物联网设备固件漏洞利用与攻击缓解

实验背景

近年来， 物联网产业迅速发展， 物联网设备已经广泛应用于人们日常的生产生活中，如智能家居设备、 工业控制设备以及无人机等。根据最新统计数据[1]，截至 2021 年底，仅在线的物联网设备就突破了 123 亿，超过了传统联网设备（台式机、笔记本和手机）。然而，物联网设备软件与系统（即固件）大多由不安全的低级语言编写， 并且其所运行在的微控制器硬件资源十分受限，对功耗、时延还有着严格的要求， 因此难以直接在设备上部署防御监控和攻击缓解措施， 如进程隔离，空间地址随机化等， 这就导致了物联网设备固件中存在大量易被攻击者利用的软件漏洞， 这些漏洞不仅会对物联网产品造成破坏，甚至对用户、 社会乃至国家造成严重危害。 例如， FDA 发现市场上的心脏起搏器及其他智能医疗设备普遍存在严重的安全漏洞，这些漏洞一旦被攻击者恶意利用，将直接威胁用户的生命安全。近年来席卷全球的大规模物联网僵尸网络和勒索病毒也正是利用了物联网设备的脆弱性。

实验目的：

面对层出不穷的物联网设备漏洞与攻击事件，作为网络空间安全专业的学生了解和掌握常见的物联网设备漏洞发现与利用方法以及利用常见的硬件特性设计简单的系统防御措施显得十分重要。

实验准备：

实验工具：

物联网设备固件逆向工具：IDA Pro

物联网设备固件开发工具：Keil （资料已上传 超星平台）

安装与使用方法参见：https://jingyan.baidu.com/article/20b68a8877403c386cec629f.html

嵌入式设备模拟器：QEMU 7.0.0

注意：仅安装 qemu-system-arm 即可 源码：https://www.qemu.org/

配置命令：

./configure --prefix=/root/Downloads/qemu-7.0.0-rc0/build --target-list=arm-softmmu --enable-debug

Make

Make install

参考学习手册：ARM Cortex-M3 M4 权威指南（资料已上传超星平台）

IDA Pro 权威指南 2

FreeRTOS 源码讲解与应用开发

背景知识

1. ARM 架构基础 权威指南第四章
   a) 操作模式和状态
   b) 寄存器
   c) 存储系统

d) 异常和中断
e) 系统复位流程

2. ARM 指令集 权威指南第五章
   a) 存储器访问指令
3. ARM 存储器内存保护单元 第九章

实验内容与分数占比：

1. 题号 1：裸机物联网设备溢出漏洞利用 30%
2. 基于 MPU 的物联网设备攻击缓解技术
   a) 题号 2 解除防止代码注入区域保护 15%
   b) 题号 3 解除指定外设区域保护 15%
3. FreeRTOS-MPU 保护绕过
   a) 题号 4 编写 C 代码实现基于 FreeRTOS-MPU v10.4 的提权和指定函数查找 20%
   b) 题号 5 利用溢出漏洞实现在 FreeRTOS MPU V10.4 版本的系统提权和 Flag 函数打印 20%
4. 附加思考题：(不作为本实验强制要求，实验报告中回答的同学每个小题加一分)
   a) 如何利用溢出漏洞实现在 FreeRTOS MPU V10.5 版本的系统提权和 Flag 函数打印？
   b) 安全的物联网实时操作系统的系统调用实现方式？
   c) 基于 ARMv7m 架构谈一谈一个函数中存在缓冲区溢出是否就一定可以被利用呢？
   d) 能否用动态调试的方法从内存中获取 flag？

注：Flag 提交和实验报告各占比百分之五十。但实验报告和 flag 提交要对应，如果 flag 提交了但实验报告没有对应题目的解析或者完全不对那么这道题无法得分。

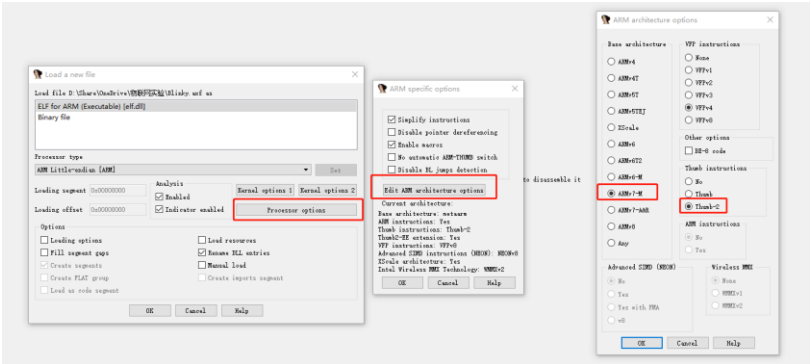
实验 1 裸机物联网设备溢出漏洞利用

实验要求：

找到固件中可以溢出的缓冲区 buffer 和目标 flag 函数，利用溢出攻击覆盖返回地址打印 Flag 函数。

实验步骤：

1. 选择 ARM v7m 架构利用 IDA Pro 根据尾号正确加载实验 1 所需固件 task1_XX.elf



2. 通过逆向固件找到可以溢出的缓冲区
3. 找到打印 flag 的函数地址(包含了 flag 字符)

4. 构造 shellcode 使其溢出返回地址到 Flag 函数
5. 在 Qemu 模拟执行固件，输入学号后四位然后输入 shellcode 填充字符长度和返回地址为 Flag 函数从而获取 flag.

   注意：先填写学号后四位（尽量快一些，否则会出错），填写填充长度（两位十进制数字），最后填写八个字符的覆盖的十六进制函数地址字母小写。填充长度会回显，地址不会回显。

模拟执行指令：./qemu-system-arm -M **netduinoplus2** -cpu cortex-m4 -m 1M -nographic -d in_asm,nochain -kernel task1_xx.elf -D log.txt

实验 1 在报告中提交的内容包括：

1. IDA 加载过程测试固件的过程
2. 存在溢出缓冲区函数和 Flag 函数截图以及发现步骤
3. 栈原始布局和溢出示意图以及原理比如长度和返回地址要标注
4. 模拟执行获取 flag 的截图

实验 2 基于 MPU 的物联网设备攻击缓解技术

实验要求：

实验的 lib 库代码中开启了多项 MPU 区域保护

请根据不同的子任务重配置 MPU 区域解除相应的区域保护

子任务 1 解除代码段不可写入保护将指定代码段改为可写

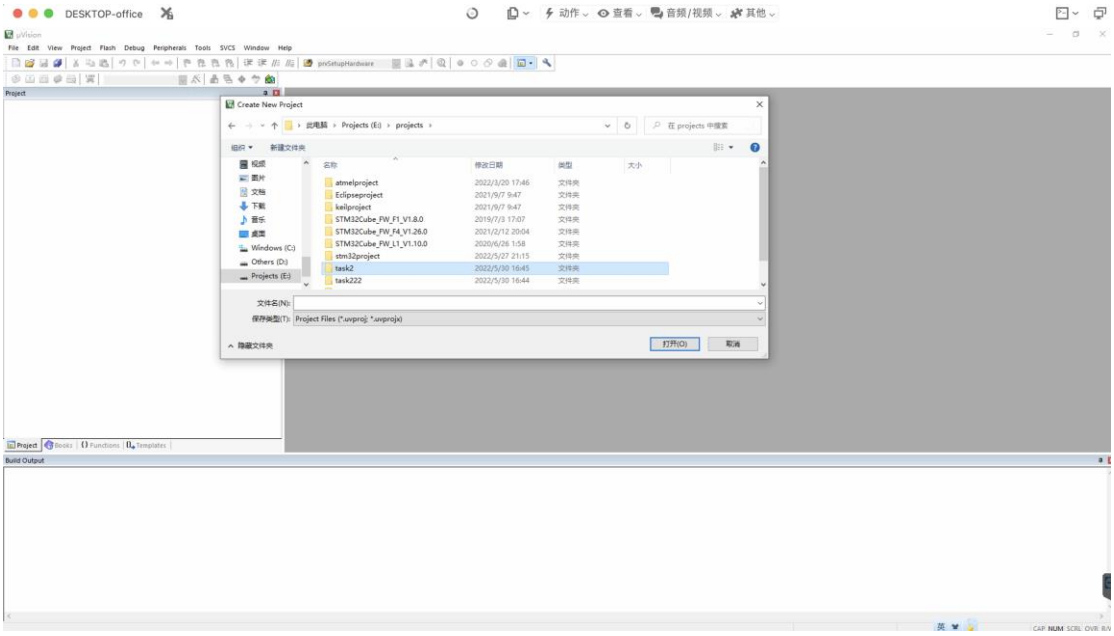
子任务 2 解除特定外设区域保护将指定外设区域改为可读写

注：请通过逆向分析出触发 Memfault 的指令，从而在满足任务要求的同时尽可能用**最少的 MPU 区域配置、最小的 MPU 区域范围，最小的特权级**。

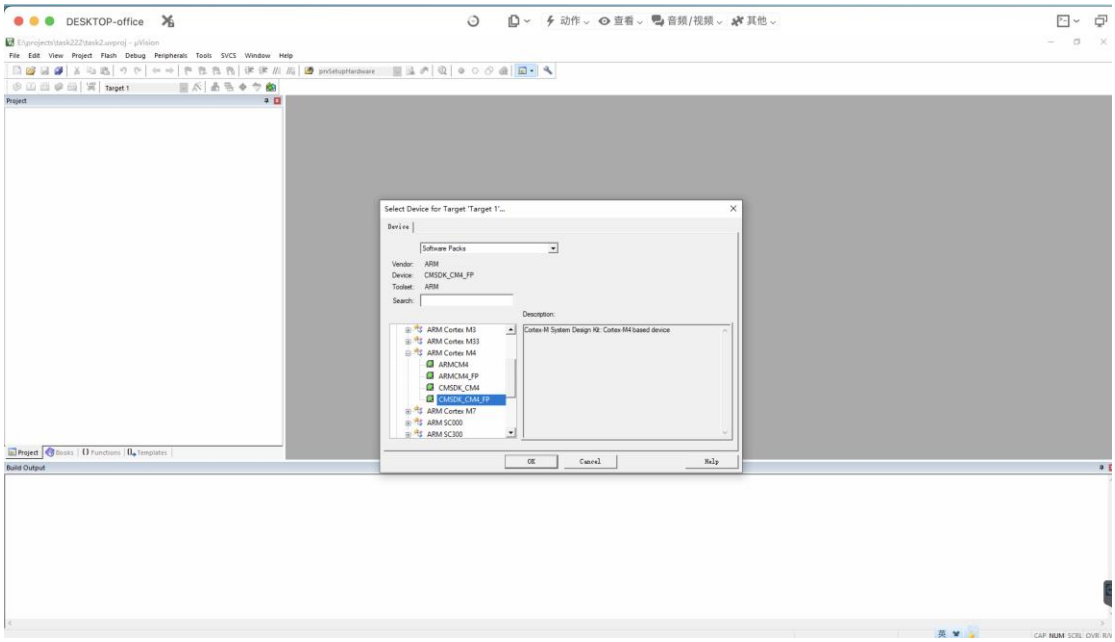
特权级排序：rwx(可读可写可执行)> rw>rx>r>无权限

实验步骤：

1. 打开 keil 创建新工程



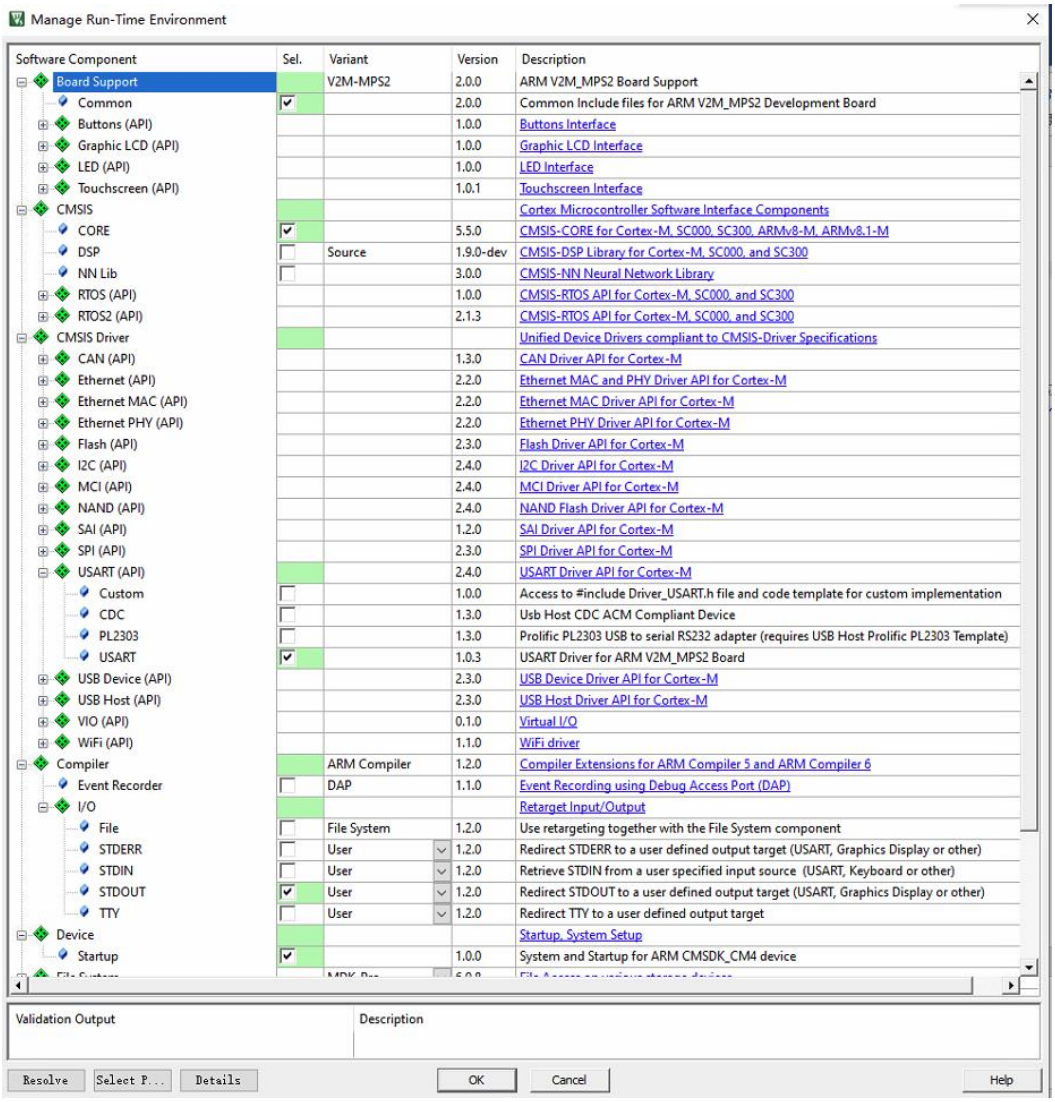
2. CPU 和 MCU 选型



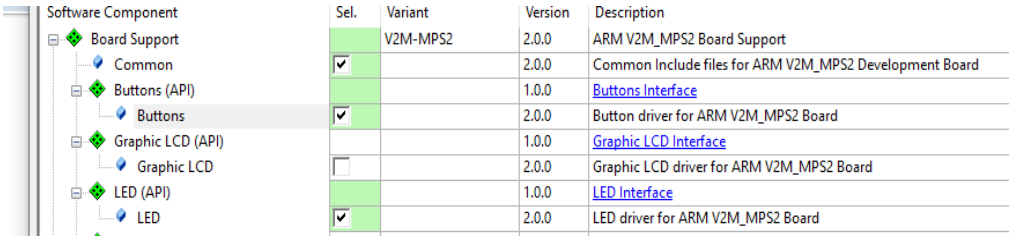
注：有的同学可能安装以后发现没有这个 cpu 型号，请下载我上传在超星平台的两个 pack。安装自动导入 keil。

选在板卡以后，右边点击安装。

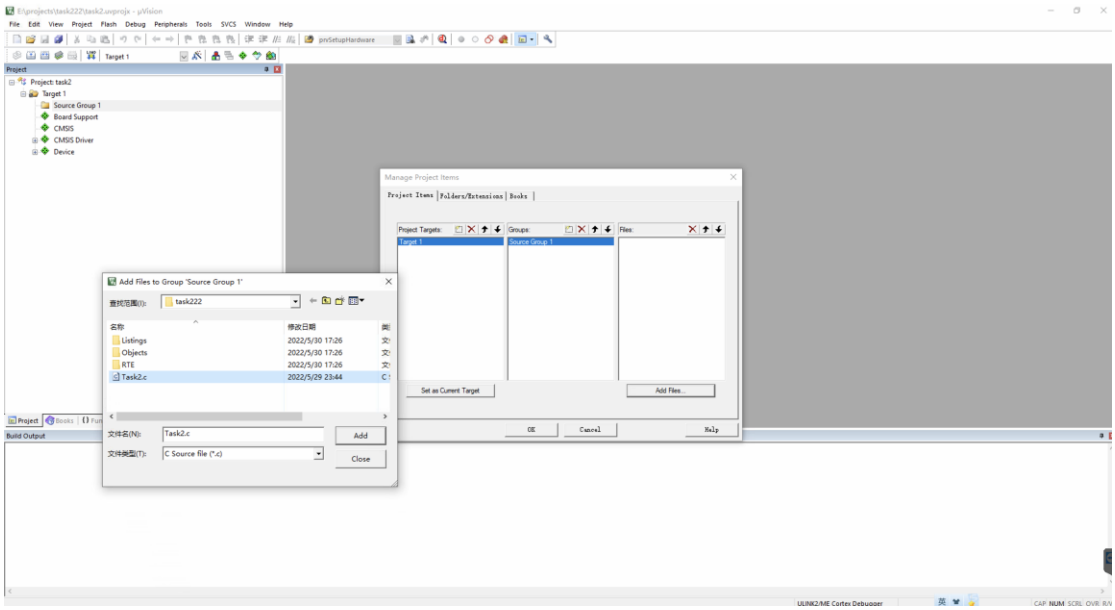
3. Keil 如下图配置必要环境和驱动程序



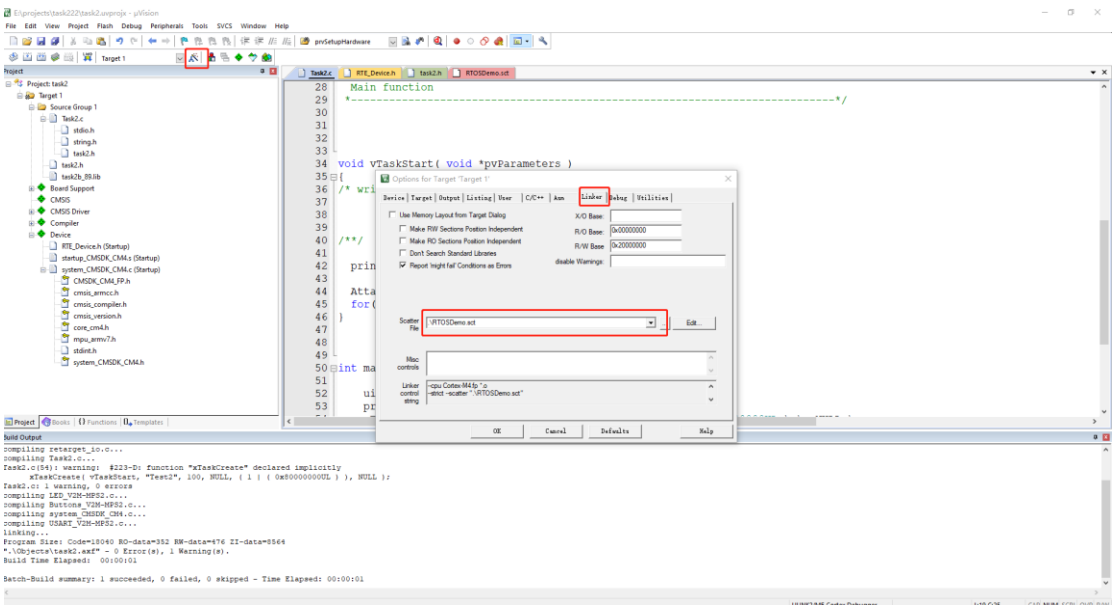
子任务 2 需要再增加勾选以下两个外设



4. 将模板代码 task2.c、task2.h 和 task2a_.xxdll/task2b_xxdll 添加到工程中（超星平台）

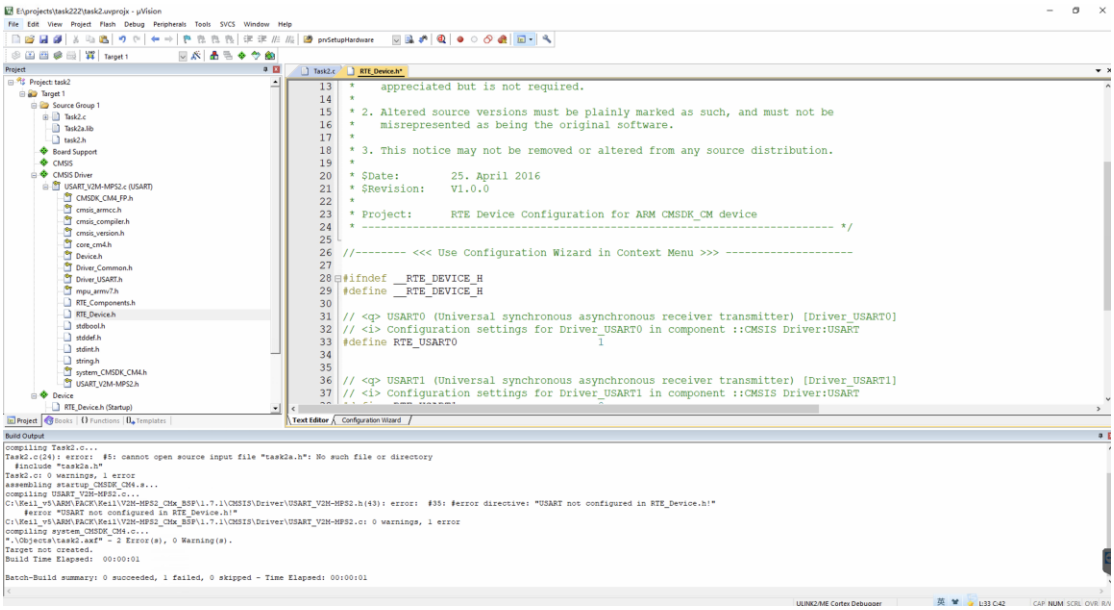


注：有些同学可能默认链接布局有问题，建议可以选用超星平台的 FreeRTOS MPU 链接脚本文件 sct，导入方式如下截图：

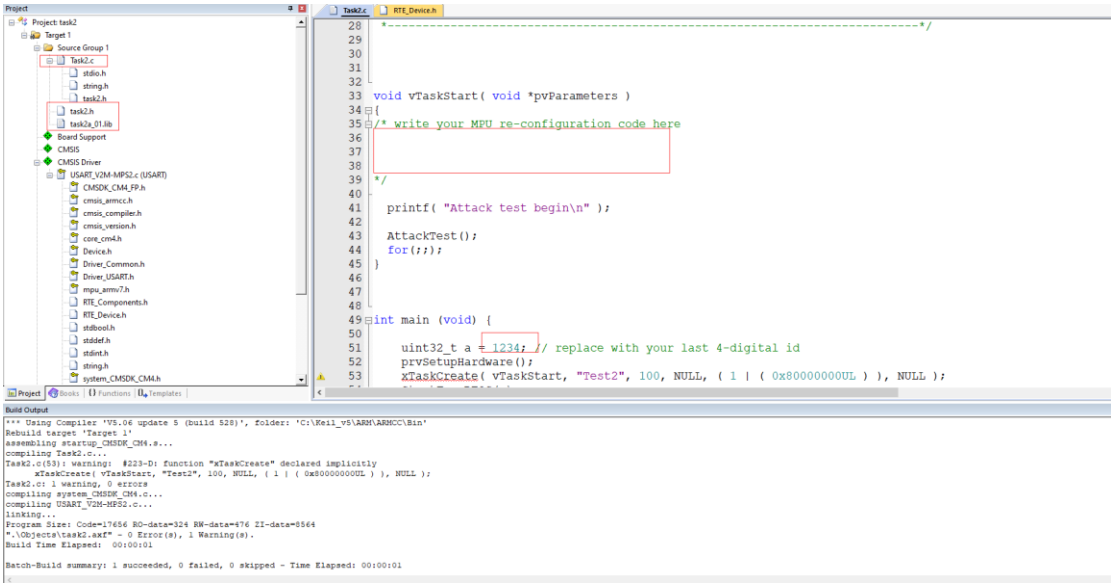


注意：有的同学是灰色的不能点，是因为没有取消勾选 use memory layout from target dialog 选项。

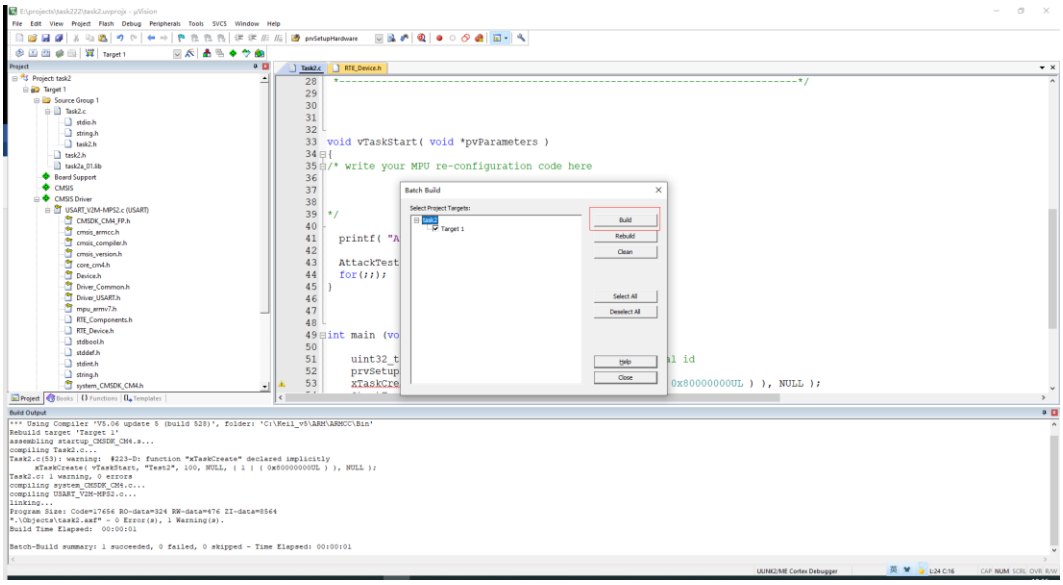
5. 编译后会提示我们用了 uart 但没配置用了哪个 UART 修改 RTE_Device.h 文件 UART0 为 1



6. 通过对 AttackTest 函数和设置 MPU 函数以及动态执行的 log 日志分析导致触发 memfault 的原因，在注释范围内根据子任务要求填写学号和 MPU 区域配置代码参考权威指南 MPU 区域寄存器使用规则，用最少的 MPU 区域、最小特权和最小地址范围来解除相应的内存保护。



7. 编译 build 生成可执行文件



注意：MPU 寄存器配置时的默认值

表 11.10　微控制器中常用的存储器属性

类型	存储器类型	常用的存储器属性
ROM，Flash(可编程存储器)	普通存储器	不可共用，写通 C=1，B=0，TEX=0，S=0
内部 SRAM	普通存储器	可共用，写通 C=1，B=0，TEX=0，S=1
外部 RAM	普通存储器	可共用，写回 C=1，B=1，TEX=0，S=1
外设	设备	可共用，设备 C=0，B=1，TEX=0，S=1

8. 在 qemu 模拟执行可执行文件获得 flag

模拟执行指令：./qemu-system-arm -M mps2-an386 -cpu cortex-m4 -m 16M -nographic -d in_asm,nochain -kernel task2.axf -D log.txt

实验 2 在报告中需要提交的内容包括：

1. 逆向分析 AttackTest 函数中违反 MPU 区域保护的指令，和对应违反的权限。
2. 用于解除 attackTest 函数违反规则的新 MPU 区域重配置代码和对应设置的区域说明，区域数目越少、地址范围越小、权限越小得分越高。
   区域数目不是最少的扣 2 分、地址范围不是最少的扣 3 分、权限不是最小的扣 2 分。
3. 附加内容：找到固件中所有 MPU 区域配置，并画出固件 MPU 配置示意图（此项不做硬性要求，写出部分 MPU 区域配置的同学额外加 1 分，写全的额外加 2 分）

实验 3 FreeRTOS-MPU 保护绕过

实验要求：

FreeRTOS 提供了基于 MPU 的内存保护方案，本实验中其设置 MPU 布局如下表所示：

区域编号	地址范围	访问属性
0	代码段	只读，可执行
1	0x00-0x00008000 RTOS kernel API	特权只读，可执行， 非特权不可访问
2	SysCall 系统调用段	全部只读可执行
3	0x2000000-0x20005000 RTOS kernel data	特权读写，可执行 非特权不可访问
4	任务栈地址范围	非特权级读写，不 可执行
5-7	自定义区域1	自定义

a) 编写 C 代码实现基于 FreeRTOS-MPU v10.4 的提权代码和指定函数查找
b) 利用溢出漏洞实现在 FreeRTOS MPU V10.4 版本的系统提权和 Flag 函数打印

子任务 1 实验步骤：

1. keil 新建工程按实验 2 相同的方法导入 task3.lib task3.h 和 task3a.c 模板
2. 编译生成固件通过逆向分析找用于提权的函数和打印 Flag 的特权函数
3. 将提权函数和打印 Flag 的特权函数填入 task3a.c 注释范围内容
4. 重新编译固件
5. 在 qemu 模拟执行固件，输入 shellcode 填充字符长度和返回地址获取 Flag

模拟执行指令：./qemu-system-arm -M mps2-an386 -cpu cortex-m4 -m 16M -nographic -d in_asm,nochain -kernel task3a.axf -D log.txt

子任务 2 实验步骤

1. 利用 IDA pro 加载实验的 task3b_xx.axf 文件（参考实验 1）
2. 找到可以用于提权和 Flag 打印的函数或指令地址
3. 找到利用的溢出缓冲区
4. 模拟执行 task3b_xx.axf 文件

模拟执行指令：./qemu-system-arm -M mps2-an386 -cpu cortex-m4 -m 16M -nographic -d in_asm,nochain -kernel task3a.axf -D log.txt

依次填写学号**后四位**、**总 buffer 长度**、从低地址到高地址 buffer 每个 Byte 的十六进制数，注意和第一题不同，每次输入完后请输入回车确认。

```
root@ubuntu:bin# ./qemu-system-arm -M mps2-an386 -cpu cortex-m4 -m 16M -nographi
c -d in_asm,nochain -kernel task3b.axf -D log.txt
input your last 4-digital id, please press 'Enter' to end
id = 1234
input Total buffer length, please press 'Enter' to end
please input your 12-bytes overflow buffer Byte by Byte in hex value, please pre
ss 'Enter' to end once input
a 0 81 4 1 2 3 1b 23 45 1
```

实验 3 在报告中需要提交的内容包括：

1. 打印 Flag 函数名称和地址
2. 用于提权的函数名称和地址
3. 子任务 1 填写的代码和原理
4. 子任务 2 找到存在在溢出的缓冲区、栈示意图、溢出提权和覆盖返回地址的解析过程

5. 模拟运行截图

总体需要提交的实验报告内容：

1. 实验环境
2. 每个实验需要提交的内容
3. 实验心得