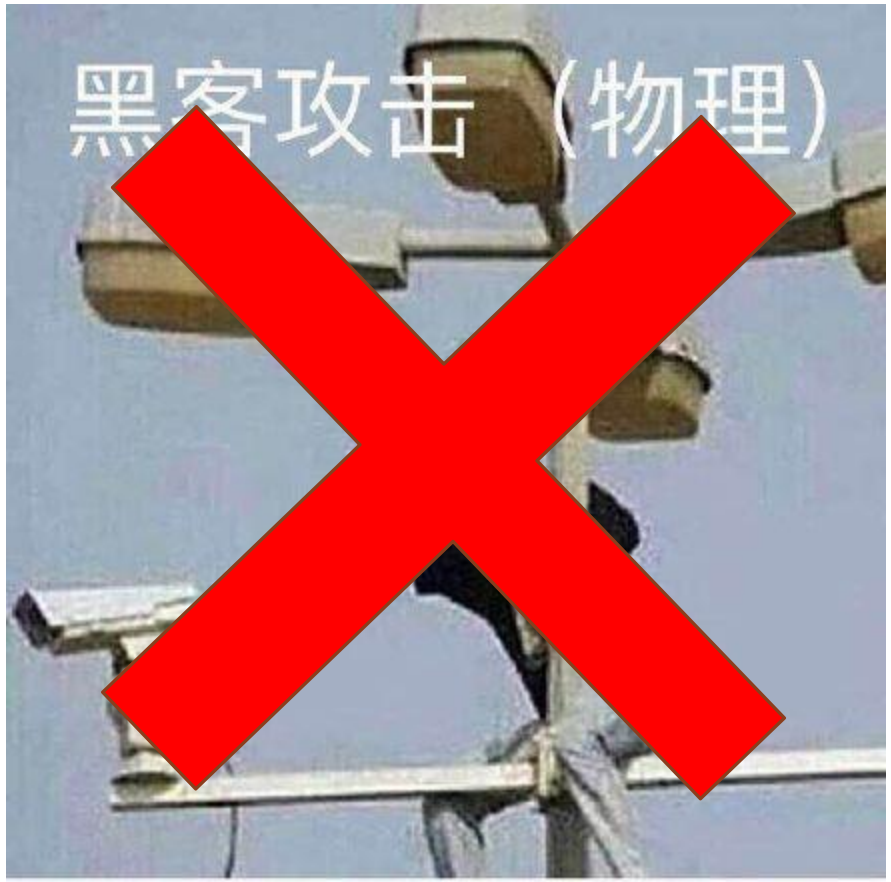
Attack Lab 实验说明

计算机组成原理 2025 秋

本次实验负责助教：黄泽文

函数调用的栈结构



缓冲区溢出

缓冲区溢出

可利用的漏洞

```
4   int getbuf() {
5       char buf[NORMAL_BUFFER_SIZE];
6       Gets(buf);
7       return 1;
8   }
```

是否只有这两种结果呢?

正常输出

```
❯ ./ctarget
Cookie: 0x59b997fa
Type string:hello
No exploit.  Getbuf returned 0x1
Normal return
```

段错误

```
❯ ./ctarget
Cookie: 0x59b997fa
Type string:It is easier to love this lab if you are a TA
Ouch!: You caused a segmentation fault!
Better luck next time
FAILED
```

Attack Lab 任务概览

你将得到两个包含缓冲区溢出漏洞的可执行文件ctarget和rtarget，与一个专属于你cookie（十六进制数）

你的任务是，构造特定的输入，使得这两个可执行文件在接受输入后，跳转到某个非预期的函数——这是一种常见的攻击手段，能够改变程序的执行结果。具体包括五项子任务

ctarget：代码注入攻击。

- level 1：跳转到touch1函数（10分）
- level 2：跳转到touch2函数，并传递cookie作为参数（25分）
- level 3：跳转到touch3函数，并传递cookie作为参数（25分）

rtarget：返回导向攻击。从已有的一系列函数（gadget）中找到可以利用的语句进行攻击。

- level 2：跳转到touch2函数，并传递cookie作为参数（35分）
- level 3：跳转到touch3函数，并传递cookie作为参数（自愿完成，不计入成绩）

代码注入攻击 Level 1

在ctarget文件中，getbuf函数被test函数调用

正常情况下，getbuf函数执行完毕后，会返回到test

你的任务是让getbuf在返回时跳转到ctarget中定义的另一个函数touch1，而非回到test函数。

这一步骤要求你改变栈中的Return Address. 你的操作可能会破坏程序的栈结构，不过在本阶段，不需要考虑此问题，因为程序在执行touch1函数后就会中止。

主要任务：
分析栈空间中字符串和Return Addr的位置关系

```
1 void test()
2 {
3     int val;
4     val = getbuf();
5     printf("No exploit.  Getbuf returned 0x%x\n", val);
6 }
```

```
1 void touch1()
2 {
3     vlevel = 1;        /* Part of validation protocol */
4     printf("Touch1!: You called touch1()\n");
5     validate(1);
6     exit(0);
7 }
```

代码注入攻击 Level 2

这一阶段，你的任务是让程序跳转到touch2的同时，传递你的cookie作为参数。

回想一下，寄存器和函数参数的关系是什么？%rdi

本质上说，栈中的信息和代码段中的代码并没有区别。如果我们将一些代码构造成特殊的字符串，并设法让程序跳转到这段特殊字符串对应的地址，就能够让程序执行我们所定义的操作。

主要任务：
通过编写自定义汇编代码传递参数
先后跳转到自定义代码和touch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 }
```

代码注入攻击 Level 3

本阶段任务与level 2基本相同，但你要传递的参数变为了一个字符串——也就是说，你需要传递一个地址作为参数，并自己管理字符串的存储。

主要任务：
存储字符串，并避免其被覆盖

```
1  /* Compare string to hex represention of unsigned value */
2  int hexmatch(unsigned val, char *sval)
3  {
4      char cbuf[110];
5      /* Make position of check string unpredictable */
6      char *s = cbuf + random() % 100;
7      sprintf(s, "%.8x", val);
8      return strncmp(sval, s, 9) == 0;
9  }
10
11 void touch3(char *sval)
12 {
13     vlevel = 3;        /* Part of validation protocol */
14     if (hexmatch(cookie, sval)) {
15         printf("Touch3!: You called touch3(\"%s\")\n", sval);
16         validate(3);
17     } else {
18         printf("Misfire: You called touch3(\"%s\")\n", sval);
19         fail(3);
20     }
21     exit(0);
22 }
```

返回导向攻击——gadget

事实上，现代操作系统和编译器采用了许多安全策略，例如对栈地址进行随机化、设置栈地址段为不可执行等。这会导致我们先前采用的简单攻击方法失效。

然而这并不意味着攻击无法开展，你将在rtarget文件中尝试另辟蹊径，完成与先前一样的任务。

例如，在rtarget中，存在一个函数setval_210，其实现和汇编代码如图所示（与你实际拿到的文件可能有所差异）。这段代码看似平平无奇，但注意其中的“48 89 c7”部分——这同时也是“movq %rax, %rdi”的指令编码。也就是说，如果你从0x400f18开始读取代码，那么这段代码会被解读为“movq %rax, %rdi; ret”。这有什么作用?

```
void setval_210(unsigned *p)
{
    *p = 3347663060U;
}
```

```
0000000000400f15 <setval_210>:
  400f15:       c7 07 d4 48 89 c7       movl   $0xc78948d4,(%rdi)
  400f1b:       c3                      retq
```

返回导向攻击 Level 2

rtarget文件中存在大量像setval_210的函数，称为gadget farm。它们的特点是，存在一小段可利用的攻击“零件”（gadget），形式通常为：

- 一条可用于攻击的汇编指令
- （可能存在）一些没有影响的指令，如nop或不影响攻击目标的操作
- ret指令

只要合理编排return addr，就可以将多个gadget串联起来，完成攻击目标。

在level 2中，你仍然需要让程序跳转到touch2函数，并传递一个数值参数。

完成任务的关键在于将栈中的数值存入$rdi——因此你可能需要一个pop指令将栈中的数值读取到某个寄存器，以及一个mov指令将其转移到$rdi中。

主要任务：
在gadget中寻找pop指令和mov指令

返回导向攻击 Level 3（自愿完成，不计成绩）

在level 3中，你仍然需要让程序跳转到touch3函数，并传递一个字符串参数。

思路是一致的：我们依然字符串存储在栈中，并将其地址作为参数传递。挑战在于，如今的栈地址是随机化的，我们如何获取字符串的地址？

答案是使用相对位置。不论栈地址如何变化，字符串和$rsp的相对位置是可以确定的——因而，你需要寻找一个合适的函数，通过$rsp和相对位置关系计算出字符串的地址。

除此之外，你依然需要用到pop和一系列mov操作。本阶段的难度更高，你可能无法在gadget farm中找到一次就将源寄存器的值移动到目标寄存器的语句，而是需要在多个寄存器间中转才能完成任务。

主要任务：
在gadget中寻找pop指令和多个mov指令
在gadget中寻找一个提供加减运算的函数，并计算出字符串地址

HEX2RAW

要完成攻击任务，需要构建特定的输入，如某个函数的地址、汇编指令等

但我们显然不能直接把地址或汇编指令的十六进制表示输入程序

为了方便大家的实验，本次大作业提供 hex2raw 程序，将十六进制字符串转换为raw：

*你所构造的输入不可以包含0a (换行符\n)，这会被gets函数视为字符串结束标志，导致在其之后的内容不被读取。

```
❯ cat ctarget01.txt
/* Pad with 0 bytes */
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
```

```
❯ ./ctarget < ctarget01.txt # 错误做法，这只是把字符串的ASCII码输入程序
Cookie: 0x59b997fa
Type string:No exploit.  Getbuf returned 0x1
Normal return
❯ ./hex2raw < ctarget01.txt | ./ctarget # 正确做法，用hex2raw进行转换，再用管道输入程序
Cookie: 0x59b997fa
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

生成字节码

example.S

```
# Example of hand-generated assembly code

        push $0xabcdef          # Push value onto stack
        add   $17,%eax          # Add 17 to %eax
        .align 4                # Following will be aligned on multiple of 4
        .long   0xfedcba98      # A 4-byte constant
```

```
$ gcc -c example.S
$ objdump -d example.o > example.d
```

example.d

```
0: 68 ef cd ab 00       push   $0xabcdef
5: 83 c0 11             add    $0x11,%eax
8: 98                   cwtl
9: ba                   .byte 0xba
a: dc fe                fdivr  %st,%st(6)
```

attack.txt

```
68 ef cd ab 00 83 c0 11 98 ba dc fe
```

调试利器 GDB

- 断点调试
- 单步调试
- 打印变量、执行表达式
- 查看/更改内存、寄存器
- 反汇编
- …

命令	效果
开始和停止	
quit	退出 GDB
run	运行程序(在此给出命令行参数)
kill	停止程序
断点	
break multstore	在函数 multstore 入口处设置断点
break * 0x400540	在地址 0x400540 处设置断点
delete 1	删除断点 1
delete	删除所有断点
执行	
stepi	执行 1 条指令
stepi 4	执行 4 条指令
nexti	类似于 stepi，但以函数调用为单位
continue	继续执行
finish	运行到当前函数返回
检查代码	
disas	反汇编当前函数
disas multstore	反汇编函数 multstore
disas 0x400544	反汇编位于地址 0x400544 附近的函数
disas 0x400540,0x40054d	反汇编指定地址范围内的代码
print /x $rip	以十六进制输出程序计数器的值
检查数据	
print $rax	以十进制输出%rax 的内容
print /x $rax	以十六进制输出%rax 的内容
print /t $rax	以二进制输出%rax 的内容
print 0x100	输出 0x100 的十进制表示
print /x 555	输出 555 的十六进制表示
print /x ($rsp+ 8)	以十六进制输出%rsp 的内容加上 8
print *(long *) 0x7fffffffe818	输出位于地址 0x7fffffffe818 的长整数
print *(long *)($rsp+ 8)	输出位于地址%rsp+8 处的长整数
x/2g 0x7fffffffe818	检查从地址 0x7fffffffe818 开始的双(8 字节)字
x/20bmultstore	检查函数 multstore 的前 20 个字节
有用的信息	
info frame	有关当前栈帧的信息
info registers	所有寄存器的值
help	获取有关 GDB 的信息

图 3-39　GDB 命令示例。说明了一些 GDB 支持机器级程序调试的方式

获取大作业相关资料

我们将为每位同学生成**专属的cookie和二进制文件**

- 访问http://nuc.kirotta.top:15513/，输入学号和邮箱并提交
- 你将获得一个名为target**k**.tar的文件，k是你的作业编号
- 请不要重复提交请求，所有target文件的难度是一样的
- 如果生成了多次target文件，请在**最后一次生成的文件**上完成实验

请全面、仔细地阅读作业说明文档，以充分了解动手前你需要知道的所有事情。

完整阅读说明文档有助于理解实验，并降低上手难度。

作业需在Linux下运行，若遇到运行错误/地址变化等意外问题，请尝试安装相关依赖库、关闭地址空间随机化（ASLR）、更换系统版本等操作。建议直接使用助教提供的虚拟机

作业提交

在完成作业期间，你可以在http://nuc.kirotta.top:15513/scoreboard查看当前的得分。作业完成先后不影响最终的评分。

此外，你还需要在网络学堂以如下格式提交一个压缩包：

文件名：学号-姓名-attacklab.zip

src/ctarget01.txt, ctarget02.txt, ctarget03.txt, rtarget02.txt：

- 分别保存各个小题的答案（原始的十六进制字符串），使用空格或换行分隔字节

pdf/report.pdf：

- 描述你的实验原理、实验过程以及实验感想（可选：课程意见）。

```
/* Pad with 0 bytes */
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
/* Address of touch1 */
b6 88 04 08 00 00 00 00
```

评分相关

分数最高分100：
- 65分，自动评测系统给出的评测分数折算。
- 35分，助教给出的文档分数。

文档采分点：
- 原理要点，关键过程，攻击字符串的意义、构造过程以及其代表的代码。

作业截止时间：
- 2025.11.17 23:59 （周一）

迟交作业：
- 迟交未超过宽限期的正常评分。
- 迟交超过宽限期，但仍在 DDL 一周内，得分仅有应得分数的80%。
- 迟交超过 DDL 一周拒收。

禁止抄袭
- 如若抄袭（代码、文档等任何提交材料），本次作业零分并上报学校处分

Good Luck!

Wish your code bug-free