计算机组成原理课后作业4

1. 目标

进一步理解代码注入攻击

2. 背景资料

程序ctarget是一个具有缓冲区溢出缺陷的程序。在执行改程序时，会调用getbuf函数。

```
unsigned getbuf()
{
  char buf[BUFFER_SIZE];
  Gets(buf);
  return 1;
}
```

其中Gets函数和C语言标准库中的gets函数功能相同，从标准输入中读入字符串，遇到'\n'或读入至文件结尾停止，并将数据存入缓冲区。因此Gets函数有缓冲区溢出缺陷。BUFFER_SIZE是一个常量，需要你对代码进行分析去确定它的值。

当ctarget程序执行时，如果从输入较短的字符串，如输入 Keep it short，程序将正常返回，不产生缓冲区溢出。此时getbuf函数返回1。并在控制台上有如下输出：

```
> ./ctarget -q
Cookie: 0x59b997fa
Type string: Keep it short!
No exploit. Getbuf returned 0x1
Normal return
```

当输入字符串较长时，产生缓冲区溢出，将会在控制台上看到如下信息:

```
> ./ctarget -q
Cookie: 0x59b997fa
Type string: This is not a very interesting string, but it has the property ...
Ouch!: You caused a segmentation fault!
Better luck next time
```

注意：输入字符中不能出现值为0xa的字节，因为0xa是'\n'的ASCII码值，当Gets函数遇到0xa就会返回，不会继续读入后面的数据。

程序hex2raw是一个工具，由于有些数据并不能以可见文本的方式通过在控制台输入至ctarget，这是可以使用hex2raw工具，利用Linux中的管道机制，实现非可见文本的输入。例如将ctarget.l2.txt文件中的数据作为输入至ctarget的标准输入中。

```
> ./hex2raw < ctarget.l2.txt | ./ctarget -q
```

hex2raw的输入文件的格式如下：

```
48 c7 c1 f0 11 40 00 /* mov $0x40011f0,%rcx */
```

使用两个字符表示一个字节的输入数据对应的十六进制编码，每个字节以空格作为分隔符。支持C语言的 /* */ 注释，注释中的所有文字会被忽略。在上例中，输入文件所表示的输入数据为7个字节，分别为0x48, 0xc7, 0xc1, 0xf0, 0x11, 0x40, 0x00。

3. 内容

你需要进行3次代码注入攻击。

任务一

在这次任务中，你不需要注入任何代码，只需要利用缓冲区溢出漏洞，实现程序控制流的重定向。具体要求如下：

在ctarget中，会执行test函数，test函数内部会调用getbuf函数。test函数实现如下：

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

你需要利用getbuf函数的漏洞，使getbuf函数返回时，不返回test函数，而是跳转至touch1函数。touch1函数实现如下：

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

任务完成条件：当控制台上有touch1函数printf输出和正确的任务通过信息显示时，说明你的任务一已经正确完成。

提示：

1. 可以使用objdump -d命令对ctarget进行反汇编。
2. 通过缓冲区溢出的手段，修改getbuf的返回地址，将返回地址修改为touch1函数的入口地址。
3. 注意字节序问题（小端序）。
4. 可以使用gdb查看运行时栈的情况。
5. 需要首先确定BUFFER_SIZE的值。

任务二

在这次任务中，你需要注入少量代码，利用缓冲区溢出漏洞，实现程序控制流的重定向至touch2函数，并进入touch2函数的validate分支。touch2函数的实现如下：

```
void touch2(unsigned val)
{
  vlevel = 2; /* Part of validation protocol */
  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);
}
```

变量cookie的值为0x59b997fa。

提示：

1. 需要进行代码注入攻击，在getbuf返回时，通过修改返回地址，使程序跳转至你注入的代码所在的地址。
2. 需要正确设置touch2的传入参数val，才能确保进入validate分支。根据函数调用规范，函数的第一个参数存储在%rdi寄存器中。
3. 使用ret指令实现控制流的跳转。将跳转的目标地址首先置入栈顶，然后执行ret指令。
4. 不要尝试使用jmp或call指令实现跳转，因为很难生成一条完整地带有目标地址的jmp或call指令。ret指令实现跳转是一种最容易实现的方式。
5. 获取机器指令所对应的二进制数据的方法请见指导书的第4部分《如何生成字节码》中的说明

任务三

在这次任务中，你需要注入少量代码，利用缓冲区溢出漏洞，实现程序控制流的重定向至touch3函数，并进入touch3函数的validate分支。touch3函数的实现如下：

```
void touch3(char *sval)
{
  vlevel = 3; /* Part of validation protocol */
  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);
}
```

其中hexmatch函数实现如下：

```
int hexmatch(unsigned val, char *sval)
{
  char cbuf[110];
  /* Make position of check string unpredictable */
  char *s = cbuf + random() % 100;
  sprintf(s, "%.8x", val);
  return strncmp(sval, s, 9) == 0;
}
```

在touch3包含一个参数，输入一个字符串，通过hexmatch对字符串进行比对，满足条件则进入validate分支。

提示：

1. 需要构造一个字符串数据作为参数传入hexmatch中，具体的字符串内容需参考hexmatch中的sprintf函数。
2. 字符串需要有串尾符'\0',即字节的值为0.
3. 当hexmatch和strncmp函数被调用时，会有数据入栈。这会导致原getbuf所在的位置的内存数据被覆盖，因此需要仔细考虑注入的字符串的存储位置。一旦注入的字符串的数据被覆盖，攻击将失败。

4. 如何生成字节码

使用gcc编译工具和objdump反汇编工具可以实现将某一条指令转换为字节码。例如：

1. 编写一小段汇编程序，并存储至exapmle.s文件中，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
```

2. 使用gcc命令编译成.o文件，并使用objdump反编译。

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

3. 在example.d文件中，则可以看到反编译后的字节码：

```
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
```

如上, 指令pushq $0xabcdef所对应的字节码为 68 ef cd ab 00；add $0x11,%rax对应的字节码为48 83 c0 11。这些字节码就可以作为hex2raw的输入，实现对ctarget的代码注入。

5 一些说明

为了保证ctarget程序可以正确运行，需要在运行时加入-q选项。

6 作业提交要求

提交一个报告，在报告中给出攻击过程的详细描述和和攻击结果的运行时截图。并讨论每一个过程攻击时所应用的基本原理。

注意：任务一和任务二是必做任务；任务三为选做，有加分。