

课程实践报告



**课程 网络攻防技术(314006040)**

**课 序 号 3**

**作业名称 缓冲区溢出攻击实验**

**评分**

**姓名** 邓嘉怡 **学号** 2022141530010

**评阅意见**

# 作业题目

本实验的学习目标是让学生将从课堂上学到的有关缓冲区溢出漏洞的知识进行实践，从而获得有关该漏洞的第一手经验。缓冲区溢出是指程序试图将数据写入预先分配的固定长度缓冲区边界之外的情况。恶意用户可利用此漏洞改变程序的流控制，甚至执行任意代码。此漏洞是由于数据存储（如缓冲区）和控件存储（如返回地址）的混合造成的：数据部分的溢出会影响程序的控制流，因为溢出会改变返回地址。

本实验将提供四台不同的服务器，每台服务器运行一个带有缓冲区溢出漏洞的程序。实验任务是开发一个利用漏洞的程序，并最终获得这些服务器上的root权限。除了进行这些攻击实验之外，还将试验几种针对缓冲区溢出攻击的对策。学生需要评估这些计划是否有效，并解释原因。

# 实验步骤

环境准备

1. 学习基本docker指令

docker-compose build # Build the container image

docker-compose up # Start the container

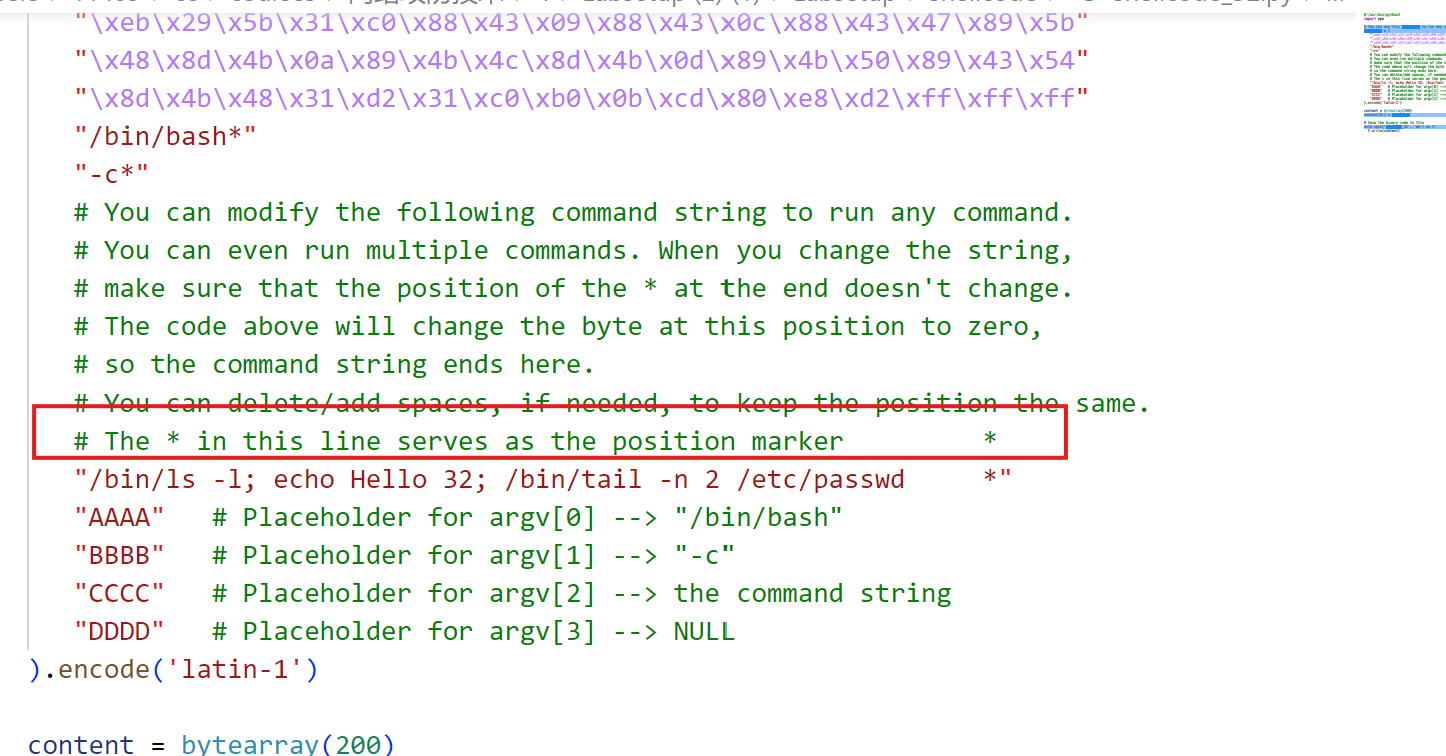
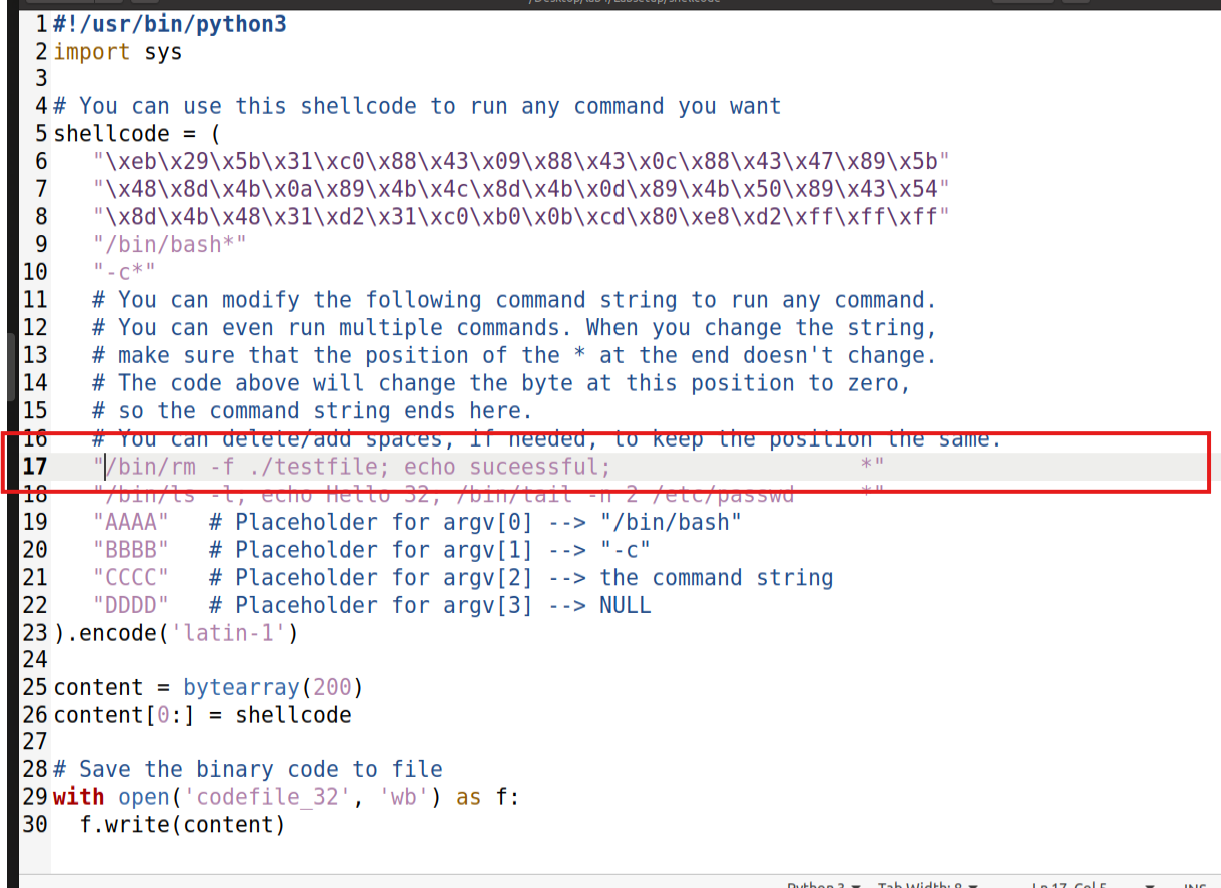
docker-compose down # Shut down the container

dcbuild # Alias for: docker-compose build

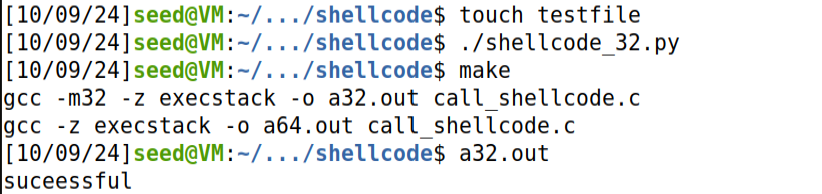
dcup # Alias for: docker-compose up

dcdown # Alias for: docker-compose down

任务 1：熟悉 shellcode

1. 观察代码  
     
   唯一可供修改的位置在这一行
2. 修改代码让他实现文件删除  
     
     
   要对齐\*的位置  
   

/bin/rm -f ./testfile; echo suceessful;

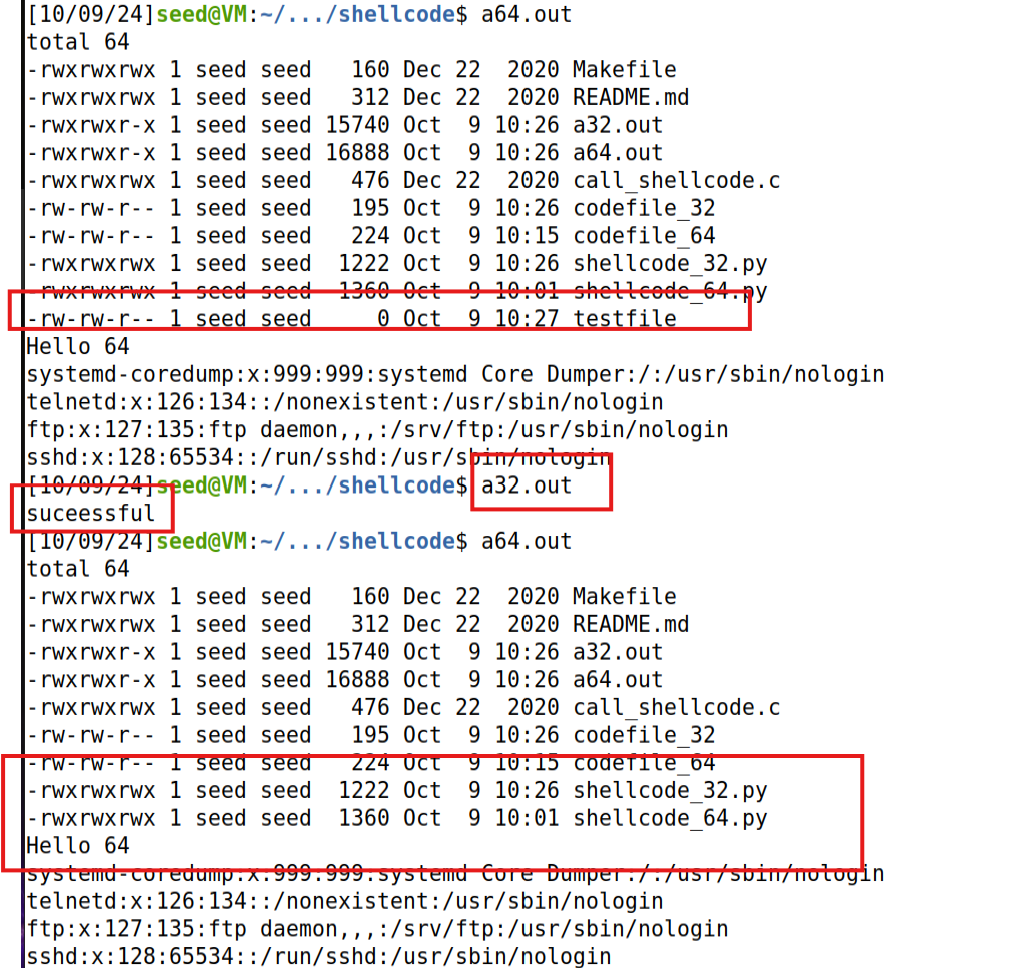
1. 尝试新建文件并运行shellcode  
   

touch testfile //创建测试文件

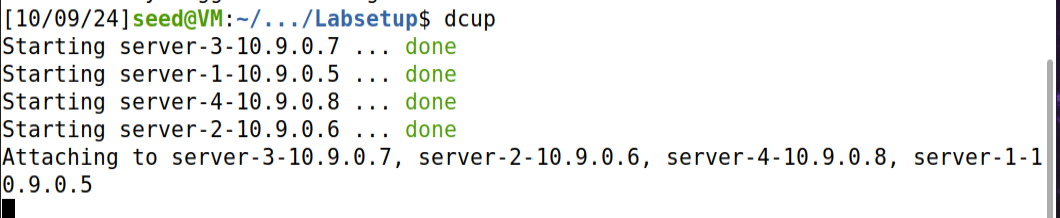
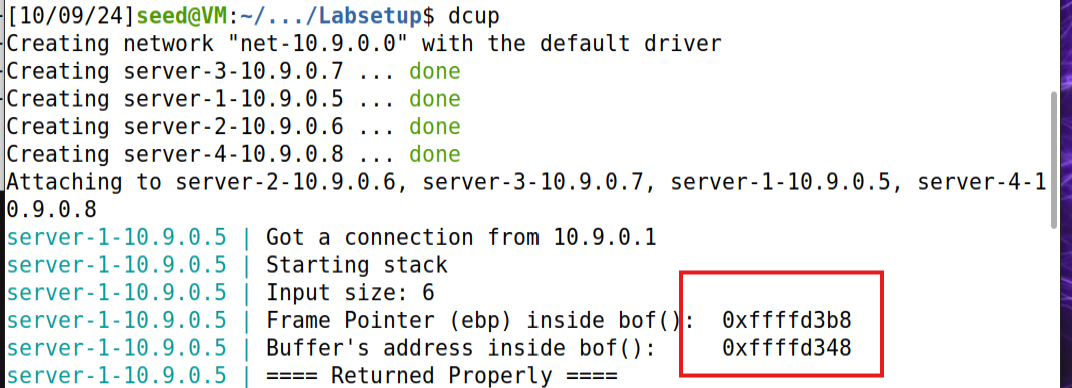
./shellcode\_32.py //运行shell

make

a32.out

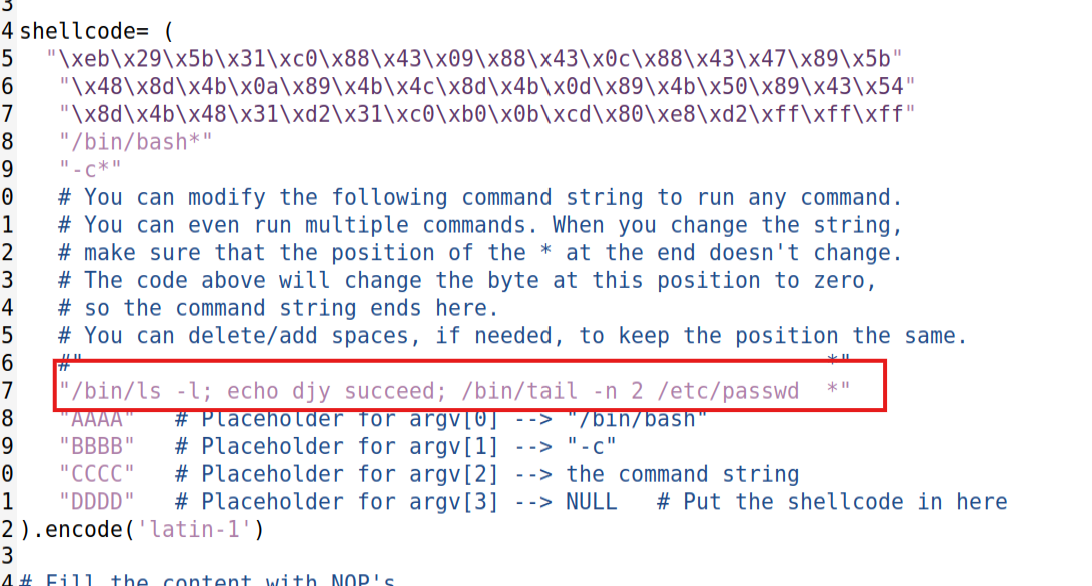
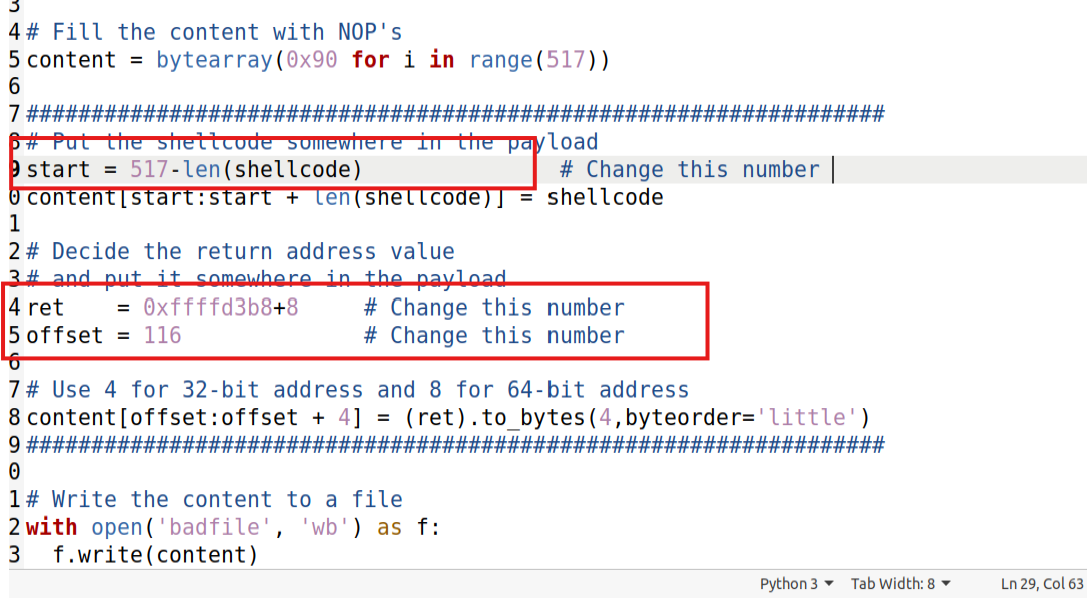
1. 运行成功  
     
   这时候用a64来验证  
   在运行完a32.out后，通过a64中的ls语句，发现testfile确实被删除了

任务二：level-1 攻击

1. 环境配置  
   首先启动系docker环境  
     
   关闭安全防护并重启docker  
     
     
     
   

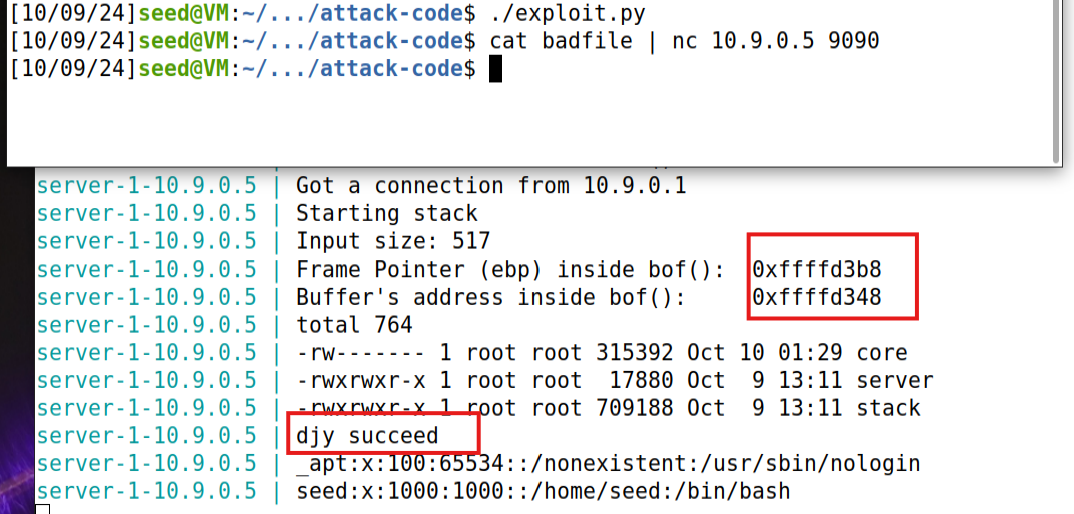
sudo /sbin/sysctl -w kernel.randomize\_va\_space=0

dcup

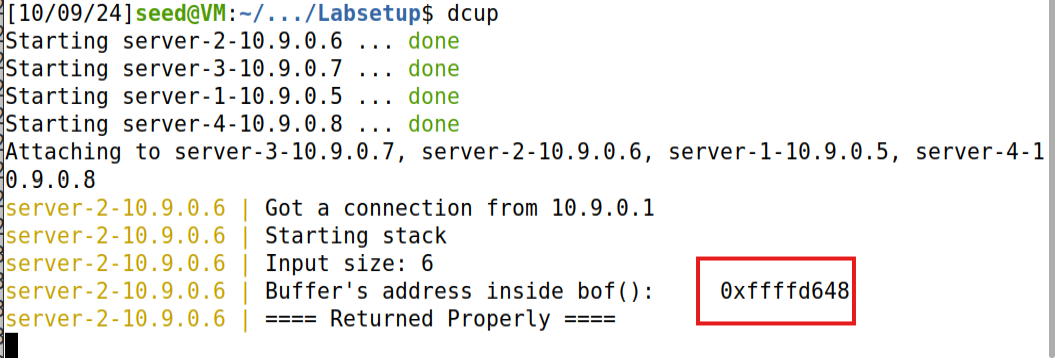
1. 修改exploit.py代码  
   修改shellcode并设定输出语句为ls -l,回应 djy succeed  
     
     
   

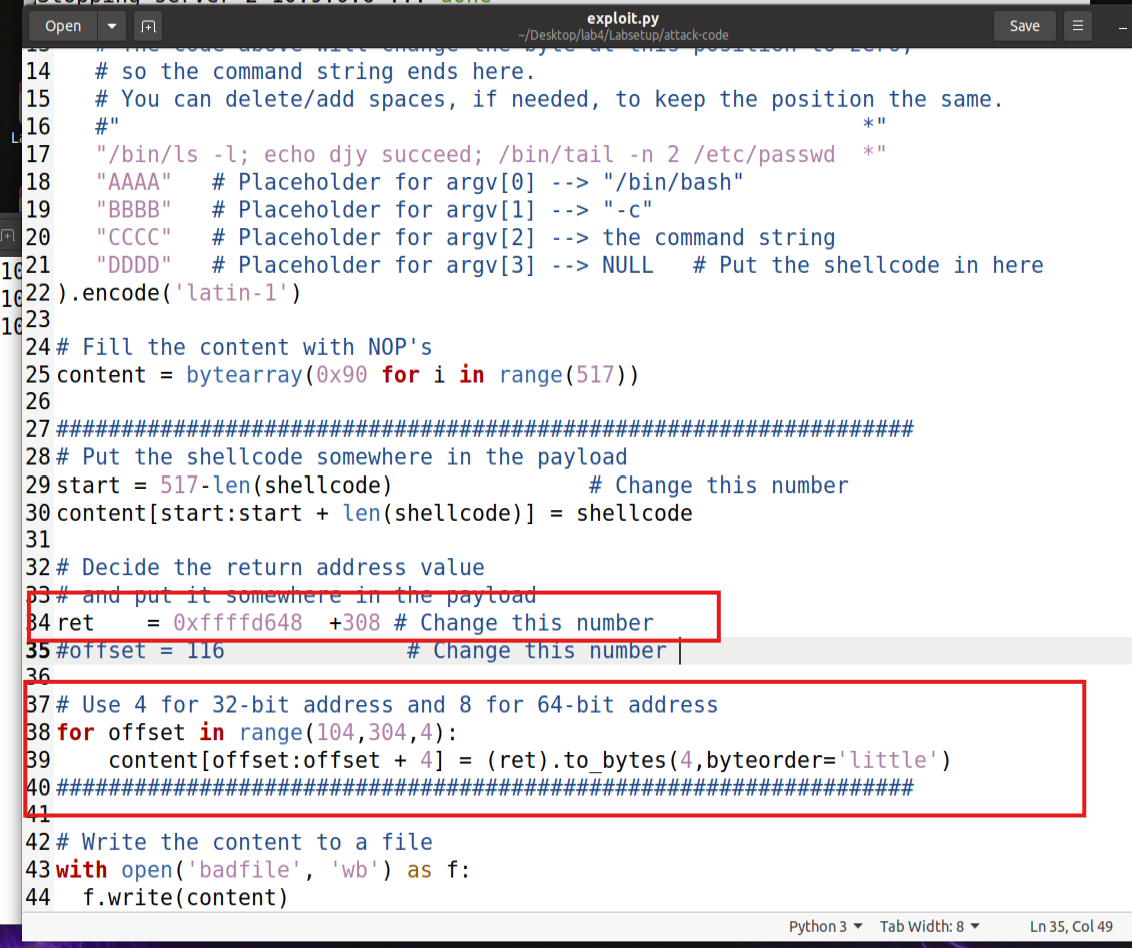
将shellcode放置在字节序列末尾（设置start长度是517-shellcode）  
设置返回地址和偏移量，返回地址在结束地址上面八位的位置

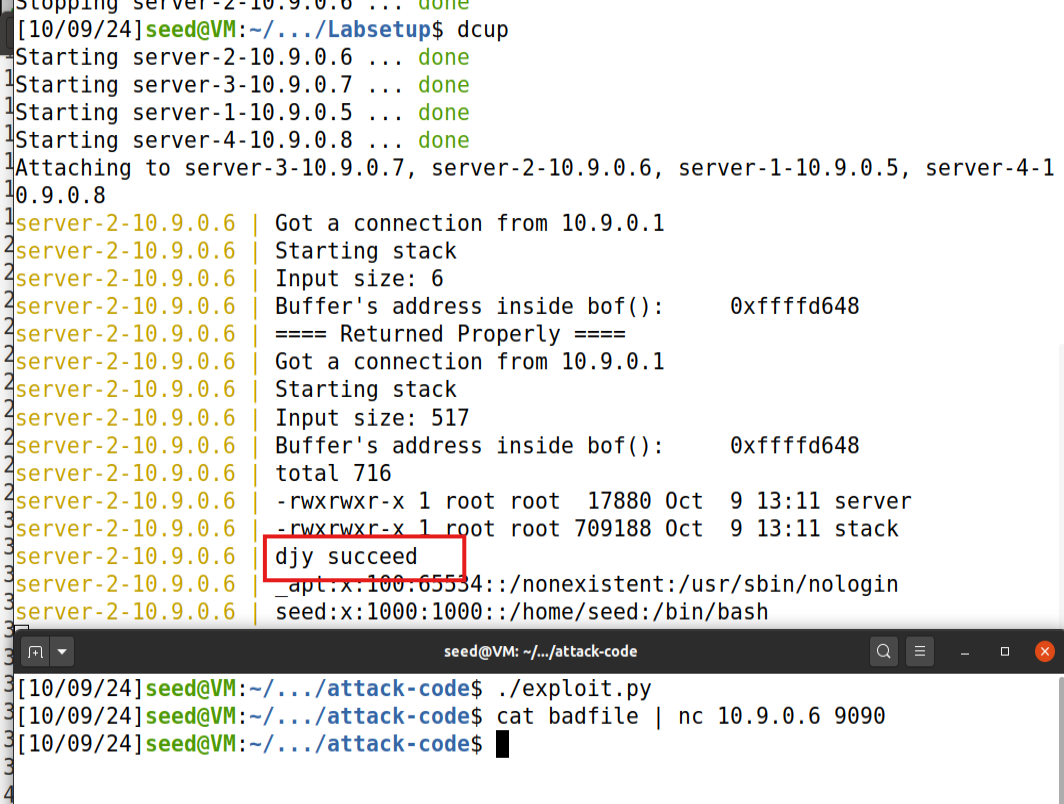
Offset要+4：因为ebp占用空间

1. 输出成功  
   

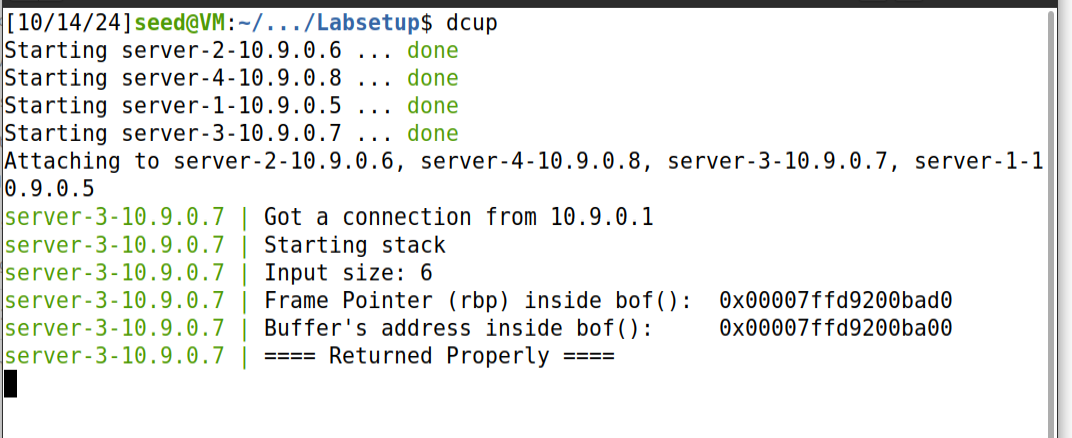
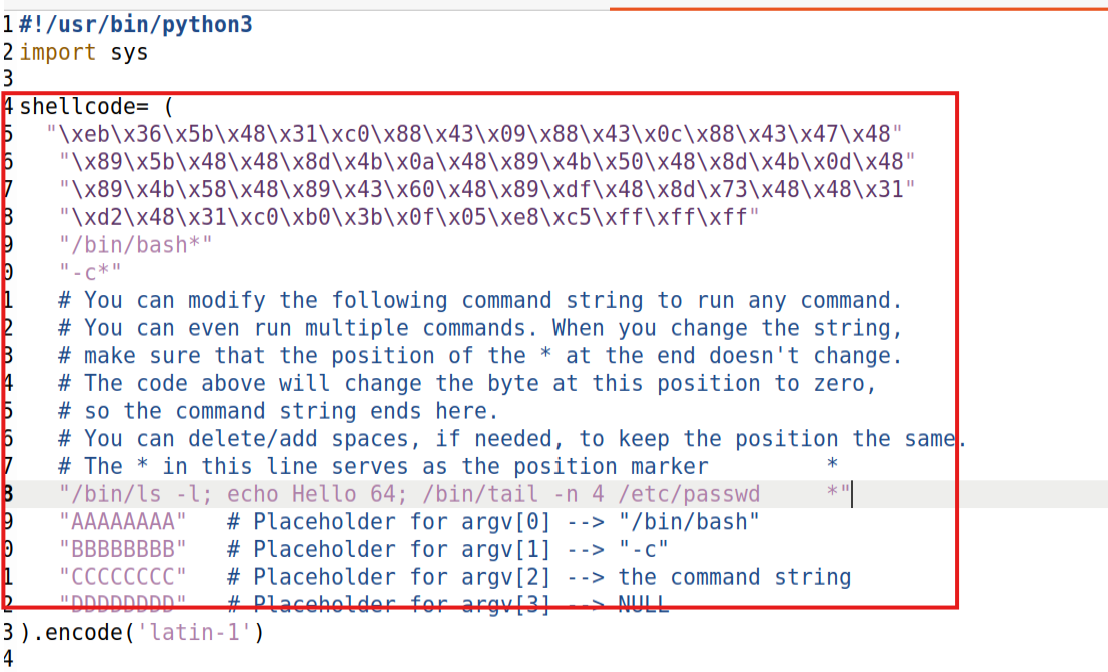
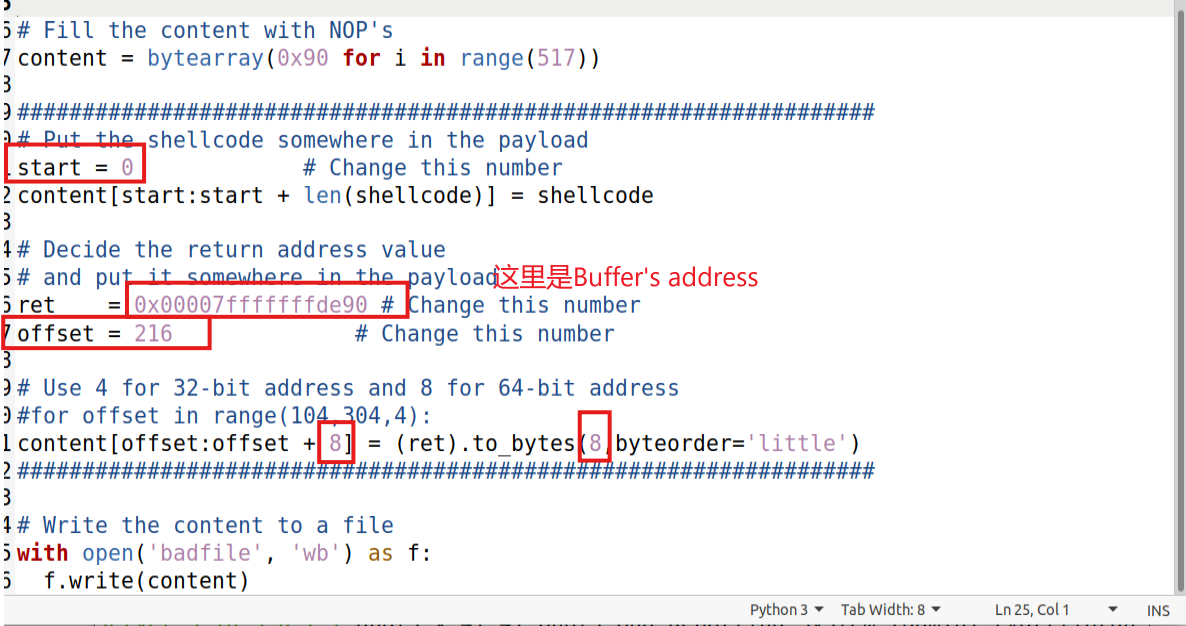
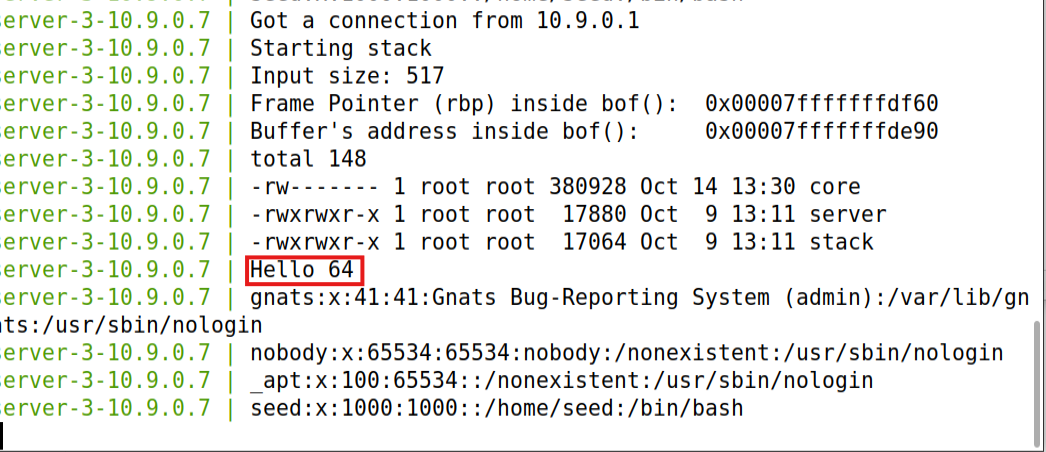
任务三：Level-2 攻击

1. 发送数据  
     
   发现服务器只给出缓冲区地址，没有ebp值，因此无法得出缓冲区大小
2. 寻找缓冲区大小  
   因为缓冲区大小在100-300之间，所以offset在[104，304]之间，而且offset是4的倍数  
   因为ret后面的高地址都是NOP或者shellcode代码，所以如果另ret为地址+304+4会指向NOP或者shellcode指令  
   所以尝试确定ret值，然后遍历offset

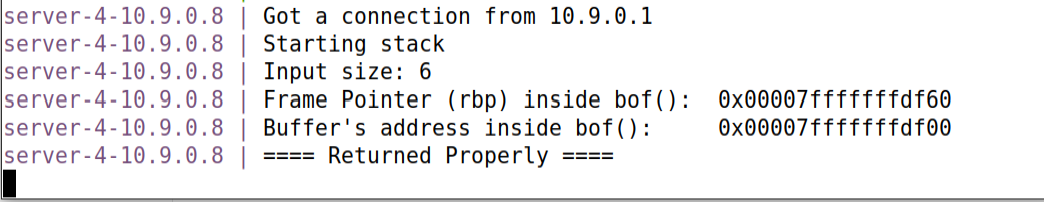
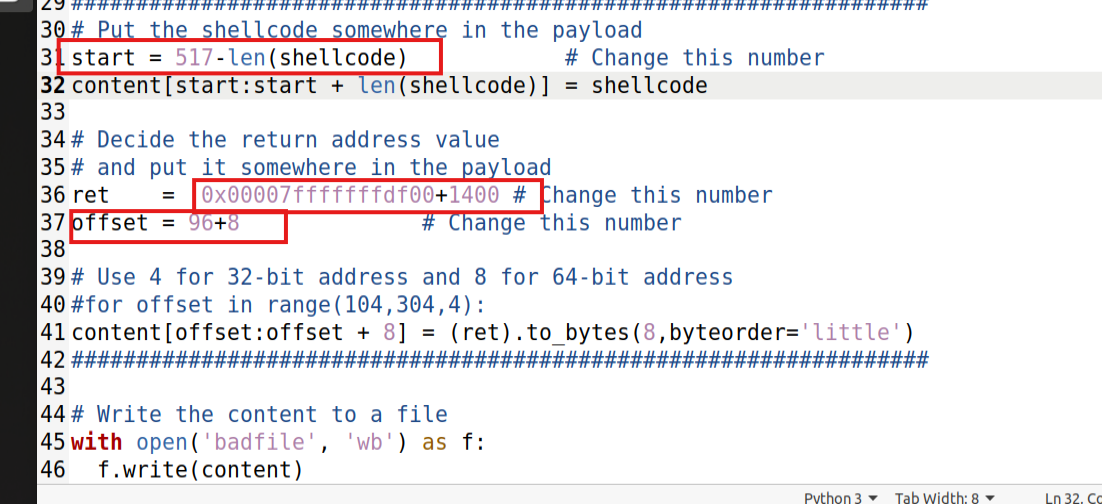
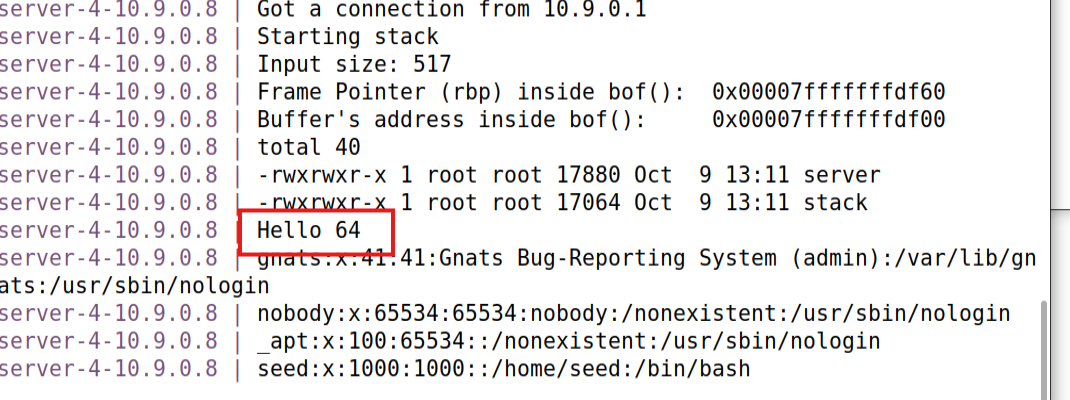


1. 得出结果  
   运行程序并查看  
   

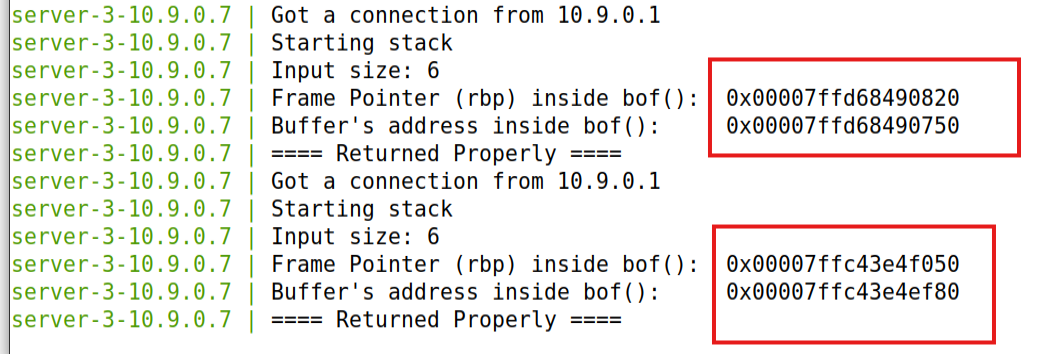
任务4：level-3 攻击

1. 打个招呼看一下  
     
   发现返回的地址是64bit的  
   而且缓冲区大小是208字节，所以可以将shellcode放在缓冲区中
2. 修改exploit.py  
   将shellcode修改成64bit的  
     
   start修改成0开始，将ret的值这样就变成缓冲区的地址  
   offset是buffer的大小+8  
     
   成功  
   

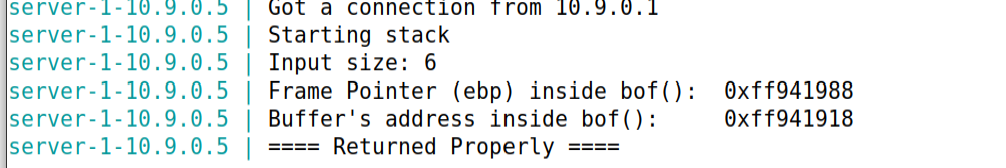
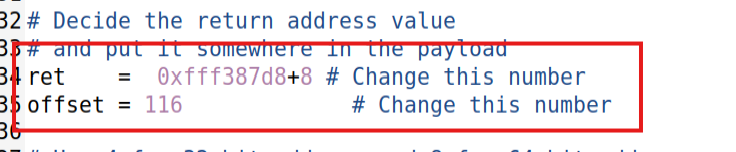
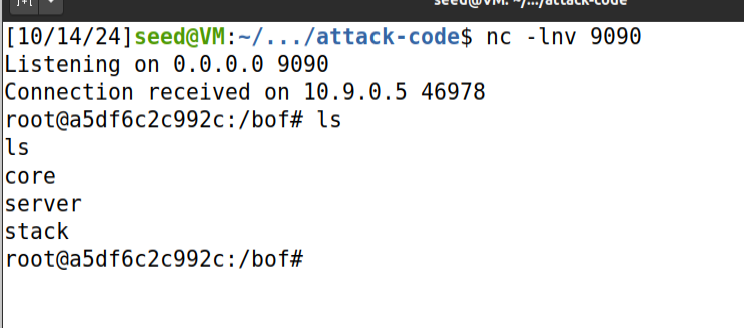
任务五：level-4

1. 打个招呼  
     
   Buffer\_size=96bytes  
   Offset=Buffer\_size+8=104bytes<len(shellcode)=165bytes
2. 修改exploit.py  
     
   ret=rbp+n，n在1184与1424之间
3. 成功  
   

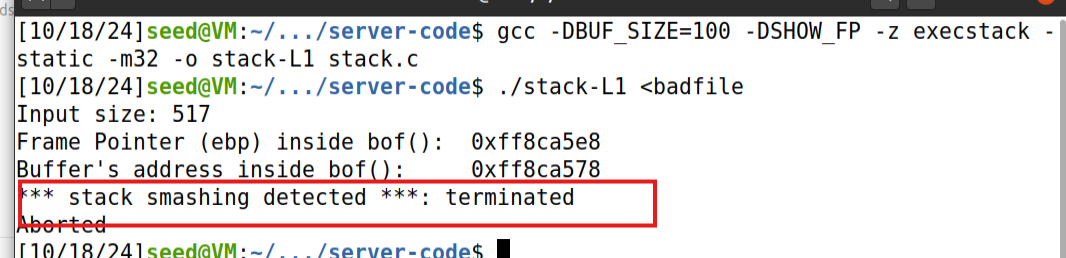
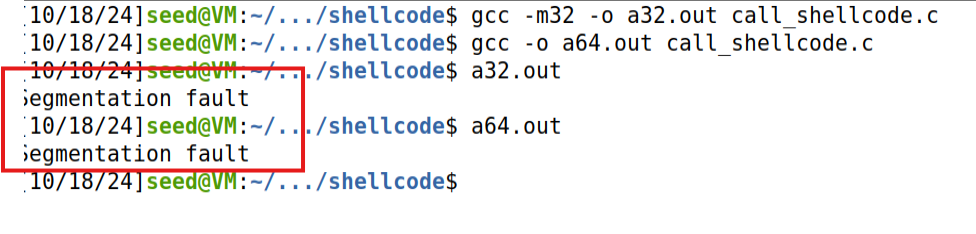
任务六：针对缓冲区溢出攻击的对策之随即地址

1. 开启内存地址随机化保护  
     
     
     
   两次地址不再相同

sudo /sbin/sysctl -w kernel.randomize\_va\_space=2

1. 目标程序放在server1上  
     
   随机选择地址  
   
2. 开启监听  
   执行攻击程序，并打开监听  
     
   使用爆破程序开始爆破  
     
     
   
3. 爆破成功，监听到反弹shell  
   

任务七：其他对策

1. 打开StackGuard保护，并将badfile文件传输到stack-L1文件中，在/Labsetup/serve-code目录下直接执行stack-L1文件  
   将badfile文件复制到/Labsetup/serve-code下  
   将stack.c直接编译为stack-L1  
   
2. 打开堆栈不可执行保护  
   堆栈不能执行  
   

# 实验结果 实验涉及缓冲区溢出攻击的模拟、不同复杂度的攻击实施，以及对各种防护机制的检验。实验旨在通过理论和实践相结合的方式，帮助理解缓冲区溢出攻击的原理和如何利用shellcode进行攻击，以及如何通过各种安全措施抵御这些攻击。