attacklab 实验说明

1. 简介

本实验要求在两个有着不同安全漏洞的程序上实现五种攻击。通过完成本实验，能够学习：

- 当程序没有对缓冲区溢出做足够防范时，攻击者可能会如何利用这些安全漏洞。
- 通过实验，能更好的理解写出安全的程序的重要性，也能了解到一些编译器和操作系统提供的帮助改善程序安全性的特性。
- 能跟深入的理解 x86-64 机器代码的栈和参数传递机制。
- 能更深入的理解 x86-64 指令的编码方式。
- 能更熟练的使用 GDB 和 OBJDUMP 这样的调试工具。

注意：在本实验中，你能亲自动手体验利用操作系统和网络服务器中安全漏洞进行攻击的方法。我们的目的是帮助你学习程序的运行时操作，以及理解这些安全漏洞的本质，这样当你以后书写系统代码时能尽量避免这些漏洞。我们不容许任何使用其他形式攻击获取对系统资源未被授权的访问！

2. 实验步骤

2.1. 第一步：获取文件

解压 target.tar 文件（tar -xvf target.tar）得到一个目录./target，其中包含如下文件：

- README.txt：描述本目录内容的文件。
- 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，是你的唯一标识符，用于验证身份。
- farm.c：你的目标“gadget farm”的源代码，在产生 return-oriented programming 攻击时会用到。
- hex2raw：一个生成攻击字符串的工具。

2.2. 要点说明

- 要在我们提供的虚拟机上完成该实验，我们不保证在其他平台上作出的结果能在我们的验证平台上成功执行。
- 你的解答不能绕开程序中的验证代码。也就是说，ret 指令使用的攻击字符串中注入的地址必须是一下几种之一：
  - 函数 touch1，touch2 或 touch3 的地址
  - 你注入的代码的地址

- gadget farm 中 gadget 的地址

➢ 只能从文件 rtarget 中地址范围在函数 start_farm 和 end_farm 之间的地址构造 gadget。

3. 目标程序

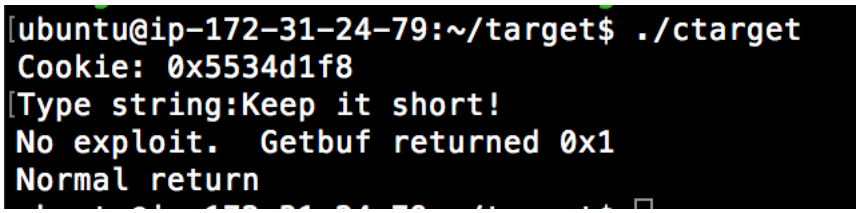
CTARGET 和 RTARGET 都是用 getbuf 函数从标准输入读入字符串，getbuf 函数定义如下：

```
1  unsigned getbuf()
2  {
3      char buf[BUFFER_SIZE];
4      Gets(buf);
5      return 1;
6  }
```

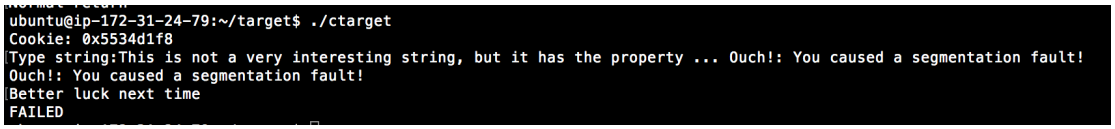
函数 Gets 类似于标准库函数 gets，从标准输入读入一个字符串（以'\n'或者 end-of-file 结束），将字符串（带 null 结束符）存储在指定的目的地址。从这段代码可以看出，目标地址是数组 buf，声明为 BUFFER_SIZE 个字节。BUFFER_SIZE 是一个编译时常量，在你的 target 程序生成时就具体确定了。

函数 Gets()和 gets()都无法确定目标缓冲区是否够大，能够存储下读入的字符串。它们都只会简单地拷贝字节序列，可能会超出目标地址处分配的存储空间的边界。

如果用户输入和 getbuf 读入的字符串足够短，getbuf 会返回 1，如下执行示例所示：



如果你输入的字符串很长，就会出错：



（注意 cookie 的值会每个人有所不同。）RTARGET 程序有类似的行为。正如错误消息提示的那样，超出缓冲区大小通常会导致程序状态被破坏，引起内存访问错误。你的任务是巧妙的设计输入给 CTARGET 和 RTARGET 的字符串，让它们做些更有趣的事情。这样的字符串称为*攻击（exploit）*字符串。

CTARGET 和 RTARGET 有这样一些命令行参数：

-h：输出可能的命令行参数列表

-q：不向打分服务器发送结果

-i FILE：输入来自于文件 FILE 而不是标准输入

一般来说，你的攻击字符串包含的字节值并不都对应着能够打印出来的字符的 ASCII 值。HEX2RAW 程序的使用见附录 A。

要点说明：

- 你的攻击字符串不能包含字节值 0x0a，这是换行符（'\n'）的 ASCII 代码。Gets 遇到这个字节时会认为你意在结束该字符串。
- HEX2RAW 要求输入的十六进制值必须是两位的，值与值之间以一个或多个空白分隔。如果你想得到一个十六进制值为 0 的字节，必须输入 00。要得到字 0xdeadbeef，必须向 HEX2RAW 输入“ef be ad de”（注意顺序相反是因为使用的是小端法字节序）。

一定要使用-q 选项，以免向并不存在的服务器发送消息。

本实验分为五个阶段，CTARGET 的三个使用的是 CI（code-injection），RTARGET 的两个阶段使用的是 ROP（return-oriented-programming），如图 1 所示。

阶　　段	程　　序	关　　数	方　　法	函　　数	分　　数
1	CTARGET	1	CI	touch1	10
2	CTARGET	2	CI	touch2	20
3	CTARGET	3	CI	touch3	20
4	RTARGET	2	ROP	touch2	35
5	RTARGET	3	ROP	touch3	15

图 1. attack lab 阶段小结

4. 实验内容第一部分：代码注入攻击

前三个阶段，你的攻击字符串会攻击 CTARGET 程序。程序被设置成栈的位置每次执行都一样，这样一来栈上的数据就可以等效于可执行代码。这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。

4.1. 第一关

在这一关中，你不用注入新的代码，你的攻击字符串要指引程序去执行一个已经存在的函数。

CTARGET 中函数 test 调用了函数 getbuf，test 的代码如下：

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

getbuf 执行返回语句时（getbuf 的第 5 行），按照规则，程序会继续执行 test 函数中的语句，而我们想改变这个行为。在文件 ctarget 中，函数 touch1 的代码如下：

```
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 }
```

你的任务是让 CTARGET 在 getbuf 执行返回语句后执行 touch1 的代码。注意，你的攻击字符串可以破坏栈中不直接和本阶段相关的部分，这不会造成问题，因为 touch1 会使得程序直接退出。

要点说明：

- 设计本阶段的攻击字符串所需的信息都从检查 CTARGET 的反汇编代码中获得。用 objdump -d 进行反汇编。
- 主要思路是找到 touch1 的起始地址的字节表示的位置，使得 getbuf 结尾处的 ret 指令会将控制转移到 touch1。
- 注意字节顺序。
- 可能需要用 GDB 单步跟踪调试 getbuf 的最后几条指令，确保它按照你期望的方式工作。
- buf 在 getbuf 栈帧中的位置取决于编译时常数 BUFFER_SIZE 的值，以及 GCC 使用的分配策略。你需要检查反汇编带来来确定它的位置。

4.2. 第二关

第二关中，你需要在攻击字符串中注入少量代码。

在 ctarget 文件中，函数 touch2 的代码如下：

```
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 }
```

你的任务是使 CTARGET 执行 touch2 的代码而不是返回到 test。在这个例子中，你必须让 touch2 以为它收到的参数是你的 cookie。

建议：

- 需要确定你注入代码的地址的字节表示的位置，使 getbuf 代码最后的 ret 指令会将控制转移到那里。

- 注意，函数的第一个参数是放在寄存器%rdi 中的。
- 你注入的代码必须将寄存器的值设定为你的 cookie，然后利用 ret 指令将控制转移到 touch2 的第一条指令。
- 不要在攻击代码中使用 jmp 或 call 指令。所有的控制转移都要使用 ret 指令，即使实际上你并不是要从一个函数调用返回。
- 参见附录 B 学习如何生成指令序列的字节级表示。

4.3. 第三关

第三阶段还是代码注入攻击，但是是要传递字符串作为参数。

ctarget 文件中函数 hexmatch 和 touch3 的 C 代码如下：

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

你的任务是让 CTARGET 执行 touch3 而不要返回到 test。要使 touch3 以为你传递你的 cookie 的字符串表示作为它的参数。

建议：

- 你的攻击字符串中要包含你的 cookie 的字符串表示。这个字符串由 8 个十六进制数字组成（顺序是从最高位到最低位），开头没有“0x”。
- 注意，C 中的字符串表示是一个字节序列，最后跟一个值为 0 的字节。“man ascii”能够找到你需要的字符的字节表示。
- 你的注入代码应该将寄存器%rdi 设置为攻击字符串的地址。

- 调用 hexmatch 和 strncmp 函数时，会将数据压入栈中，覆盖 getbuf 使用的缓冲区的内存，你需要很小心把你的 cookie 字符串表示放在哪里。

5. 实验内容第二部分：面向返回的编程

对程序 RTARGET 进行代码注入攻击要难一些，它采用了以下两种技术对抗攻击：

- 采用了随机化，每次运行栈的位置都不同。所以无法决定你的注入代码应该放在哪里。
- 将保存栈的内存区域设置为不可执行，所以即使你能把注入的代码的起始地址放到程序计数器中，程序也会报段错误失败。

幸运的是，聪明的人们设计了一些策略，通过执行现有程序中的代码来做他们期望的事情，而不是注入新的代码。这种方法称为面向返回的编程（ROP）。

例如，rtarget 可能包含如下代码：

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

这个函数不太可能会攻击到一个系统，但是这段代码反汇编出来的机器代码是：

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

字节序列 48 49 c7 是指令 movq %rax, %rdi 的编码。图 2A 展示了一些有用的 movq 指令的编码。你的 RTARGET 的攻击代码由一组类似于 setval_210 的函数组成，我们称为 gadget farm。你的工作是从 gadget farm 中挑选出有用的 gadget 执行类似于前述第二关和第三关的攻击。

要点说明：

函数 start_farm 和 end_farm 之间的所有函数构成了你的 gadget farm。不要用程序代码中的其他部分作为你的 gadget。

5.1. 第二关

在第四阶段，你将重复第二阶段的攻击，不过要使用 gadget farm 里的 gadget 来攻击 RTARGET 程序。你的答案只使用如下指令类型的 gadget，也只能使用前八个 x86-64 寄存器（%rax-%rdi）。

movq：代码如图 2A 所示。

popq：代码如图 2B 所示。

ret：该指令编码为 0xc3。

nop：该指令编码为 0x90。

建议：

- 只能用两个 gadget 来实现该次攻击。

- 如果一个 gadget 使用了 popq 指令，那么它会从栈中弹出数据。这样一来，你的攻击代码能既包含 gadget 的地址也包含数据。

A. Encodings of `movq` instructions

`movq` *S*, *D*

Source *S*	Destination *D*							
	%rax	%rcx	%rdx	%rbx	%rsp	%rbp	%rsi	%rdi
%rax	48 89 c0	48 89 c1	48 89 c2	48 89 c3	48 89 c4	48 89 c5	48 89 c6	48 89 c7
%rcx	48 89 c8	48 89 c9	48 89 ca	48 89 cb	48 89 cc	48 89 cd	48 89 ce	48 89 cf
%rdx	48 89 d0	48 89 d1	48 89 d2	48 89 d3	48 89 d4	48 89 d5	48 89 d6	48 89 d7
%rbx	48 89 d8	48 89 d9	48 89 da	48 89 db	48 89 dc	48 89 dd	48 89 de	48 89 df
%rsp	48 89 e0	48 89 e1	48 89 e2	48 89 e3	48 89 e4	48 89 e5	48 89 e6	48 89 e7
%rbp	48 89 e8	48 89 e9	48 89 ea	48 89 eb	48 89 ec	48 89 ed	48 89 ee	48 89 ef
%rsi	48 89 f0	48 89 f1	48 89 f2	48 89 f3	48 89 f4	48 89 f5	48 89 f6	48 89 f7
%rdi	48 89 f8	48 89 f9	48 89 fa	48 89 fb	48 89 fc	48 89 fd	48 89 fe	48 89 ff

B. Encodings of `popq` instructions

Operation	Register *R*							
	%rax	%rcx	%rdx	%rbx	%rsp	%rbp	%rsi	%rdi
popq *R*	58	59	5a	5b	5c	5d	5e	5f

C. Encodings of `movl` instructions

`movl` *S*, *D*

Source *S*	Destination *D*							
	%eax	%ecx	%edx	%ebx	%esp	%ebp	%esi	%edi
%eax	89 c0	89 c1	89 c2	89 c3	89 c4	89 c5	89 c6	89 c7
%ecx	89 c8	89 c9	89 ca	89 cb	89 cc	89 cd	89 ce	89 cf
%edx	89 d0	89 d1	89 d2	89 d3	89 d4	89 d5	89 d6	89 d7
%ebx	89 d8	89 d9	89 da	89 db	89 dc	89 dd	89 de	89 df
%esp	89 e0	89 e1	89 e2	89 e3	89 e4	89 e5	89 e6	89 e7
%ebp	89 e8	89 e9	89 ea	89 eb	89 ec	89 ed	89 ee	89 ef
%esi	89 f0	89 f1	89 f2	89 f3	89 f4	89 f5	89 f6	89 f7
%edi	89 f8	89 f9	89 fa	89 fb	89 fc	89 fd	89 fe	89 ff

D. Encodings of 2-byte functional nop instructions

Operation	Register *R*			
	%al	%cl	%dl	%bl
andb *R*, *R*	20 c0	20 c9	20 d2	20 db
orb *R*, *R*	08 c0	08 c9	08 d2	08 db
cmpb *R*, *R*	38 c0	38 c9	38 d2	38 db
testb *R*, *R*	84 c0	84 c9	84 d2	84 db

图 2. 指令的字节编码。所有的值均为十六进制。

5.2. 第三关

阶段五要求你对 RTARGET 程序进行 ROP 攻击，用指向你的 cookie 字符串的指针，使程序调用 touch3 函数。

这一关，允许你使用函数 start_farm 和 end_farm 之间的所有 gadget。除了第四阶段允许的那些指令，还允许使用 movl 指令（如图 2C 所示），以及图 2D 中的 2 字节指令，它们可以作为有功能的 nop，不改变任何寄存器或内存的值，例如，andb %al, %al，这些指令对寄存器的低位字节做操作，但是不改变它们的值。

提示：

- 官方答案需要 8 个 gadgets。

附录 A HEX2RAW 的使用

HEX2RAW 的输入是一个十六进制格式的字符串，用两个十六进制数字表示一个字节值。例如，字符串“012345”，必须输入“30 31 32 33 34 35 00”。十六进制字符之间以空白符（空格或新行）分隔。

可以把攻击字符串存入文件中，例如 exploit.txt，以下列几种方式调用：

1. unix> *cat exploit.txt | ./hex2raw | ./ctarget*

2. unix> *./hex2raw < exploit.txt > exploit-raw.txt*

   unix> *./ctarget < exploit-raw.txt*

   这种方法也可以结合 gdb 使用。

   unix> *gdb ctarget*

   (gdb) *run < exploit-raw.txt*

3. unix> *./hex2raw < exploit.txt > exploit-raw.txt*

   unix> *./ctarget -i exploit-raw.txt*

   这种方法也可以和 gdb 一起使用。

附录 B 生成字节代码

假设编写一个汇编文件 example.s，代码如下：

```
# Example of hand-generated assembly code
pushq    $0xabcdef        # Push value onto stack
addq     $17,%rax         # Add 17 to %rax
movl      %eax,%edx       # Copy lower 32 bits to %edx
```

可以汇编和反汇编文件：

```
unix> gcc -c example.s
unix> objdump -d example.o > example.d
```

生成的 example.d 包含如下内容：

```
example.o: file format elf64-x86-64

Disassembly of section .text:
0000000000000000 <.text>:
   0: 68 ef cd ab 00          pushq   $0xabcdef
   5: 48 83 c0 11             add     $0x11,%rax
   9: 89 c2                   mov      %eax,%edx
```

由此可以推出这段代码的字节序列：

68 ef cd ab 00 48 83 c0 11 89 c2

可以通过HEX2RAW生成目标程序的输入字符串。也可以手动修改example.d的代码，得到下面的内容：

```
    68 ef cd ab 00     /* pushq    $0xabcdef    */
    48 83 c0 11        /* add      $0x11,%rax */
    89 c2              /* mov      %eax,%edx    */
```

这也是合法的HEX2RAW的输入。