

**实验报告**



**题目： 缓冲区溢出攻击**

**班 级： 2023211307**

**学 号： 2023211075**

**姓 名： 魏生辉**

**学 院：计算机学院（国家示范性软件学院）**

**2024 年 11 月 18 日**

一、实验目的  
1.理解C语言程序的函数调用机制，栈帧的结构。

2.理解x86-64的栈和参数传递机制

3.初步掌握如何编写更加安全的程序，了解编译器和操作系统提供的防攻击手段。  
4.进一步理解x86-64机器指令及指令编码。

1. 实验环境
2. Linux
3. Objdump命令反汇编
4. GDB调试工具

4.Windows PowerShell（10.120.11.12）

三、实验内容

登录bupt1服务器，在home目录下可以找到一个targetn.tar文件，解压后得到如下文件：

README.txt；

ctarget；

rtarget；

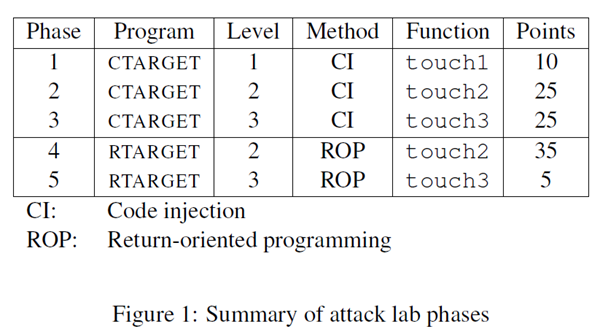
cookie.txt；

farm.c；

hex2raw。

ctarget和rtarget运行时从标准输入读入字符串，这两个程序都存在缓冲区溢出漏洞。通过代码注入的方法实现对ctarget程序的攻击，共有3关，输入一个特定字符串，可成功调用touch1，或touch2，或touch3就通关，并向计分服务器提交得分信息；通过ROP方法实现对rtarget程序的攻击，共有2关，在指定区域找到所需要的小工具，进行拼接完成指定功能，再输入一个特定字符串，实现成功调用touch2或touch3就通关，并向计分服务器提交得分信息；否则失败，但不扣分。因此，本实验需要通过反汇编和逆向工程对ctraget和rtarget执行文件进行分析，找到保存返回地址在堆栈中的位置以及所需要的小工具机器码。实验2的具体内容见实验2说明，尤其需要认真阅读各阶段的Some Advice提示。

本实验包含了5个阶段（或关卡），难度逐级递增。各阶段分数如下所示：



1. 实验步骤及实验分析

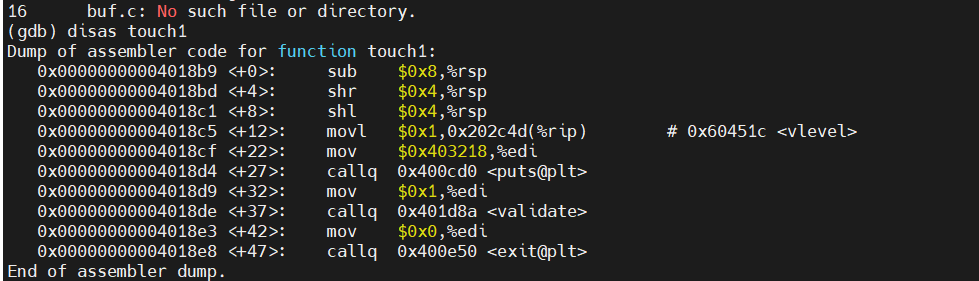
**准备工作**

①首先通过命令 解压得到 6 个文件：

②阅读所给材料，了解注意事项：

**第一阶段：**

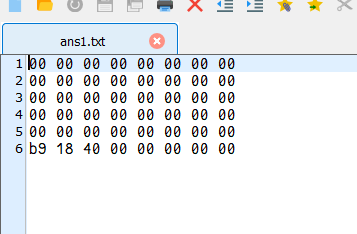
从说明文件上我们可以看到有两个函数getbuf和touch1，我们进入gdb模式反汇编一下这两个函数看看内容。



其中对我们最有用的是touch1的地址和栈的大小了--我们只要**，填满缓冲区，冲破缓冲区进行覆盖**，就可以实现目的了。

可以看到开栈的时候%rsp进行了减0x28的操作，说明栈的大小确定了。

用touch1覆写返回地址，函数返回后就到了touch1。Touch1的答案内容和结果如下：



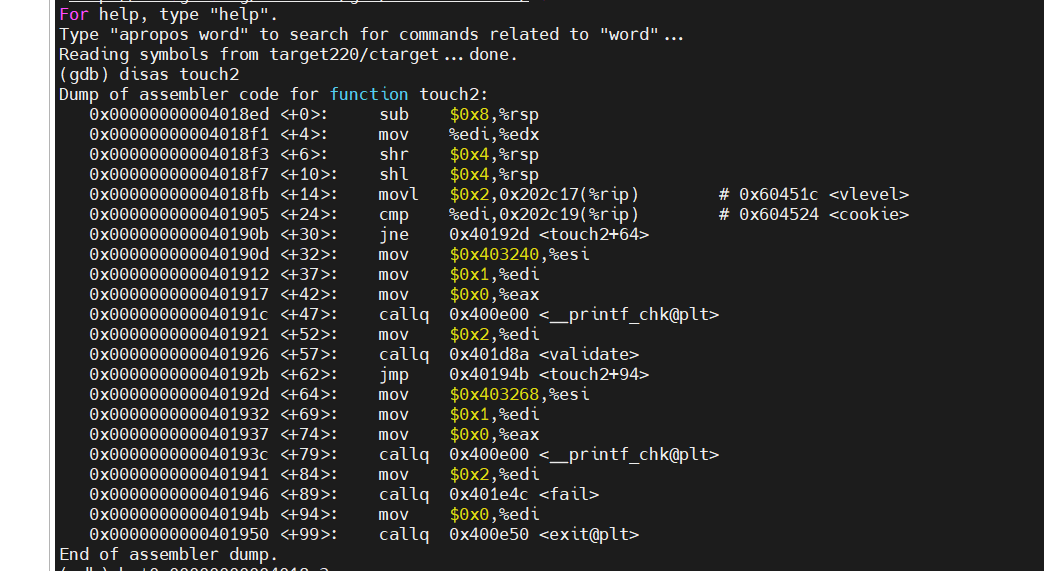
**阶段二**

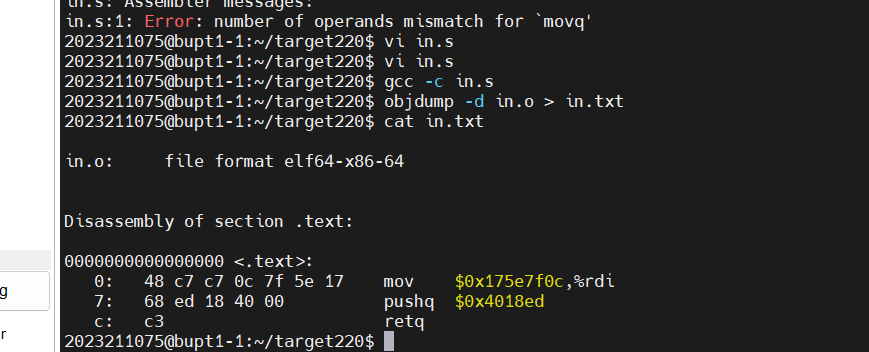
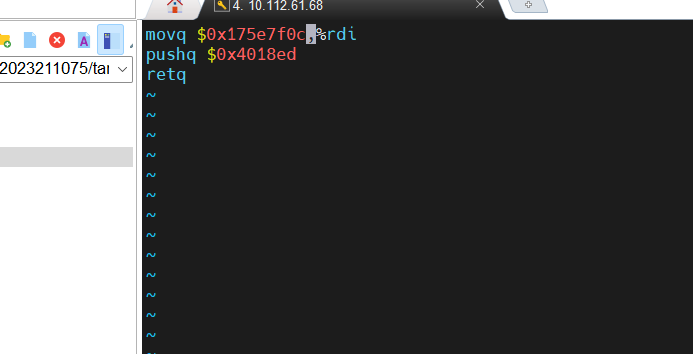
根据实验提示的内容，本阶段我们需要将 cookie 识别码注入到 touch2 函数中，并使

函数返回之后转而执行 touch2 函数而不是返回继续执行 test 函数；

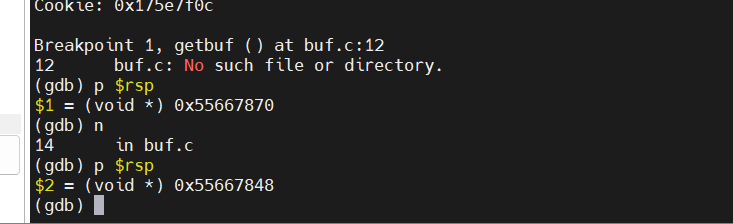
题目建议我们不使用jmp和call指令进行代码跳转，也就是说，只能通过在栈中保存目

标代码的地址，然后以ret的形式进行跳转；

先查看 touch2 函数（重点关注地址）：

接下来我们需要获取机器代码，可以写好之后编译再objdump反汇编，过程如下

然后我们找一下栈顶的地址，我们先进入gdb模式，然后在函数执行前打上断点，我们运行到函数开始时，看看此时的%rsp的值。



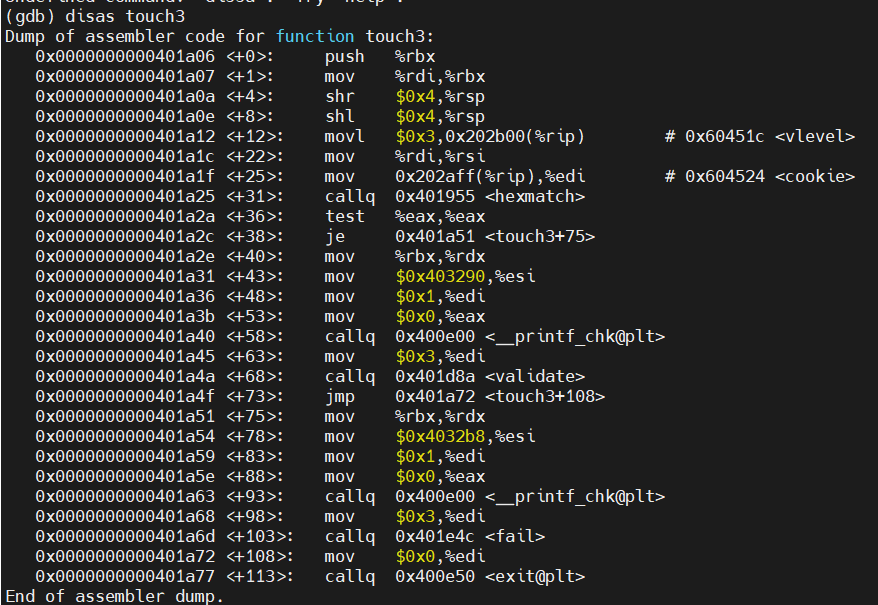
最后得出答案：将机器代码放到 40 个字节中的开头，代码地址放到末尾。

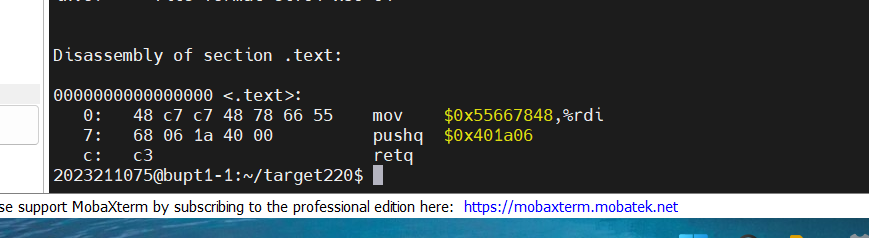
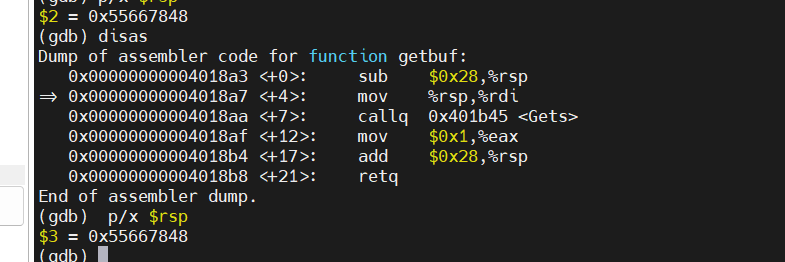
**阶段三**

进入第三阶段。与之前的第二阶段相似，我们将编写一段代码，将其转化为机器指令并存储在栈顶。然后在返回时跳转到栈顶，我们这一阶段的目标是关注 touch3 的地址。需要注意的是，touch3 的参数需要一个指针，随后我们将取这个指针的值。换句话说，这次的代码修改不仅仅是将 touch2 的地址变成 touch3，同时 %rdi 的内容也是由传递 cookie 的值变为传递 cookie 的地址。

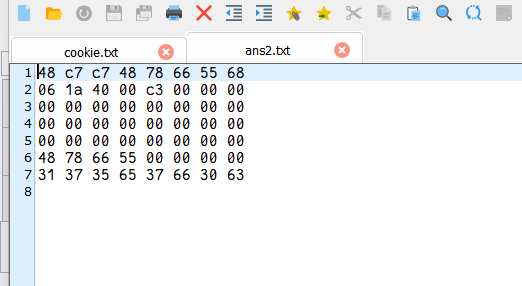
对此，我们便用第二阶段找到的，开栈前栈顶的上一个栈帧地址。这正好是 cookie 的地址。在 touch3 函数中，有个名为 hexmatch 的函数，用于比较 cookie 和它地址中的存值。可值得注意的是，我们只传入了第二个参数，而第一个参数也必须传递，因此，我们将其放置在 getbuf 的返回地址的上一个栈帧中。选择这个位置的原因是，所有输入已经结束，恰好位于结尾，尚未被包含在缓冲区内，同时这也与前面提到的 cookie 地址呼应。

当然，关于代码和 cookie 值的表示，也需要进行调整。此时，代码必须被改变为十六进制的机器指令，而 cookie 的值则需要转化为 ASCII 码的十六进制表示（这实际上是由于说明文件的建议）。根据说明文件的指导，我们查阅 ANSI 码时，注意不需要书写 0x，将其结果放在输入的最后，以确保输入格式的正确性。

以下是做题过程和截图：



最后得出答案:



**阶段四**

在这个阶段，我们将利用机器代码片段，将返回值与多个段落相结合，构建出一段完整的代码以实现我们的攻击目标。首先，我们将使用 pop %rax 作为返回地址，这也就是我们希望执行的命令的起始位置。接下来，我们将弹出上一个栈帧的 cookie 值，并存入 %rax 中。这里需要注意的是，或许 cookie 并不需要转换为 ASCII 码（虽然具体原因尚不明，但这一点在上一阶段的说明文件中是明确要求的）。

接下来，我们使用 mov %rax, %rdi 指令，将 cookie 的值传递给 %rdi，完成参数传递后，接下来就是 touch2 的跳转地址了。

很明显，不可能找到带特定立即数的 gadget，只能采用别的办法，思路如下：

getbuf 执行 ret，从栈中弹出返回地址，跳转到我们的 gadget1；

gadget1 执行，将 cookie 弹出，赋值给 %rax，然后执行 ret，继续弹出返回地

址，跳转到 gadget2；

gadget2 执行，将 cookie 值成功赋值给寄存器 %rdi，然后执行 ret，继续弹出

返回地址，跳转到 touch2。

上述思路用到的两个 gadget 为：

popq %rax ret和movq %rax, %rdi ret

接下来的工作就是很无聊的查表和对照，过程和截图如下：

**阶段五：**

经过查阅，我们发现并没有适合实现加法的 gadget。不过，仔细分析 gadget farm，我们注意到其中自带了一条实现 %rsp 偏移的指令。这一指令的地址为 401b93，我们将其命名为 gadget0，以便后续引用。

通过该 gadget，我们能够实现加法，因为 lea (%rdi, %rsi, 1), %rax 的效果是将 %rdi 和 %rsi 的值相加，并将结果存入 %rax。因此，只需将 %rdi 和 %rsi 中的一个寄存器设置为 %rsp，另一个设置为偏移值，即可表示 cookie 存放的地址，随后将这个地址移动到 %rdi 中。

通过进一步查阅，我们发现从 %rax 不能直接移动到 %rsi，而是需要通过 %eax、%ecx、%edx，再到 %esi 的顺序完成。因此，我们会将 %rsp 存放到 %rdi 中，将偏移值存入 %rsi。接着，使用 lea 指令将这两个结果的和存放在 %rax 中，最后再将 %rax 的值移动到 %rdi 中。

综上得出流程：

①movq %rsp, %rax

②movq %rax, %rdi

③popq %rax

④偏移值：48

⑤movl %eax, %ecx

⑥movl %ecx, %edx

⑦movl %edx, %esi

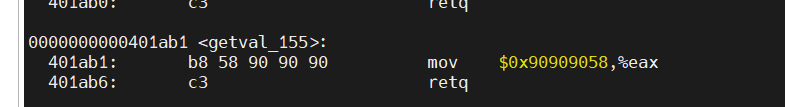
⑧lea (%rsi, %rdi, 1) %rax

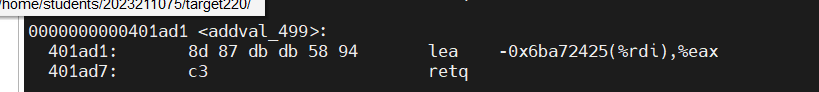
⑨movq %rax, %rdi

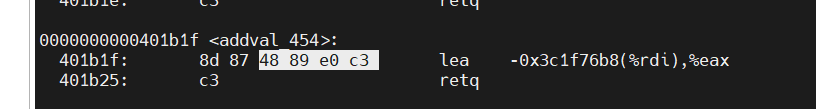
⑩touch3

别忘了Cookie

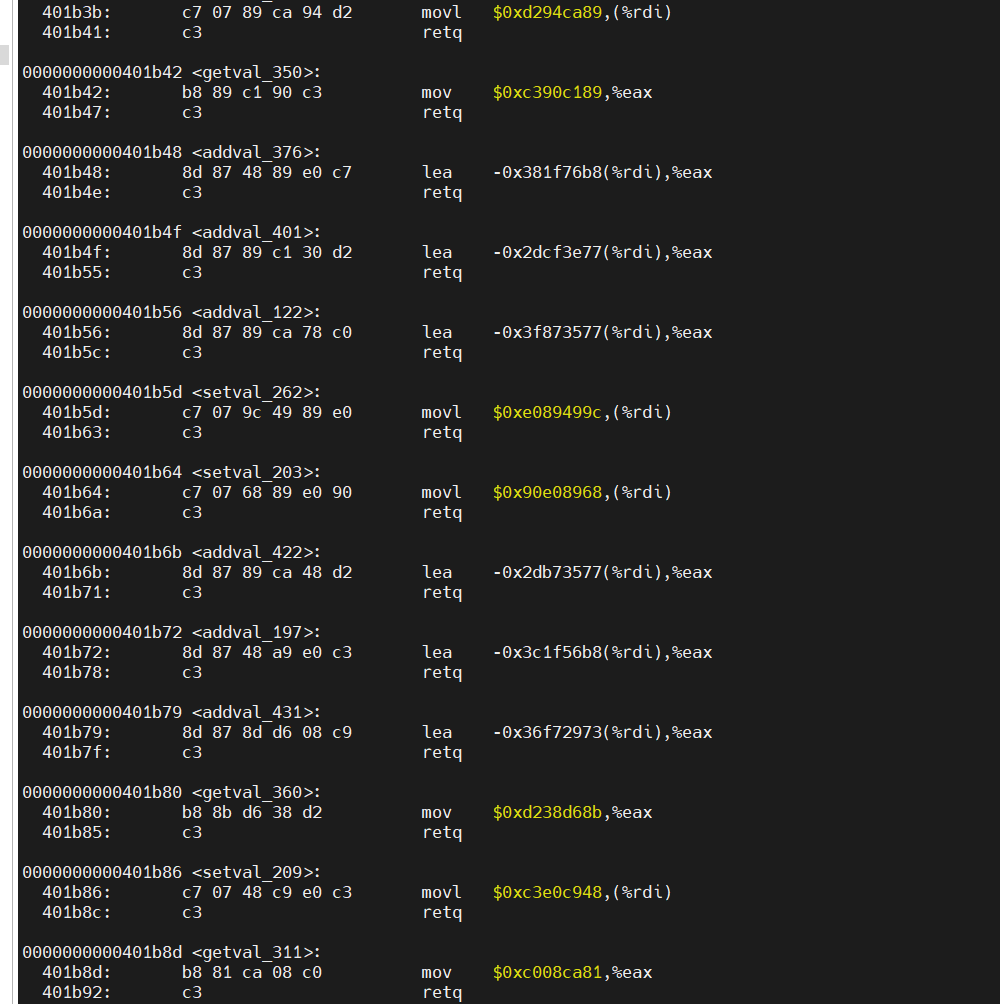
下面的截图是查找部分：



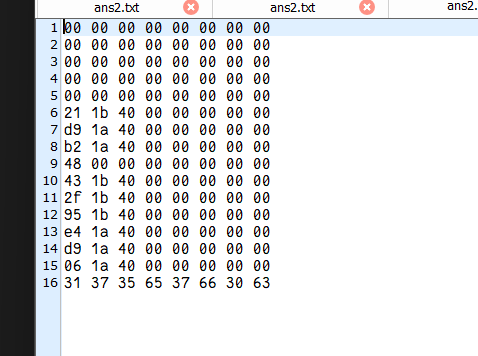








**最后得出答案：**



五、总结体会

总结心得（包括实验过程中遇到的问题、如何解决的、过关或挫败的感受、实验投入的时间和精力、意见和建议等）

无能狂怒-第四阶段我直接“红温”--想了好久发现输入时ctarget没改成rtarget。

虽然最后一个阶段只有五分，一直纠结过花费这么多时间在这五分上值不值得，但最终看到 “100”分后，心情其实是很激动的，这就是学习的成就感!

