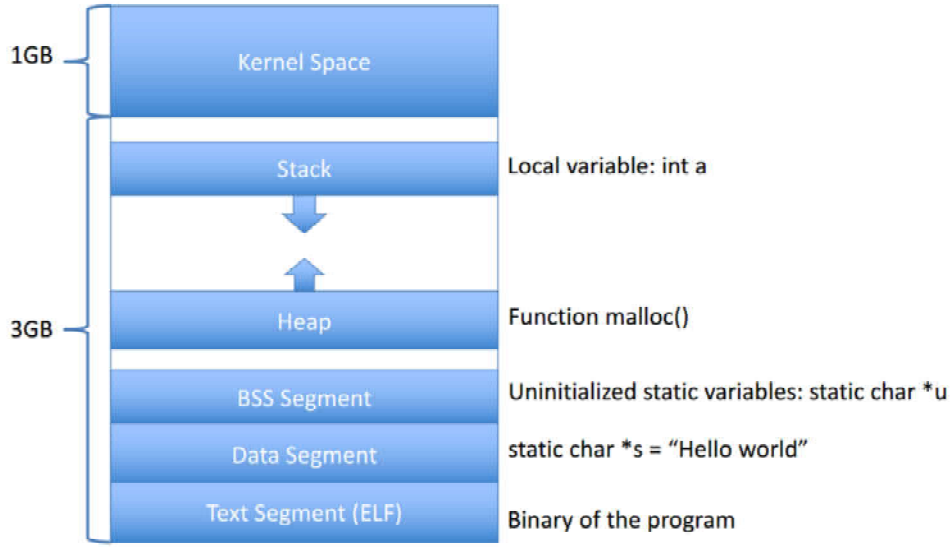
《计算机安全》课程笔记

2019年9月9日 19:11

Log：9/20从头复习到了2.lec3

1. Lec1 Packet Sniffing and Wireshark
    1. 讲了网络的五层结构和怎么使用Wireshark
    2. 课程信息
        a. 课程站点
        https://fengweiz.github.io/19fa-cs315/index.html
        b. 密码
            i. WSU-security.
            ii. 注意结尾有 .
        c. 王子勤把lab1-8 做了一个镜像源：
            https://mirrors.sustech.rocks/courses/CS315-2019Fall/
        d. 上课前要交作业。late penalty：10% per day
    3. 如果要用虚拟机上网就可以恢复VM的默认设置
    4. 要关闭虚拟机就需要先关闭虚拟机内的电脑再叉掉虚拟机
    5. 短线的英语：dash、hyphen
2. Lec2 Buffer Overflows （栈溢出攻击初步）
    1. 基础概念：
        a. vulnerability：漏洞
        b. shell code：
            i. 是一段用于利用软件漏洞而执行的16进制机器码，CPU执行的shellcode后可以让攻击者获得shell权限，执行攻击者的任意指令
        c. dissert：会议论文
        d. NVD: National Vulnerability Datebase，美国国家通用漏洞数据库
        e. ELF: Executable and Linkable Format , 是Linux系统下的可执行文件
        f. 栈溢出在操作系统中是很常见的vulnerability，曾经是最常见的
    2. Memory Layout for 32-bit Linux



        a. 栈是从高地址往低地址的
        b. 一个栈帧（stack frame）中保存着一个还没运行完的函数的返回地址和局部变量
        c. ESP（extension stack pointer）：是一个寄存器，记录栈顶的地址

d. EBP(extension base pointer)：是一个寄存器，记录最上面的那个栈帧的栈基值

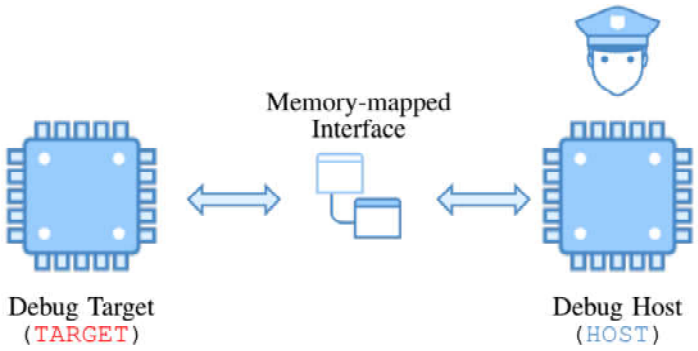
3. segmentation fault （段错误）
   a. 就是访问了错误的内存段
   b. 一般是你没有权限，或者根本就不存在对应的物理内存
   c. 尤其常见的是访问0地址
4. 应付栈溢出攻击的方法
   a. NX – non-executable stack
      i. 规定栈中的代码不能执行
      ii. 破解方法
      –在堆中执行代码
      – Data Execution Prevention (DEP)
      – Return Oriented Programming (ROP)
      1） 利用已有程序组合成恶意程序
   b. StackGuard: Cannaries （金丝雀）
      i. 在栈中的返回值边上加一个guard，如果guard变了就说明return被篡改了
      ii. 破解方法
      – 用一样的值重写canary
      – Brute force attack：遍历所有的guard值
   c. ASLR（Address space layout randomization）
      a. 通过对 堆、 栈、共享库映射等线性区布局的随机化，防止攻击者直接定位攻击代码位置
      b. 可以有效的降低缓冲区溢出攻击的成功率，如今 Linux、 FreeBSD、 Windows等主流 操作系统都已采用了该技术。
      c. Linux下的ASLR总共有3个级别
         i. 0就是关闭ASLR，堆栈基地址每次都相同
         ii. 1是普通的ASLR。mmap基地址、栈基地址、.so加载基地址都将被随机化，但是堆没有随机化
         iii. 2是全部随机化
5. 攻击步骤
   a. 先通过反汇编获得程序运行的每一步的情况
   b. 然后得到eax的地址，在这一步打断点，显示出所有寄存器此时的值，推断出buffer的地址
   c. buffer 和返回地址相隔4个byte，所以可以增加四个单位
   d. 由于buffer是char数组，而我们是32位，所以赋值四次
   e. 找到shellcode的内存地址
   f. 把shellcode的地址重写入bufferOverflows( ) 中的return address

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


int bufferOverflow(const char * str)
{
   char buffer[12];


   /* This line has a buffer overflow vulnerability. */
   strcpy(buffer, str);
   return 1;
}

int main(int argc, char ** argv)
{
           char aString[512];
           FILE *badfile;

           printf("Buffer overflow vulnerability starting up...\n");
           badfile = fopen("badfile", "r");
           fread(aString, sizeof(char), 512, badfile);

           bufferOverflow(aString);

           printf("bufferOverflow() function returned\n");

           return 1;

}
```

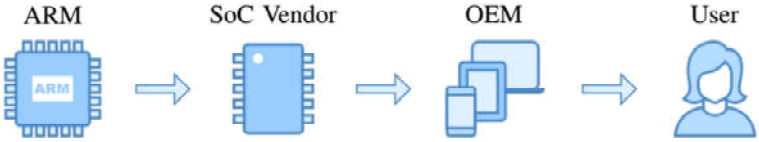
6. 怎么找到返回地址的内存地址
    a. 在编译了BOF和其中涉及到的其他文件（badfile）
        i. gcc -g -z execstack -fno-stack-protector BOF.c -o BOF
    b. 运行
        i. ./BOF
    c. 使用GDB查看汇编码
        i. 也可以使用 tool,objdump（dump：垃圾、堆砌物）来阅读汇编码
        ii. 对BOF 程序进入 gdb 模式：gdb BOF
        iii. 在gdb模式下：disas main
        iv. 然后dias bufferOverflow
        v. 理解汇编码的意思，然后找到返回地址
        vi. 接着跑：run
        vii. 也可以了解各个寄存器的情况：info register ，

3. Lec 3 Nailgun：Breaking the Privilege Isolation on ARM
    1. background
        a. privilege isolation：特权分割
        b. exception：指OS领域的exception，就是进程间断
        c. privilege level：
            i. Normal mode：
                1) normal EL0：user-level
                2) normal EL1：OS kernel
                3) normal EL2：Hypervisors（虚拟机什么的……
            ii. Secure mode
                1) secure EL0：user-level apps
                2) secure EL1：OS kernel
                3) secure EL3：是gatekeeper
            iii. 每一层都有自己的memory，指纹存储在secure level
            iv. secure level只run很少的代码，因为如果buggy code恰好run在secure mode，secure mode就再也不安全了
            v. TCB：trust computing base，信赖的代码的数量，越少越好
            vi. run在secure中的代码可以使用normal mode中的代码、memory，而normal mode中的代码却不能使用碰secure mode的

2. introduction
    a. 使用的工具：nailgun（钻孔机）
        i. normal mode和secure mode之间就有这种isolation，nailgun打破了这种isolation
        ii. 这样在normal mode也可以使用secure mode中的memory
    b. 基于硬件的debugging feature
        i. 现代的处理器配有一些基于硬件的debug方式
        ii. 如：hardware breakpoints 和 hardware-based trace
        iii. 它们通常要求 cable connection
    c. 传统的硬件debugging
        i. 使用另一台电脑作为HOST来debug
        ii. 通过JTAG Interface来连接debug host和debug target
        iii. 这样的话，host就对target有很高的privilege了，有可能存在恶意debug
            1) ARM中的debug authentication机制，要target同意才能进行debug
        iv. 人们很少研究这其中的安全问题，因为物理连接不能像网络病毒那样大面积传染
3. approach
    a. 有两个obstacle
        i. 物理连接
            1) 解决方式：inter-processor debug
                i. 可以使用CPU的一个core 来debug另一个 core，因为它们是直接相连的，所以不用JTAG了
                ii. 具体方式：memory-mapped debugging registers
                    1) 使用软件可以控制一部分的寄存器，而不只是靠汇编指令来控制
                    2) introduced since ARMv7



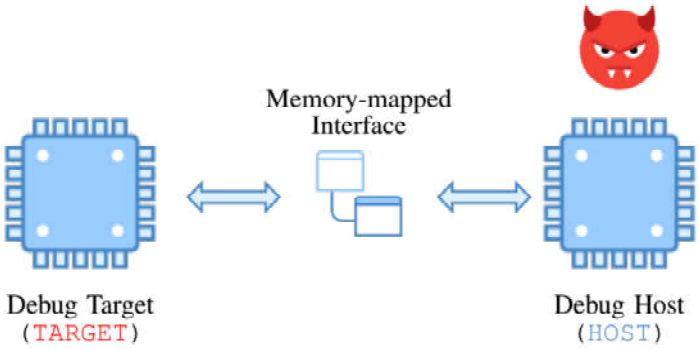
        ii. debug authentication mechanism（鉴权机制）
            1) ARM处理器的模式
                i. Normal state：target执行program counter的指令
                ii. non-invasive debugging
                    i) monitoring without control
                iii. invasive debugging
                    i) control and change state
            2) debug authentication signal
                i. 决定了是否允许别的设备来给他debug
                ii. 有四种状态
                    i) secure/non-secure * invasive/non-invasive
            3) ARM生态



                i. ARM 定义（define）这些架构（比如 debug authentication signal 要有四种模式）
                ii. 高通等SoC（System-On-Chip） vendor 实现（implement）了这个架构

iii. OEMs（原始设备制造商）装配（configure）了这个架构
iv. 消费者购买

4. Nailgun attack



a. 现在，我们假设有两个core，分别是host、target
   i. host发出一个Resource Access Request
   ii. target就进入了debug state，获得了高权限
   iii. host可以要求target获取高权限的资源，然后传给host
b. 现在，侵入了secure mode，我们要获得指纹的位置
   i. 通过逆向工程

5. Mitigation
   a. disable那些authentication signal?
      i. 不可能，因为很多tools依赖于这个debug authentication signal
   b. 应该做的
      i. ARM：应该增加对inter-processor debug model的一些限制
      ii. SoC vendor：更加精细的信号管理和debug控制
      iii. OEM：控制基于软件的access

4. lab3 scanning and reconnaissance（侦察）
   1. 学习目标
      a. 使用 nmap 和 OpenVAS 来探测受害者的OS以及其他软件的型号，看看有哪些漏洞
   2. 基本知识
      a. ifconfig等价于Windows的ipconfig
      b. nmap ("Network Mapper")
         i. 开源，用来探索网络和进行安全监听
         ii.

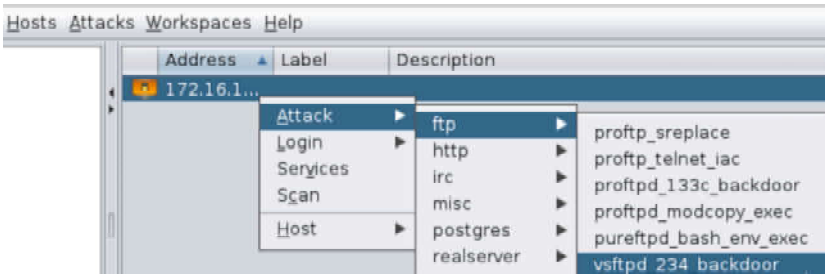
T1	Sneaky mode. Can be used to avoid IDS.
T2	Polite mode. It can avoid the crash of the system and thus it's far slower than other modes.
T3	Normal mode. It is the Default mode. It can scan the target parallel and thus it's faster.
T4	更快的执行命令
O	探测OS的版本信息

      c. OpenVAS
         i. 开源，有很多漏洞扫描与管理的工具
   3. 过程
      a. 使用 ifconfig 获取 Metasploitble2-Linux 的 IP 地址
      b. 使用 nmap 对这个 IP 地址进行扫描，看看它各个端口上都运行了哪些软件
      c. 使用 OpenVAS 对这个 IP 地址进行扫描
      d. OpenVAS 就会反馈出这个 IP 地址的主机上有哪些软件有哪些漏洞

5. Lab 4 Metasploit Framework
   1. 学习目标
      a. 学会使用 metasploit framework，成为脚本小子，不必 reinvent the wheel（重复造轮子）

b. 学会使用 Armitage，连命令行都不用输，直接使用丢人的 GUI

2. 基础知识
   a. MSF（metasploit framework）
      i. 安全漏洞检测工具，黑客的利器，可以一键攻击
      ii. 是一个基础框架，可以用来定制化
      iii. 使用了 PostgreSQL
   b. Armitage
      i. MSF 的 GUI 工具
3. 攻击过程
   a. 命令行版
      i. 打开 PostgreSQL：service postgresql start
      ii. 看看 PostgreSQL有没有打开：service postgresql status
      iii. 初始化 MSF 的数据库：msfdb init
      iv. 看看 Metasploitable2-Linux 的 IP 地址：ifconfig
      v 启动 MSF：msfconsole
      vi 设置想使用的 module：msf > use exploit/unix/irc/unreal_ircd_3281_backdoor
      vii 设置 remote host：msf exploit(unreal_ircd_3281_backdoor) > set RHOST 172.16.108.172
      viii 发动攻击：msf exploit(unreal_ircd_3281_backdoor) > exploit
      ix. 看看有没有攻击成功： whoami 或者 uname -a
   b. GUI 版
      i. 启动Armitage：armitage
      ii. 在Host选项选择 add ost，加入受害者的 IP 地址
      对其进行扫描
      iii. 在 Attack 里面选择 find attack
      iv. 右击这个 受害者，选择attack攻击方式



6. Reverse Engineering and Obfuscation
   1. Reverse Engineering（RE）
      a. 作用
         i. White hats
         – Malware analysis
         – Vulnerability discovery
         ii. Black hats
         – Cracking, hacking
         – Malware re-engineering (shell code injection/reuse)
      b. 种类
         i. 行为分析（动态分析）
         ii. 代码分析（静态分析）
            1）Java允许在运行的时候修改代码：self-modify code，而动态分析就无法分析这种东西
      c. 对病毒的反向工程
         i. 创建一个独立的实验环境，比如一个虚拟机中
         ii. 把病毒放入沙盒中，如：Anubis，查看病毒的行为
         iii. 看病毒调用的 API，猜测它的类型

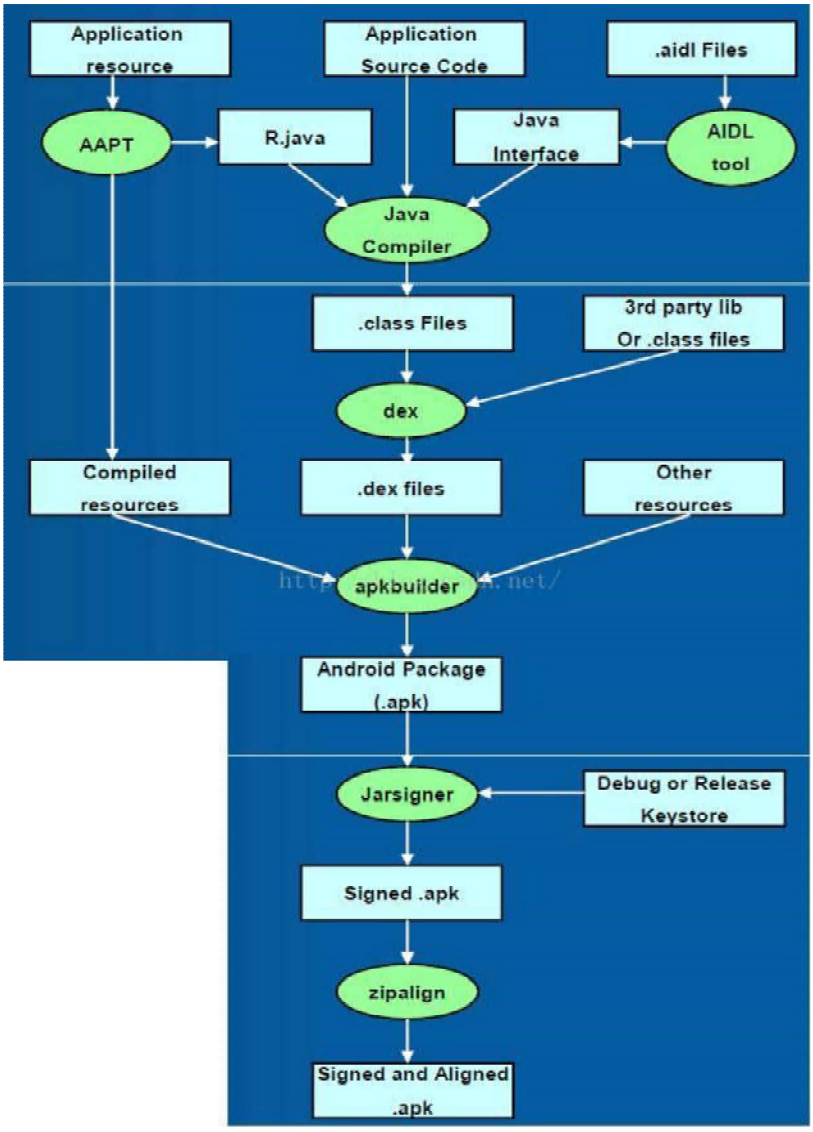
iv. 识别出它的 packer，并且对其 unpack（可手动实现或自动实现）
v. 对它 Disassemble/decompile，追踪它使用的 API
vi. 二分地调试，触发各种情况，追踪其 control flow
vii. 二分地打补丁

d. 另外
i. 病毒可能会检测当前的环境，如果是虚拟机，就不做任何事
ii. 病毒可能会一层层地加固，加几十层

2. Obfuscation
a. White hats
– Copyright issues
b. Black hats
– 增加病毒的分析难度
– Multiple layers of obfuscation
c. 商业的 packer
i. 360、Alibaba、Tencent、Baidu、Bangcle

3. DexLego 个案研究（case study）
a. 动态分析的缺点
i. 成本（Performance Overhead）大
ii. 安卓运行程序运行时修改自己的代码，这个目前无法动态分析
iii. 在运行时新的 DEX 文件可能会从 cloud 上下载下来
b. DexLego 是静态分析的，而且是指令级的
i. 以往的 Android Unpacking Systems 如：DexHunter、AppSpear 都是函数级的
c. 处理 self-modified code
i. 建立一棵 collecting tree
d. Evaluation
i. DexLego 在同类中表现最好，能破解市场上所有常用的商业 packer

7. Lab 5: Android Application Reverse Engineering and Obfuscation
1. 学习目标
a. 对安卓的 app 进行反向工程，加入恶意代码，再把它重新打包运行
2. 工具
a. JDK、Android SDK
b. Android Studio（可以使用 eclipse 代替）
c. Android Emulator System（可以拿安卓手机代替）
d. Smali ar
i. 将 dex 文件转换成 smali 文件（一种中间码）
e. back
3. 步骤

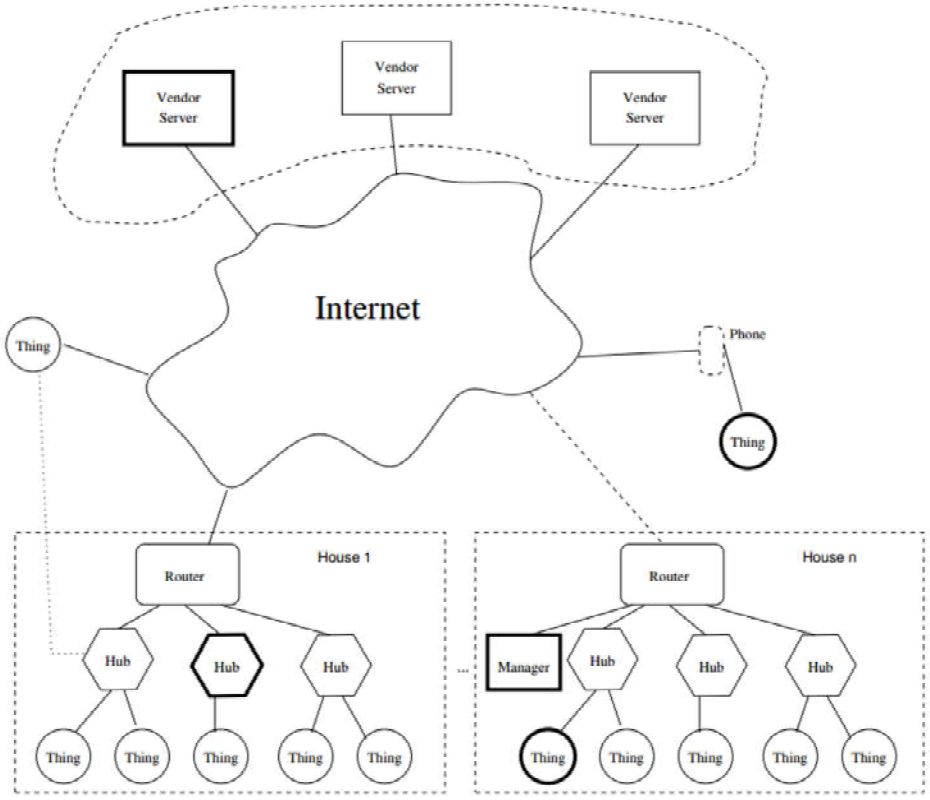


a. 在 Android Studio 上创建一个新的 project
b. 选好 form factor 和 login activity 界面，Android Studio 会为我们生成相应的代码
c. 使用 AVD Manager 创建一个 Android Virtual Device (AVD)：
    i. 在 terminal 中输入：android avd
    ii. 点击 create 按钮，定制化各种参数，然后 start
    iii. 使用真的安卓设备替代的话在这次的攻击中是没有问题的，而且在虚拟机中生成一个安卓模拟器速度比较慢
d. 回到 Android Studio，在代码界面点击 run，选一个 emulator 运行
e. Unzip 这个 apk 文件：unzip –d app-debug app-debug.apk
f. 使用 baksmali.jar 将 dex 文件转换成 smali 文件： java –jar ~/Desktop/baksmali.jar classes.dex
    i. 默认生成的文件夹名字叫 out，里面会根据调用的包名来阶级地安放 smali 文件
g. 打开这个 smali 文件，修改代码
    i. 使用 android.util.Log.d(String, String) 记录登入的用户名和密码
h. 删掉之前的 dex 文件，使用 smali ar 将 out 重新转换成 dex 文件，再把 out 删掉
i. 删掉含有这个application的文件之前的 signature 的目录：rm –r META-INF
j. 打包： zip –r app-debug.zip ./*
k. 改名字：mv app-debug.zip app-debug.apk
l. 生成自己的 RSA 签名
m. 对齐，把之前没对齐的那个删掉
n. 卸载之前的那个应用，重新安装这个新的

8. Lec IoT MB（物联网 服务器消息块）
    1. IoT定义：

a. 非计算设备
b. 有CPU
c. 能连接（否则就只是一个嵌入式设备）

2. 新的问题
   a. 人们对此的防范意识低，不觉得电视什么的能有什么问题
   b. 密码都是默认的
   c. 也没有反病毒软件
   d. 也不会去打补丁patch
      i. 根本想不到要打补丁
      ii. 一些设备也很难打补丁
      iii. 这些老设备的预期使用时间通常长于软件的预期支持时间
   e. 不同的设备有不同的问题，情况太多了
3. IoT的架构
   a. things一般都会连接hub，而非直接联网
   b. hub 连manager（可能是app）
      i. Hub 通过私有的协议与设备交流
   c. 设备比较简陋，很多功能都是靠vendor进行的



4. 思考点
   a. 设备的主人如何认定？要转卖怎么办？肯定有一个reset everything的选项！入侵！
   b. 物理-网络攻击（比如温度）
   c. 收集隐私数据
   d. DoS攻击

9. Lab O ecurit for t e nternet of ings
   1. 学习目标
      a. 了解 Brillo、Contiki
      b. 了解 Zephyr 目前有哪些安全措施
      c. 对 Zephyr 发动几种基本的攻击
   2. 基础知识
      a. Brillo、Contiki、Zephyr都是用于物联网的OS

b. Brillo
    i. 由谷歌提出，和安卓绑在一起
c. Zephyr
    i. 是Linux基金会的一个项目
    ii. 针对资源受限情况进行了优化，支持多种微处理器架构，安全性高
d. FreeRTOS
    i. 针对嵌入式设备的开源实时操作系统，支持众多的微处理器

10. Lec 802.11 Security & Pen Testing
    1. 计算机安全的“gold standard”
        a. Butler Lampson：authentication、authorization、audit
    2. WiFi（IEEE 802.1）
        a. Deauthentication（简称为 deauth） 攻击
            i. 假装是AP，发个 deauthentication frame 命令 client 断开连接
            ii. Client 可以拒绝断开，也可以断开后重新连上
            iii. 可以不断地发出这种 frame，发动 DoS 攻击
        b. Evil Twin 攻击
            i. 攻击者只需要一台笔记本就可以使用相同的 SSID （service set identifier）创建一个欺诈性接入点，发出 beacon frame
            ii. 因为与受害者常用SSID名称一样，并且具有更强的信号，因此可以轻易欺骗受害者与之连接
            iii. 建立连接后，攻击者可以替换网页，比如亚马逊付费界面替换成攻击者自制的类似界面
            iv. 此种攻击难以侦查，双因素身份验证和VPN可以帮助抵御，但是成本较大，只适用于大型公司
            v. 通常 OS 会警告使用者
        c. War Driving 攻击
            i. 只要具备一辆车、一台笔记本电脑、一个无线网卡，和一个全球定位系统的装置
            ii. 开着车兜一圈，就能够得到沿途所有设备名称，加密方式及经纬度信息，以分辨哪个是可攻击项
            iii. 现在，WEP加密能在几分钟内被破解，WAP相对要长一点
        d. Packet Sniffing 数据包嗅探攻击
            i. 对无线连接进行拦截
            ii. WEP密码破解就是收集足够的数据包后可以反推出密钥
    3. DH加密（Diffie-Hellman）

        1、甲方构建密钥对（公钥+私钥），公布公钥
        2、乙方以甲方的公钥构造新的（公钥+私钥），公布新的公钥
        3、甲方用“甲私+乙公”构造本地的密钥
        4、乙方用“乙私+甲公”构造本地的密钥
        5、这时甲乙的（公钥+私钥）都一样
        双方可以使用对称加密了，因为对称加密速度快

	对称加密	非对称加密
加解密速度	快	慢
安全性	低	高
常见算法	DES、3DES、Blowfish、IDEA RC4、RC5、RC6、AES	RSA、DSA、ECC、Diffie-Hellman、El Gamal

    4. WiFi 加密算法
        a. WEP（有线等效加密，Wired Equivalent Privacy）
            i. 由IEEE 802.11规定，因为历史原因广泛使用在WLAN中
            ii. 采用流加密机制 RC4
                1) 将短密钥扩展成任意长度的伪随机密钥流，用其与报文异或来产生密文
                2) 接收端用相同的密钥产生同样的密钥流，并用异或运算解码

b. WPA（Wi-Fi Protected Access）
   i. 用作WEP的暂时性改善方案
   ii. 通常 使用预共享密钥（PSK）和 临时密钥完整性协议（TKIP）
      1) TKIP 为了对现有 WEP 设备进行固件升级，重复利用 WEP 中的某些元素
c. WPA2
   i. 使用高级加密标准（AES）
      1) AES获得美国政府批准，对绝密信息进行加密，因此足以保护家庭网络
d. 安全性排名：

WPA2 + AES
WPA + AES
WPA + TKIP/AES (TKIP 作为回退方法)
WPA + TKIP
WEP
开放网络 (一点也不安全)

11. Lab 7 Wireless Exploitation & Defenses
    1. 学习目标
       a. 使用 Aircrack-ng 来进行无线网络攻击
    2. 基础知识
       a. Aircrack-ng
          i. WiFi 的攻击套件
          ii. 装有detector, packet sniffer, WEP 、WPA/WPA2-PSK cracker 和 WiFi 分析工具
       b. 网卡模式
          i. promiscuous（混杂）模式
             1) 要求连上 WiFi
             2) 接收所有经过网卡的数据包，包括不是发给本机的包，即不验证MAC地址
          ii. monitor（监听） 模式
             1) 允许网卡不用连接wifi就可以抓取特性频道的数据
             2) 就是在空气中抓取某个波段的数据
          iii. 普通模式
             1) 网卡只接收发给本机的包
       c. 无线网接口种类
          a) 内部网卡：大多数笔记本的选择
          b) 外部网卡：如天线
    3. 过程
       a. 使用路由器设置AP的SSID
          i. 将PC和路由器相连
             1) 有的路由器要求使用网线物理相连，连上路由器的LAN端口
             2) 有的路由器只要通过SSID相连就可以
          ii. 打开setup网页
             1) 输入login IP或hostname（一般在路由器的背面就有）
          iii. 登入路由器
          iv. 配置SSID
             1) 重命名SSID，比如：命名为“Hack3r”
          v. 配置无线安全相关的选项
             1) 例如：将安全协议配置为WPA/WPA2，使用AES作为encryption，使用“password”作为passphrase. 安全协议默认是WEP
       b. 通过wireshark捕获无线packets
          i. 条件

1) 为了捕获无线packets，需要有一张无线网卡

ii. 步骤

1) 授予wireshark root权限

a) 在命令行中打出 sudo wireshark

2) 在wireshark中选择WiFi接口

3) 启动monitor 模式 或 promiscuous模式

4) 开始捕获

c. 捕获四次握手

i. 把wireshark配置成monitor 模式来抓包

ii. 通过passphrase连上AP

iii. 停止抓包，并且找到四次握手

1) filter：eapol

iv. 保存结果为test.pcap

d. 使用 aircrack-ng 来攻破 WPA2

i. 将test.pcap文件拷贝进Kali

1) 也可以使用镜像中预存的 test-instructor-monitor.pcap 和 test-instructor-promiscuous

ii. 使用 aircrack-ng ： aircrack-ng -w /usr/share/wordlists/fern-wifi/common.txt ~/Desktop/test-instructor-monitor.pcap

12. lec8 Firewalls & Intrusion Detection Systems

1. 概论

a. 防火墙

i. 分析报文headers，来控制报文的出入

b. IDS（Intrusion Detection Systems，入侵检测系统）

i. 对网络、系统、运行状况进行监视

ii. 分析整个报文，如果报文符合要求，就生成一条log message

c. IPS（Intrusion Prevention Systems，入侵防御系统）

i. 比 IDS 更高级，能够阻止恶意报文

ii. 分析整个报文，如果不合要求，就会被拒绝

d. “零日漏洞”(zero-day)

i. 又叫零时差攻击

ii. 漏洞发现时，补丁存在的天数是0

2. IDS类型

a. host-based IDS（HIDS）

i. 装在用户的主机上，负责主机的安全

b. network-based IDS（NIDS）

i. 装在网络上，监控网络活动

ii. Signature-based IDS

1) 将收到的包和已知的签名进行比较

2) 问题：

a) zero-day attack

b) Polymorphic attack

i) 变种，比如加壳、换别的语言写

3) 比如：Snort, Bro, Suricata

iii. Anomaly（异常）-based IDS

1) 对系统的异常行为产生alert

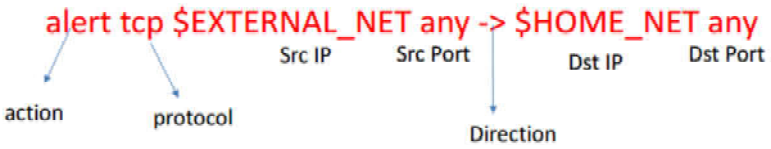
2) 能够识别出zero-day attack

3) 问题：

a) High false positive rates

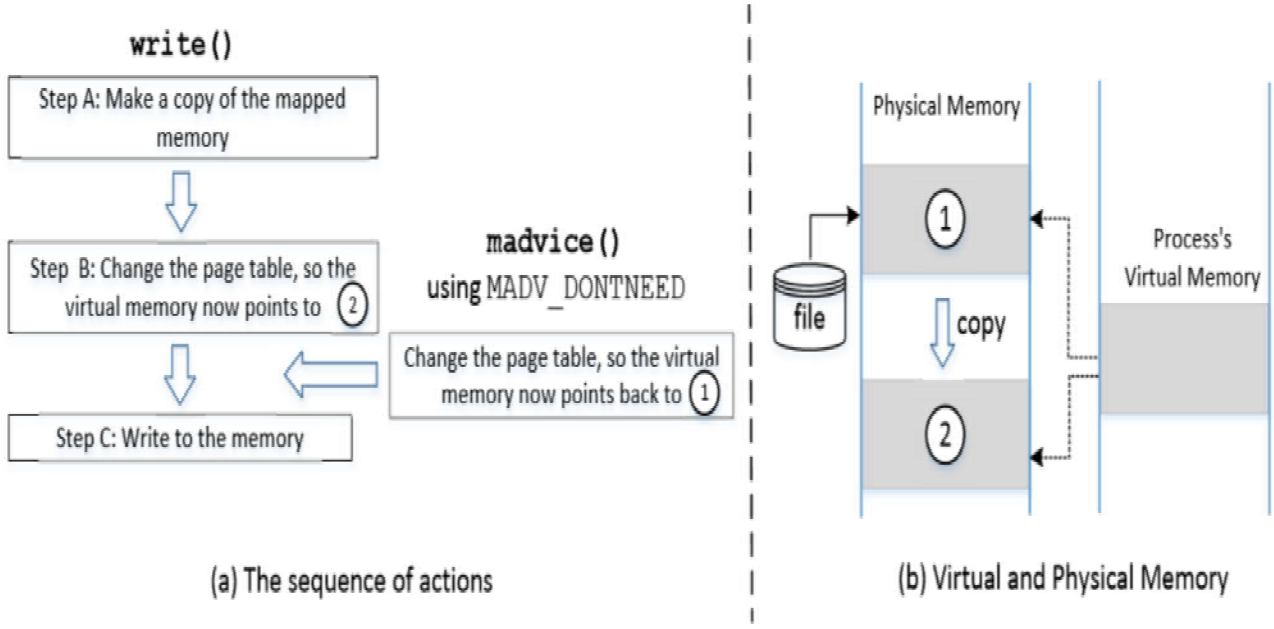
b) Labeled training data

3. IDS评估标准
   a. 二维坐标
      i. True Positives (TP)：检测出了攻击
      ii. True Negatives (TN)：识别出了正常的活动
      iii. False Positives (FP)：误识正常活动
      iv. False negatives (FN)：没发现攻击
   b. 描述
      i. accurate：能够找到所有的攻击
      ii. precise：不会误识正常行为
   c. 计算
      i. true positive rate: TP / (TP + FN)
      ii. false positive rate: FP / (FP + TN)
4. snort
   a. 是基于签名的 IDS
   b. 能作为 IPS 或者 IDS
   c. rules



13. lab8 Firewalls & Intrusion Detection Systems
    1. 学习目标
       a. 使用snort来探测分组
    2. 基础知识
       a. Snort IDS
          i. 基于签名，可以用于分组嗅探
    3. 过程
       a. 打开 snort
          i. service snort start 或者 /etc/init.d/snort start
       b. 打开装有 snort 规则的文件： vim /etc/snort/rules
       c. 加入新规则：alert icmp any any -> any any (msg:"ICMP Packet found"; sid:1000001; rev:1;)
          i. 这条规则表示如果找到了一个ICMP的包就log这个alert
          ii. SID要大于1000，000才行
       d. 重启snort
          i. service snort restart 或者 /etc/init.d/snort restart
       e. 触发新规则的alert
          i. 给VM发一条ICMP报文就可以了
             1) 先使用 ifconfig 找到VM的IP地址
             2) 然后使用我的主机host在cmd中去 ping 刚才的那个IP地址，如 ping 172.16.108.242
       f. 验证触发成功
          i. 打开 log message
             1) 这个log文件是二进制的，需要使用 u2spewfoo 打开：u2spewfoo /var/log/snort/snort.log
14. Lab 9 dirty cow
    1. 学习目标
       a. 使用 dirty COW 发动攻击
    2. 基础知识

a. dirty cow
   i. 一种 race condition 漏洞
   ii. 自从2007年就出现在了Linux内核中，2016年10月才被发现
   iii. 所有的Linux系统都有可能被攻击，包括安卓
   iv. 通过这种攻击，攻击者可以得到root权限
   v. 原理：通过进程竞争和 COW 来修改原本只读的权限文件
b. 在SEED LAB的 Ubuntu 16.04 VM中这个漏洞已经被修复了，只能用 Ubuntu 12.04 VM 或之前的 Kali
c. 每一个进程都是从别的进程中产生的，使用fork（）表示这种system call
d. COW（copy on write，写时复制）
   i. 如果有多个调用者（callers）同时要求相同资源（如内存或磁盘上的数据存储），他们会共同获取相同的指针指向相同的资源
   ii. 直到某个调用者试图修改资源的内容时，系统才会真正复制一份专用副本（private copy）给该调用者，而其他调用者所见到的最初的资源仍然保持不变。这过程对其他的调用者都是透明的（transparently）
   iii. 优点：如果调用者没有修改该资源，就不会有副本（private copy）被创建，因此多个调用者只是读取操作时可以共享同一份资源
e. /etc/passwd 记录了不同用户的角色，只有 root用户才能修改
   i. 其中 root 用户和普通用户的记录分别是：
      root:x:0:0:root:/root:/bin/bash
      seed:x:1000:1000:Seed,123,,:/home/seed:/bin/bash
      1) 通过冒号分隔出了七个字段
      2) 第三个字段是UID value，value为0就说明用户是root用户



3. 任务1的过程
   a. 创建一个dummy read-only文件
      i. 尽管我们可以在任何只读文件中实现攻击，但是为了防止修改了一些重要的文件，我们选择创建一个这样的文件。
      ii. 在根目录下创建一个名为zzz的文件，将其设为对普通用户只读
         $ sudo touch /zzz
         $ sudo chmod 644 /zzz
      iii. 向其中写入一些东西。示例：
         $ sudo gedit /zzz
         $ cat /zzz
         111111222222333333
         $ ls -l /zzz

-rw-r--r-- 1 root root 19 Oct 18 22:03 /zzz

iv. 测试

$ echo 99999 > /zzz

bash: /zzz: Permission denied

1) 写入失败，说明权限设置成功了

b. 设置内存映射线程（ Memory Mapping Thread）

i. 从实验室的网站中下载 cow_attack.c 程序

ii. 这个程序有三个线程： main thread, write thread, madvise thread

1) 主线程可以匹配内存中的 /zzz，找到“222222”模式的位置

2) 另外两个线程用来攻击 OS 内核

c. 设置 write 线程

i. 将“222222”替换成“******"，因为这个mapped memory是COW类型的，所以我们只能写在这个内存的一份拷贝上

d. 设置memory advise（madvise）线程

i. madvise线程建议 OS kernel 丢弃 mapped memory 的私有拷贝，从而page table就会指回原始的匹配内存

e. 发动（launch）攻击

i. 设置死循环，不断地同时运行 write 线程和 madvise 线程

4. 任务2的过程

i. 创建一个叫charlie 的账号

1) 如果使用当前的 seed 账号来攻击，那么如果攻击完忘记恢复就会对将来的实验造成影响

ii. 保存这个password文件

1) 如果攻击出意外了，还能恢复

2) 也建议保存一个这个虚拟机的snapshot，以便出意外了也能回滚回来

iii. 如果攻击成功了，就可以看到shell中的prompt有一个#，如：

root@ubuntu# id

15. lec10 Format-String Vulnerability

1. 样例：

```
#include <stdio.h>
#include <stdarg.h>

int myprint(int Narg, ... )
{
  int i;
  va_list ap;                                ①

  va_start(ap, Narg);                        ②
  for(i=0; i<Narg; i++) {
    printf("%d  ", va_arg(ap, int));         ③
    printf("%f\n", va_arg(ap, double));      ④
  }
  va_end(ap);                                ⑤
}

int main() {
  myprint(1, 2, 3.5);                        ⑥
  myprint(2, 2, 3.5, 3, 4.5);                ⑦
  return 1;
}
```

a. 解读

i. ①的ap获得了这个可选参数

ii. 第二行的va_start() 这个macro通过Narg计算了va_list的初始地址

2. 问题

```c
#include <stdio.h>

int main()
{
    int id=100, age=25; char *name = "Bob Smith";

    printf("ID: %d, Name: %s, Age: %d\n", id, name);
}
```

a. 原理
   i. printf（）函数先将参数逆序放入栈中，然后每一次扫描打印时都将%d之类的的东西替换成下一个可选参数的值
b. 如果缺省一部分的参数
   i. va_args 不知道自己已经达到了边界，它仍然持续地向前推进，并且不断地从栈中获取数据
c. 如果用户输入成为输出的一部分

```c
printf(user_input);
```

```c
sprintf(format, "%s %s", user_input, ": %d");
printf(format, program_data);
```

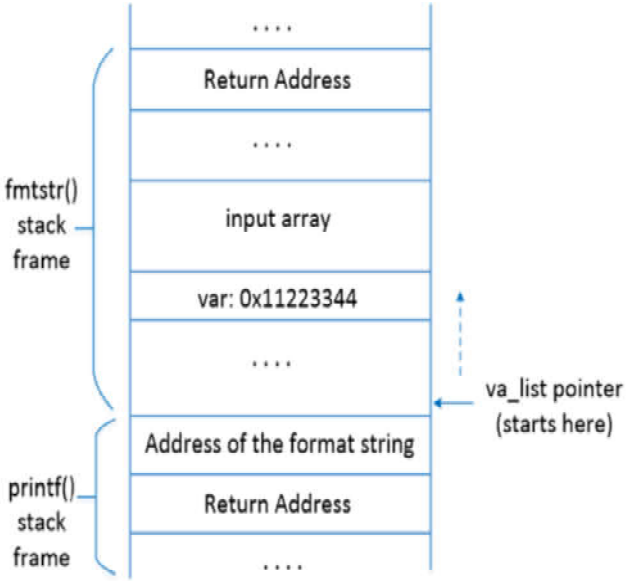
```c
sprintf(format, "%s %s", getenv("PWD"), ": %d");
printf(format, program_data);
```

   i. 如果用户输入中包含有format specifier那就可能会出事

3. 攻击代码

```c
#include <stdio.h>

void fmtstr()
{
    char input[100];
    int var = 0x11223344;

    /* print out information for experiment purpose */
    printf("Target address: %x\n", (unsigned) &var);
    printf("Data at target address: 0x%x\n", var);

    printf("Please enter a string: ");
    fgets(input, sizeof(input)-1, stdin);

    printf(input); // The vulnerable place     ①

    printf("Data at target address: 0x%x\n",var);
}

void main() { fmtstr(); }
```

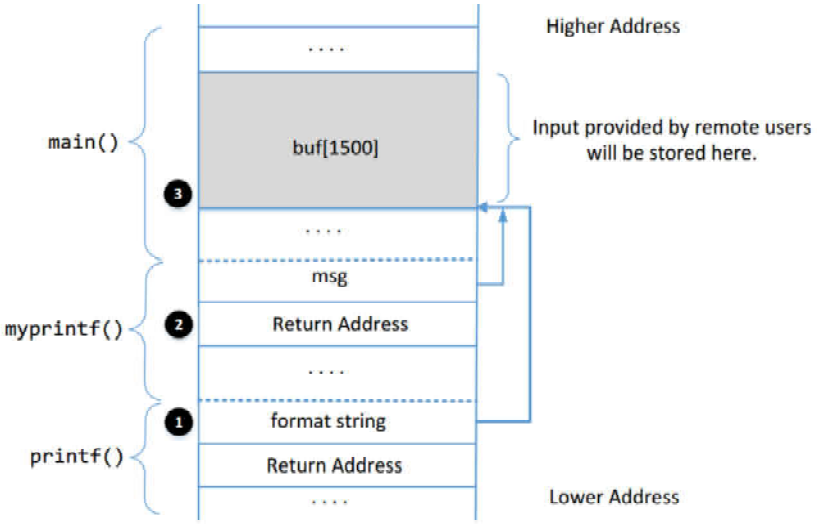
a. 问题代码的栈的示意图



4. 能够实现的攻击

a. crash program
    i. %s%s%s%s%s%s%s%s
    ii. 每一次程序都从栈中fetch一个值作为string的地址，如果这个值不是有效的地址，那程序就炸了
b. print put data on the stack
    i. %x%x%x%x%x%x%x%x
    ii. 这样就会把栈中的数据逐一打印出来
c. 改变内存中这个程序的数据
    i. 补充知识：
        1) %.5d 表示打印5位小数
        2) %n 在printf（）中，会打印出至今为止已经打印的字符的数量
            a) 例如：printf(“hello%n”,&i) 这样i就会存储5了
            b) %hn 使用2bytes来存储数据
            c) %hhn 使用1byte来存储数据
        3) $(command): 命令替换，允许command的输出替换command本身，一般用来表示无法显示的ASCII码
        4) “\x04”:表示04是数字而非两个ASCII码
    ii. 思想：
        1) 将va_list 的指针指向这个地方，然后通过%n来写入
        2) 使用%x来推进指针，要使用5个，而非四个
    iii. 假设var的地址是 0xbffff304 （能够通过gbd找到），那么指令可以是
```
$ echo $(printf "\x04\xf3\xff\xbf").%x.%x.%x.%x.%x.%n > input
```
d. 将这个程序的数据改成特定的值
e. inject malicious code

16. Format String Vulnerability Lab
    1. 概述
        a. C中的格式化字符串（format string）包括许多函数，比如：printf(), sprintf(), fprintf(), scanf()
        b. 实验环境：pre-built的Ubuntu 16.04 VM
    2. 实验任务
        a. 设置
            i. 为了简化任务，关闭了address randomization
                sudo sysctl -w kernel.randomize_va_space=0
        b. 任务1：有漏洞的程序
            i. 假设这个程序运行在服务器上，监听9090的UDP端口，一旦收到相应packet就invoke myprint( )函数。这个服务器是一个 root daemon，即它有root privilege，在myprint( ) 函数中，有一个格式化字符串漏洞
            ii. 编译
                1) 指令为 gcc -z execstack -o server server.c
                2) 要加上 -z execstack ，不然栈中可能无法执行程序。当然，通过使用 return-to-libc 技术，这个也可以被攻破
                3) gcc 会发出警告，在这里我们忽略这个警告
            iii. 在服务器VM上
                1) sudo ./server
            iv. 在客户VM上
                1) nc -u 10.0.2.5 9090
                2) -u表示使用UDP
                3) 然后就可以输入字符串了，服务器都可以收到
        c. 任务2：理解栈中的布局



17. web security
    1. 概论
        a. 攻击种类
            i. cross-site scripting (XSS), 跨站点脚本
            ii. session hijacking （会话劫持）
            iii. cross-site request forgery(CSRF)（跨站点请求伪造）
        b. 防范核心：验证输入
        c. defense in depth：到处防范，每一层都防范
    2. 网络概述
        a. 请求类型
            i. GET的所有data都在URL中，对服务器没有影响
            ii. POST则可能有影响
        b. 服务器
            i. SQL injection

```
$result = mysql_query("select * from Users
        where(name='$user' and password='$pass');");


$result = mysql_query("select * from Users
        where(name='frank' OR 1=1); --
            and password='whocares');");
```

            ii. 关键在于区分代码和数据，不要把它们搞混了
    3. 方法:
        a. 权限约束
        b. 加密敏感数据
    4. CSRF
        a. 让用户点击一个链接
    5. XSS
        a. 入侵网站，让网站发一下东西给用户
    6. js防范
        a. 这个js是哪里来的，他就只能使用哪里来的数据
18. Return-to-libc Attacks　lec12　12/2
    1. Non-executable Stack countermeasure

```
char buffer[sizeof(code)];
strcpy(buffer, code);
((void(*)( ))buffer)( );
```

Calls shellcode

        a. 把buffer数组转换成一个指向函数的指针并且执行这个函数
    2. main idea

a. 虽然不能在栈中执行，但是可以跳转到原有的代码中
b. 实验要求
    i. 打开non executable stack
    ii. 但是要关闭 地址随机化和stackGuard
    iii. 授予Set-UID程序root权限
        1) Set-UID：当一个Set-UID程序运行的时候，它被假设为具有拥有者的权限。
            a) 例如，如果程序的拥有者是root，那么任何人运行这个程序时都会获得程序拥有者的权限

3. 任务概述
    a. 任务1：任务2
        i. 找到栈中用来放置“bin/sh”的位置，这是system（）的参数
4. 任务1：找到system（）的地址，这样才能直接把这个位置写到return address上
    a. 使用gdb来打印system（）和exit（）的位置

```
$ gdb stack
(gdb) run
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7e5f430 <system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7e52fb0 <exit>
(gdb) quit
```

5. 任务2：找到“bin/sh”这个string的位置
    a. 为了方便，在cmd中声明这个环境变量参数，再在程序中获取它

```
$ export MYSHELL=/bin/sh
$ env | grep MYSHELL
MYSHELL=/bin/sh
```

    b.

```
void main(){
   char* shell =  getenv("MYSHELL");
   if (shell)
      printf("%x\n", (unsigned int)shell);
}
```

6. 任务3：给system（）构造参数

```
$ gcc -o exploit exploit.c
$./exploit          // create the badfile
$./retlib           // launch the attack by running the vulnerable program
# <---- You've got a root shell!
```