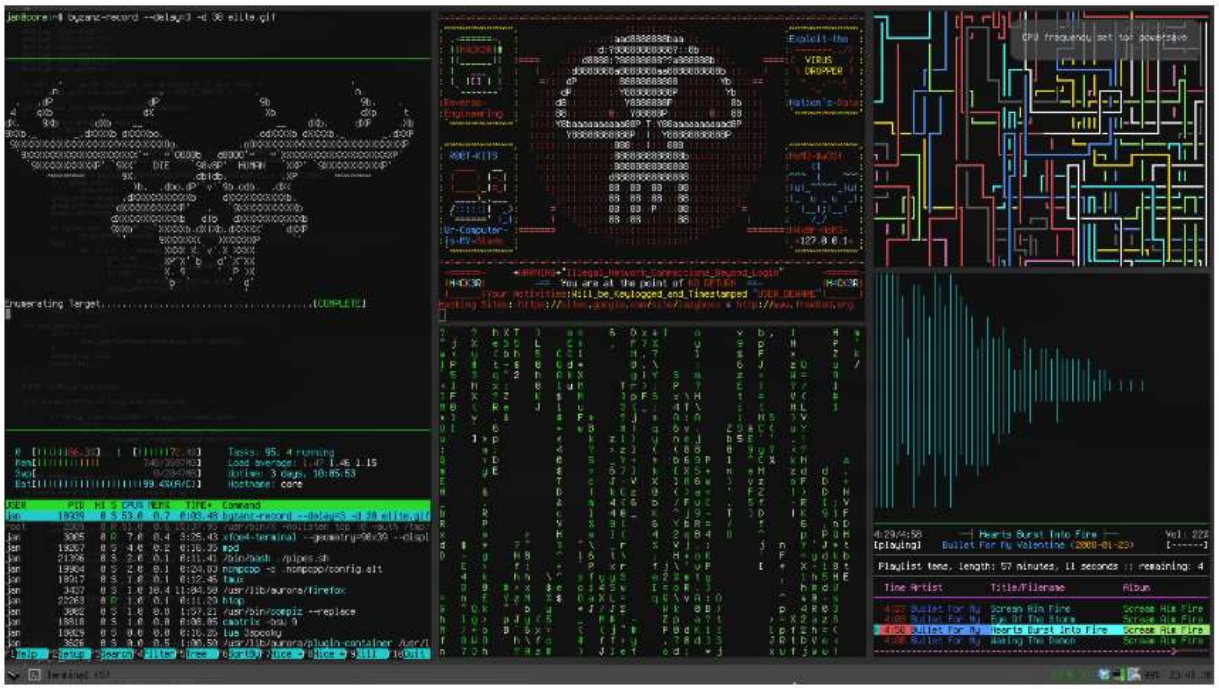
Khorina · 4月26日

[Trustzone]-Arm Trustzone的安全扩展介绍-一篇就够了

PSA TrustZone

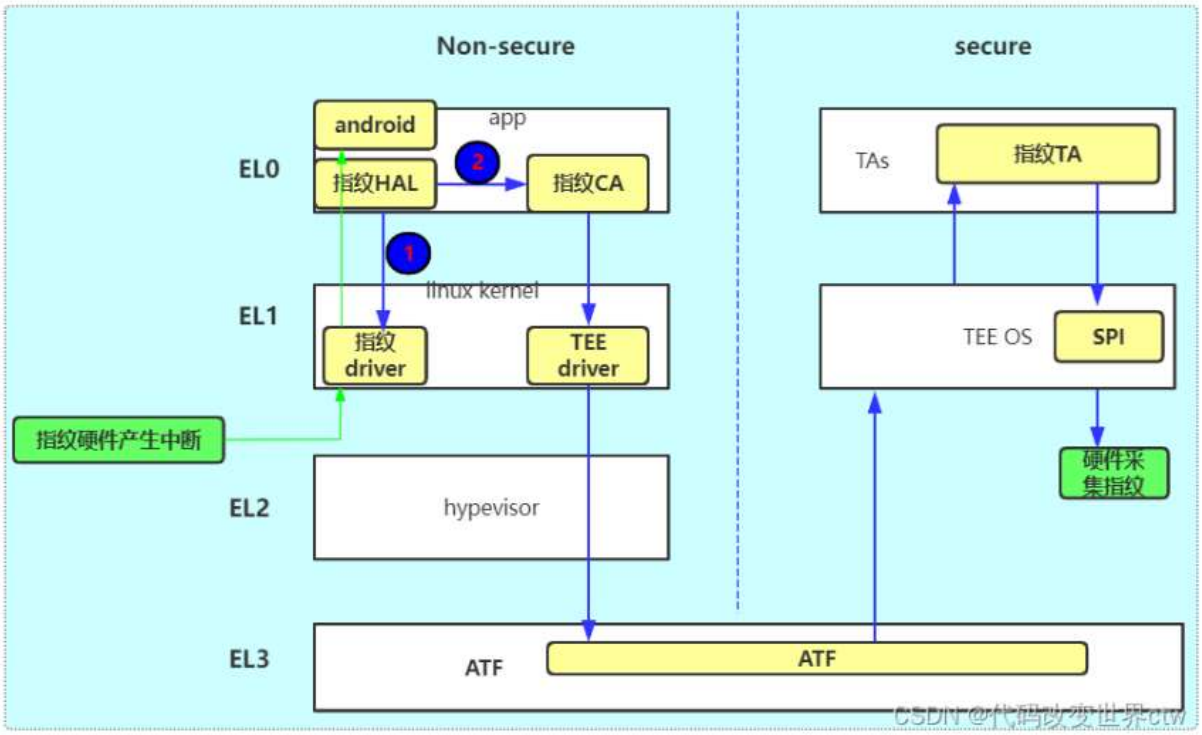


1、背景：

随着时代的发展、科技的进步，安全需求的趋势也越来越明显，ARM也一直在调整和更新其新架构，很多都是和安全相关的。如下列出了一些和安全相关的架构

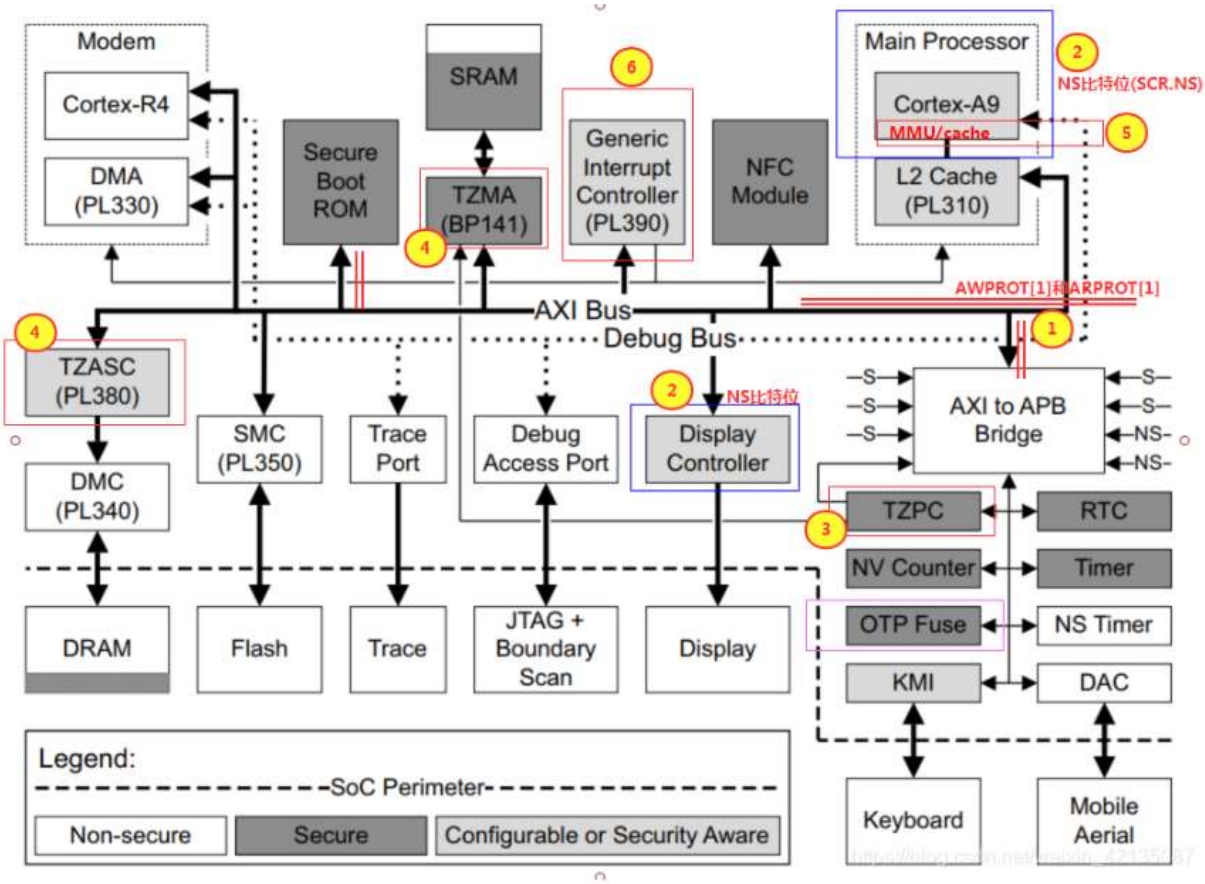
Trustzone做为ARM安全架构的一部分，从 2008 年 12月 ARM 公司第一次 release Trustzone 技术白皮书。(trustzone white paper – ARM Trustzone安全白皮书百度网盘下载, 密码:1234)

2013 年 Apple 推出了第一款搭载指纹解锁的 iPhone：iPhone 5s，用以保证指纹信息安全的 Secure Enclave 技术据分析深度定制了 ARM trustzone 架构，印象中这大概是 Trustzone 技术第一次走进大众视线。到如今 Trustzone 技术已经成为移动安全领域的重要基础技术，你也许不了解它的技术原理，但它一直默默为你守护你的指纹信息，账户密码等各种敏感数据。 如下也列出了一张在Trustzone架构下的一张指纹的框图，这也是这些年(2015-至今)比较流行的一张软件框图。



2、ARM Trustzone的安全扩展简介

从上文我们已经知道， ARM Trustzone不具体指一个硬件，也不是一个软件，而是一个技术架构，在支持ARM Trustzone的SOC中，需按照ARM Trustzone技术对各个子模块进行设计。如下便展示了一个SOC的Trustzone架构下的设计框图



其中：

(1)、AMBA-AXI总线的扩展, 增加了标志secure读和写地址线：AWPROT[1]和ARPROT[1]

(2)、processor的扩展(或者说master的扩展)，在ARM Core内部增加了SCR.NS比特位，这样ARM Core发起的操作就可以被标记“是以secure身份发起的访问，还是以non-secure身份发起的访问”

(3)、TZPC扩展，在AXI-TO-APB端增加了TZPC，用于配置apb controller的权限(或者叫secure controller)，例如将efuse(OTP Fuse)配置成安全属性后，那么processor以non-secure发起的访问将会被拒绝，非法的访问将会返回给AXI总线一个错误

rose配置成安全属性后，那么processor以non-secure发起的访问将会被拒绝，非法的访问将会返回给AXI总线一个错误。

(4)、TZASC扩展，在DDRC(DMC)之上增加一个memory filter，现在一般都是使用TZC400，或由SOC厂商自己设计一个这样的IP，或叫MPU，或集成在DMC内部，它的作用一般就是配置DDR的权限。 如果配置了DDR中某块region为安全属性，那么processor以non-secure发起的访问将会被拒绝。

(5)、MMU/Cache对安全扩展的支持

在软件架构的设计中，就分为: Non-secure EL0&1 Transslation Regime 和 Secure EL0&1 Transslation Regime，即normal world和secure world侧使用不同的Transslation Regime，其实就是使用不同的TTBRx_ELn寄存器，使用不同得页表。

注意：在armv7上，TTBRx_EL0、TTBRx_EL1是banked by Security State，也就是说在安全世界和非安全世界各有一组这样的寄存器，所以在linux和tee中可以各自维护一张自己的内存页表.

在armv8/armv9上，TTBRx_EL0、TTBRx_EL1不再是banked了，但是world switch时会在ATF中switch cpu context, 所以从hypervisror或os的视角来看，依然还是两套不同的TTBRx_ELn寄存器，linux和tee各有各的页表。

而在TLB中，又为每一个entry增加了Non-secure属性位，即标记当前翻译出的物理地址是secure还是non-secure；

cache的扩展：在cache的entry中的TAG中，有一个NON-Secure Identifier标记为，表示当前缓存数据的物理地址是属于non-secure还是secure。

(6)、gic对安全扩展的支持，在gicv2、gicv3的版本中，都增加了对安全扩展的支持. 以gicv3为例，将中断划分成了group0、secure group1和non-secure group1. 在软件的配置下，group0和secure group1的中断将不会target到REE(linux)中处理

3、ARM Trustzone的安全扩展详细解剖

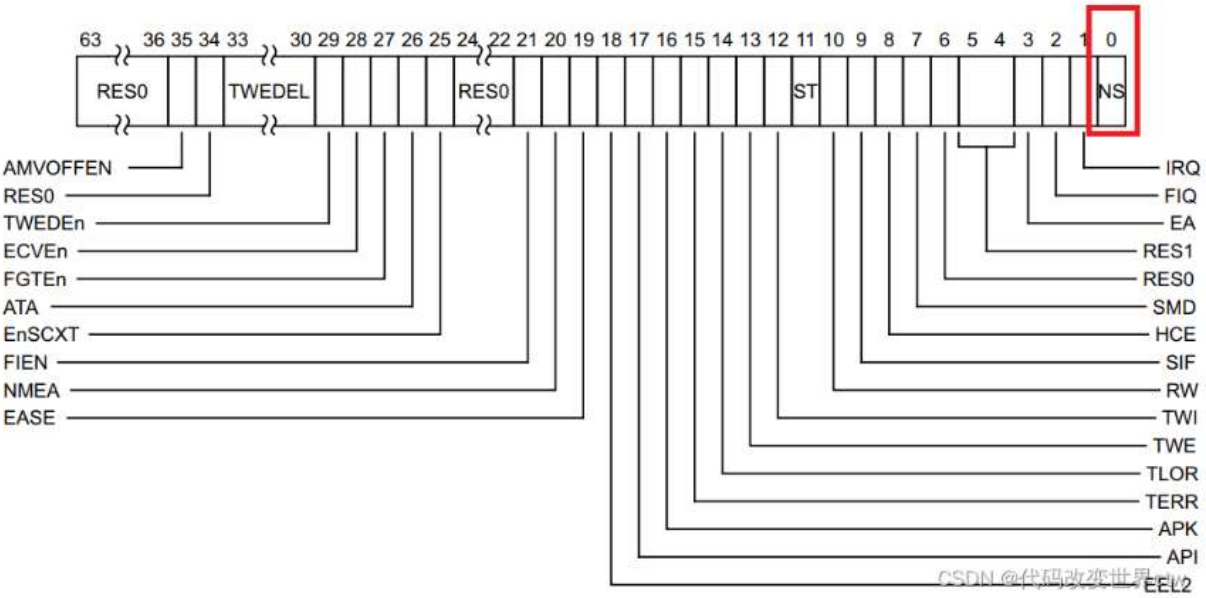
3.1 AMBA-AXI对Trustzone的支持

ARPROT[2:0]和AWPROT[2:0] 分别是读通道和写通道中的关于权限的信号，例如他们中的BIT[1]则分别表示正是进行secure身份的读或secure身份的写操作。

AxPROT	Value	Function
[0]	0	Unprivileged access
	1	Privileged access
[1]	0	Secure access
	1	Non-secure access
[2]	0	Data access
	1	Instruction access

3.2 Processor的SCR.NS比特位

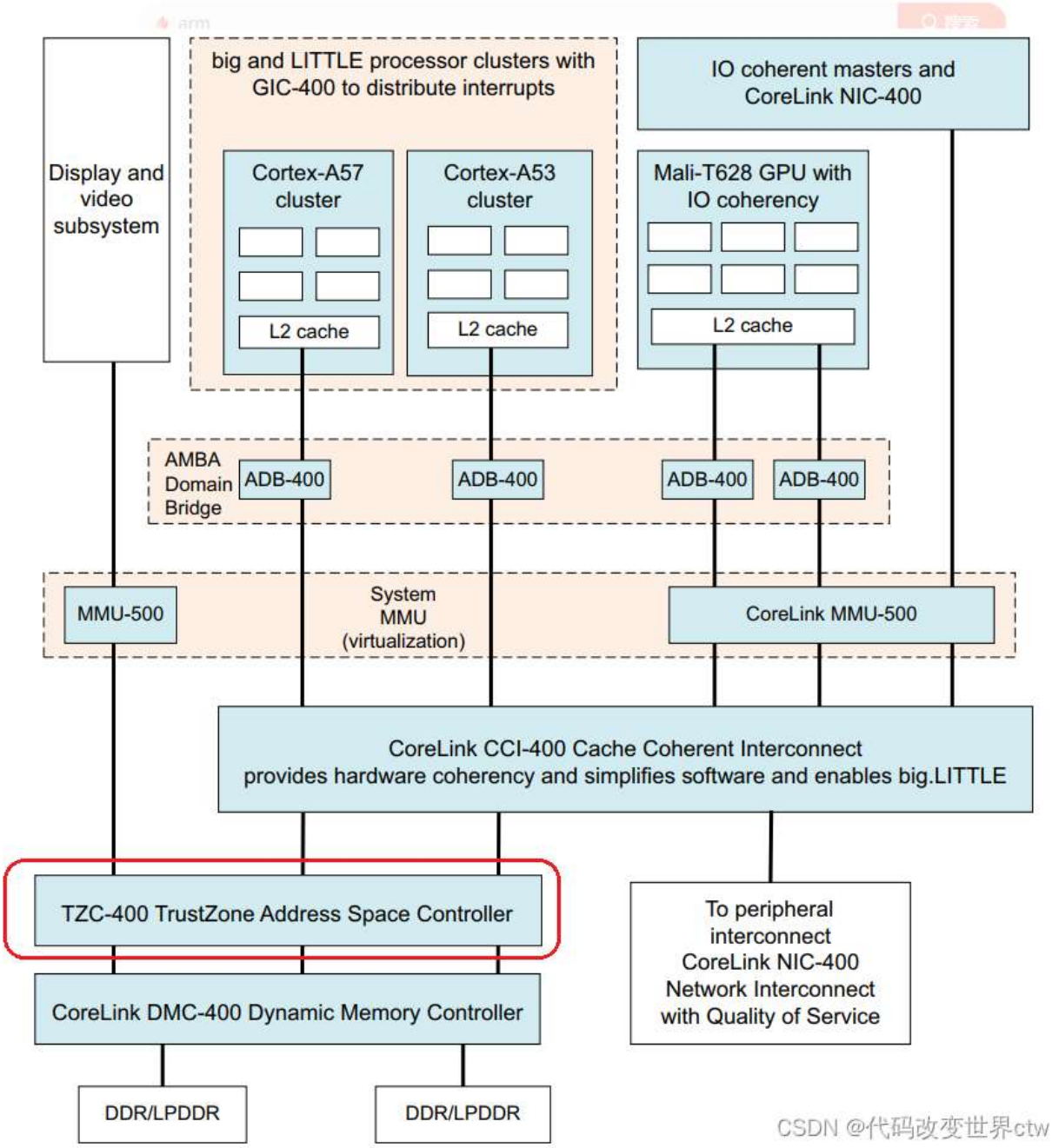
SCR_EL3.NS 表示当前processor的安全状态，NS=1表示是non-secure的，NS=0表示是Secure的



3.3 TZC400和TZPC简介

TZC400接在core和(DMC)DDR之间，相当于一个memory filter。

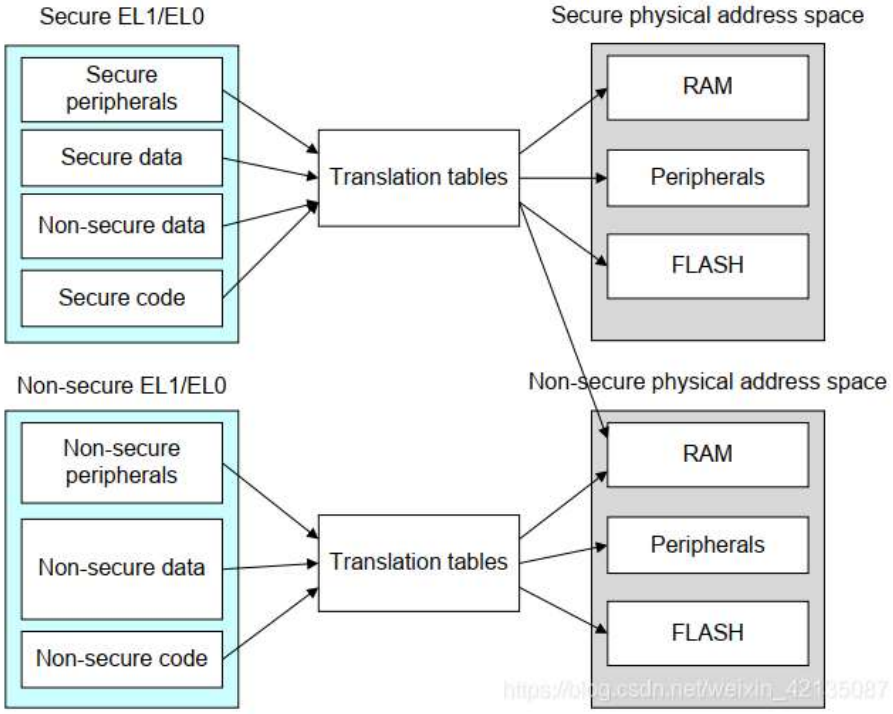
TZC400一般可以配置8个region（算上特殊region0, 也可以说9个），然后可以对每一个region配置权限。例如讲一块region配置成secure RW的，那么当有non-secure的master来访问这块内存时，将会被TZC挡住。



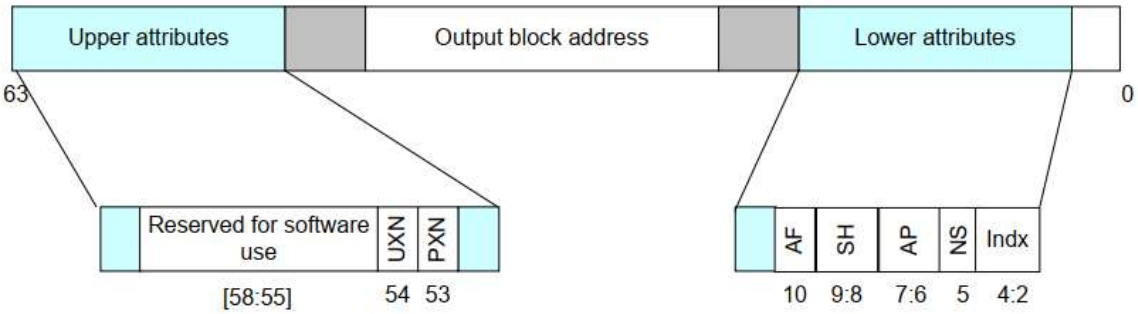
3.4 MMU对Trustzone的支持

首页，在软件架构的设计中，就分为: Non-secure EL0&1 Transslation Regime 和 Secure EL0&1 Transslation Regime，即normal world和secure world侧使用不同的Transslation Regime，其实就是使用不同的TTBRx_ELn寄存器，使用不同得页表

其次，在MMU使用的页表中，也有NS比特位。 Non-secure Transslation Regime 只能翻译NS=1的页表项，secure Transslation Regime 可以翻译NS=1和NS=0的页表项。即secure的页表可以映射non-secure或secure的内存，而non-secure的页表只能去映射non-secure的内存，否则在转换时会发生错误



在Page Descriptor中(页表entry中)，有NS比特位（BIT[5]），表示当前的映射的内存属于安全内存还是非安全内存：



- Unprivileged eXecute Never (UXN) and Privileged eXecute Never (PXN) are execution permissions.
- AF is the access flag.
- SH is the shareable attribute.

3.5 cache对Trustzone的支持

如下所示，以为cortex-A78为例，L1 Data Cache TAG中，有一个NS比特位（BIT[33]），表示当前缓存的cacheline是secure的还是non-secure的

Bit field	Description
[63:41]	0
[40:34]	ECC
[33]	Non-secure identifier for the physical address
[32:5]	Physical address [39:12]
[4:3]	Reserved
[2]	Transient/WBNA
[1:0]	MESI **00** Invalid **01** Shared **10** Exclusive **11** Modified with respect to the L2 cache

3.6 TLB对Trustzone的支持

如下所示，以为cortex-A78为例，L1 Data TLB entry中，有一个NS比特位（BIT[35]），表示当前缓存的entry是secure的还是non-secure的

L1 data TLB cache format for data register 0

Bit field	Description
[63:62]	Virtual address [13:12]
[58]	Outer-shared
[57]	Inner-shared
[52:50]	Memory attributes: **000** Device nGnRnE **001** Device nGnRE **010** Device nGRE **011** Device GRE **100** Non-cacheable **101** Write-Back No-Allocate **110** Write-Back Transient **111** Write-Back Read-Allocate and Write-Allocate
[38:36]	Page size: **000** 4KB **001** 16KB **010** 64KB **011** 256KB **100** 2MB **101** Reserved **110** 512MB **111** Reserved
[35]	Non-secure
[34:33]	Translation regime: **00** Secure EL1/EL0 **01** Secure EL3 **10** Non-secure EL1/EL0 **11** Non-secure EL2
[32:17]	ASID
[16:1]	VMID
[0]	Valid

3.7 gicv的安全中断

在gicv2/gicv3中，支持了安全中断，配置有如下：

（1）、Group分组(GICD_IGROUPRn) – gicv2

- group0：安全中断，由nFIQ驱动
- group1：非安全中断，由nIRQ驱动

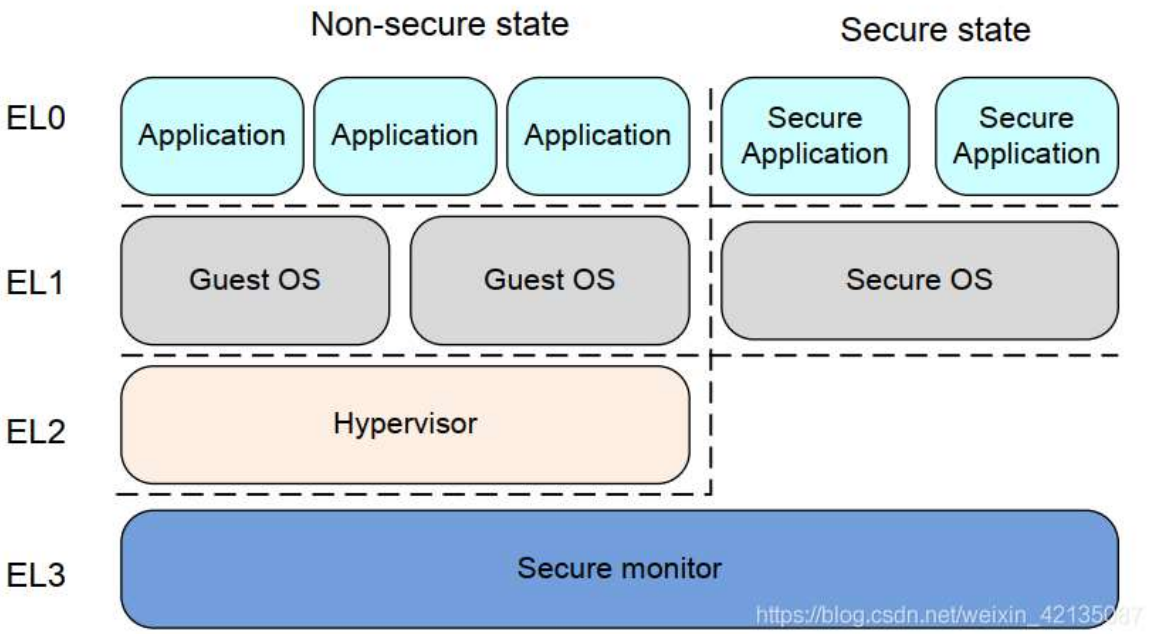
（2）、Group分组(GICD_IGROUPRn)– gicv3

- group0：安全中断
- non-secure group1：非安全中断
- secure group1：安全中断

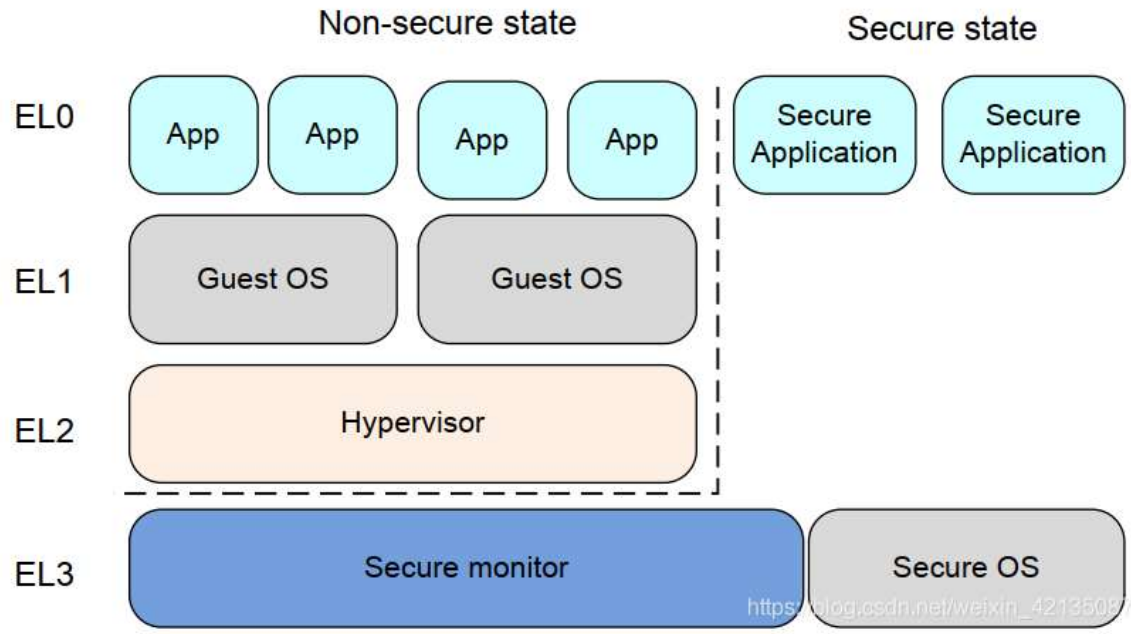
4、ARM Trustzone技术对软件带来的变化

ARM Trustzone技术对软件框架带来了变化

4.1、EL3 is AArch64：



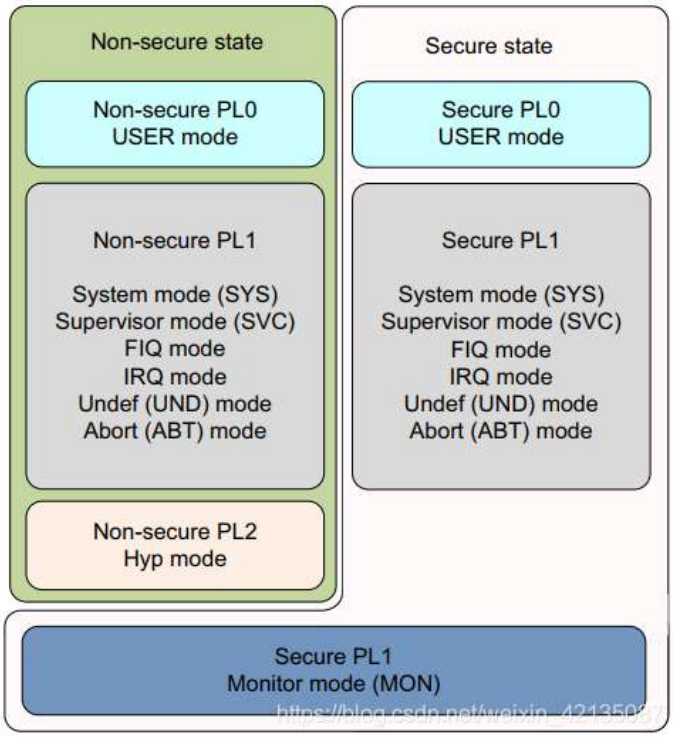
4.2、EL3 is AArch32：



AArch32和AArch64 secure monitor的理解：

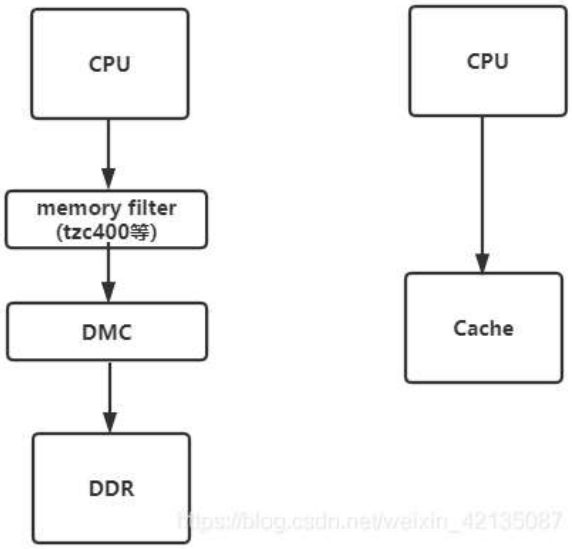
如果secureos和monitor都是64位，secureos跑在el1, monitor跑在el3;
如果secureos和monitor都是32位，secureos和monitor都跑在EL3（secureos在svc模式、monitor在svc模式），它俩共用页表；
如果monitor是64位，secureos是32位，那么secureos跑在svc模式(el1)，monitor跑在el3，他俩不共用页表

4.3、armv7：

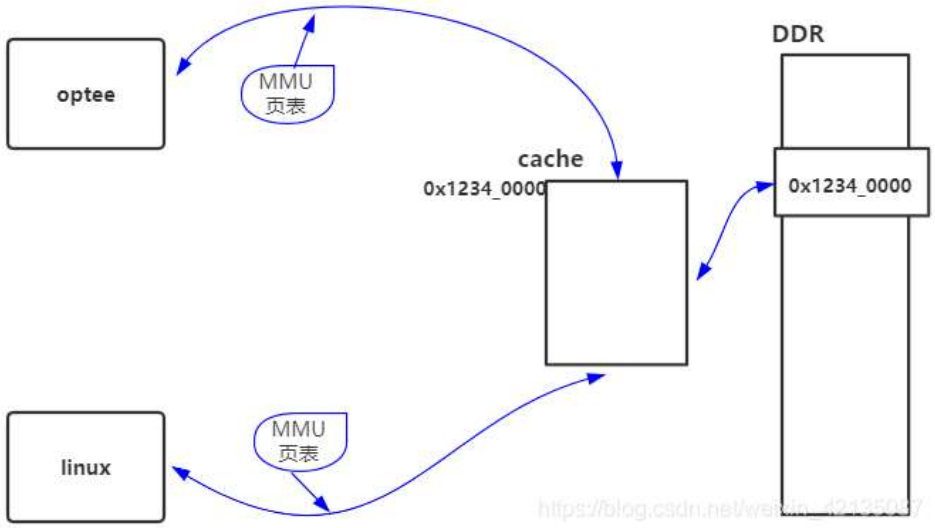


5、思考：通过MMU/TLB/Cache对安全内存攻击的可能性

在安全架构的设计时，我们在Core和DDR之间增加了一个TZC做为memory filter，数据流为:Core ---> TZC---->DDR, 这种架构下，core以非安全身份发起的对安全内存的读写，将会被TZC挡住。



但是这都是在理想的情况下，事实上Core发起对内存的读写，未必经过TZC未必到DDR，有可能到cache阶段就完成了，即数据流变成了Core ---> MMU(TLB+Addtress Translation)---->Cache，那么这种情况下，没有TZC的事了，你也许会说MMU/Cache中都有NS比特，但是你真的理解这里NS比特的用法吗？ 如果core以非安全身份对安全内存发起的读写时，我强制将MMU页表中的安全属性标记位强制改成NS=0，会如何呢？



事实上我们只要理清原理、理清数据流，就不会问上面那么S13的问题了。 下面来开始剖析:

假设一个安全core 读取了一个安全物理内存0x2000_0000数据(虚拟地址可能是0x_xxxx_xxxx)，那么将产生一下行为:

在读写之前，势必做好了MMU map，如物理地址0x2000_0000 MAP成了0x_xxxx_xxxx地址, 此时Page Descriptor中的atrribute中的NS=0
TLB缓存该翻译，即TLB的entry中包含: 0x2000_0000、0x_xxxx_xxxx、NS=0
安全内存0x2000_0000数据将会被缓存到cache中，entry中的TAG包含0x2000_0000、NS=0
同时，我有一个非安全core 发起读写虚拟地址0x_yyyy_yyyy，我自行修改该页表，让0x_yyyy_yyyy强制映射到安全物理内存0x2000_0000，此时有两种配置:
(1)、0x_yyyy_yyyy—0x2000_0000, NS=0
(2)、0x_yyyy_yyyy—0x2000_0000, NS=1
我们分别看下这两种配置，是否能读到安全内存:
针对(1)，非安全的core发起访问，发现TLB中的条目是0x_yyyy_yyyy—0x2000_0000, NS=0,自然不会被命中，然后使用Address Translation转换，MMU发现非安全的Core要来访问安全属性NS=0 将会被直接拒绝掉。
针对(2)，非安全的core发起访问，由于NS=1，TLB可能会被命中，即能翻译出0x2000_0000物理地址来，即使没有被命中，在经过Address Translation转换，由于NS=1，此时也是可以正确转换出正确的0x2000_0000物理地址。 然后接着会去cache中查询这个地址，但是此时cache的entry中的NS=0，所以cache不会被命中，接下来就要走TZC流程了，很显然，你一个非安全的core想访问安全的内存，TZC将会挡住你。

综上所述：安全就是安全，不要再瞎想漏洞了。

> 作者：代码改变世界ctw
> **原文链接**：https://blog.csdn.net/weixin_42135087/article/details/109272384

推荐阅读

TrustZone和PSA之间是什么关系？ 8
Arm平台安全架构(PSA)详解（视频+PPT)
11月1日-一起来听PSA专家给你介绍Arm平台安全架构 2
中国自主设计之山海安全方案(视频)
极术干货|张晓楠-Arm 平台安全架构(PSA)（PPT下载+视频回放）
TrustZone是如何支持安全中断的？ 5