PWN!

- 什么是pwn?
  - 利用程序中存在的漏洞，攻击和控制对方服务器
  - 模拟真实情况，较难的一类题目

- 常见漏洞
  - 栈漏洞
  - 堆漏洞
  - Libc泄漏
  - 整数溢出
  - 格式化字符串
  - 数组越界读写

- 基础知识
  - 汇编语言（X86, X64）
  - 编译原理
  - gdb,IDA等工具的使用
  - Python等脚本语言
  - 常用命令：ssh，nc

Ⅰ：汇编简介&gdb的简单使用

GDB: The GNU Project Debugger

- gdb ./file_name 启动gdb并对file_name文件进行调试
- disas func_name 对func_name函数进行反汇编
- b 下断点
- r 运行程序
- n 执行下一语句（不进入函数）
- s 执行下一语句（进入函数）
- c 继续运行程序到下一断点
- x 查看内存
- q退出
- set disassembly-flavor intel
- set disassembly-flavor att

32位汇编语言：Assembly language

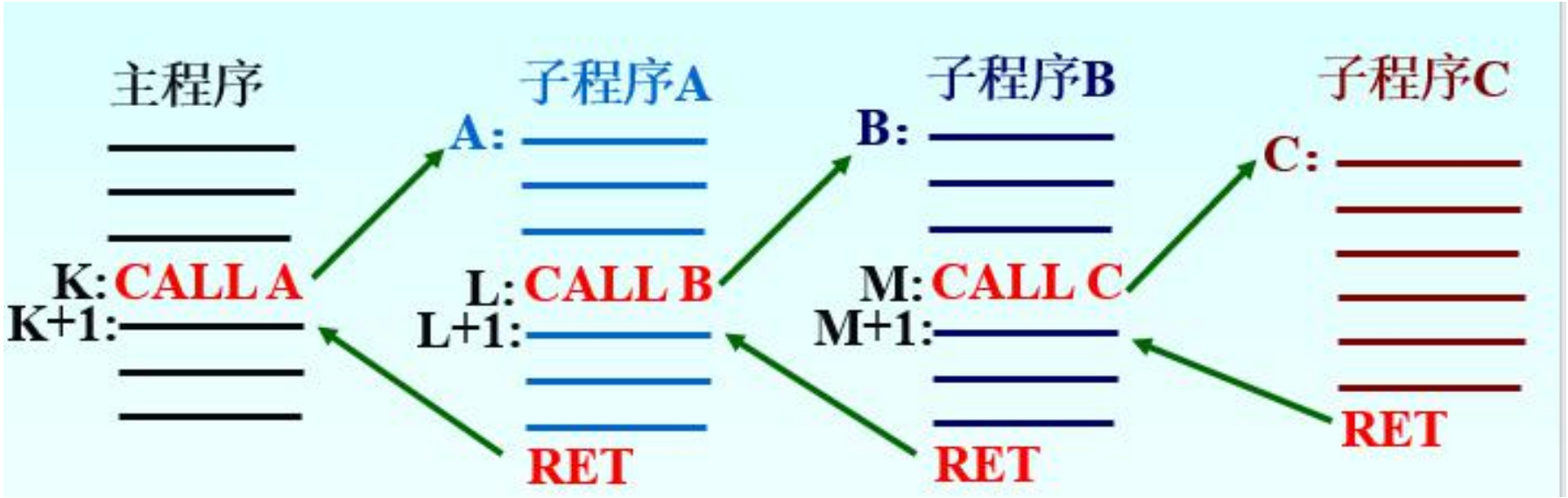
- call 调用函数，将下一条指令压入栈，然后跳转到被调函数
- push 压栈操作
- cmp 比较指令
- jmp, jne, je 跳转
- mov eax, 0xffffffff Intel汇编中，将0xffffffff赋给eax寄存器
- esp 栈指针寄存器，指向栈顶
- ebp 基址指针寄存器，指向栈底
- eip 指向下一条指令

```
test   eax,eax
jne    0x8048539 <func+78>
```

II. 编译原理:

函数的调用约定:

- 栈帧
- 函数传递参数的方式



如果在函数A中调用了函数B，我们称函数A为主调函数，函数B为被调函数，如果函数B的声明为int B (int arg1, int arg2, int arg3)，那么函数A中的调用函数B时的汇编指令的形式如下：

push arg3
push arg2
push arg1
call B

连续三个push将函数的参数按照**从右往左**的顺序进行压栈，然后执行call B来调用函数B。注意在gdb中看到的效果可能不是三个push，而是三个mov来对栈进行操作，这是因为**Linux采用AT&T风格的汇编**，而上面的指令使用的是Intel风格的汇编，比较容易理解。

```
...

F1( arg1, arg2 );

...
```

```
    push arg2

    push arg1

    call F1
```

```
void F1( arg1, arg2 ) {

  char buffer[8];

  ...

}
```

```
    push ebp

    mov ebp, esp

    sub esp, 8

    ...
```

Function Call

```
...

F1( arg1, arg2 );

...

    push arg2

    push arg1

    call F1
```

Function Call

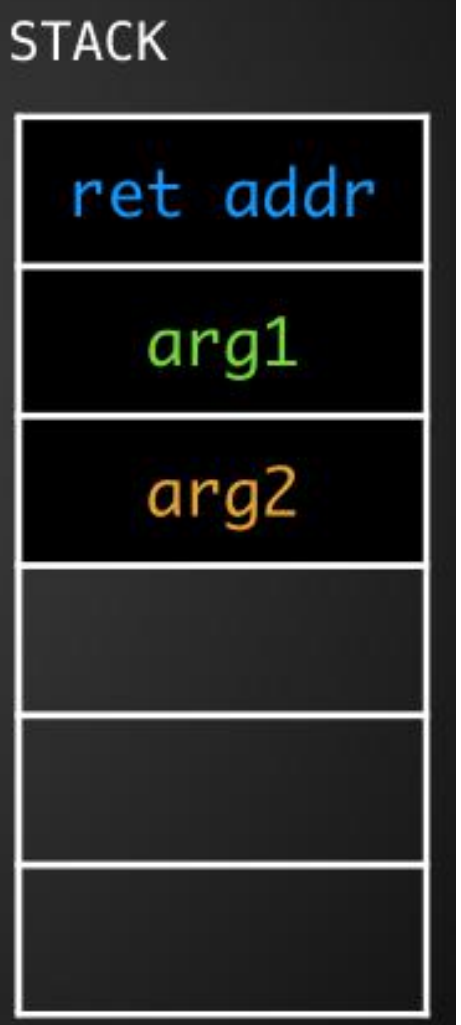
```
...

F1( arg1, arg2 );

...

    push arg2

    push arg1

    call F1
```

Function Call

```
...

F1( arg1, arg2 );

...

    push arg2

    push arg1

    call F1
```

Function Call

```
...

F1( arg1, arg2 );

...

   push arg2

   push arg1

   call F1
```



Function Call

```
void F1( arg1, arg2 ) {
   char buffer[8];
   ...
}
```

```
    push ebp
    mov ebp, esp
    sub esp, 8
    ...
```

Function Call

```
void F1( arg1, arg2 ) {
   char buffer[8];
   ...
}
```

```
   push ebp
   mov ebp, esp
   sub esp, 8
   ...
```

Function Call

```
void F1( arg1, arg2 ) {
   char buffer[8];
   ...
}
```

```
   push ebp
   mov ebp, esp
   sub esp, 8
   ...
```

Function Call

```
void F1( arg1, arg2 ) {
   char buffer[8];
   ...
}
```

```
   push ebp
   mov ebp, esp
   sub esp, 8
   ...
```

Function Call

```
void F1( arg1, arg2 ) {
   char buffer[8];
   ...
}
```

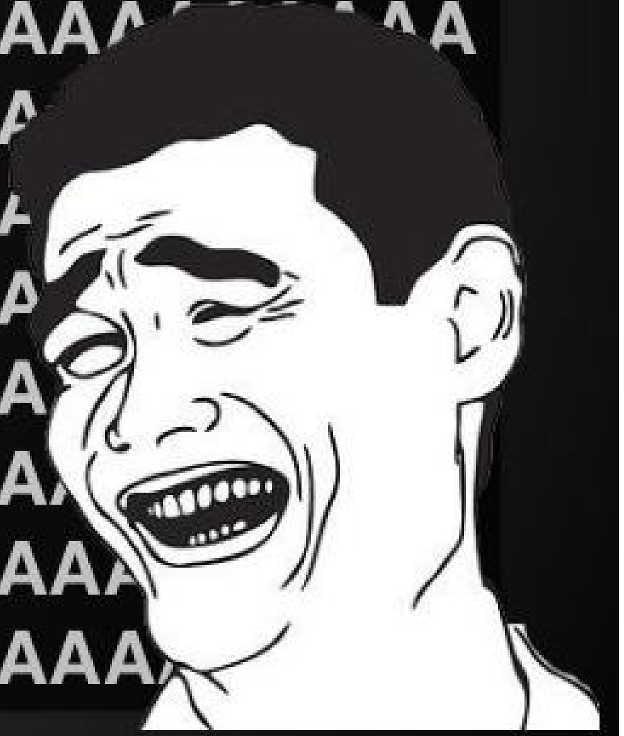
```
   push ebp
   mov ebp, esp
   sub esp, 8
   ...
```

Buffer Overflow

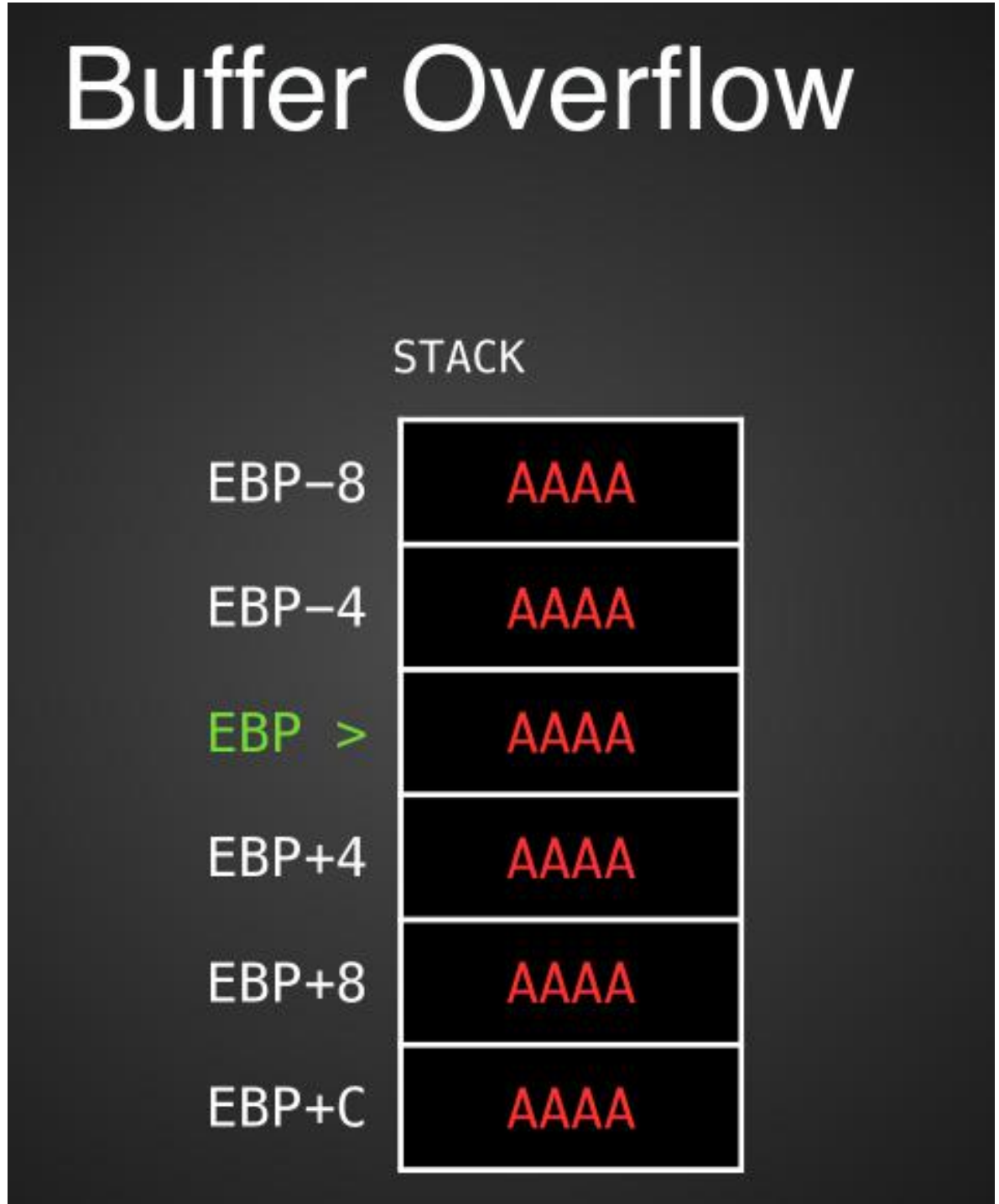
```
void F1( arg1, arg2 ) {
   char buffer[8];
   ...
   scanf( “%s”, buffer );
   ...
}
```

Buffer Overflow

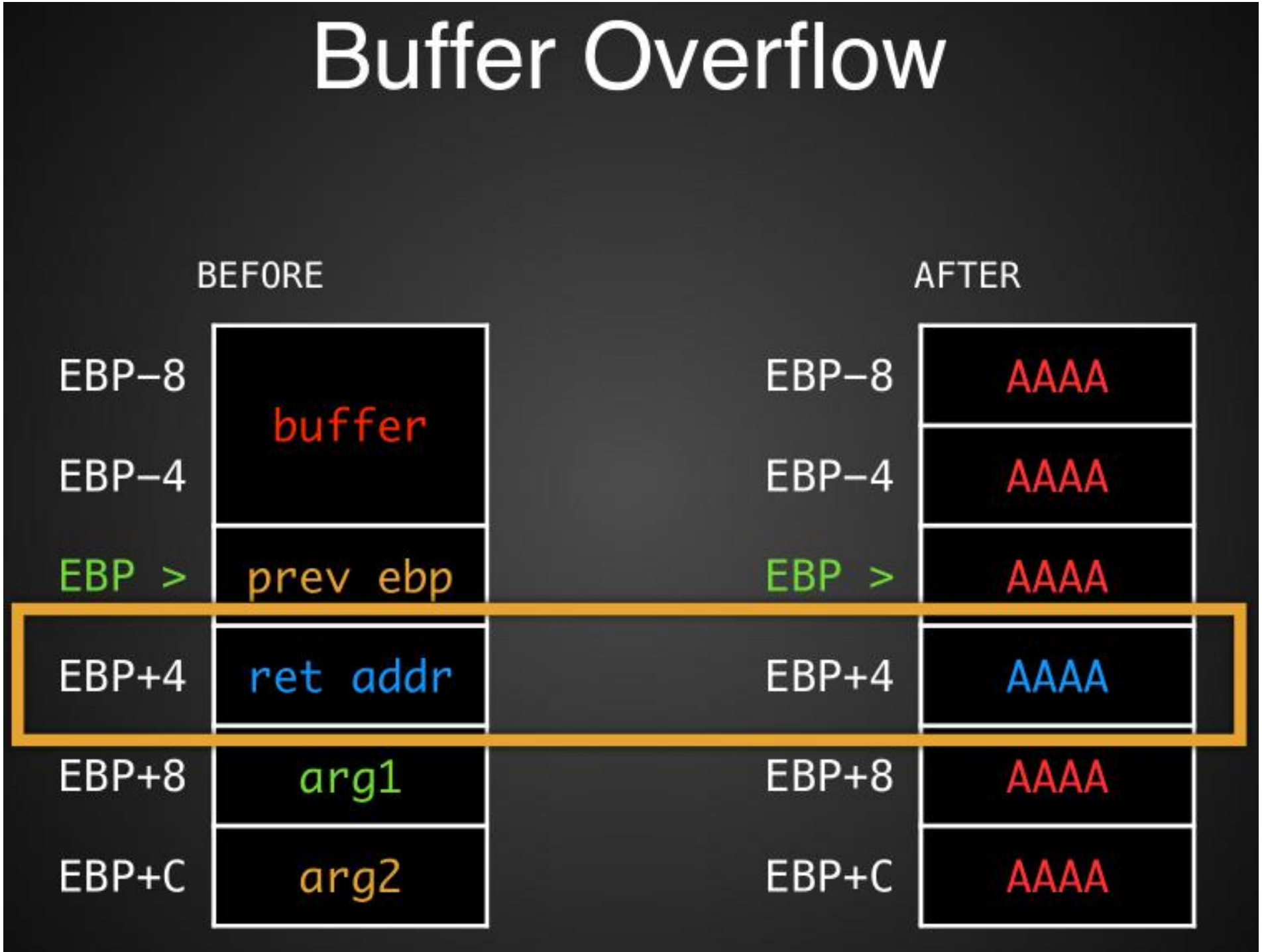
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA



Buffer Overflow



Buffer Overflow



作业：

Ⅰ. 学习GDB的使用

Ⅱ. 通过分析汇编代码尝试还原程序逻辑

Ⅲ. 查阅了解大小端字节序相关知识

Ⅲ. Sniper Oj pwn bof-x86-64