攻防视角下的MTK安全启动

Le Wu

2021年9月

1 背景/动机

Secure boot chain下的安全问题开始浮出水面，在业界崭露头角：

1 背景/动机

Secure boot是trustzone必不可少的部分，没有secure boot，trustzone的安全性无从谈起。

5.2 Booting a secure system

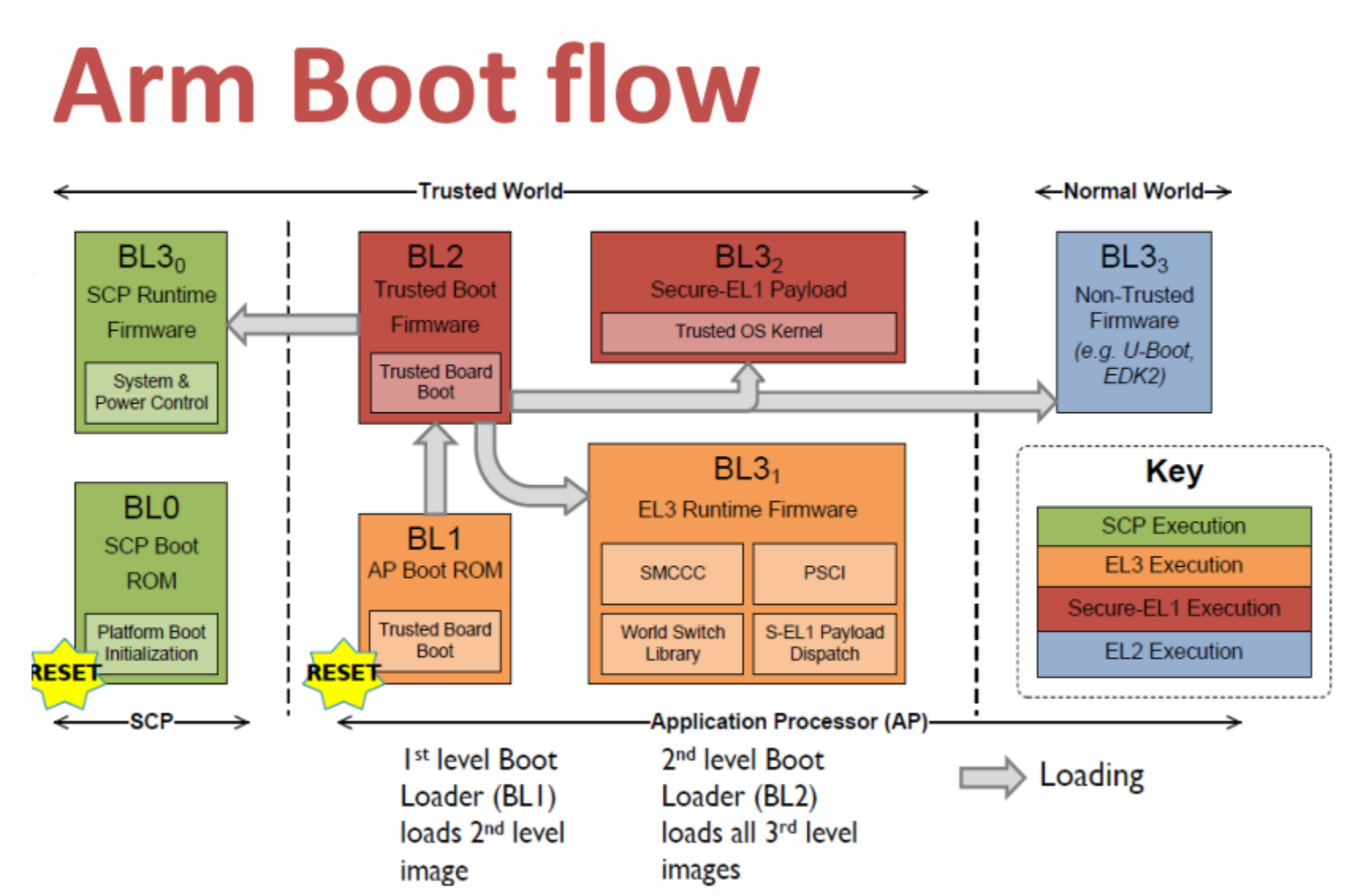
One of the critical points during the lifetime of a secure system is at boot time. Many attackers attempt to break the software while the device is powered down, performing an attack that, for example, replaces the Secure world software image in flash with one that has been tampered with. If a system boots an image from flash, without first checking that is it authentic, the system is vulnerable.

One of the principles applied here is the generation of a chain of trust for all Secure world software, and potentially Normal world software, established from a root of trust that cannot easily be tampered with. This is known as a secure boot sequence. See *Secure boot* on page 5-6.

（From trustzone_security_whitepaper）

2 安全启动

一般的启动流程：



2 安全启动

MTK secure boot 流程（以mt8768为例，没有teeos，exception level角度来看）

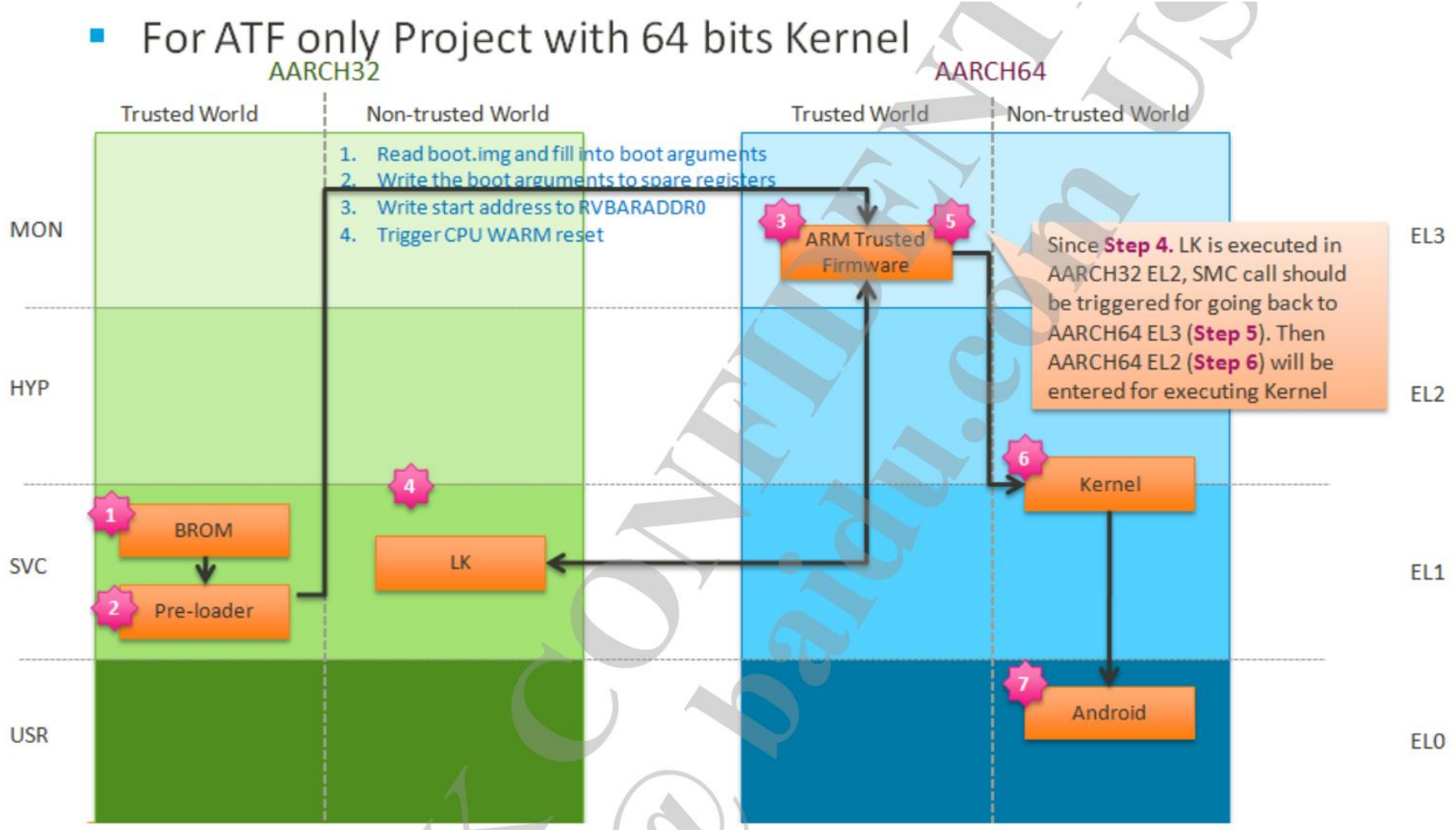
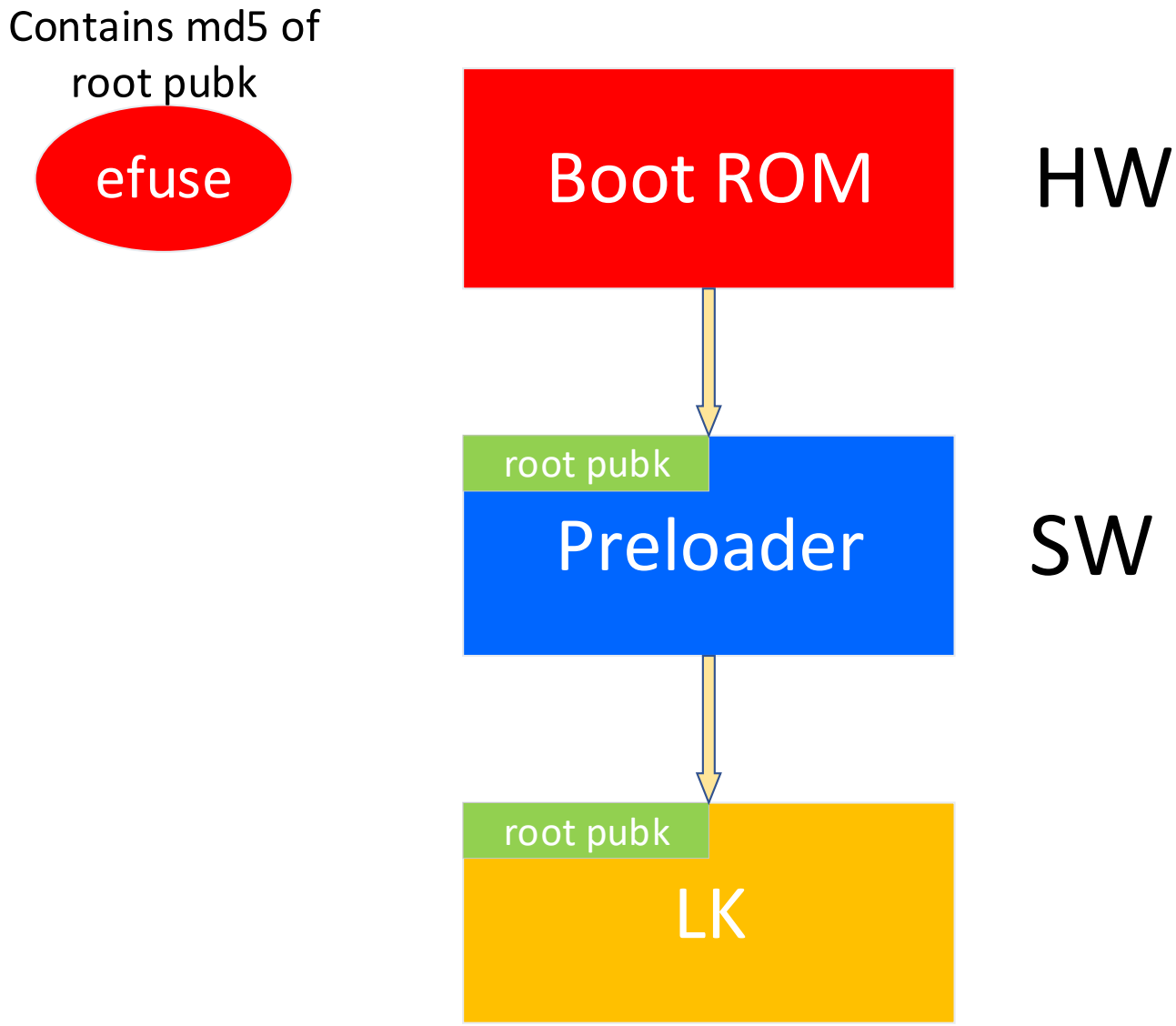


Figure 4 – Boot up flow

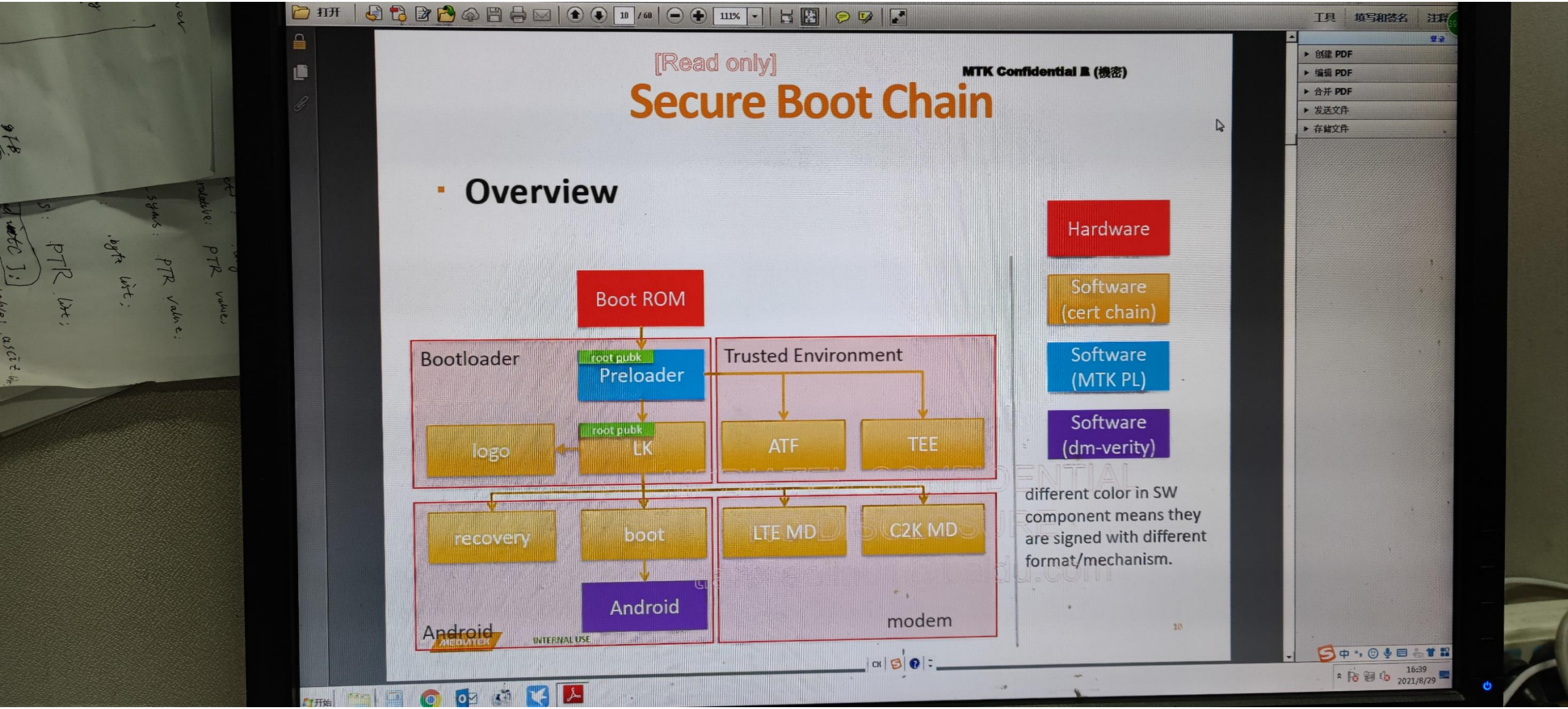
2 安全启动

Root of trust：



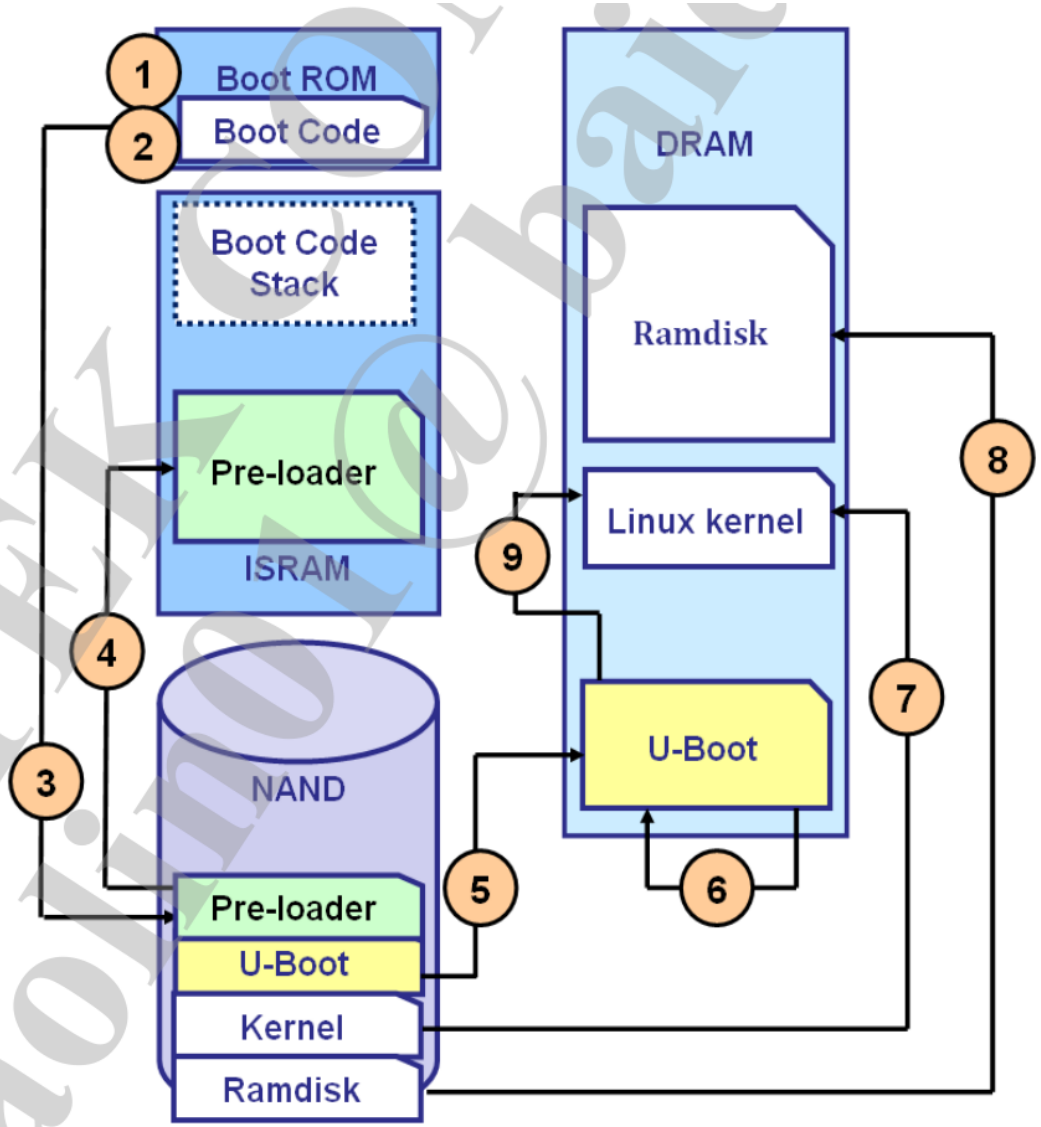
2 安全启动

总流程：


[Read only]
MTK Confidential B (機密)
Secure Boot Chain
Overview
Boot ROM
Bootloader
root pubk
Preloader
Trusted Environment
logo
root pubk
LK
ATF
TEE
recovery
boot
LTE MD
C2K MD
Android
Android
modem
Hardware
Software (cert chain)
Software (MTK PL)
Software (dm-verity)
different color in SW component means they are signed with different format/mechanism.
INTERNAL USE

2 安全启动

分区加载视角：



3 攻击及防御——攻击面1

攻击面1：image加载时，发生在image签名校验前+签名校验时的数据解析，
包括image文件格式解析，签名校验的流程等。
（根本原因是storage中的image可能是非法image）

举一个栗子：preloader阶段的write-where-what问题

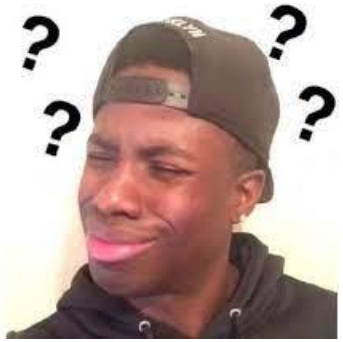
```
ret = get_part_info((u8 *)part_hdr, &maddr, &dsize, &mode, 1);
if (ret) {
    pal_log_err("[%s] image doesn't exist\n", MOD);
    return ret;
}
```

```
if (0 == (ret = blkdev_read(bdev, src, dsize, (u8*)maddr, part->part_id))) {
    if (addr) *addr = maddr;
    if (size) *size = dsize;
}
```

```
#ifdef MTK_SECURITY_SW_SUPPORT
    ms = get_timer(0);
    if (img_auth_required) {
        sec_malloc_buf_reset();
        ret = sec_img_auth_init(cur_part_name, part_hdr->info.name);
        if (ret) {
            pal_log_err("[%s] cert chain vfy fail...\n", MOD);
            ASSERT(0);
        }
        #ifdef MTK_SECURITY_ANTI_ROLLBACK
        ret = sec_rollback_check(0);
        if (ret) {
            pal_log_err("[%s] img ver check fail...\n", MOD);
            ASSERT(0);
        }
        #endif
    }
    ms = get_timer(ms);
    pal_log_info("[%s] part: %s img: %s cert vfy(%d ms)\n", MOD, cur_pa
#endif
```

3 攻击及防御——攻击面1

开个玩笑，这个栗子是假的。Write操作发生在签名校验后：



```
ret = get_part_info((u8 *)part_hdr, &maddr, &dsize, &mode, 1);
if (ret) {
    pal_log_err("[%s] image doesn't exist\n", MOD);
    return ret;
}
```

```
if (0 == (ret = blkdev_read(bdev, src, dsize, (u8*)maddr, part->part_id))) {
    if (addr) *addr = maddr;
    if (size) *size = dsize;
}
```

```
#ifdef MTK_SECURITY_SW_SUPPORT
    ms = get_timer(0);
    if (img_auth_required) {
        sec_malloc_buf_reset();
        ret = sec_img_auth_init(cur_part_name, part_hdr->info.name);
        if (ret) {
            pal_log_err("[%s] cert chain vfy fail...\n", MOD);
            ASSERT(0);
        }
        #ifdef MTK_SECURITY_ANTI_ROLLBACK
        ret = sec_rollback_check(0);
        if (ret) {
            pal_log_err("[%s] img ver check fail...\n", MOD);
            ASSERT(0);
        }
        #endif
    }
    ms = get_timer(ms);
    pal_log_info("[%s] part: %s img: %s cert vfy(%d ms)\n", MOD, cur_pa
#endif
```

3 攻击及防御——攻击面1

攻击手段：

此类问题一般的攻击手段是在el0阶段提权后写storage，对特定分区image进行破坏。在设备重新启动后，进入preloader等el3高权限阶段会进行image解析操作，从而触发el3权限下的代码任意执行。属于典型的权限提升（el0 ->el3）。

防护手段：

提升代码安全性；

验证签名之前尽量不做解析操作；

mtk write protect——保护核心分区在el0期间只读。

推荐论文：

Attacking Nexus 6 & 6P Custom Bootmodes

Roee Hay & Michael Goberman
IBM Security

3 攻击及防御——攻击面2

如何拯救变砖的设备？——download mode

总有一些特殊的情况（开发阶段或者用户使用期间）会破坏secure boot中的核心分区，如preloader，lk等，因为这些分区会被严格进行完整性检查，检查不过，设备就会变砖。

为了解决这类问题，一般厂商都会提供一个救砖的途径——download mode。这个模式一般会在bootrom阶段或者preloader早期，通过usb或uart协议来连接flash tool等，完成设备的image传输、刷机等，完成救砖。在这个模式下提供的定制化command自然成为了一个攻击面。

3 攻击及防御——攻击面2

漏洞1：secure el1任意代码执行

成因：

download模式下的两个特殊的command：

CMD_SEND_IMAGE:

接收来自远端flash tool的特定image存放在物理内存中，包括lk.img, tee1.img, boot.img等；并不对image的签名做校验；

CMD_BOOT_IMAGE:

跳转至特定image物理内存处执行；

CMD_SEND_IMAGE + CMD_BOOT_IMAGE就可以完成el3下代码任意执行；

3 攻击及防御——攻击面2

漏洞1利用——解锁刷机某平板：

3 攻击及防御——攻击面2

漏洞2：绕过write region保护，做物理内存读写操作

成因：

download mode中存在两个特殊的command：CMD_READ, CMD_WRITE

CMD_READ:可以用于读取特定物理内存地址的内容；读取物理内存地址为base_addr, 读取长度为length，base_addr和length均由flash tool控制。读取到的数据通过usb协议传输到flash tool端。

CMD_WRITE:可以用于向特定物理内存地址写入内容；写入物理内存地址为base_addr, 写入长度为length，base_addr和length均由flash tool控制。要写入的数据通过usb协议由flash tool传输到pc端。

为了保证CMD_READ， CMD_WRITE指令只能读写特定的物理内存， base_addr和length均会被限定在一个白名单中：

```
struct region {
    unsigned int start;
    unsigned int size;
};

static struct region g_read_whitelist[] = {
    {RGU_BASE, 0x1000},
    {USBDL_FLAG, 0x4},
    {DEVINFO_BASE, 0x1000}
};

static struct region g_write_whitelist[] = {
    {RGU_BASE, 0x1000},
    {USBDL_FLAG, 0x4}
};
```

3 攻击及防御——攻击面2

漏洞2成因：

base_addr=RGU_BASE;
length=0xffffffff;

```
/* check whether region 1 in in region 2 */
unsigned int is_in_region(struct region *region1, struct region *region2)
{
    unsigned int in = 0;

    if ((region1->start >= region2->start) &&
        (region1->start + region1->size) <=
        (region2->start + region2->size))
        in = 1;

    return in;
}

int whitelist_check(U32 addr, U32 len, U32 is_read)
{
    unsigned int i = 0;
    unsigned int whitelist_size = 0;
    struct region region;

    region.start = (unsigned int)addr;
    region.size = (unsigned int)len;

    if (is_read) {
        whitelist_size = sizeof(g_read_whitelist) / sizeof(struct region);
        for (i = 0; i < whitelist_size; i++) {
            if (1 == is_in_region(&region, g_read_whitelist + i))
                return 0;
        }
    } else {
        whitelist_size = sizeof(g_write_whitelist) / sizeof(struct region);
        for (i = 0; i < whitelist_size; i++) {
            if (1 == is_in_region(&region, g_write_whitelist + i))
                return 0;
        }

    }

    return -1;

} « end whitelist_check »
```

integer overflow

3 攻击及防御——攻击面2

其他漏洞：

其他的integer overflow；

逻辑编码错误；

……

如何缓解：

安全编码；

救砖&安全性的一个权衡（越早启动dl mode就可以拯救更多的砖，但是越早就意味着权限越大，安全风险越大）

3 攻击及防御——攻击面3

Fastboot mode下的厂商定制命令：

Fastboot mode是 LK执行阶段的一个特殊mode，此mode下可以进行刷机、解锁等操作。

如何进入Fastboot mode：

开机期间特定组合键或者adb reboot bootloader；

厂商基本都会有自己定制的command或者特殊的实现，比如常见的解锁命令：

fastboot oem unlock；

推荐读一读这篇论文，专门讲fastboot定制化问题：

fastboot oem vuln: Android Bootloader Vulnerabilities in Vendor Customizations

Roee Hay
Aleph Research, HCL Technologies

3 攻击及防御——攻击面4

Android刷机的奥秘——设备解锁

Android为了方便开发者进行刷机等操作，专门搞了一个“**锁**”（一块存储）。其有两个状态：LOCK, UNLOCK.

LOCK状态下： LK末尾阶段加载boot等分区时会对image做严格的签名校验，校验通过系统才会正常启动。
UNLOCK状态下：LK末尾阶段加载boot等分区时不对image做签名校验，系统依然可以正常启动。所以，我们可以在UNLOCK状态下随意对设备的android系统分区刷写。

“锁”状态存放在哪里？

mt8768存放在了seccfg分区中，通过md5+aes加密方式做了一定的保护（aes方式是否存在一机一密性？存在一机一密，则相对安全，否则可以在el0下root权限直接写seccfg分区，解锁设备）；

seccfg分区存在write protect；

缺陷：

在emmc的普通分区中直接存储“锁”信息，“锁”信息没有签名校验的流程做保护；

存在已知的解锁api，在获取代码执行能力的情况下可以直接调用api解锁；

3 攻击及防御——攻击面4

Android刷机的奥秘——设备解锁

更安全的“锁”实现？

“锁”信息存放在RPMB中（Replay Protected Memory Block），锁信息需要进行签名校验（trust of root类似），锁状态由签名验证的结果直接决定。需要trustzone参与。

4 Future work

- Bootrom 本身安全问题
- Lk阶段后的安全启动流程（Android Verified Boot）
- Mtk其他芯片？
- 其他手机厂商的安全启动流程？
- 很多细节问题

  整个权限模型，一些驱动，类似于kernel层面的一些问题等

Thanks