计算机系统基础 Lab3 实验报告

1. 实验内容:

   完成 AttackLab。

2. 实验步骤:

   1. 阅读 Handout 中的文档。
   2. 链接至服务器并下载题目。
   3. 将题目下载至本地。
   4. 在本地对题目进行分析，在服务器对程序进行调试与攻击。
   5. 撰写实验报告。

3. 解题步骤:

先使用 ida 与 objdump 指令获得程序的源代码与汇编代码，为程序分析构造基本条件。

1. Ctarget

对源代码进行分析，发现 test 函数调用了 getbuf 函数,即为为待攻击函数，应对其进行缓冲区溢出攻击。分别调用 touch1，touch2，touch3 函数并完成其要求。

```
1 void __cdecl test()
2 {
3   unsigned int v0; // eax
4
5   v0 = getbuf();
6   printf("No exploit.  Getbuf returned 0x%x\n", v0);
7 }
```

```
1 unsigned int __cdecl getbuf()
2 {
3   char buf[16]; // [rsp+0h] [rbp-18h] BYREF
4
5   Gets(buf);
6   return 1;
7 }
```

Touch1：只是简单的缓冲区溢出，使用 touch1 函数地址覆盖原返回地

址即可。

Touch2：阅读反编译代码，发现 touch2 函数需要接受 cookie 值作为参数才能正常工作。因此我们需要修改 rdi 寄存器为 cookie 值。直接代码注入 mov rdi，cookie 与 ret 的机器码，并将返回地址修改为在 gdb 过程中获得的栈地址即可成功过关。

```
void __fastcall __noreturn touch2(unsigned int 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);
}
```

Touch3：读反编译代码，发现 touch2 函数需要接受 cookie 值字符串的地址作为参数才可正常工作。因此我们先将 cookie 值存储进入内存，再修改 rdi 寄存器为其地址。阅读指导 pdf 提示，发现若将 cookie 字符串存储于栈上会导致调用 strcmp 函数时发生传参错误，故将 cookie 字符串保存于 test 函数中，使用 touch2 相同的手法将 rdi 寄存器修改为字符串保存地址即可正常调用 touch3 函数。

```
void __fastcall __noreturn 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);
}
```

2. Rtarget

对文件进行反编译，发现程序内容基本与 ctarget 保持一致，但是该程序通过禁止栈上指令执行与随机栈地址等手段进行了保护，原

ctarget 中调用 touch2，touch3 的方式不再适用。阅读 pdf，发现我们应该在 gedget farm 中寻找代码片段并依次调用以实现目标。

Touch2：依据提示从 start_farm 至 mid_farm 间寻找 gedget 以实现将输入内容（cookie）导入至 rdi 寄存器中。最直接的思路为将输入内容 pop 到 rdi 寄存器中，但对 gedget 的寻找未果，因此先将输入内容 pop 至 rax 寄存器中，再将 rax 寄存器移动至 rdi 寄存器中，从而成功达成目标。

Touch3：依据提示从 start_farm 至 end_farm 间寻找 gedget 以实现将输入内容（cookie）的地址导入至 rdi 寄存器中。但由于栈地址随机，因此需选用间接寻址法。发现仅有 lea (%rdi,%rsi,1),%rax 指令满足要求，故需要分别将 cookie 相对地址与 rsp 所标示的地址传入 rdi，rsi 寄存器，经过查找均无一步到位的代码片段，故选择适当寄存器进行中继。最终选择以下代码片段

```
118 0000000000401a30 <setval_436>:
119   401a30: c7 07 48 89 e0 90     movl   $0x90e08948,(%rdi)      401a32: mov rsp,
      rax         ///
120   401a36: c3                    retq
121 0000000000401972 <addval_130>:
122   401972: 8d 87 48 89 c7 c3     lea    -0x3c3876b8(%rdi),%eax   401974: mov rax,
      rdi         ///
123   401978: c3                    retq
124
125 000000000040196b <setval_301>:
126   40196b: c7 07 6f e1 58 90     movl   $0x9058e16f,(%rdi)      40196f: pop rax
127   401971: c3                    retq                                    0x48
128
129 00000000004019de <addval_212>:
130   4019de: 8d 87 50 89 c1 90     lea    -0x6f3e76b0(%rdi),%eax   4019e1: mov eax,
      ecx         ///
131   4019e4: c3                    retq
132 00000000004019d0 <addval_200>:
133   4019d0: 8d 87 09 3b 89 ca     lea    -0x3576c4f7(%rdi),%eax     4019d4: mov ecx,
      edx         ///
134   4019d6: c3                    retq
135 00000000004019ad <setval_216>:
136   4019ad: c7 07 89 d6 90 90     movl   $0x9090d689,(%rdi)     4019af: mov edx,esi
137   4019b3: c3                    retq
138
139 00000000004019a8 <add_xy>:
140   4019a8: 48 8d 04 37           lea    (%rdi,%rsi,1),%rax     you surely want this
141   4019ac: c3                    retq
142 0000000000401972 <addval_130>:
143   401972: 8d 87 48 89 c7 c3     lea    -0x3c3876b8(%rdi),%eax   401974: mov rax, rdi
144   401978: c3                    retq
```

（如图所示），并分别将其地址，偏移量，cookie 的 ascii 码，touch3 地址作为内容进行缓冲区攻击，（其中偏移量为 cookie 值以前所输入的其余内容长度）从而实现目标。

4. 实验收获与心得:

收获：初步掌握了缓冲区攻击的常见方法及手段，对相关汇编函数与其机器码进行了了解。

心得：应熟练掌握 ida 等高级工具，但同时也应兼顾 objdump 等初级指令。还应熟练掌握基于 linux 的命令行指令与 nano 文本编辑器。