Android内核攻击面临的挑战

张延清 武汉科锐

2021 SDC分会场-公开课

Android App防御现状

- 用户权限保护
- 无法获取内核权限保护

攻击方式

- 内核攻击

用户权限如何进入内核权限?

- 重写系统代码，编译系统代码刷入手机
- 加载内核驱动进入内核

如何绕过驱动加载验证？

- 重新编译内核，通用性较弱
- 内核文件二进制打补丁，通用性较强

Android系统加载内核驱动的验证函数

- 内核函数check_version版本校验
- 内核函数module_sig_check签名校验

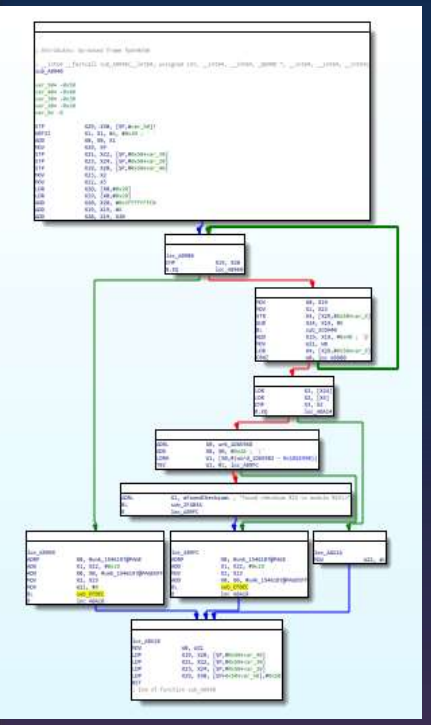
check_version实现代码

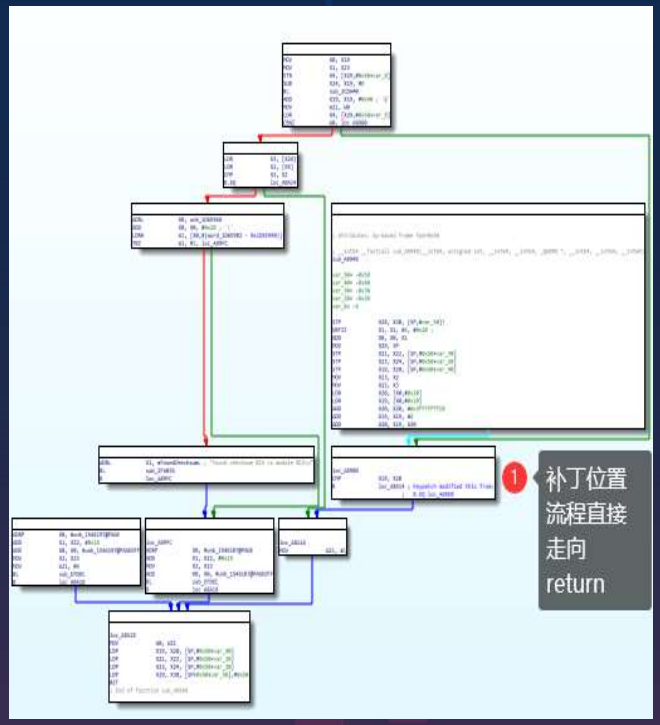
```
static int check_version(const struct load_info *info,
                         const char *symname,
                         struct module *mod,
                         const s32 *crc)
```

```
    /* Exporting module didn't supply crcs?  OK, we're already tainted. */
    if (!crc)
            return 1;

    /* No versions at all?  modprobe --force does this. */
    if (versindex == 0)
            return try_to_force_load(mod, symname) == 0;

    versions = (void *) sechdrs[versindex].sh_addr;
    num_versions = sechdrs[versindex].sh_size
            / sizeof(struct modversion_info);

    for (i = 0; i < num_versions; i++) {
            u32 crcval;

            if (strcmp(versions[i].name, symname) != 0)
                    continue;

            if (IS_ENABLED(CONFIG_MODULE_REL_CRCS))
                    crcval = resolve_rel_crc(crc);
            else
                    crcval = *crc;
            if (versions[i].crc == crcval)
                    return 1;
            pr_debug("Found checksum %X vs module %lX\n",
                     crcval, versions[i].crc);
            goto bad_version;
    }

    /* Broken toolchain. Warn once, then let it go.. */
    pr_warn_once("%s: no symbol version for %s\n", info->name, symname);
    return 1;
```

该函数主要用于判断驱动文件中内核函数crc与内核文件编译时生成的函数crc是否一致。

check_version打补丁思路





check_version函数头部进行补丁使其强制返回 1 即可。

load_module实现代码

```
/* Allocate and load the module: note that size of section 0 is always
   zero, and we rely on this for optional sections. */
static int load_module(struct load_info *info, const char __user *uargs,
                       int flags)
```

加载内核驱动实现代码。

```
    err = elf_header_check(info);
    if (err)
            goto free_copy;

    err = setup_load_info(info, flags);
    if (err)
            goto free_copy;

    if (blacklisted(info->name)) {
            err = -EPERM;
            goto free_copy;
    }

    err = module_sig_check(info, flags);
    if (err)
            goto free_copy;
```

1 校验返回失败则不能正常加载驱动

module_sig_check实现代码

```
#ifdef CONFIG_MODULE_SIG   通常情况厂商编译内核时默认打开此宏
static int module_sig_check(struct load_info *info, int flags)
{
```

```
/*
 * Require flags == 0, as a module with version information
 * removed is no longer the module that was signed
 */
if (flags == 0 &&
    info->len > markerlen &&
    memcmp(mod + info->len - markerlen, MODULE_SIG_STRING, markerlen) == 0) {
        /* We truncate the module to discard the signature */
        info->len -= markerlen;
        err = mod_verify_sig(mod, info);
}

if (!err) {
        info->sig_ok = true;   1 关键点
        return 0;
}
```

load_module正常执行需要保证module_sig_check函数返回 0，所以需要使内联的module_sig_check函数跳过验证部分，并使load_info->sig_ok = true。

module_sig_check函数在多个内核版本中均以内联形式出现在load_module函数中，因此需要对load_module函数进行分析。

Load_module打补丁思路

```
239  __int64 v243; // [xsp+108h] [xbp+108h]
240
241  LOBYTE(v11) = a3;
242  v243 = 0i64;
243  v12 = 0xFFFFFFFFFFFFFF82ui64;
244  v13 = *a1;
245  if ( !a3 )
246  {
247    v14 = a1[1];
248    if ( v14 > 0x1C
249      && !(unsigned int)sub_2CD3A0(v13 + v14 - 0x1C, (unsigned __int64)"~Module signature appended~\n", 0x1Cui64) )
250    {
251      a1[1] = v14 - 0x1C;
252      v20 = sub_ACF20(v13, a1 + 1, v15, v16, v17, v18, v19);
253      v12 = v20;
254      if ( !v20 )
255        *((_BYTE *)a1 + 0x44) = 1;
256    }
257  }
```

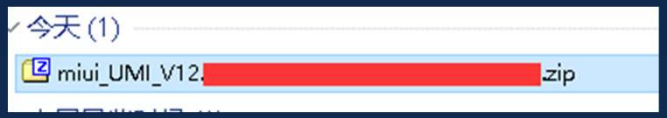
补丁前后对比。

```
229  __int64 v233; // [xsp+108h] [xbp+108h]
230
231  LOBYTE(v11) = a3;
232  v233 = 0i64;
233  LODWORD(v12) = 0xFFFFFF82;
234  v13 = *a1;
235  if ( !a3 )
236    *((_BYTE *)a1 + 0x44) = 1;
237  while ( 2 )
```

内核文件二进制打补丁流程

1. 申请手机解锁
2. 下载官方系统镜像
3. 提取kernel文件
4. 对kernel文件打补丁
5. 重打包系统镜像
6. 将系统镜像刷入手机

从系统镜像中提取boot.img文件




```
PS C:\Users\Tester\Desktop\新建文件夹> adb push .\boot.img /data/local/tmp
.\boot.img: 1 file pushed, 0 skipped. 27.4 MB/s (134217728 bytes in 4.668s)
PS C:\Users\Tester\Desktop\新建文件夹> adb push .\magiskboot /data/local/tmp
.\magiskboot: 1 file pushed, 0 skipped. 553.5 MB/s (294592 bytes in 0.001s)
PS C:\Users\Tester\Desktop\新建文件夹>
```

将提取boot.img传送到手机

从boot.img中提取kernel文件

```
chiron:/data/local/tmp # ./magiskboot unpack boot.img
Parsing boot image: [boot.img]
HEADER_VER      [2]
KERNEL_SZ       [50585612]
RAMDISK_SZ      [1324805]
SECOND_SZ       [0]
RECOV_DTBO_SZ   [0]
DTB_SZ          [1581709]
OS_VERSION      [11.0.0]
OS_PATCH_LEVEL  [2021-08]
PAGESIZE        [4096]
NAME            []
CMDLINE         [console=ttyMSM0,115200n8 androidboot.hardware=qcom androidboot.console=ttyMSM0 androidboot.memcg=1 lpm_
levels.sleep_disabled=1 video=vfb:640x400,bpp=32,memsize=3072000 msm_rtb.filter=0x237 service_locator.enable=1 androidbo
ot.usbcontroller=a600000.dwc3 swiotlb=2048 loop.max_part=7 cgroup.memory=nokmem,nosocket reboot=panic_warm buildvariant=
user]
CHECKSUM        [faa3e5561cf7df30403ead9ceef71efc2fb8414b00000000000000000000000]
KERNEL_FMT      [raw]
RAMDISK_FMT     [gzip]
chiron:/data/local/tmp #
```

从boot.img中提取kernel文件

```
-rw-r--r-- 1 root  root    1581709 2021-10-08 16:04 dtb
-rwxrwxrwx 1 shell shell  45187600 2021-09-30 12:10 fs
-rw-r--r-- 1 root  root   50585612 2021-10-08 16:04 kernel
-rwxrwxrwx 1 shell shell    294592 2021-01-17 06:12 magiskboot
```

```
PS C:\Users\Tester\Desktop\新建文件夹> adb pull /data/local/tmp/kernel .
/data/local/tmp/kernel: 1 file pulled, 0 skipped. 29.1 MB/s (50585612 bytes in 1.660s)
PS C:\Users\Tester\Desktop\新建文件夹>
```

补丁后重打包

通过magiskboot重打包。

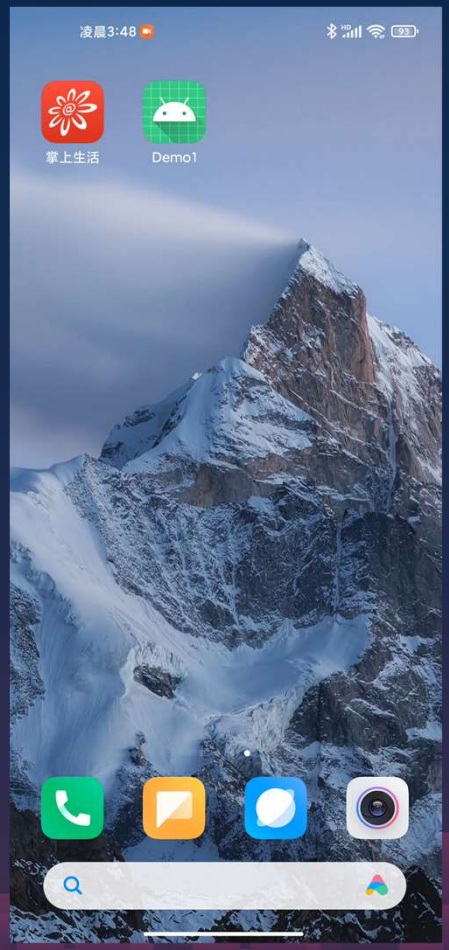
```
1|chiron:/data/local/tmp # ./magiskboot repack boot.img
Parsing boot image: [boot.img]
HEADER_VER      [2]
KERNEL_SZ       [50585612]
RAMDISK_SZ      [1324805]
SECOND_SZ       [0]
```

刷机

重启进入bootloader并使用fastboot刷机。

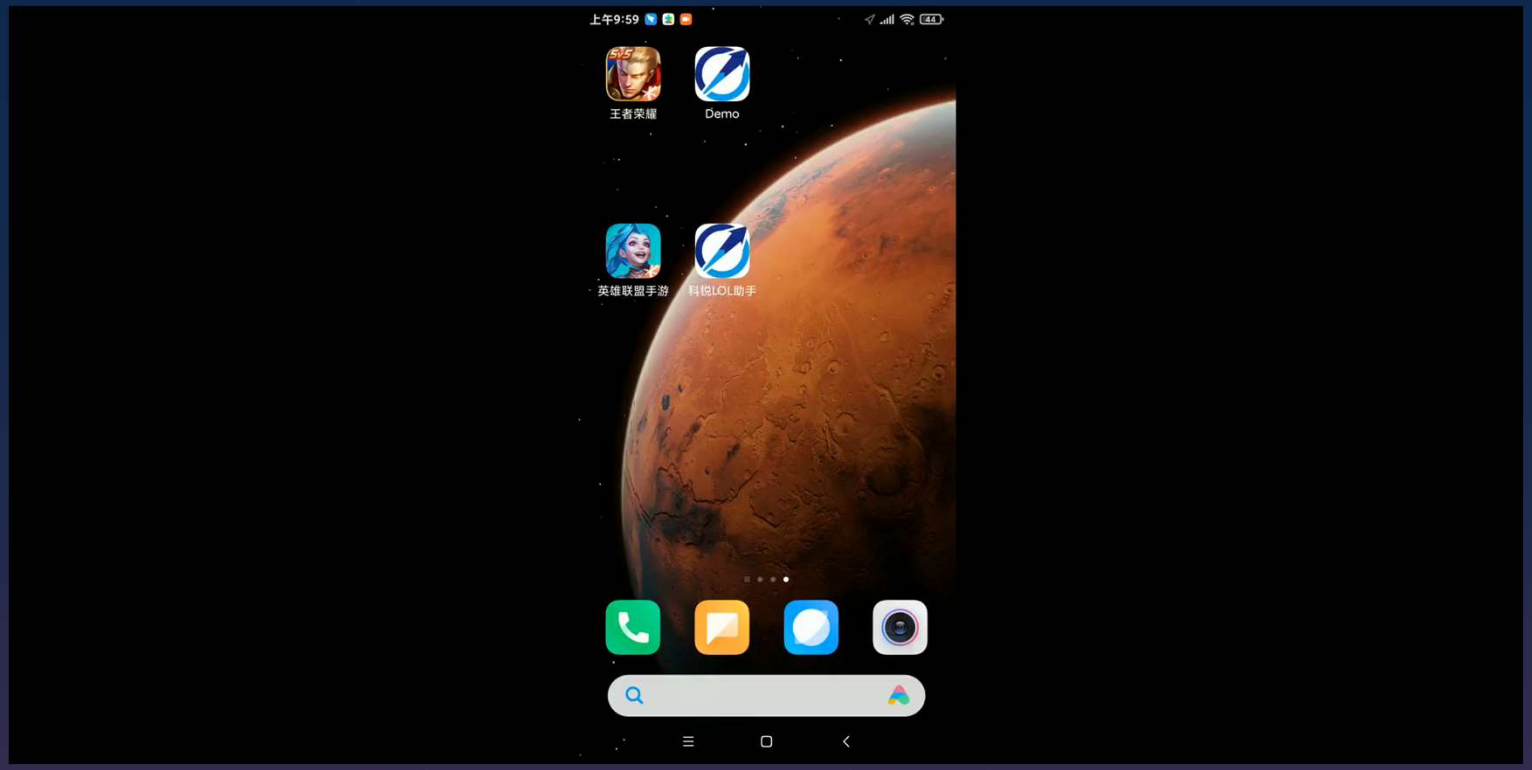
```
PS C:\Users\Tester\Desktop\新建文件夹> adb reboot bootloader
PS C:\Users\Tester\Desktop\新建文件夹> fastboot devices
8977205e        fastboot
PS C:\Users\Tester\Desktop\新建文件夹> fastboot flash boot new-boot.img
```

通过内核攻击网银APP



通过内核攻击社交APP

通过内核攻击手游APP



谢谢大家！