信息安全导论 hw4

小组成员：

- PB20111704 张宇昂
- PB20111647 鲍润晖
- PB20000103 王炳勋
- PB20111651 何泽昊
- PB19071508 唐思渝

Problem 1

NOP sled是指在想要插入的shellcode or malicious code前插入一段nop指令，不需要跳转到精确的地址，使其"足够接近"，这样ASLR等随机化技术无法破坏输入的malicious code

在缓冲区溢出攻击中，通常要做的是在存在漏洞的地方输入一串超过数组长度的输入串，覆写栈上的信息，最重要的是覆写RIP(x86架构)到后面shellcode存在的地址上，但是ASLR等技术会将地址随机化，来防止这样的攻击，这时候我们可以插入一段什么都不做的nop指令，这样ASLR等技术做的小地址位移将不会使得插入的shellcode指令无效(也就是说，小位移后的nop指令在继续执行后，仍然会运行到植入的shellcode上)

Problem 2

我调研了在Windows/Linux的五种最新的、不同的shellcode：

- Linux/MIPS N32 MSB Reverse Shell Shellcode：该shellcode通过使用一些Linux的系统调用函数(`socket/connect/dup2/execve`等)，来建立网络连接并执行特定的命令。它将attacker指定的<IP,port> pair编码到malicous code中，然后用`socket`创建一个套接字并用`connect`函数连接到攻击者指定的<IP,port>；建立连接完成后，shellcode使用`dup2`将标准输入/输出/错误重定向到套接字中，使得攻击者可以远程控制机器；最后使用`execve`函数执行一个新的shell，从而执行攻击者想执行的任何命令
- Linux/x86 Polymorphic Netcat Shellcode：该shellcode利用了`NetCat`工具中的反向shell功能，它可以在受害者计算机上启动一个绑定到攻击者机器的Shell。这个Shellcode将`Netcat`命令的输出重定向到一个随机生成的文件名中，使用多态技术来隐藏恶意代码，并使用`socket`编程在受害者计算机和攻击者计算机之间建立一个TCP连接。然后，它将文件名和<Ip,port>发送给攻击者计算机，以便攻击者可以连接到受害者计算机并获取Shell访问权限。
- Linux/x86_64 Bash Shellcode：该shellcode利用了bash中的一个漏洞，当执行用户输入的命令时，bash shell会将用户的输入数据存储在栈中，并使用一个指向栈顶的指针来跟踪栈的状态。攻击者可以通过向bash shell输入超过栈缓冲区大小的数据来覆盖栈顶指针，从而控制程序的执行流程。攻击者可以利用这个漏洞来注入恶意代码并执行任意命令，例如下载和安装恶意软件、窃取敏感数据等。值得一提的是，这个shellcode使用了XOR加密来隐藏恶意代码的名称，使用系统调用来执行攻击者想要的命令等。
- Windows/x86 Create Administrator User Shellcode：该shellcode通过WIndows下的函数`NetUserAdd`创建一个用户，由于默认情况下Windows会创建一个具有管理员权限的用户"Administrator"，攻击者可以利用这个漏洞来创建一个与预定义管理员账户相同的账户，并将其添加到管理员组中以获取管理员权限，这样通过`NetUserAdd`创建的用户就可以通过调用`NetLocalGroupAddMembers`被添加到本地管理员组并获得管理员权限
- Windows/x86 Delete File / Dynamic PEB Method NULL-Free Shellcode：Windows系统中每个进程都有的一个数据环境块PEB(于存储进程的环境变量、命令行参数等信息)，该shellcode通过动

态PEB方法来获取PEB的地址，从而获取`kernal32.dll`的基地址，然后通过获取`kernal32.dll`的函数地址来调用`DeleteFileW`，从而删除指定文件

通过对这几种来自Packet Storm的最新的shellcode的调研，我总结近年来攻击者设计shellcode的方式如下：

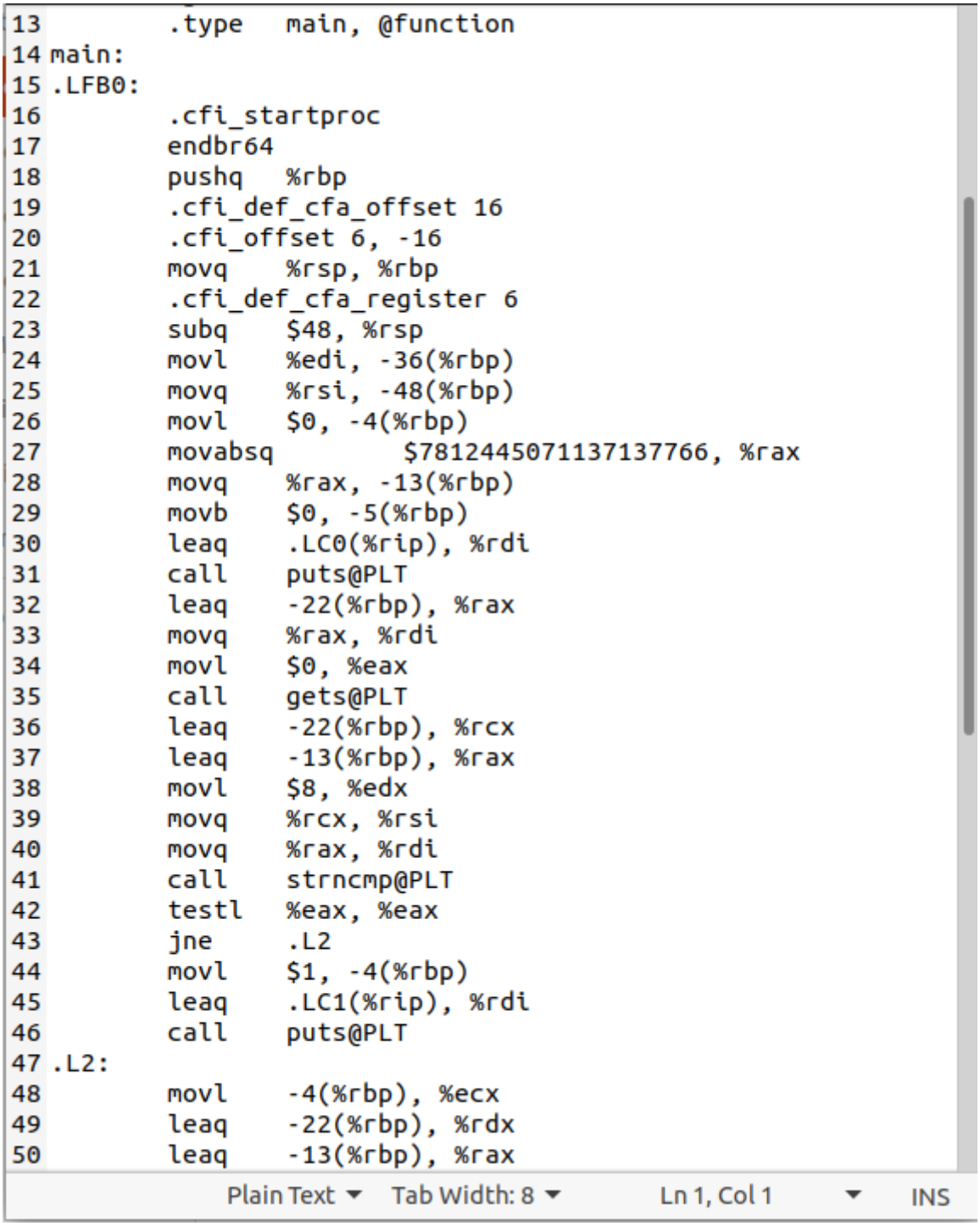
- 提权：提升攻击者用户的权限来维持对目标机器的持久访问
- 系统破坏：通过系统中存在的漏洞或者是系统中链接文件暴露的信息来获得函数地址，并借此调用函数进行删除文件等对系统破坏的操作
- 安装恶意软件：在获得shell后进行恶意软件的安装，例如键盘记录器、木马、僵尸网络等
- 远程控制：用于建立反向Shell或绑定Shell，从而让攻击者可以通过网络连接远程控制目标机器，执行任意命令，下载、上传或删除文件等操作；

我认为值得一提的是：**自Aug,2020以来，关于通过网络套接字进行攻击的shellcode似乎占据了四成左右**，我认为这可以说明关于建立连接这方面漏洞似乎是关于构造shellcode方面最值得关注的问题

Problem 3

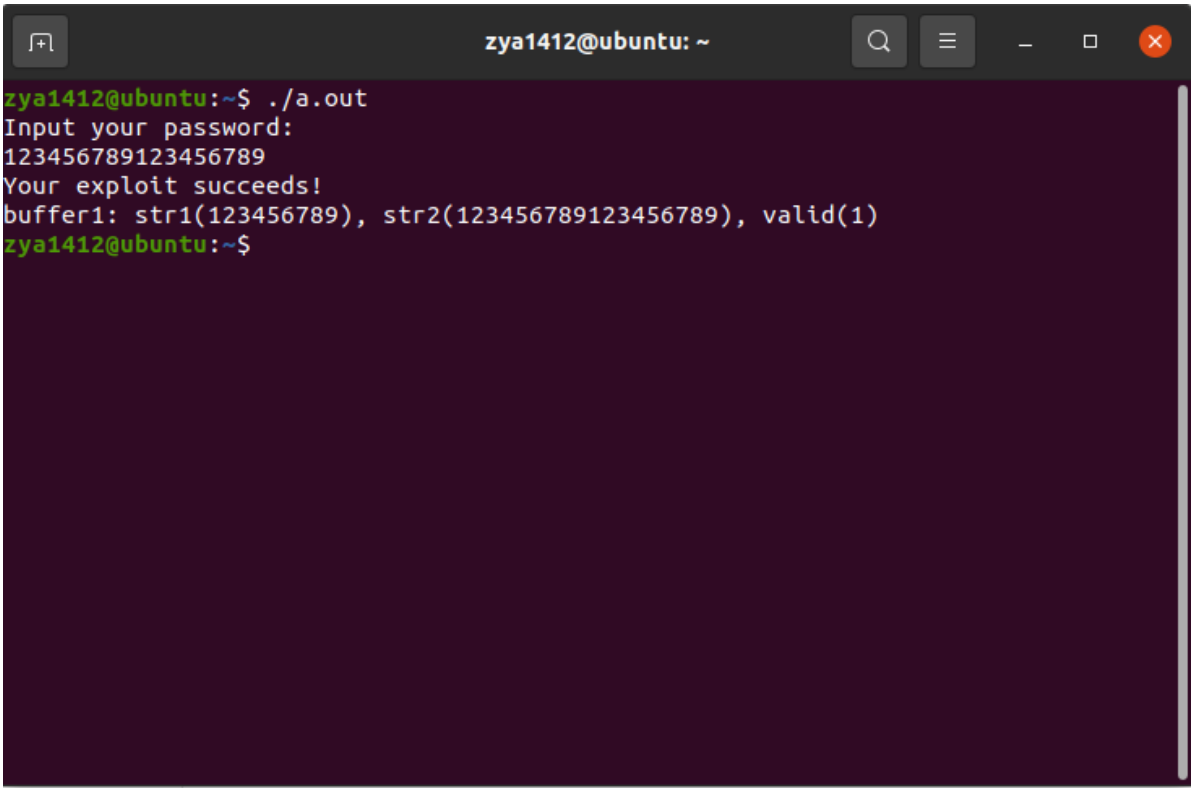
Sub Problem a

我将源代码以`gcc -fno-stack-protector`指令进行了编译，得到汇编代码如下



```
13          .type   main, @function
14  main:
15  .LFB0:
16          .cfi_startproc
17          endbr64
18          pushq   %rbp
19          .cfi_def_cfa_offset 16
20          .cfi_offset 6, -16
21          movq    %rsp, %rbp
22          .cfi_def_cfa_register 6
23          subq    $48, %rsp
24          movl    %edi, -36(%rbp)
25          movq    %rsi, -48(%rbp)
26          movl    $0, -4(%rbp)
27          movabsq         $7812445071137137766, %rax
28          movq    %rax, -13(%rbp)
29          movb    $0, -5(%rbp)
30          leaq    .LC0(%rip), %rdi
31          call    puts@PLT
32          leaq    -22(%rbp), %rax
33          movq    %rax, %rdi
34          movl    $0, %eax
35          call    gets@PLT
36          leaq    -22(%rbp), %rcx
37          leaq    -13(%rbp), %rax
38          movl    $8, %edx
39          movq    %rcx, %rsi
40          movq    %rax, %rdi
41          call    strncmp@PLT
42          testl   %eax, %eax
43          jne     .L2
44          movl    $1, -4(%rbp)
45          leaq    .LC1(%rip), %rdi
46          call    puts@PLT
47  .L2:
48          movl    -4(%rbp), %ecx
49          leaq    -22(%rbp), %rdx
50          leaq    -13(%rbp), %rax
```


可以看出，str2的地址在str1的低地址区域，地址差为9，也就是说输入的str2的内容如果是两个以长度为18循环节为9的相同内容，那么将会覆写str1为str2后半段相同的内容，所以strncmp对前八位的比较将会相同，从而跳转到`.L2`也即if块内的内容执行，完成Exploit



可以看到这样成功跳转到if语句内，完成攻击

Sub Problem b

有两种方法避免这种问题

- 将gets()函数改为fgets()`char *fgets(char *str, int n, FILE *stream)`，控制语句最大读取的字符数目
- 将gets()函数改为gets_s()`char *gets_s( char *str, rsize_t n )`，同样控制语句最大读取的字符数目

Problem 4

寄存器表：

name	value
ecx	0x3000
edx	0x9000
eip	0x4000

内存表：

address	value
0x9000	
0x9004	0x3333
0x9008	0x5000
0x900c	0x4000
0x9010	0x6666
0x9014	0x6000

Problem 5

这段代码可以造成缓冲区溢出的原因是：当`nlen=8192`时，`8192 - (nlen+1) = -1`，这时`vlen`的值可以取0到$2^{32}-1$之间的任何一个数，从而轻松超过`buf`的限制`8264`去覆写栈上的数据

Problem 6

选择论文一：*Spectre attacks: Exploiting speculative execution*

一.内容简述

本文发现了一种针对CPU的Spectre攻击，它诱导处理器执行错误的指令来泄露信息。具体实现包括设计诱导处理器进行错误推测的方法，以及秘密通道（microarchitectural covert channel，用于提取信息）的使用。本文的实际攻击结合了侧信道攻击、故障攻击及BTB，同时给出了可能的解决方案。

攻击过程

1.准备工作：

对推测器进行误导训练。目标是使推测器推测执行条件成立，但实际不成立。

贮备秘密通道；在进程地址空间定位或引入一串指令，作为这些秘密通道的发射器。

2.CPU推测执行指令，从而泄露内存、寄存器内容。

3.机密信息恢复，但错误执行会留下痕迹（eg:cache的变化）。

两种主要变体

1.条件分支：预测程序执行的方向，并提前执行代码。可能的攻击方案：使推测器在越界检查前，错误地推测执行代码，从而进行越界访问。

2.间接分支：使用BTB（分支目标缓冲区）预测未来代码的地址，并提前访问执行，因此允许任意内存的读取。可能的攻击方案：在可执行内存中植入"gadgets"（特指一个恶意代码片段，执行后将机密信息转移到秘密通道）。在分支指令执行前，将分支预测器引导到gadgets的地址并执行。

二.主要贡献

Spectre攻击揭示了分支预测带来的巨大安全隐患。该基于推测执行的攻击，能够绕过现代软件安全机制所基于的安全假设（eg:软件容器化，进程分离）。同时表明，现有的解决方案都是浅层次、混乱且脆弱的。彻底的解决方案是完全放弃分支预测带来的性能提升，或从根本上改变指令集架构，代价都十分巨大。

三.针对Spectre攻击的评价

优点

1.现代CPU追求极致性能，而将安全性放在相对次要的位置，Spectre攻击利用了这一软肋。理论上禁用分支预测就能完全禁止该攻击，但带来的是数倍甚至数十倍的性能代价。

2.能够绕过CPU安全机制，且不依赖于任何软件漏洞。

3.应用范围广泛：几乎所有类型的推测性执行代码都存在潜在隐患；实际攻击中对市面上大多数Intel及AMD的CPU都能发挥作用。同时，由于不同处理器的执行逻辑等的差异，统一的应对方案也难以设计。

缺点

1.攻击者对推测器的训练效果易受干扰。除了直接禁止分支预测外，通过限制分支预测结果共享、定时刷新BTB状态等都能限制影响。此外偶发因素（eg:其他程序对某些指令的过度使用）也能影响训练效果。

2.只对"正常的"代码有效，要求指令能够顺利执行并能正常访问内存。相较于Meltdown能够利用指令的失序执行进行攻击，该攻击存在局限性。

3.基于条件分支的Spectre攻击依赖于检测错误执行所留下的痕迹，而这些痕迹易于被模糊化或消除。（eg:刻意降低高精度计时器的准确度）