

**实验报告**



**题目： 缓冲区溢出攻击实验**

**班 级： 2023211321**

**学 号： 2023212872**

**姓 名： 计子毅**

**学 院： 计算机学院**

**2024年 12 月 2 日**

**2**

一、实验目的

1. C语言程序的机器级表示。

2. 掌握GDB调试器的用法。

3. C编译器生成的x86-64机器代码，理解不同控制结构生成的基本指令模式，过程的实现。

4. 掌握两种缓冲区攻击方法，进一步理解软件漏洞的危害。

1. 实验环境
2. 服务器：10.112.61.68（关机时间：2024年12月5日00：00）
3. Linux
4. Objdump命令反汇编
5. GDB调试工具
6. 积分榜：

9-12班：<http://10.120.11.13:19330/scoreboard>

13-21班：[http://10.120.11.13:19340/scoreboard](http://10.120.11.13:19330/scoreboard)

说明：由于服务器及OS的差异，存在如下问题：

ctarget的第2、3关，在攻击成功后，显示Failed信息如下：

Valid solution for level 2/3 with target ctarget

Ouch!: You caused a segmentation fault!

Better luck next time

FAILED

**请同学们关注记分牌信息，得分后表示攻击成功**

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；

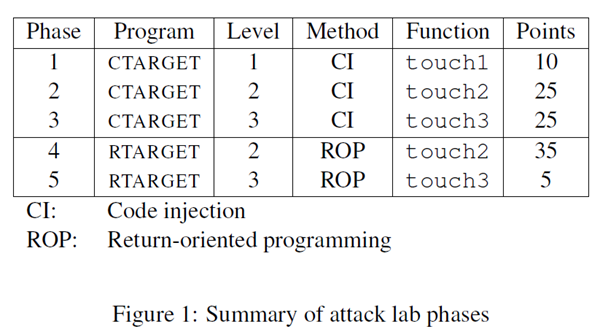
cookie.txt；

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。实验的具体内容见实验说明，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：



四、实验步骤及实验分析

**准备工作：**解压targe543.tar压缩包，得到target543文件夹，文件夹中有README.txt；ctarget；rtarget；cookie.txt；farm.c；hex2raw文件，使用cd target指令进入该文件夹中的环境，在此进行实验。首先使用vim编辑器查看README文件，明确各个文件的作用，可知提交答案需要将字符串转换为2进制文件进行提交。

**Phase1：**重定向touch1

首先使用objdump -d ctarget指令查看ctarget的汇编代码，得到函数的调用顺序main --> stable\_launch --> launch --> test --> getbuf --> touch1。各函数对应c语言代码如下：

test函数：

void test(){

int val;

val = getbuf();

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

}

getbuf函数：

unsigned getbuf(){

char buf[BUFFER\_SIZE];

Gets(buf);

return 1;

}

touch1函数：

void touch1(){

vlevel = 1; /\* Part of validation protocol \*/

printf("Touch1!: You called touch1()\n");

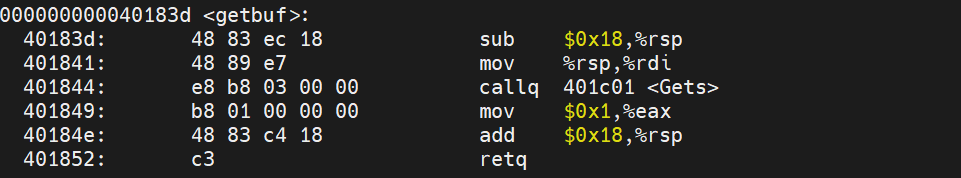
validate(1);

exit(0);

}

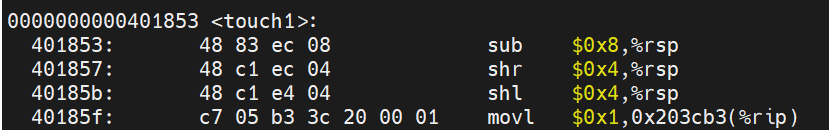
getbuf函数会为即将输入的字符串开辟缓冲区，我们的目的就是需要攻击缓冲区使之溢出，并将返回地址替换为touch1的地址，从而成功的调用touch1函数，所以我们得先明确BUFFER\_SIZE的大小。

通过查看getbuf的汇编代码，知道buf数组的大小为0x18个字节的大小，即24个字节。因此我们只需要填充24个字节，就可以将getbuf函数的栈帧填满，再往上增加%rsp栈指针的值，就会到达存储返回值的位置，然后再将touch1函数的地址填充进去。



buf数组的40个字节可以随便填充任何内容，只要在填充完毕之后以小端的方式将touch1的地址再放入栈中即可。

通过汇编代码得到touch1的地址为0x401853。

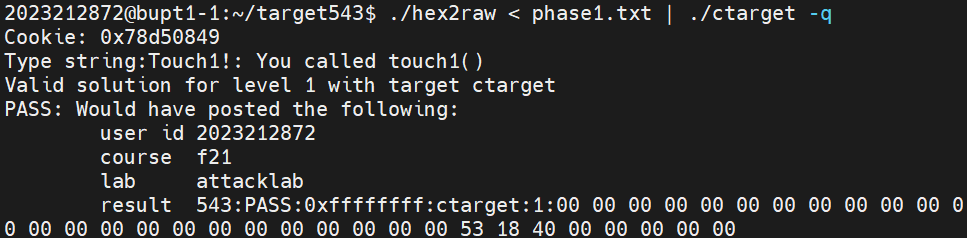


则可以确定需要输入的字符串可以是：

图片包含 户外, 街道, 城市, 键盘

描述已自动生成

将字符串通过vim编辑器写入phase1.txt文件中，然后执行命令./hex2raw < phase1.txt | ./ctarget -q，答案正确通过。



然后通过hex2raw文件，执行指令./hex2raw < phase1.txt > out1.txt，转换为2进制文件out1.txt，然后./rtarget < out5.txt执行，获得nicejob信息表示通过。

文本

描述已自动生成

**Phase2:**重定向touch2

首先可得touch2的c语言代码：

void touch2(unsigned val){

vlevel = 2;

if (val == cookie) {

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

validate(2);

} else {

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

fail(2);

}

exit(0);

}

可知，相较于touch1，touch2需要传入一个参数，根据程序的判断部分可知，当传入的这个参数等于cookie时，就正确。

使用vim编辑器查看cookie文件的内容，得到cookie为0x78d50849



那么我们可以编写一段代码，将cookie放入%rdi寄存器中，并设返回地址为touch2，编写代码如下：

文本

中度可信度描述已自动生成

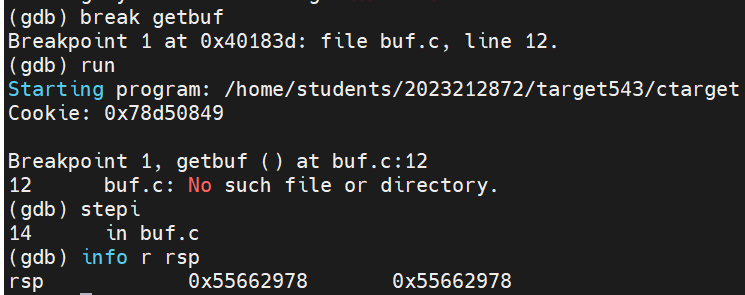
并使用gcc -c phase2.s将其编译为16进制表示到phase2.o文件中，然后使用objdump查看反汇编代码。

图形用户界面, 文本

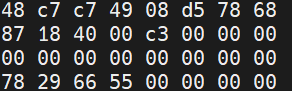
描述已自动生成

那么我们可以将这段代码的地址放到栈顶，将返回地址设为栈顶地址，就可以成功的调用touch2，所以我们要获取getbuf执行结束之前的栈顶指针的位置。

通过GDB获取执行getbuf函数创建buf数组之后，栈顶指针rsp的地址为0x55662978



则可以确定我们需要填入的字符串为



最后使用phase1中的提交方式进行提交。

**Phase3：**重定向touch3

首先我们由汇编代码可以得到touch3和hexmatch的c语言代码：

int hexmatch(unsigned val, char \*sval){

char cbuf[110];

char \*s = cbuf + random() % 100;

sprintf(s, "%.8x", val);

return strncmp(sval, s, 9) == 0;

}

void touch3(char \*sval)

{

vlevel = 3;

if (hexmatch(cookie, sval)) {

printf("Touch3!: You called touch3(\"%s\")\n", sval);

validate(3);

} else {

printf("Misfire: You called touch3(\"%s\")\n", sval);

fail(3);

}

exit(0);

}

可知touch3是以一串字符串为参数，且该字符串为cookie，然而cookie是一个16进制数，则我们要根据ascii码表获取到cookie字符串的16进制表示：37 38 64 35 30 38 34 39 00

由于touch3会调用hexmatch和strncmp，他们会把数据压入到栈中，有可能会覆盖getbuf栈帧的数据，所以我们不能把字符串数据存放在缓冲区中，考虑将其存放在test函数的栈帧中。test函数的栈顶是返回地址，则我们将字符串存放在栈顶+8的地址处。

首先通过gdb获取test栈顶的地址：0x55662990，再+8得0x55662998

文本

描述已自动生成

则我们可以编写代码：

图片包含 文本

描述已自动生成

并将其编译和反汇编：

图片包含 图形用户界面

描述已自动生成

然后如同phase2，将返回地址设为getbuf执行时的栈顶，并将这段函数的地址放在栈顶处，则可得到输入的字符串为：

文本

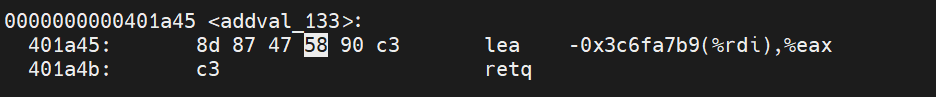
描述已自动生成

**Phase4：**在随机栈中重定向touch2

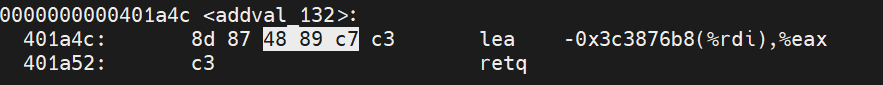
rtarget使用了随机栈和标记可执行内存位置来防止代码注入，因此不能像之前那样可以精确定位代码的地址。因此只能从源代码产生的汇编代码中寻找特殊的指令序列来帮助完成攻击，即寻找一系列特定的指令序列，这些指令最后以ret结尾(这种序列被称为gadget)。rtarget实验中所使用的所有gadget序列都由farm.c提供，则我们可以通过反汇编知道这些gadget在start\_farm到end\_farm之间。

而由实验提示，phase4所需gadget在start\_farm到mid\_farm之间，那么我们可以先pop %rax，然后mov %rax %rdi，再调用touch2.

pop %rax对应机器码为58，且保证58后面的机器码为90（不执行动作）或c3（ret）



mov %rax %rdi对应机器码为48 89 c7，后续机器码同理



则二者地址分别为401a48，401a4e，那么我们的具体思路是先将缓冲区溢出，第一个返回地址为pop，将cookie弹出到rax中，然后mov，最后调用touch2

则输入的字符串为：

文本

中度可信度描述已自动生成

**Phase5：**在随机栈中重定向touch3

和phase4同理，在phase5中需要8条gadget来将cookie字符串放到rdi中，再调用touch3

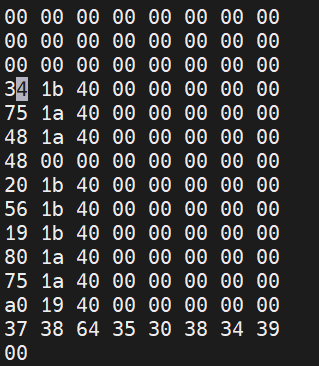
图示

描述已自动生成

具体流程如图，但具体地址我们可以通过rtarget的反汇编代码获取。

我们再字符数组前有10个地址存入，然而rip会指向下一条返回地址，那么偏移量应为9\*8=72字节，16进制为0x48.

最后我们应该输入的字符串为：



分别为：填满缓冲区，rsp->rax rax->rdi pop rax 0x48->rax eax->ecx ecx->edx edx->esi

lea(rdi,rsi,1)->rax rax->rdi touch3

五、总结体会

本次实验题目为“缓冲区溢出攻击实验”，旨在深入理解C语言程序的机器级表示、掌握GDB调试器的用法、理解C编译器生成的x86-64机器代码、以及掌握两种缓冲区攻击方法，从而进一步理解软件漏洞的危害。

在实验过程中，我首先通过反汇编工具objdump和GDB调试器对目标程序ctarget和rtarget进行了分析。这两个程序都存在缓冲区溢出漏洞，需要通过代码注入和ROP（Return-Oriented Programming）方法对其进行攻击。

在攻击ctarget程序时，我首先分析了其汇编代码，确定了缓冲区的大小以及需要覆盖的返回地址。通过填充特定的字符串，我成功地使程序跳转到了touch1、touch2和touch3函数，从而完成了对ctarget的攻击。

在攻击rtarget程序时，由于程序使用了随机栈和标记可执行内存位置来防止代码注入，我采取了ROP方法。通过寻找特定的指令序列（gadget），我成功地构造了攻击字符串，使程序跳转到了touch2和touch3函数，从而完成了对rtarget的攻击。

理论与实践结合：通过本次实验，我深刻体会到了理论与实践相结合的重要性。虽然在课堂上学习了相关的理论知识，但在实际操作过程中仍然遇到了许多困难。通过不断地调试和修改，我逐渐掌握了攻击方法，并成功地完成了实验。

在本次实验中，我熟练掌握了objdump和GDB等工具的使用方法。这些工具对于分析程序的汇编代码和调试程序非常有用，能够帮助我快速定位问题并找到解决方案。

在实验过程中，我遇到了许多问题，如缓冲区大小计算不准确、攻击字符串构造不正确等。通过查阅资料和不断尝试，我逐渐解决了这些问题，并积累了宝贵的经验。

通过本次实验，我深刻认识到了软件漏洞的危害性以及安全防护的重要性。缓冲区溢出是一种常见的安全漏洞，如果不加以防范，可能会导致程序崩溃、数据泄露等严重后果。因此，在开发过程中必须注重安全性，采取适当的措施来防止缓冲区溢出等漏洞的出现。

本次实验投入了大量的时间和精力。在实验过程中，我不断地调试和修改代码，查阅相关资料，与老师和同学交流讨论。虽然过程有些艰难，但最终的成功让我倍感欣慰和满足。

六、诚信声明（不签扣10分）

需要填写如下声明，并在底部给出手写签名的电子版。

在完成本次实验过程中，我曾分别与以下各位同学就以下方面做过交流：

1、与陈子容同学交流栈指针的转移机制

此外，我还参考了以下资料：

1. 网络查找机器码表

在我提交的程序中，还在对应的位置以注释形式记录了具体的参考内容。

我独立完成了本次实验除以上方面之外的所有工作，包括分析、设计、编码、调试与测试。

我清楚地知道，从以上方面获得的信息在一定程度上降低了实验的难度，可能影响起评分。

我从未使用他人代码，不管是原封不动地复制，还是经过某些等价转换。

我未曾也不会向同一课程（包括此后各届）的同学复制或公开我这份程序的代码，我有义务妥善保管好它们。

我编写这个程序无意于破坏或妨碍任何计算机系统的正常运行。

我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按照0分计。

（签名）图片包含 图示

描述已自动生成