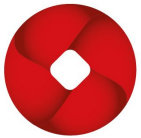
int execve(const char *filename, char *const argv[], char *const envp[])

- R0 → /bin/sh
- R1 → NULL
- R2 → NULL
- R7 → execve call number 11



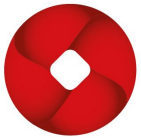
Shellcoding Techniques

- Shellcode being **Machine Code** depends on the processor
- Relative addressing for data using **PC offset** instead of **Label Addressing**.
- Register PC points to the **instruction being fetched** and not the one being executed
- Machine code output in objdump is reversed (little endian)
- Optimization Technique involves writing shellcode as much as possible in THUMB MODE(2 byte) and rest of it in ARM mode (4 byte)



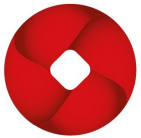
Shellcoding Techniques: Remove NULL BYTE

- ARM system call instruction
  - svc #0 => ef000000
  - svc #010101 => ef010101 → instead
- mov r0, #1 => e3a00001 #oops!
- Here's a random workaround
  - mov r7, #65536 → e3a07801
  - add r7, r7, #1 → e2877001
  - sub r0, r7, #65536 → e2470801



Lab : Spawn a TCP shell

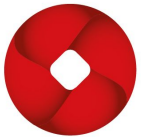
1. Start a socket server and listen for clients
2. Accepts the client connection
3. Once Connected
    a. Duplicate the client descriptor for STDIN, STDOUT, STDERR
    b. Open shell '/bin/sh'



Lab : Reverse TCP shell

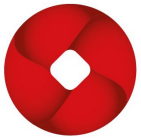
1. Try to connect to External server on TCP socket
2. Once Connected
    a. Duplicate the client descriptor for STDIN, STDOUT, STDERR
    b. Open shell '/bin/sh'

Bare-Metal Exploits



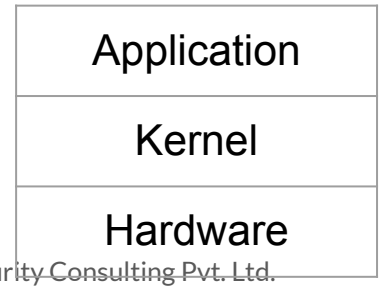
1. Embedded OS
2. Static Analysis
3. Dynamic Analysis
4. Lab : Activate a Peripheral
5. Exploit Mitigations

Embedded OS



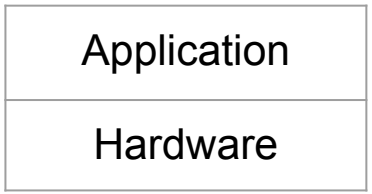
Full-Fledged OS

- OS - Linux, Unix, QNX, Windows, OpenWrt, etc
- Feature rich applications
- Hardware: microprocessor
- Heavy processing more advanced OS features required.

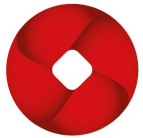


Bare Metal OS

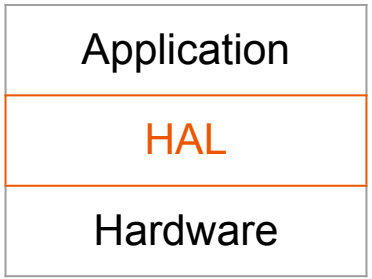
- OS - FreeRTOS, mbed-os,
- Not very sophisticated features
- Hardware: micro-controllers
- Lightweight processing



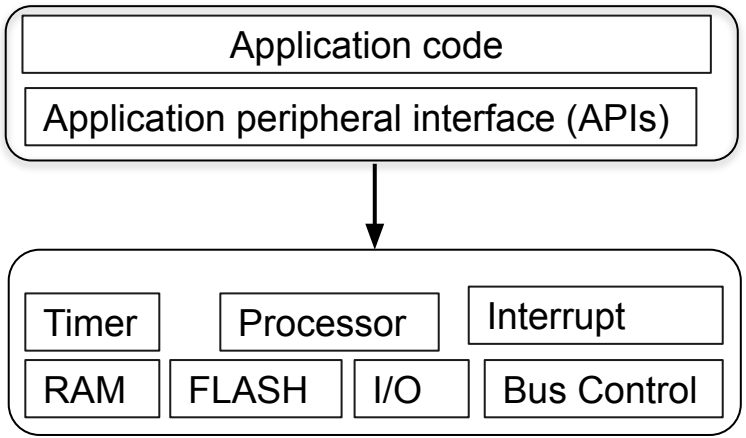
Bare-Metal Firmware



- Application directly talk to hardware
- SDK's released by the vendor
- Toolchains released by vendors
- OS - FreeRTOS, mbed-os
- Popular generic framework have HAL layer - mbed-os, etc.



Bare Metal Firmware



Hardware

Memory Mapped IO

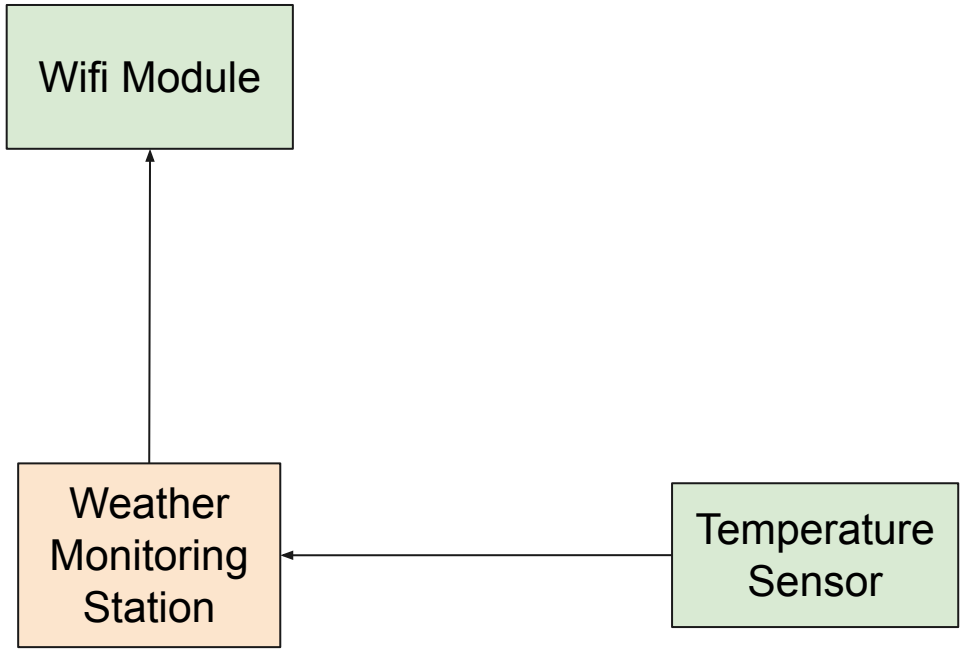
Boundary address	Peripheral	Bus
0x4004 0000 - 0x4007 FFFF	USB OTG HS	AHB1
0x4002 9000 - 0x4002 93FF	ETHERNET MAC	
0x4002 8C00 - 0x4002 8FFF		
0x4002 8800 - 0x4002 8BFF		
0x4002 8400 - 0x4002 87FF		
0x4002 8000 - 0x4002 83FF		
0x4002 6400 - 0x4002 67FF	DMA2	
0x4002 6000 - 0x4002 63FF	DMA1	
0x4002 4000 - 0x4002 4FFF	BKPSRAM	
0x4002 3C00 - 0x4002 3FFF	Flash interface register	
0x4002 3800 - 0x4002 3BFF	RCC	
0x4002 3000 - 0x4002 33FF	CRC	
0x4002 2000 - 0x4002 23FF	GPIOI	
0x4002 1C00 - 0x4002 1FFF	GPIOH	
0x4002 1800 - 0x4002 1BFF	GPIOG	

Memory Mapped Peripherals

Addr. offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	RCC_CR	Reserved				PLL I2SRDY	PLL I2SON	PLL RDY	PLL ON	Reserved				CSSON	HSEBYP	HSERDY	HSEON	HSICAL 7	HSICAL 6	HSICAL 5	HSICAL 4	HSICAL 3	HSICAL 2	HSICAL 1	HSICAL 0	HSITRIM 4	HSITRIM 3	HSITRIM 2	HSITRIM 1	HSITRIM 0	Reserved	HSIRDY	HSION
0x04	RCC_PLLCFGR	Reserved				PLLQ 3	PLLQ 2	PLLQ 1	PLLQ 0	Reserved	PLLSRC	Reserved				PLLP 1	PLLP 0	Reserved	PLLN 8	PLLN 7	PLLN 6	PLLN 5	PLLN 4	PLLN 3	PLLN 2	PLLM 5	PLLM 0	PLLM 5	PLLM 4	PLLM 3	PLLM 2	PLLM 1	PLLM 0
0x08	RCC_CFGR	MCO2 1	MCO2 0	MCO2PRE2	MCO2PRE1	MCO2PRE0	MCO1PRE2	MCO1PRE1	MCO1PRE0	I2SSRC	MCO1 1	MCO1 0	RTCPRE 4	RTCPRE 3	RTCPRE 2	RTCPRE 1	RTCPRE 0	PPRE2 2	PPRE2 1	PPRE2 0	PPRE1 2	PPRE1 1	PPRE1 0	Reserved		SWS 1	SWS 0	SW 1	SW 0					
0x0C	RCC_CIR	Reserved								CSSC	Reserved	PLLI2SRDYC	PLLRDYC	HSERDYC	HSIRDYC	LSERDYC	LSIRDYC	Reserved		PLLI2SRDYIE	PLLRDYIE	HSERDYIE	HSIRDYIE	LSERDYIE	LSIRDYIE	CSSF	Reserved	PLLI2SRDYF	PLLRDYF	HSERDYF	HSIRDYF	LSERDYF	LSIRDYF
0x10	RCC_AHB1RSTR	Reserved		OTGHSRST	Reserved		ETHMACRST	Reserved		DMA2RST	DMA1RST	Reserved				CRCRST	Reserved			GPIOIRST	GPIOHRST	GPIOGRST	GPIOFRST	GPIOERST	GPIODRST	GPIOCRST	GPIOBRST	GPIOARST					
0x14	RCC_AHB2RSTR	Reserved																							OTGFSRST	RNGRST	HSAHRST	CRYPRST	Reserved			DCMIRST	
0x18	RCC_AHB3RSTR	Reserved																															FSMCRST

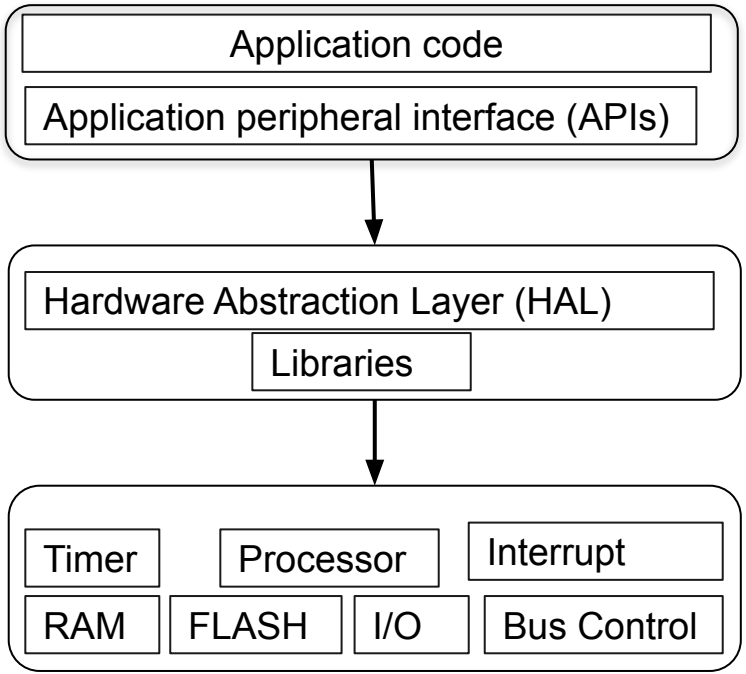
Peripheral Register Mapping

src : STM32F4x Reference Manual

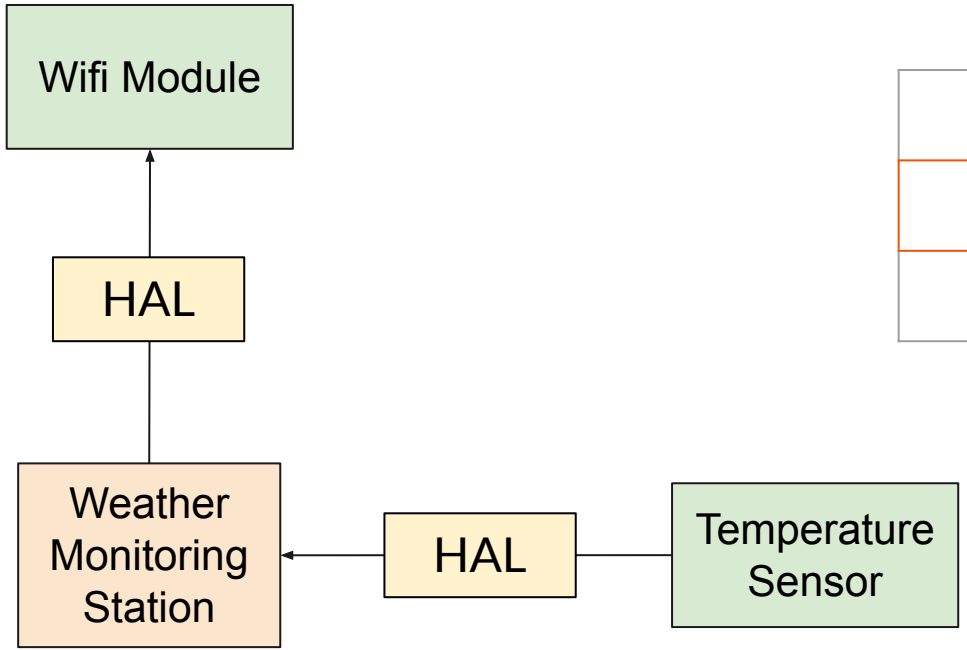


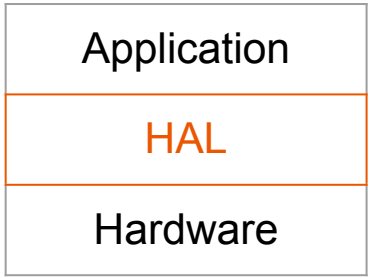
Weather Monitoring System

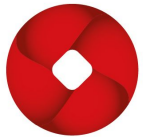
Bare Metal Firmware



Hardware

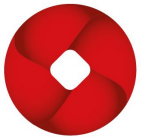






Weather Monitoring System

Application Peripheral Interface (APIs)



Functions

void **usart_set_baudrate** (uint32_t usart, uint32_t baud)
USART Set Baudrate. More...

void **usart_set_databits** (uint32_t usart, uint32_t bits)
USART Set Word Length. More...

void **usart_set_stopbits** (uint32_t usart, uint32_t stopbits)
USART Set Stop Bit(s). More...

void **usart_set_parity** (uint32_t usart, uint32_t parity)
USART Set Parity. More...

void **usart_set_mode** (uint32_t usart, uint32_t mode)
USART Set Rx/Tx Mode. More...

void **usart_set_flow_control** (uint32_t usart, uint32_t flowcontrol)
USART Set Hardware Flow Control. More...

void **usart_enable** (uint32_t usart)
USART Enable. More...

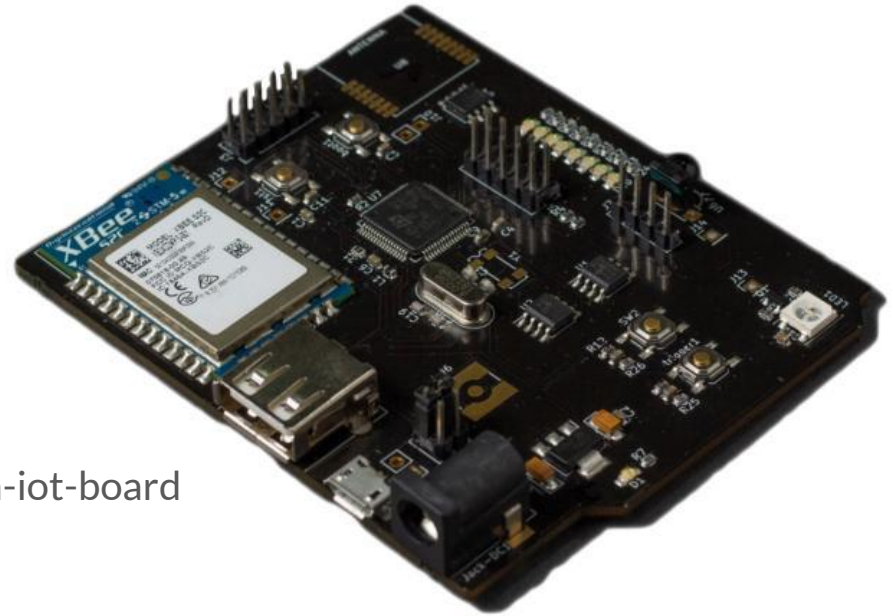
void **usart_disable** (uint32_t usart)
USART Disable. More...

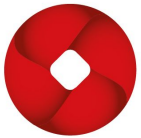
void **usart_send_blocking** (uint32_t usart, uint16_t data)
USART Send Data Word with Blocking. More...

src - http://libopencm3.org/

Target Device

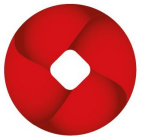
- Name : DIVA (vulnerable board)
- Processor : ARM
- Chip ID : Stm32f411
- Variant : Cortex / Thumb little
- Bits : 32
- Endian: little
- More info : https://expliot.io/products/diva-iot-board





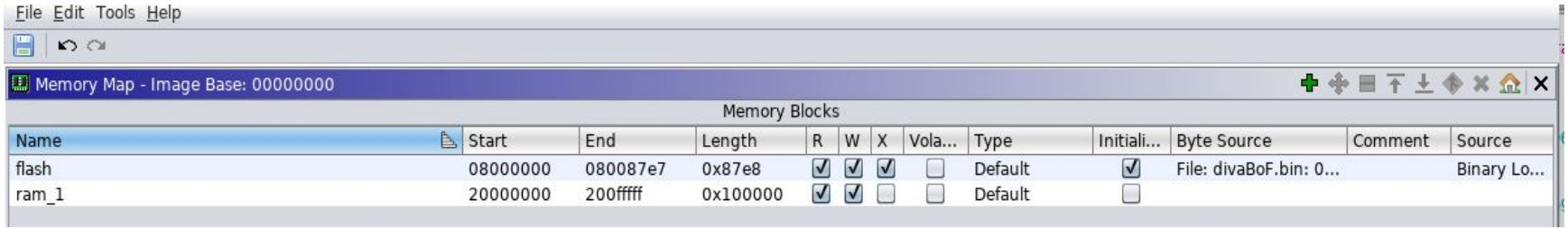
Static Analysis

- Gaining understanding of the software by reading code
- Program is not executed
- You can explore any part of program you want
- Finding patterns in code
- Time consuming
- Tools
    - Disassembler - Ghidra, IDA pro, radare2, Hopper, Capstone
    - Decompile - Hex-Ray Decompiler with IDA pro, Ghidra

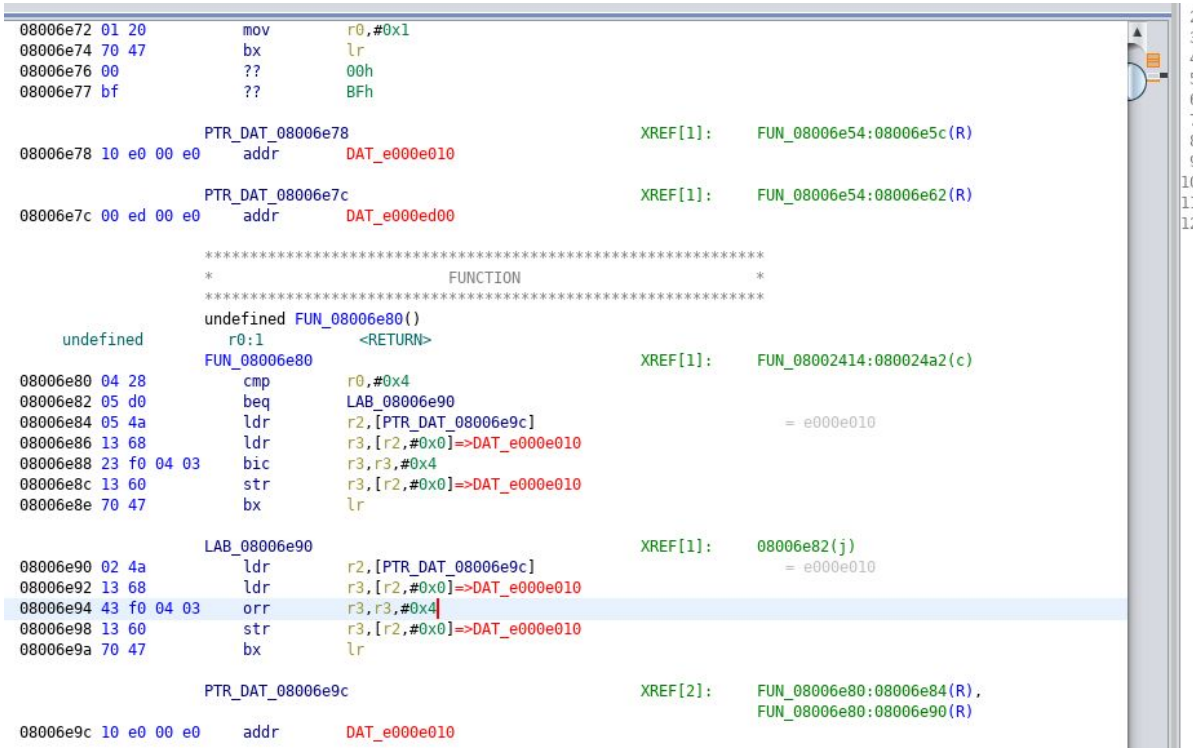


Ghidra

- Define Memory Mapping
- Decompiled code
- Create Peripheral Register Mapping Structures



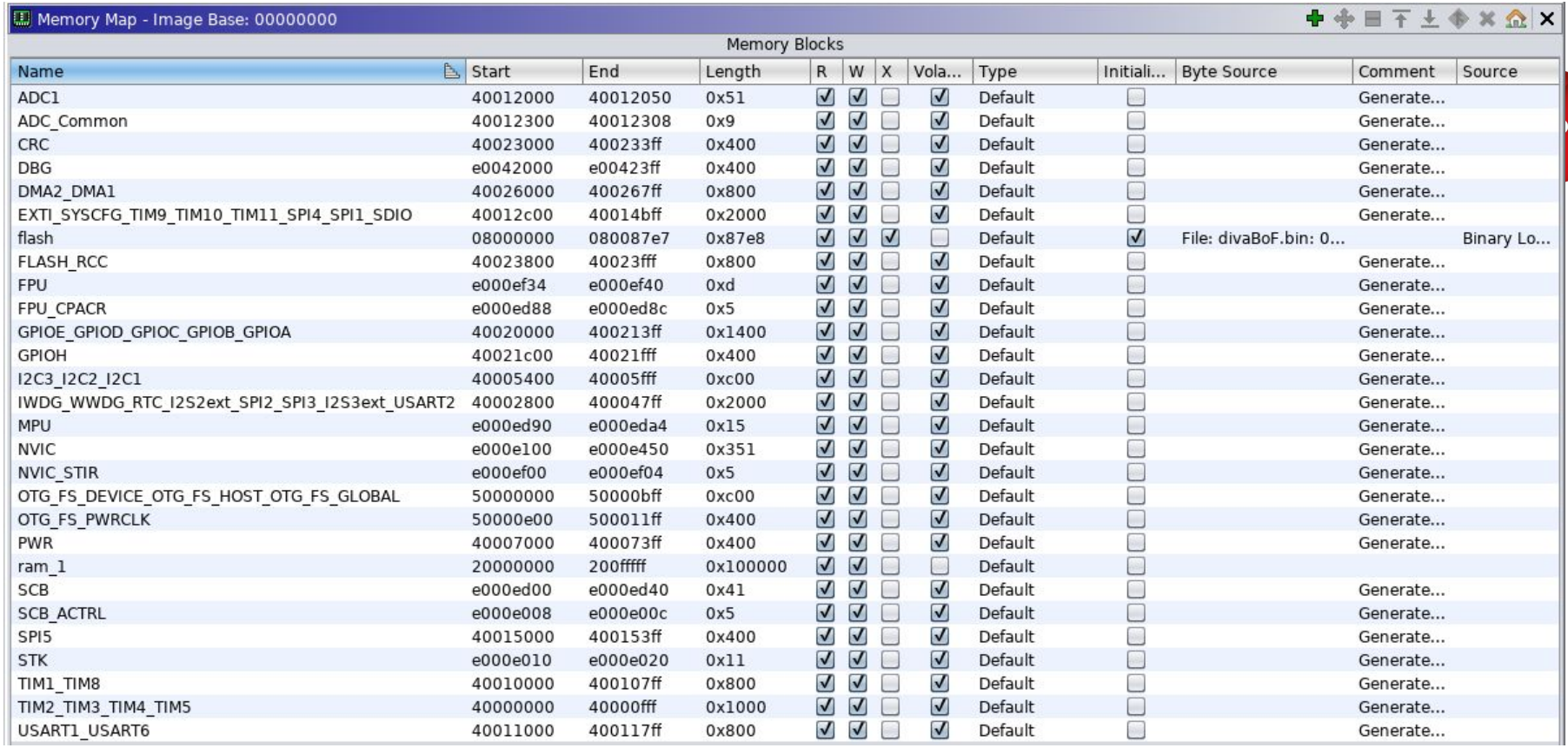
Memory Segments



```
08006e72 01 20          mov         r0,#0x1
08006e74 70 47          bx          lr
08006e76 00             ??          00h
08006e77 bf             ??          BFh

                    PTR_DAT_08006e78                                XREF[1]:     FUN_08006e54:08006e5c(R)
08006e78 10 e0 00 e0    addr        DAT_e000e010

                    PTR_DAT_08006e7c                                XREF[1]:     FUN_08006e54:08006e62(R)
08006e7c 00 ed 00 e0    addr        DAT_e000ed00

                    *************************************************************
                    *                          FUNCTION                         *
                    *************************************************************
                    undefined FUN_08006e80()
    undefined         r0:1          <RETURN>
                    FUN_08006e80                                    XREF[1]:     FUN_08002414:080024a2(c)
08006e80 04 28          cmp         r0,#0x4
08006e82 05 d0          beq         LAB_08006e90
08006e84 05 4a          ldr         r2,[PTR_DAT_08006e9c]                        = e000e010
08006e86 13 68          ldr         r3,[r2,#0x0]=>DAT_e000e010
08006e88 23 f0 04 03    bic         r3,r3,#0x4
08006e8c 13 60          str         r3,[r2,#0x0]=>DAT_e000e010
08006e8e 70 47          bx          lr

                    LAB_08006e90                                    XREF[1]:     08006e82(j)
08006e90 02 4a          ldr         r2,[PTR_DAT_08006e9c]                        = e000e010
08006e92 13 68          ldr         r3,[r2,#0x0]=>DAT_e000e010
08006e94 43 f0 04 03    orr         r3,r3,#0x4
08006e98 13 60          str         r3,[r2,#0x0]=>DAT_e000e010
08006e9a 70 47          bx          lr

                    PTR_DAT_08006e9c                                XREF[2]:     FUN_08006e80:08006e84(R),
                                                                                 FUN_08006e80:08006e90(R)
08006e9c 10 e0 00 e0    addr        DAT_e000e010
```

```
2  void FUN_08006e80(int param_1)
3
4  {
5    if (param_1 != 4) {
6      *(uint *)PTR_DAT_08006e9c = *(uint *)PTR_DAT_08006e9c & 0xfffffffb;
7      return;
8    }
9    *(uint *)PTR_DAT_08006e9c = *(uint *)PTR_DAT_08006e9c | 4;
10   return;
11 }
12
```

Static Analysis of firmware binary



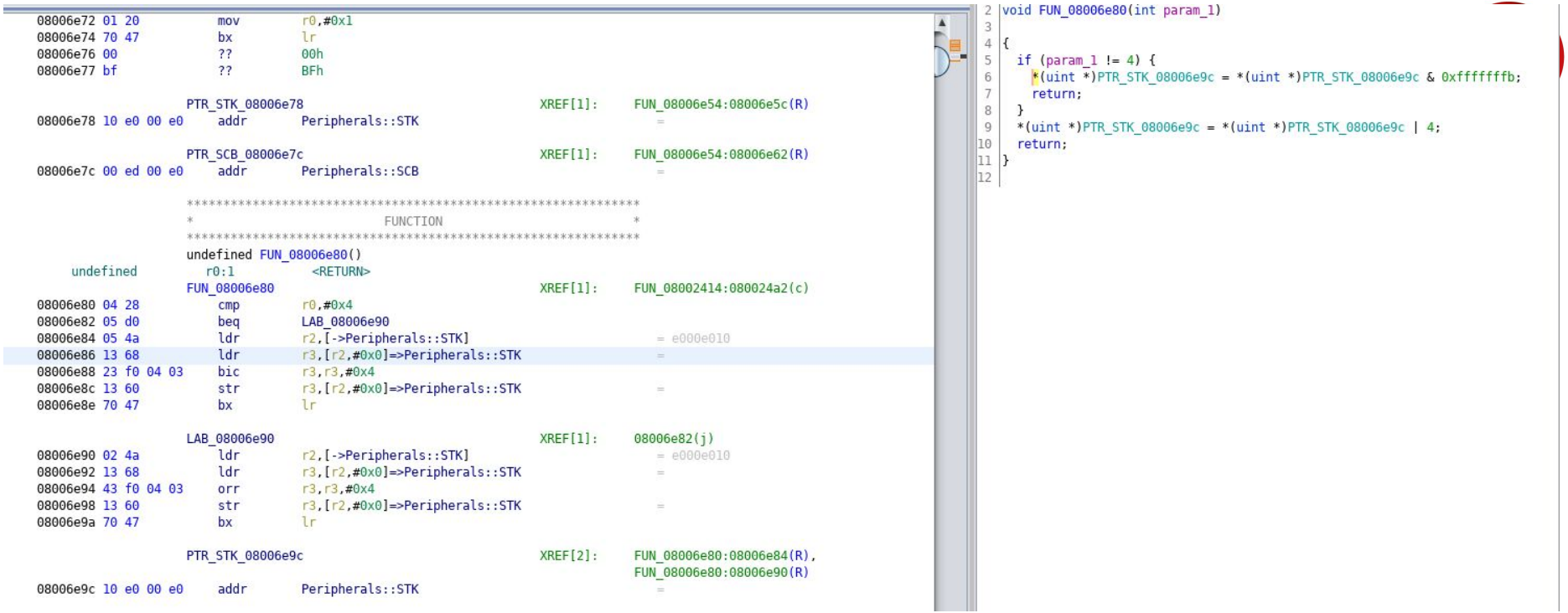
Memory Map - Image Base: 00000000

Memory Blocks

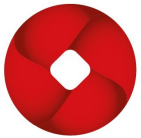
Name	Start	End	Length	R	W	X	Vola...	Type	Initiali...	Byte Source	Comment	Source
ADC1	40012000	40012050	0x51	☑	☑	☐	☑	Default	☐		Generate...	
ADC_Common	40012300	40012308	0x9	☑	☑	☐	☑	Default	☐		Generate...	
CRC	40023000	400233ff	0x400	☑	☑	☐	☑	Default	☐		Generate...	
DBG	e0042000	e00423ff	0x400	☑	☑	☐	☑	Default	☐		Generate...	
DMA2_DMA1	40026000	400267ff	0x800	☑	☑	☐	☑	Default	☐		Generate...	
EXTI_SYSCFG_TIM9_TIM10_TIM11_SPI4_SPI1_SDIO	40012c00	40014bff	0x2000	☑	☑	☐	☑	Default	☐		Generate...	
flash	08000000	080087e7	0x87e8	☑	☑	☑	☐	Default	☑	File: divaBoF.bin: 0...		Binary Lo...
FLASH_RCC	40023800	40023fff	0x800	☑	☑	☐	☑	Default	☐		Generate...	
FPU	e000ef34	e000ef40	0xd	☑	☑	☐	☑	Default	☐		Generate...	
FPU_CPACR	e000ed88	e000ed8c	0x5	☑	☑	☐	☑	Default	☐		Generate...	
GPIOE_GPIOD_GPIOC_GPIOB_GPIOA	40020000	400213ff	0x1400	☑	☑	☐	☑	Default	☐		Generate...	
GPIOH	40021c00	40021fff	0x400	☑	☑	☐	☑	Default	☐		Generate...	
I2C3_I2C2_I2C1	40005400	40005fff	0xc00	☑	☑	☐	☑	Default	☐		Generate...	
IWDG_WWDG_RTC_I2S2ext_SPI2_SPI3_I2S3ext_USART2	40002800	400047ff	0x2000	☑	☑	☐	☑	Default	☐		Generate...	
MPU	e000ed90	e000eda4	0x15	☑	☑	☐	☑	Default	☐		Generate...	
NVIC	e000e100	e000e450	0x351	☑	☑	☐	☑	Default	☐		Generate...	
NVIC_STIR	e000ef00	e000ef04	0x5	☑	☑	☐	☑	Default	☐		Generate...	
OTG_FS_DEVICE_OTG_FS_HOST_OTG_FS_GLOBAL	50000000	50000bff	0xc00	☑	☑	☐	☑	Default	☐		Generate...	
OTG_FS_PWRCLK	50000e00	500011ff	0x400	☑	☑	☐	☑	Default	☐		Generate...	
PWR	40007000	400073ff	0x400	☑	☑	☐	☑	Default	☐		Generate...	
ram_1	20000000	200fffff	0x100000	☑	☑	☐	☐	Default	☐			
SCB	e000ed00	e000ed40	0x41	☑	☑	☐	☑	Default	☐		Generate...	
SCB_ACTRL	e000e008	e000e00c	0x5	☑	☑	☐	☑	Default	☐		Generate...	
SPI5	40015000	400153ff	0x400	☑	☑	☐	☑	Default	☐		Generate...	
STK	e000e010	e000e020	0x11	☑	☑	☐	☑	Default	☐		Generate...	
TIM1_TIM8	40010000	400107ff	0x800	☑	☑	☐	☑	Default	☐		Generate...	
TIM2_TIM3_TIM4_TIM5	40000000	40000fff	0x1000	☑	☑	☐	☑	Default	☐		Generate...	
USART1_USART6	40011000	400117ff	0x800	☑	☑	☐	☑	Default	☐		Generate...	

Populated Memory Mapping

Ghidra Plugin :
https://github.com/leveldown-security/SVD-Loader-Ghidra

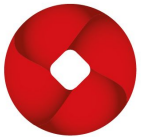


Memory Mapping can Import the code Readability



Dynamic Analysis

- Gaining understanding of the program by executing it
- Not all part of the program can be executed.
- Will only give you information about the path of program which is executed.
- Tools
  - OpenOCD
  - GDB

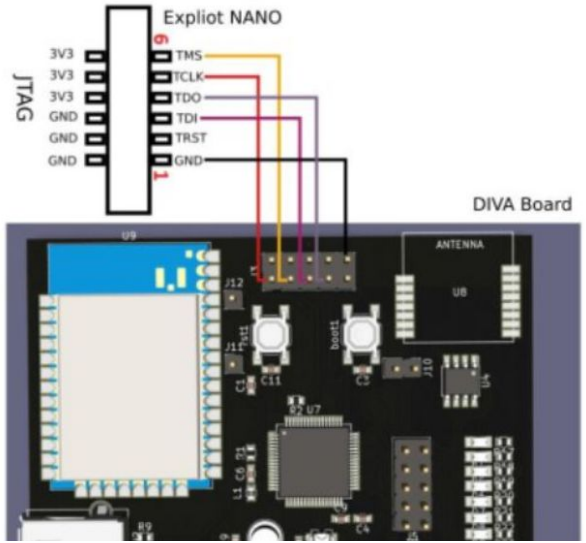


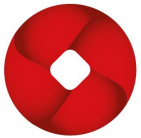
What do you need?

- Core family
- Hardware architecture
- Instruction set
- Peripheral Mapping

Device Connection

PIN DESCRIPTION	Arduio Nano	DIVA PIN (J3 Header)	Expliot Nano
TCLK	D2	1	5
TMS	D3	3	6
TDI	D4	5	3
TDO	D5	7	4
GND	GND	10	1





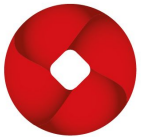
Lab : Activate a Peripheral

Objective

Using stack overflow vulnerability activate LED connected to **GPIOC**

Location

lab-shell



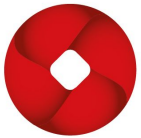
Lab : Activate a Peripheral

Objective

Using stack overflow vulnerability activate LED connected to **GPIOC**

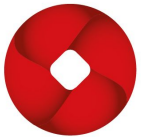
Hint

GPIOC is located a address 0x40020814



Shellcode

```
mov    r5, #0xFF
movw  r6, #0x0814
movt   r6, #0x4002
str      r5, [r6]
```



Exploit Mitigations

- ASLR
- Non-Executable Stack (ESP)
- Stack Canaries
- Some Micro-Controller have MPU and MMU which have page permissions

Thank You