

**实验报告**

**物联网设备固件安全实验**

**专业班级： 信安2101班**

**学 号： U2021XXXXX**

**姓 名： MTX**

**指导教师： 周威**

**报告日期： 2023年1月16日**

**网络空间安全学院**

目 录

[实验内容 3](#_Toc156589159)

[实验过程 3](#_Toc156589160)

[一、 实验一 3](#_Toc156589161)

[二、 实验二 6](#_Toc156589162)

[实验心得 9](#_Toc156589163)

[附加题 9](#_Toc156589164)

# 实验内容

1. 物联网设备溢出漏洞利用
2. 利用溢出漏洞绕过FreeRTOS v10.4.4系统保护绕过
3. 利用溢出漏洞绕过FreeRTOS v10.4.5系统保护绕过

# 实验过程

## 实验一

操作系统：windows 11 反汇编工具：IDA\_7.0

1. 按照任务要求选择指定指令集，如图1。

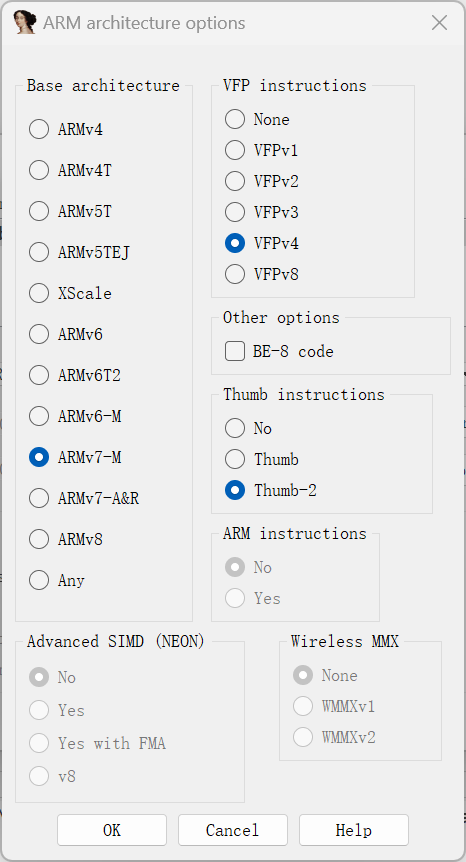


图1 指令集选择界面

1. 打开Task1\_67.axf后，找到main函数，如图2。

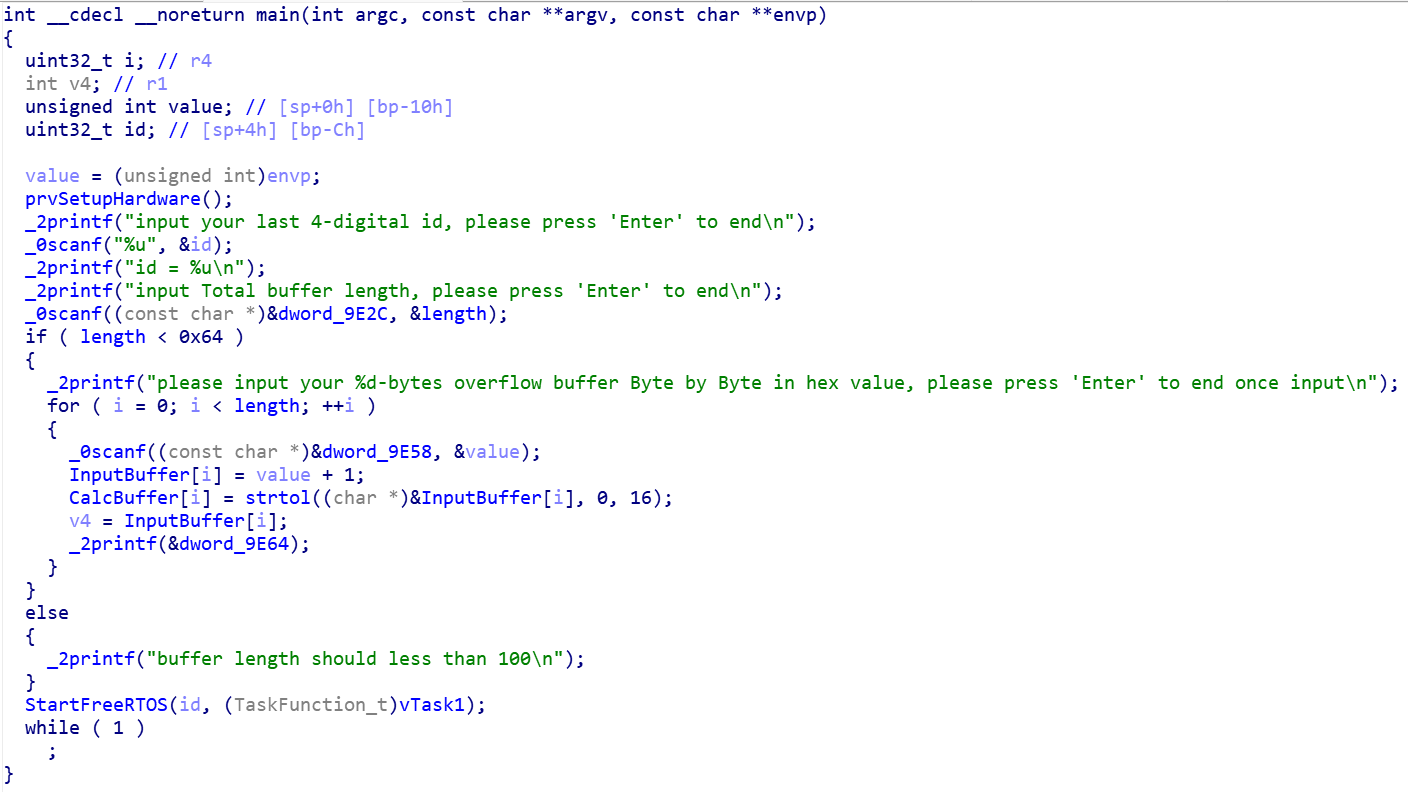


图2 main函数代码

1. 对main函数进行分析，先输入一个4位数的id，然后输入一个整数length，length小于100，然后输入一串字符到InputBuffer，长度为length，**且存入的每个字符的值比输入的大1**。然后先调用vTask1，再调用StartFreeRTOS。
2. 分析vTask1函数，函数如图3。首先对特权级进行了一次判断，然后执行if语句，依次调用Helper与Function，最后调用gcd函数，并输出gcd信息以及攻击失败的信息。据此可推测，若攻击成功，该函数不会执行到调用gcd。

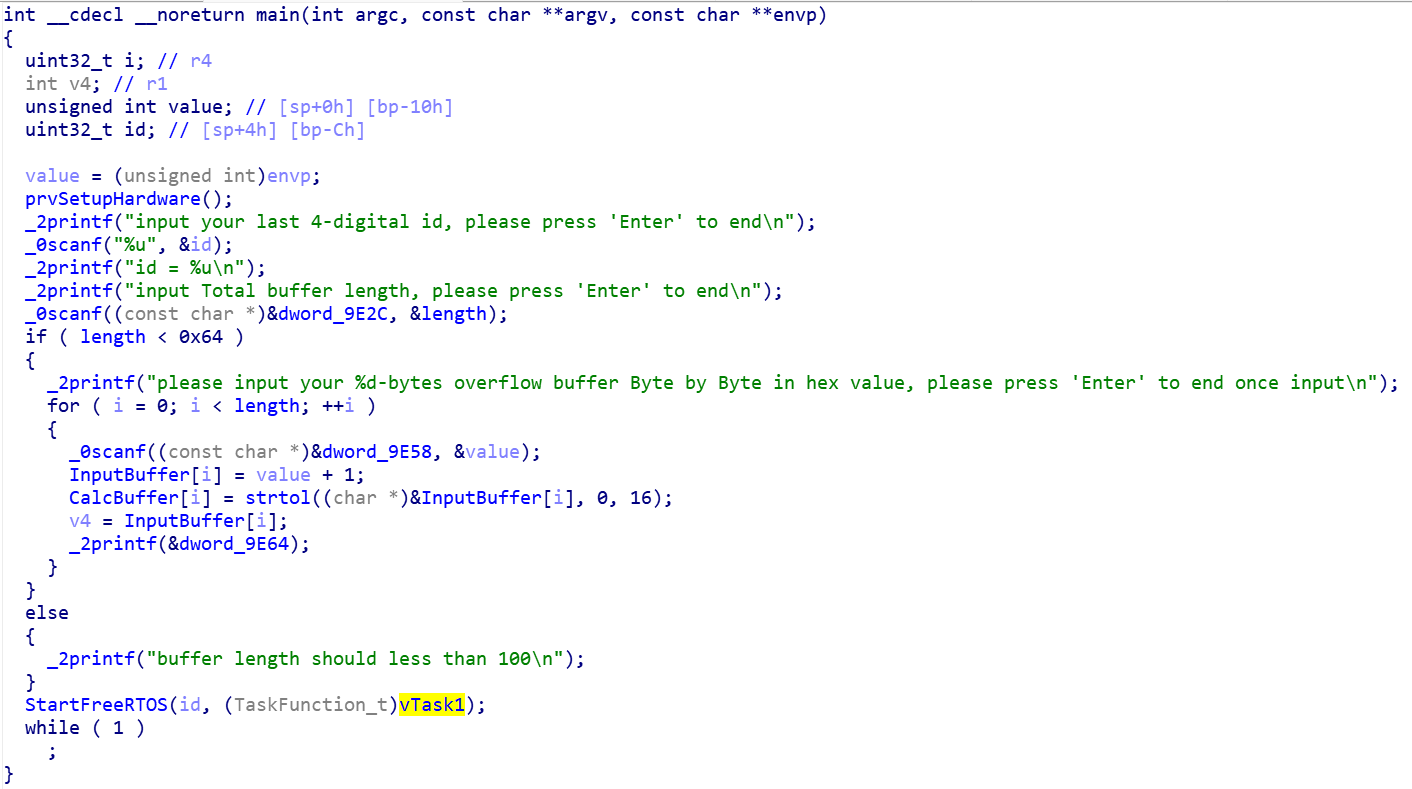


图3 vTask1代码

1. 分析Helper函数，函数如图4。该函数定义了一个16字节大小的缓冲区HelperBuffer，先对其赋了一组初值，然后将InputBuffer中的所有字节复制到该缓冲区中。由于InputBuffer的长度可自定义，且可以超过16字节，因此这里可能发生缓冲区溢出。

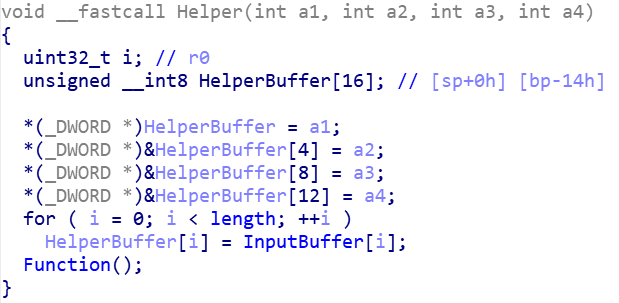


图4 Helper函数代码

1. 对Helper函数进一步分析，查看其汇编代码，如图5。注意到最后一行将栈中数据压出到R0-R3和PC，其中R0-R3对应着a1-a4，而返回地址可以由溢出部分覆盖。至此，发现只需要将打印flag的函数地址对应地覆盖到缓冲区，即可完成实验。栈的表示如图6。

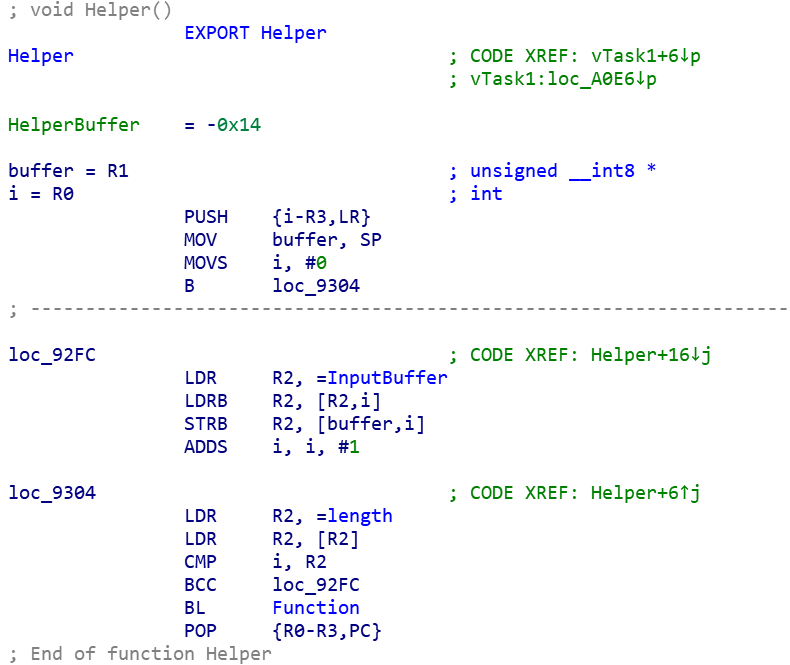


图5 Helper函数汇编代码

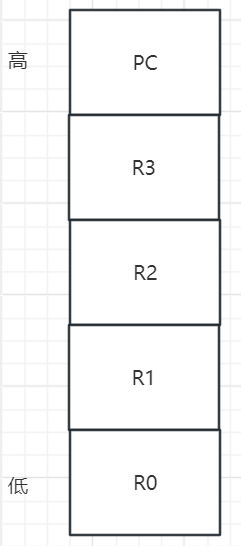


图6 栈的示意图

1. 搜索打印flag的函数：打开strings window，找到含flag的项，如图7。

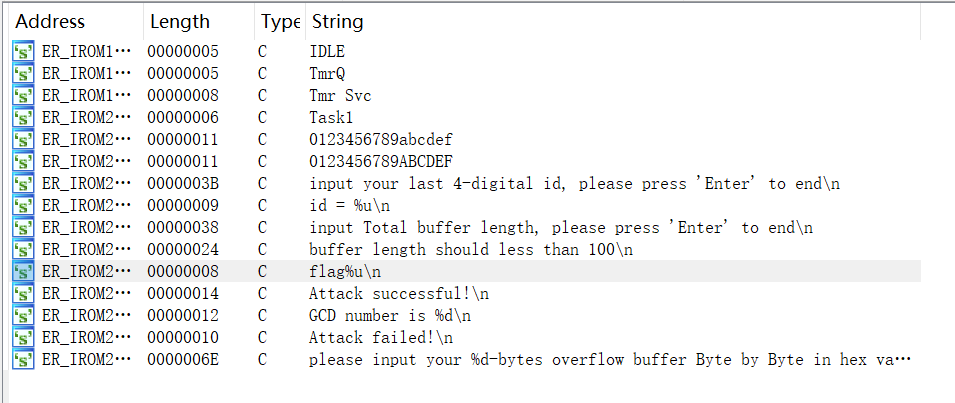


图7 strings window窗口

1. 找到flag的位置，并找到其对应的函数，函数首地址为00009f94，如图8。

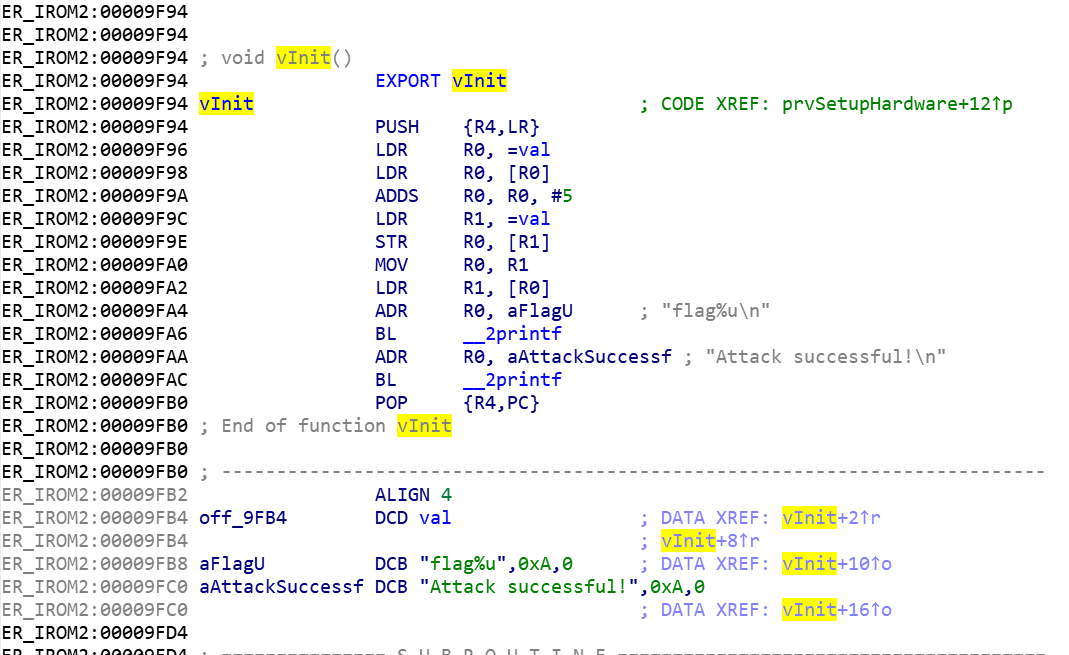


图8 vInit函数

1. 综合考虑以上内容，以及thumb代码函数地址最低位都是1，如图9，可得输入的length为：20，缓冲区数据为：（任意16字节） + 94 9e ff ff。测试结果如图10。

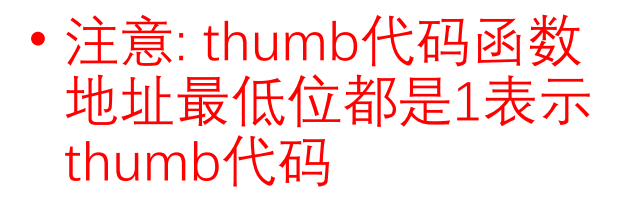


图9 实验讲解内容截图

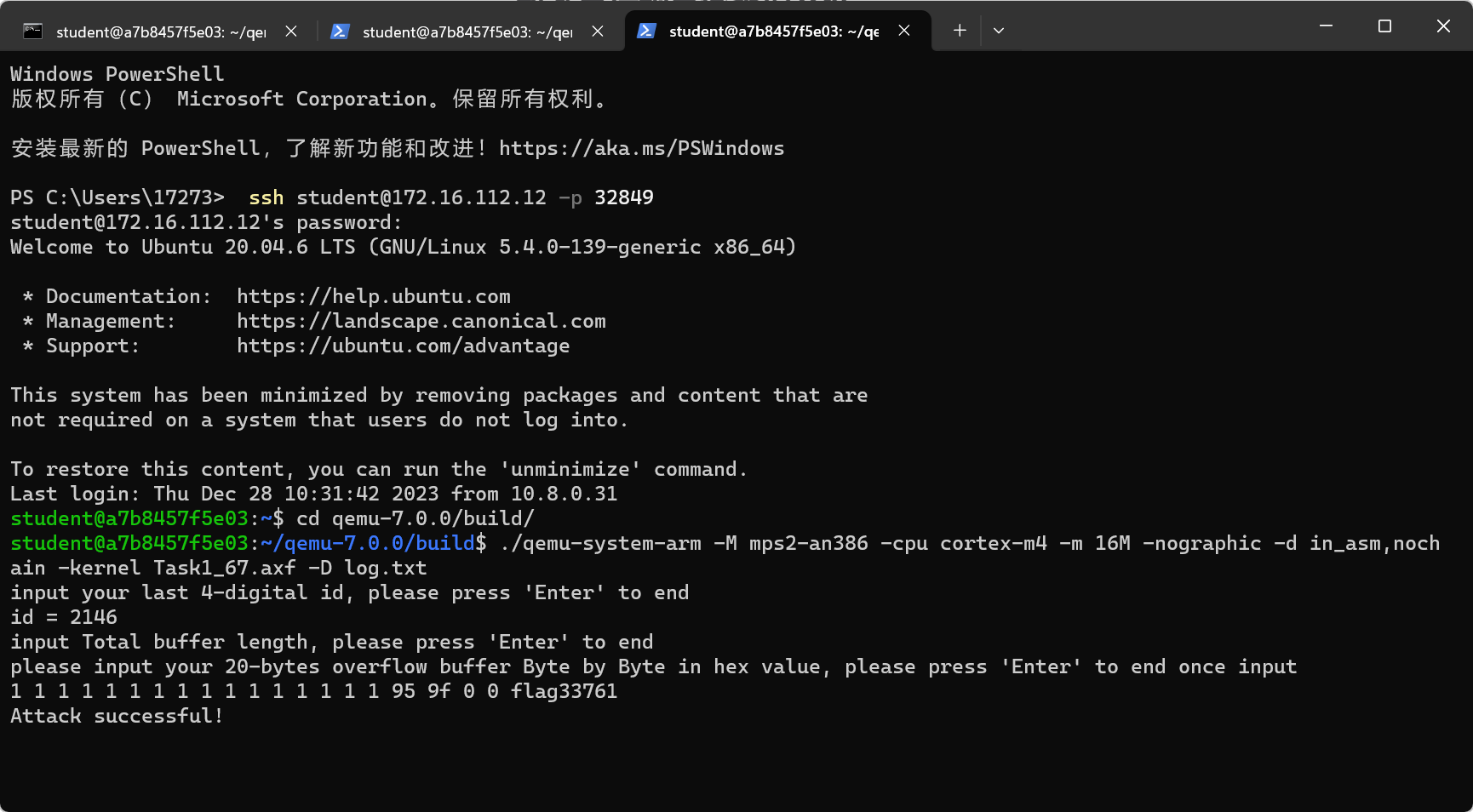


图10 测试截图

成功获取flag，为33761。

## 实验二

操作系统：windows 11 反汇编工具：IDA\_7.0

1. 按照任务要求选择指定指令集，如图11。

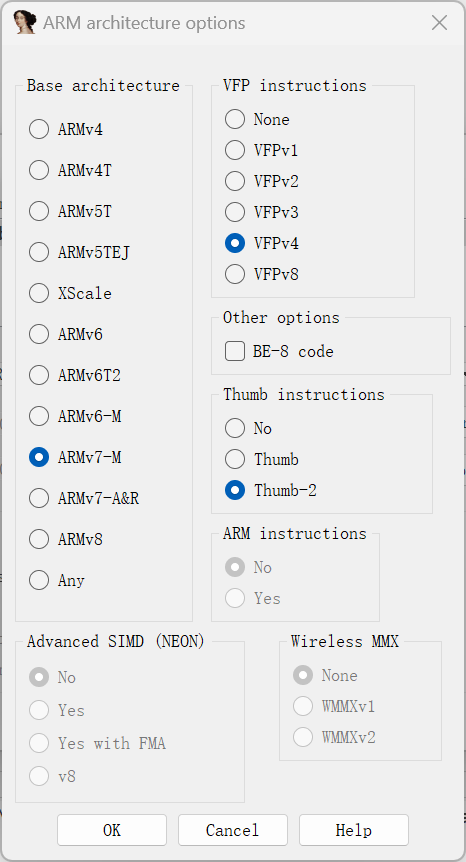


图11 指令集选择界面

1. 打开Task2\_67.axf后，找到main函数，如图12。

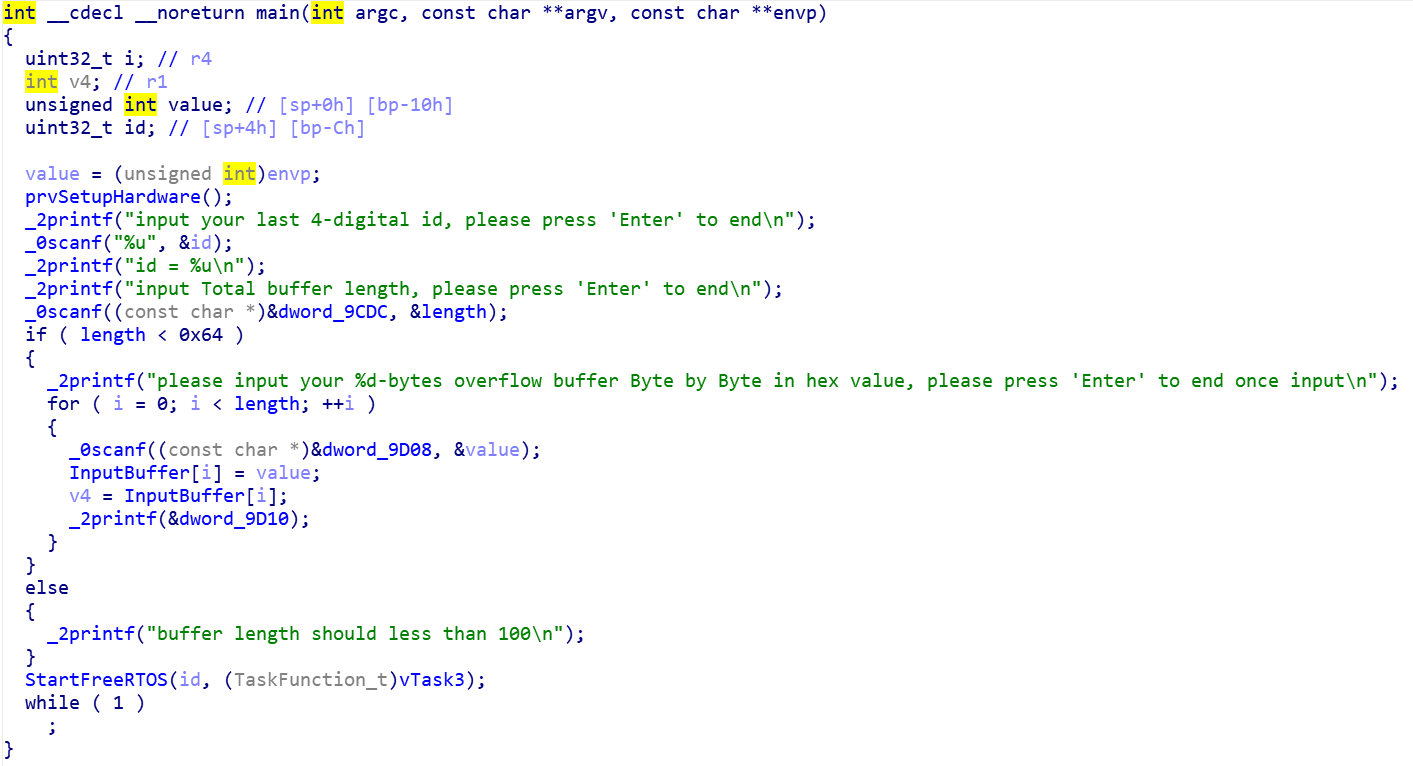


图12 main函数代码

1. 对main函数进行分析，输入部分的分析与实验一基本相同，唯一不同在于输入与存入InputBuffer的数据一致。然后先调用vTask3，再调用StartFreeRTOS。
2. 分析vTask3函数，函数如图13。由于v4初值为0，因此if语句判断后，先调用Transfer，再调用MPU\_vTaskDelay。

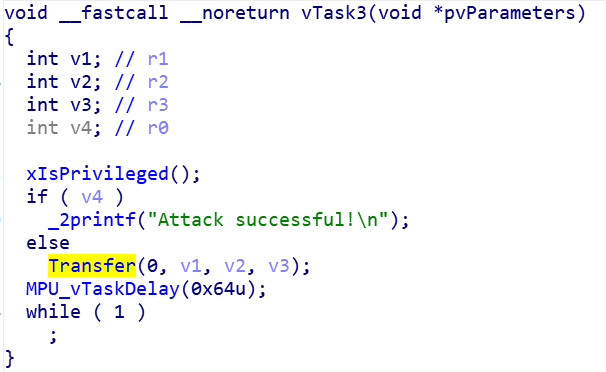


图13 vTask3函数代码

1. 分析Transfer函数，函数如图14。该函数定义了一个大小为8字节的缓冲区HelperBuffer，先赋了一组初值，然后将InputBuffer中的所有内容复制到HelperBuffer。类比实验一，可知这里可能出现缓冲区溢出。与前一个实验不同的是，每一个存入HelperBuffer的字节，都比输入的小1。

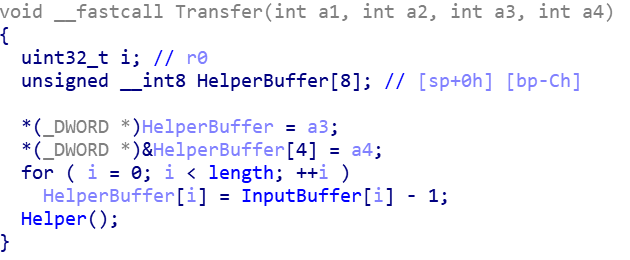


图14 Transfer函数代码

1. 搜索打印flag的函数：打开strings window，找到含flag的项，如图15。

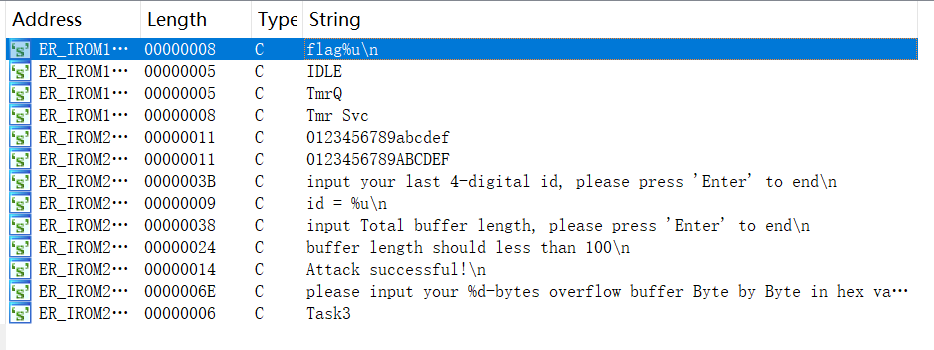


图15 strings window窗口

1. 找到flag的位置，并找到其对应的函数vTaskPool，函数首地址为00001c7e，如图16、17。



图16 aFlagU定义

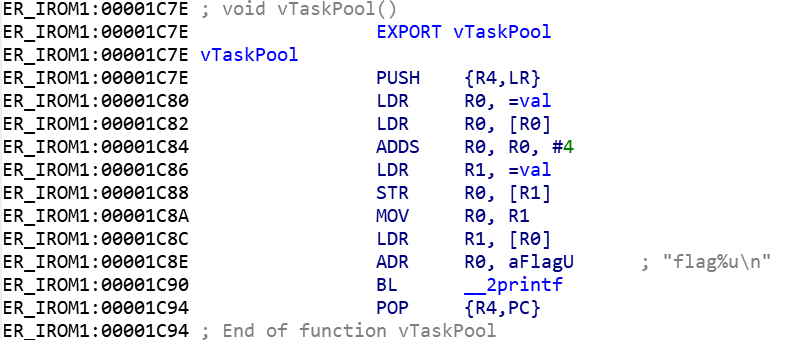


图17 vTaskPool函数汇编代码

1. 注意到00001c90行的代码，结合实验任务要求，发现打印flag需要先进行提权。依据实验二讲解pdf，如图18，得知提权函数为xPortRaisePrivilege。

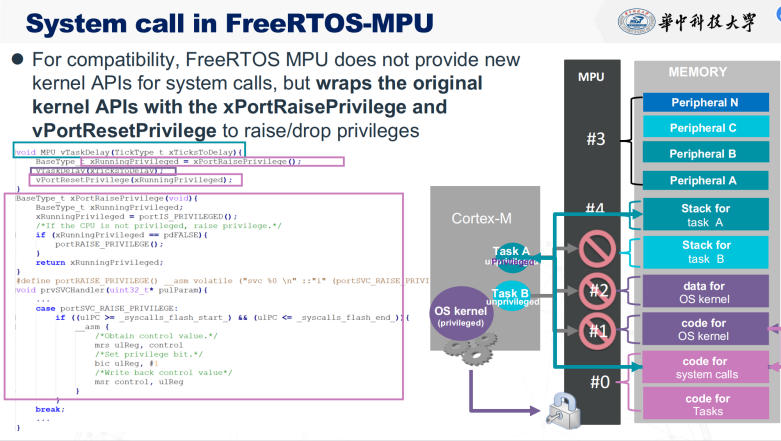


图18 实验二讲解内容截图

1. 根据name查找提权函数，首地址为：000086e2，如图19。

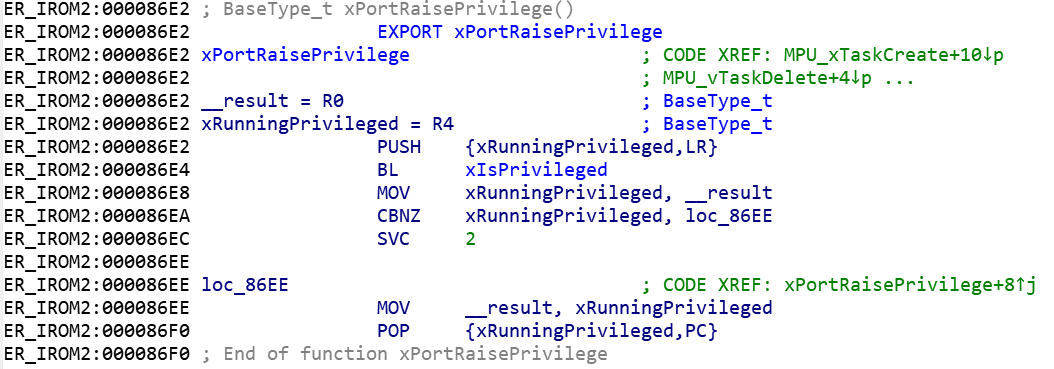


图19 xPortRaisePriviege函数汇编代码

1. 注意到上图中000086e2行代码将会push 8字节数据到栈，这个时候栈顶部分的数据就不受自己掌控了。在该函数末尾pop语句中，赋给PC的值就由LR来决定，几乎不可能跳转到打印flag的vTaskPool函数。因此，为了确保栈顶数据的可控性，需要阻止该函数压入新数据到栈中。最简单的办法是跳过该行代码，从下一行代码开始执行，下一行代码的地址为：000086e4。此时，可以在缓冲区溢出时，多覆盖一些数据，确保跳转的正确性。根据pop的内容，需要先压入4字节任意内容，再压入打印flag的函数地址。
2. 综合考虑以上内容（包括：字节均减一，地址加一等），可得输入的length为：20，缓冲区数据为：（任意8字节） + e6 87 01 01 + （任意4字节） + 80 1d 01 01。栈的表示如图20，测试结果如图21。

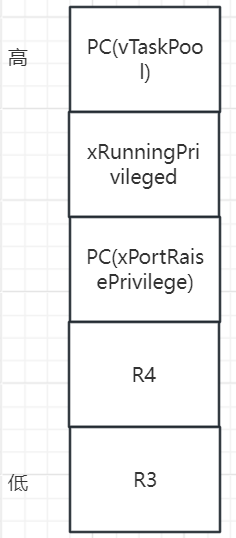


图20 栈的示意图

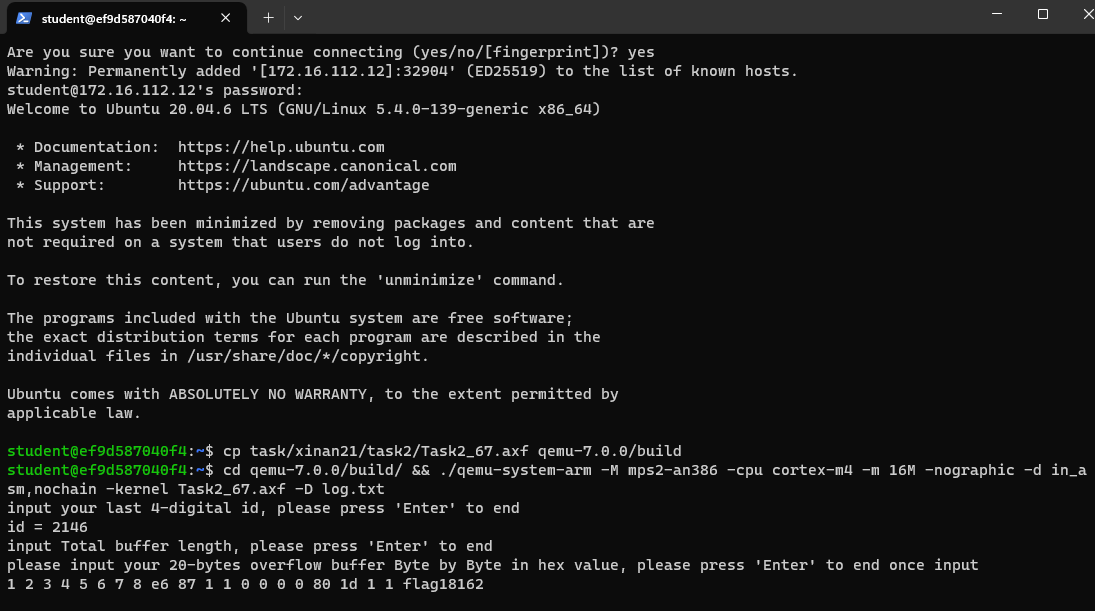


图21 测试结果截图

成功获取flag，为18162。

# 实验心得

本次实验的整体完成过程还算比较顺利，实验的过程中我也有一些收获。首先，我对缓冲区溢出漏洞有了比较深刻的了解。当一个程序没有正确验证输入数据的长度，或者在向缓冲区写入数据时没有正确限制其大小，恶意攻击者可以通过构造恶意输入来覆盖其他内存区域中的数据。主要的攻击方式有：执行恶意代码、覆盖重要数据、提升权限等。在实验过程中，我也有多次阅读实验讲解。其实，实验中遇到的问题很多都在讲解pdf中有提示。我个人认为实验相关的提示可以集中放在一处，这样也可以减少同学们反复阅读的时间。另外，我也对基于MCU的固件开发做了一些了解，包括架构、外设、寄存器配置、中断处理等。

# 附加题

1. MPU和MMU对于内存保护上的主要差别是什么？简述ARM、RISC-V或者MIPS上除了MPU以外的其他可⽤于系统保护的硬件特性？

答：MPU和MMU是用于内存保护的两种不同的硬件特性。它们在内存保护方面有以下主要差别：

1）功能范围：MPU主要用于提供基于地址的内存访问权限控制，它可以根据预定义的规则限制对内存区域的访问。MPU通常较为简单，功能有限，可以用于设置区域的读、写和执行权限等。MMU则更加复杂，用于虚拟内存管理，包括地址转换、页面映射等功能。

2）内存管理方式：MPU通常使用分段或分区的方式进行内存保护。它将内存划分为多个区域，并为每个区域定义访问权限。MMU则使用页表和虚拟内存机制，将虚拟地址映射到物理地址，并提供细粒度的内存保护和隔离。

除了MPU以外，ARM、RISC-V和MIPS等体系结构上还有其他可用于系统保护的硬件特性，例如：1）特权级别：这些体系结构定义了多个特权级别，如内核模式和用户模式，在不同特权级别下，对系统资源的访问权限有所不同。特权级别可用于限制用户态程序对系统资源的直接访问，提高系统的安全性。2）安全扩展：一些体系结构引入了安全扩展，如ARM的TrustZone技术，RISC-V的TrustZone扩展等。这些扩展提供了硬件级别的安全隔离，将系统划分为安全和非安全的域，以保护敏感数据和关键系统资源免受非授权的访问。3）访问控制指令：这些体系结构提供了特定的指令，用于进行访问控制和权限检查，如ARM的CPS指令、RISC-V的CSR指令等。这些指令可以用于在程序执行过程中进行权限检查和控制，实现更细粒度的系统保护。

1. 如何利⽤MPU实现对任务栈的溢出保护？请描述设置⽅法和简要步骤。

答：利用MPU实现对任务栈的溢出保护可以通过以下步骤进行设置：1）内存区域：首先，根据任务栈的需求，将内存划分为多个区域，其中包括任务栈所在的特定区域。MPU提供了区域划分的功能，可以根据地址范围进行设置。2）配置访问权限：针对任务栈所在的内存区域，配置合适的访问权限。通常，我们希望任务栈是可写的，但不允许执行。因此，可以将该区域的执行权限设置为禁止。3）设置MPU寄存器：根据所使用的处理器架构和MPU的具体实现，设置MPU寄存器以启用和配置内存保护。这通常涉及设置区域的起始地址、结束地址和访问权限等参数。4）启用MPU：在合适的时机，例如在操作系统的初始化阶段，启用MPU功能。这将使MPU开始监视对内存的访问，并根据配置的权限规则进行检查。

当任务运行时，MPU将检查每次访问是否符合权限规则。如果任务尝试访问任务栈以外的内存区域，或者尝试在任务栈上执行代码，MPU将引发异常或触发相应的保护机制，从而防止栈溢出和相关的安全问题。

需要注意的是，具体的MPU配置和设置方法可能因处理器架构和MPU实现而异。因此，在实际应用中，需要参考所使用的处理器和MPU的文档，并按照其规范和要求进行设置。

1. 基于ARMv7m架构谈⼀谈安全的物联⽹操作系统的系统调⽤设计与实现。

答：基于ARMv7-M架构的安全物联网操作系统的系统调用设计和实现需要考虑以下几个方面：1）特权级别：ARMv7-M架构定义了特权级别，包括特权级和非特权级。在安全的物联网操作系统中，系统调用应该在特权级别下执行，以确保对系统资源和关键功能的控制和保护。2）系统调用接口：设计合适的系统调用接口是安全物联网操作系统中的关键任务。接口应该提供对操作系统功能的受控访问。这可能包括对设备驱动程序、网络协议栈、安全服务和文件系统等的访问。3）访问控制：系统调用的实现应该包含适当的访问控制机制，以限制对敏感资源和功能的访问。这可以基于角色、访问权限或其他安全策略来进行。4）系统调用参数和返回值：对于ARMv7-M架构，系统调用的参数和返回值通常通过寄存器进行传递。在设计和实现系统调用时，需要定义和遵循参数传递的约定，并确保正确处理返回值。5）内存保护：使用ARMv7-M的内存保护单元（MPU）可以实现对任务栈和其他内存区域的保护。MPU可以以区域为基础定义内存访问权限，防止栈溢出和非授权访问。

1. 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？

答：在ARMv7-M架构或任何其他架构上，一个函数中存在缓冲区溢出并不意味着一定可以被利用。缓冲区溢出是一种常见的安全漏洞，但其利用能否成功取决于多种因素，包括架构的特点、编译器的行为、内存布局等。

以下是一些因素，可能会影响缓冲区溢出漏洞是否可被利用：1）缓冲区溢出的位置和大小：缓冲区溢出只有在溢出的数据能够覆盖敏感数据或重要的控制信息时才会带来问题。如果溢出的数据超出了目标缓冲区的边界，但并未影响关键数据或控制流，那么它可能不会被成功利用。2）内存布局和随机化：现代操作系统和编译器经常使用地址空间布局随机化（ASLR）等技术，随机化内存布局可以增加攻击者成功利用缓冲区溢出漏洞的难度。如果目标系统启用了ASLR或其他随机化技术，攻击者需要克服这些障碍才能成功利用漏洞。3）编译器的缓冲区溢出保护：一些编译器提供了针对缓冲区溢出的保护机制，如栈保护、堆保护等。这些机制可以检测和阻止缓冲区溢出漏洞的利用，从而增加攻击者的难度。4）输入的控制和验证：如果输入数据经过充分的验证和过滤，可以防止或限制缓冲区溢出的发生。例如，输入长度检查、字符串截断等可以减轻缓冲区溢出的潜在影响。