## 1 **操作关键数据度量模块**

### 1.1MMU使能位

1. 功能描述

基于ARMv8硬件架构和Linux内核结构体，实现MMU使能位动态度量模块，设计实现在进程启动前，对MMU使能位进行勾取，将勾取信息从内核发送TA，并且设计实现了MMU使能位动态度量存储事件日志Event hash值以及链式存储事件日志的功能。

1. 实现方式

关键函数：

check\_MMU()：该函数的作用是读取系统的 SCTLR\_EL1 寄存器，并将其第 2 至 31 位清零，然后将修改后的值存储到一个名为 data 的 Mem\_Critical\_Data 结构体的 MMU\_status 字段中，并返回 0。

该函数首先声明了两个整型变量 sctlr\_el1 和 st\_MMU，其中 sctlr\_el1 未被使用。接下来，使用内嵌的汇编代码来读取 SCTLR\_EL1 寄存器的值，并将其存储到 st\_MMU 变量中。

内嵌汇编代码的具体过程如下：

"mrs %0, SCTLR\_EL1 \n\t"：使用 MRS 指令将 SCTLR\_EL1 寄存器的值读取到寄存器 %0 中。

"bic %0,%0,#0xfffffffe \n\t"：使用 BIC 指令将 %0 中第 2 至 31 位的值清零。

最后将 st\_MMU 变量的值作为函数返回值，同时将其存储到 data 结构体的 MMU\_status 字段中。

流程图：



### 1.2 页表读写执行控制位

1. 功能描述

基于ARMv8硬件架构和Linux内核结构体，实现页表读写执行控制位动态度量模块，设计实现在进程启动前，对页表读写执行控制位进行勾取，将勾取信息从内核发送TA，并且设计实现了页表读写执行控制位动态度量存储事件日志Event hash值以及链式存储事件日志的功能。

1. 实现方式

关键函数：

check\_PageRW()：该函数实现了检查内核页表中的页面是否具有写入权限的功能，并将结果存储在传入的结构体 Mem\_Critical\_Data 中。函数主要的实现过程如下：

使用汇编语言内嵌代码，读取 TTBR1\_EL1 寄存器的值，即内核页表的基地址，存储在变量 kernel\_pagetable\_base 中。

使用 kallsyms\_lookup\_name 函数分别获取内核代码段的起始地址 \_text 和结束地址 \_etext。

从 \_text 开始，每次增加一个页面（大小为 4KB），扫描内核页表中的每个页面，获取该页面对应的物理地址，检查对应的 PTE（Page Table Entry）的第 51 位是否为 1，若为 1 则表示该页面具有写入权限，否则表示只有只读权限。将页面的写入权限信息存储在 write\_flags 数组中，数组中每个元素代表 8 个页面的权限信息。

用 SM3 哈希函数，计算 write\_flags 数组的哈希值，将结果存储在 Hash 数组中。

将Hash数组的值复制到传入的 Mem\_Critical\_Data 结构体中的 kernel\_page\_rw\_status 成员变量中。

打印调试信息，包括内核页表基地址、扫描的页面数量、内核代码段的起始和结束地址等。

返回 0 表示执行成功。

流程图：



### 1.3 中断描述符表

1. 功能描述

基于Linux内核结构体和硬件资源分配，实现中断描述符表动态度量模块，设计实现在进程启动前，对ARMv8架构下Linux内核的中断描述符表数据结构进行勾取，将勾取信息从内核发送TA，并且设计实现了中断描述符表动态度量存储事件日志Event hash值以及链式存储事件日志的功能。

1. 实现方式

关键函数：

check\_idt()：该函数的作用是检查系统的中断描述符表(IDT)是否存在，并将其复制到一个内存关键数据结构中。

函数参数 data 是一个指向结构体 Mem\_Critical\_Data 的指针，该结构体包含了 IDT 的数据。函数首先声明并初始化一个无符号长整型变量 vbar\_el1，用于存储 VBAR\_EL1 寄存器的值，该寄存器存储了内核的地址空间中 IDT 表的基地址。

然后，使用汇编指令 mrs 将 VBAR\_EL1 寄存器的值加载到变量 vbar\_el1 中。接着，使用 memcpy 函数将 IDT 数据从内核的地址空间中复制到 dataIDT\_data 所指向的内存区域中，这个区域大小为 128 字节。

最后，函数返回值为 0 表示检查和复制 IDT 表操作成功完成。如果复制操作失败，则打印错误信息并返回错误代码。

流程图：



### 1.4 系统调用表

1. 功能描述

基于Linux内核结构体和进程资源调度方式，实现系统调用表动态度量模块，设计实现在进程启动前，对ARMv8架构下Linux内核的系统调用表数据结构进行勾取，将勾取信息从内核发送TA，并且设计实现了系统调用表动态度量存储事件日志Event hash值以及链式存储事件日志的功能。

1. 实现方式

关键函数：

check\_syscall()：这段代码定义了一个名为check\_syscall的函数，该函数接受一个指向Mem\_Critical\_Data结构体的指针data作为参数，并返回一个整数值。以下是这段代码的详细解释：

定义一个指针变量sys\_call\_table，并将其初始化为0。

使用kallsyms\_lookup\_name函数查找内核符号表中名为"sys\_call\_table"的符号，并将其地址赋值给sys\_call\_table指针变量。

使用memcpy函数将sys\_call\_table指向的内存区域中的内容复制到datasyscall\_table\_data指向的内存区域中。其中，datasyscall\_table\_data是一个指向存储系统调用表数据的缓冲区的指针，MAX\_SYSCALL\_NUM表示系统调用的最大编号（通常为一个固定的值，比如\_\_NR\_syscall\_max），8表示每个系统调用表项的大小（通常为64位）。

返回整数值0，表示函数执行成功。

流程图：



### 1.5 SELinux使能和配置

1. 功能描述

基于Linux内核结构体和SElinux功能模块，实现SELinux使能和配置动态度量模块，设计实现在进程启动前，对ARMv8架构下Linux内核的SELinux使能和配置数据结构进行勾取，将勾取信息从内核发送TA，并且设计实现了SElinux使能和配置动态度量存储事件日志Event hash值以及链式存储事件日志的功能。

1. 实现方式

关键函数：

check\_Selinux():它接收一个指向 Mem\_Critical\_Data 结构体的指针 data，并返回整数值 0。函数的目的是检查 SELinux 是否开启并执行一些操作。

函数中的第一行代码使用 kallsyms\_lookup\_name 函数从内核符号表中查找 selinux\_state 的地址，并将其存储在 selinux\_state\_address 变量中。这个变量是一个指向 selinux\_state 结构体的指针。

然后，memcpy 函数将 selinux\_state 结构体的内容复制到 data 指向的 Mem\_Critical\_Data 结构体中的 selinux\_state 字段中。这个字段的大小应该等于 selinux\_state 结构体的大小。

最后，函数使用 printk 函数打印一些调试信息。它打印 selinux\_state 字段的大小以及 enforcing 字段的值，该值指示 SELinux 是否处于执行强制访问控制（enforcement）模式。

流程图：



### 1.6 关键数据结构

#### 1.6.1 主要常量

关键数据度量中主要包含以下数据常量

#define KERNEL\_TEXT\_SLICE\_SIZE 4096 \* 16

#define KERNEL\_TEXT\_OFFSET 0x10000

#define PAGE\_SIZE 4096

#define WRITE\_FLAG\_ARRAY\_SIZE 625

#define MAX\_SYSCALL\_NUM 451

#### 1.6.2 主要数据结构

ACL自主访问控制包含以下两个主要数据结构

**selinux\_state\_tmp：**SELinux状态缓存的数据结构

struct selinux\_state\_tmp {

bool enforcing;

bool checkreqprot;

bool initialized;

bool policycap[8];

struct page \*status\_page;

struct mutex status\_lock;

struct page \*avc;

struct page \*policy;

struct mutex policy\_mutex;

}

其中各成员表示：

bool enforcing 表示是否启用SELinux强制模式

bool checkreqprot 表示是否启用请求保护模式。

bool initialized 表示SELinux子系统是否已初始化

bool policycap[8] 表示SELinux策略的能力

struct page \*status\_page 指向内存中的SELinux状态页

struct mutex status\_lock 用于保护状态页的互斥锁

struct page \*avc 指向内存中的访问向量缓存（AVC）页

struct page \*policy 指向内存中的SELinux策略页

struct mutex policy\_mutex 用于保护策略页的互斥锁

下面对上述数据结构作个说明：

在Linux内核中，selinux\_state结构体是安全扩展（SELinux）子系统的一部分，用于存储SELinux的当前状态信息。enforcing：一个布尔值，表示SELinux是否启用强制模式。如果enforcing为true，则SELinux会强制执行安全策略。如果为false，则SELinux只会警告违反策略的操作，而不会强制执行。checkreqprot：一个布尔值，指示SELinux是否要检查进程请求的权限。如果checkreqprot为true，则SELinux会检查进程的请求是否与安全策略一致。如果为false，则SELinux不会检查请求的权限。initialized：一个布尔值，指示SELinux是否已初始化。如果initialized为true，则SELinux已经完成初始化。如果为false，则还未完成初始化。policycap：一个布尔数组，表示当前系统支持的SELinux策略功能。数组的每个元素都对应一个特定的功能，如果该元素为true，则表示该功能可用。

此外，该结构体还包含其他一些成员变量。status\_page：一个指向系统状态页的指针。状态页包含SELinux的状态信息，如当前强制模式和支持的策略功能等status\_lock：一个互斥锁，用于保护状态页的访问。avc：一个指向访问向量缓存（AVC）的指针。AVC包含了SELinux规则引擎用于决策的缓存数据。policy：一个指向当前SELinux策略的指针。SELinux策略指定了如何保护系统资源和进程的安全规则。policy\_mutex：一个互斥锁，用于保护策略的访问。

这些成员变量共同协作，以便SELinux能够正确执行安全策略。例如，enforcing变量指示SELinux是否启用强制模式，而policy变量指定了安全策略。在执行安全决策时，SELinux将使用策略来确定允许或拒绝访问，并使用AVC缓存来提高性能。status\_page和status\_lock用于跟踪SELinux的当前状态，而policy\_mutex用于保护策略的访问，以确保并发访问时的数据完整性。

**Mem\_Critical\_Data：**存储内存关键数据状态的数据结构

struct Mem\_Critical\_Data{ //3848

bool MMU\_status;

u32 kernel\_page\_rw\_status;

u32 IDT\_data[32];

struct selinux\_state\_tmp selinux\_state;

long long syscall\_table\_data[MAX\_SYSCALL\_NUM];

};其中各成员表示：

bool MMU\_status 表示MMU使能状态

u32 kernel\_page\_rw\_status 表示内核页面读写控制位状态

u32 IDT\_data[32] 存储中断描述符表项

struct selinux\_state\_tmp selinux\_state 存储SELinux状态

long long syscall\_table\_data[MAX\_SYSCALL\_NUM] 存储中断描述符表项

#### 1.6.3 内部接口

##### 检查MMU使能位状态

|  |  |
| --- | --- |
| 名称/编号 | MMU状态检查函数 |
| 接口定义 | int check\_MMU(struct Mem\_Critical\_Data \*data) |
| 参数 | data为全局变量struct Mem\_Critical\_Data指针 |
| 返回值 | 返回检查结果 |
| 功能描述 | 读取当前系统MMMU使能位的状态 |

##### 检查页表读写控制位状态

|  |  |
| --- | --- |
| 名称/编号 | 页表读写控制位检查函数 |
| 接口定义 | int check\_PageRW(struct Mem\_Critical\_Data \*data) |
| 参数 | data为全局变量struct Mem\_Critical\_Data指针 |
| 返回值 | 返回检查结果 |
| 功能描述 | 读取当前系统内核空间页表读写控制位的状态 |

##### 检查中断描述符表内容

|  |  |
| --- | --- |
| 名称/编号 | 中断描述符表检查函数 |
| 接口定义 | int check\_idt(struct Mem\_Critical\_Data \*data) |
| 参数 | data为全局变量struct Mem\_Critical\_Data指针 |
| 返回值 | 返回检查结果 |
| 功能描述 | 读取当前系统中断描述符表内容 |

##### 检查系统调用表内容

|  |  |
| --- | --- |
| 名称/编号 | 系统调用表检查函数 |
| 接口定义 | int check\_syscall(struct Mem\_Critical\_Data \*data) |
| 参数 | data为全局变量struct Mem\_Critical\_Data指针 |
| 返回值 | 返回检查结果 |
| 功能描述 | 读取当前系统系统调用表内容 |

##### 检查SELinux配置

|  |  |
| --- | --- |
| 名称/编号 | SELinux配置检查函数 |
| 接口定义 | int check\_SELinux(struct Mem\_Critical\_Data \*data) |
| 参数 | data为全局变量struct Mem\_Critical\_Data指针 |
| 返回值 | 返回检查结果 |
| 功能描述 | 读取当前系统系统调用表内容 |

## 2 **应用进程代码段度量模块**

### 2.1 功能描述

基于TEE的可信性，设计实现了对REE应用进程代码段数据的获取功能，设计实现了在TEE中完成应用进程代码段的主动度量功能，并且将度量结果存在TEE报文中。

### 2.2 实现方式

设计思路：

从0号进程遍历进程树，通过读取task\_struct结构中mm\_struct结构体，读取该进程运行时代码段的起始地址和结束地址。

获取虚拟地址对应的物理地址：每个页面的页表项中记录了该应用进程的该虚拟页面与物理页框的映射关系，读取虚拟页对应页表项的数据项，判断页表项数据项是否读取完毕，若读取完毕，转入进入下一个应用进程的代码段的地址获取流程。若未读取完毕，判断当前项表示的数据页是否被加载到内存中，如果未加载到内存，则继续读取下一项，如果已加载到内存，则获取该虚拟地址对应的物理地址。

调用kmap()在内核空间建立与该物理地址的持久映射，调用measurement\_page函数对该内存页数据进行读取。

将读取到的内存页的数据通过内核通信模块，发送到TEE进行可信度量，调用SM3算法对该内存页数据进行摘要值计算，与基准库中的应用进程代码段相应页的基准值进行可信校验。如果相同，允许应用进程正常运行，如果不同，则打印日志信息，报告进程被篡改的信息。

关键函数：

1. pstreea(struct task\_struct \*p, int b, struct TEE\_Message \*mess)

包含三个参数，第一个参数struct task\_struct \*p表示深度优先遍历的起点进程，第二个参数int b表示深度优先遍历的层级计数，方便打印输出调试，第三个参数struct TEE\_Message \*mess表示需要填充的TEE报文指针。

这个函数是递归函数，从0号进程出发，使用深度优先算法遍历进程树，在遍历过程中判断该task struct是内核线程还是应用进程，遍历结束后本轮代码段度量结束。

1. check\_process\_text(struct task\_struct \*p, struct mm\_struct \*mm, struct TEE\_Message \*mess)

包含三个参数，第一个参数struct task\_struct \*p表示需要度量的进程的指针，struct mm\_struct \*mm表示目标进程的mm\_struct指针， struct TEE\_Message \*mess表示需要填充的TEE报文信息。该函数是进程代码段度量的主函数。首先从mm中获得代码段的起始地址和结束地址，均存储在unsigned long型变量中。再创建游标指针用来遍历代码段空间，存储在unsigned long型变量中。

首先进行判断代码段起始地址是否处于正常的进程用户地址空间的合法位置，如果不合法则直接返回ERROR。如果是合法的起始地址，那么将游标设置为起始地址处，开始遍历度量进程代码段。将游标所处的内存虚拟地址通过地址转化机制转换为物理地址后，会进行页面有效标志位的判断，如果有效位为1，说明该地址有效，进行后续映射操作，如果有效位为0，说明缺页，进行后续对应操作。获得合法的物理地址后，将该物理地址进行切割获得物理页帧号，调用kmap()函数把该物理页面映射至内核空间的vmalloc区，每次映射一个页面大小的数据。映射结束后，通过SM算法计算该物理页面数据的摘要值，并存入TEE报文的对应区域中，发送该报文即可。

流程图：



### 2.3 关键数据结构

#### 2.3.1 主要常量

关键数据度量中主要包含以下数据常量

#define GET\_BIT(x, bit) ((x & (1 << bit)) >> bit) /\* 获取第bit位 \*/

#define CLEAR\_BIT(x, bit) (x &= ~(1 << bit)) /\* 清零第bit位 \*/

#define SET\_BIT(x, bit) (x |= (1 << bit)) /\* 置位第bit位 \*/

#define KERNEL\_TEXT\_SLICE\_SIZE 4096 \* 16

#define KERNEL\_TEXT\_OFFSET 0x10000

#define PAGE\_SIZE 4096

#### 2.3.2 主要数据结构

代码段度量包含以下几个主要数据结构

unsigned long curr\_va;

unsigned long start\_va;

unsigned long end\_va;

unsigned long pa;

其中curr\_va表示游标所在的虚拟地址，start\_va表示需要度量的起始虚拟地址，end\_va表示需要度量的结束虚拟地址，pa代表经过地址转换机制转换后的物理地址。

#### 2.3.3 内部接口

##### 检查内核代码段

|  |  |
| --- | --- |
| 名称/编号 | 内核代码段检查函数 |
| 接口定义 | void check\_k\_text(struct TEE\_Message \*mess) |
| 参数 | mess为全局变量struct TEE\_Message指针 |
| 返回值 | 返回检查结果 |
| 功能描述 | 对当前系统内核的代码段进行一页一页的度量 |

##### 检查内核模块代码段

|  |  |
| --- | --- |
| 名称/编号 | 内核模块代码段检查函数 |
| 接口定义 | int check\_m\_text(struct TEE\_Message \*mess) |
| 参数 | mess为全局变量struct TEE\_Message指针 |
| 返回值 | 返回检查结果 |
| 功能描述 | 对当前系统内核已安装的内核模块代码段进行一页一页的度量 |

##### 检查应用进程代码段

|  |  |
| --- | --- |
| 名称/编号 | 应用进程代码段检查函数 |
| 接口定义 | int check\_process\_text(struct task\_struct \*p, struct mm\_struct \*mm, struct TEE\_Message \*mess) |
| 参数 | p是需要进行代码段检查的进程指针  mm是对应进程的mm\_struct指针  mess为全局变量struct TEE\_Message指针 |
| 返回值 | 返回检查结果 |
| 功能描述 | 对p所指向进程的代码段进行一页一页的度量 |

## 3 **内核与TA通信模块**

### **3.1** 功能描述

基于Linux内核和OP-TEE通信机制，实现REE侧内核与TA通信模块，设计实现在系统启动后，将内核中度量代理钩取的度量数据通过共享内存传输至TA，且当TA端主动度量模块完成判定时通过此通信模块控制内核中度量程序的运行状态，即判定无误时继续程序运行，判定错误时终止程序， 并且设计实现了REE内核和与TA通信存储事件日志的Event hash值以及存储事件日志的功能。

### **3.2** 实现方式

函数分支：

init()：内核模块初始化函数。

tee\_client\_open\_context()：该函数初始化了TEE（可信执行环境）客户端操作所需的必要资源和环境。这可能包括设置内存缓冲区、配置安全通信通道和执行TEE平台所需的任何必要设置程序。

tee\_client\_open\_session()：该函数创建一个TEE客户端上下文，用于与TEE交互。上下文包括TEE运行时环境的状态和配置信息，以及与TEE相关的安全上下文。

tee\_client\_invoke\_func()：该函数打开一个TEE会话，用于与TEE中的特定应用程序或服务进行通信。会话包括用于与TEE通信的安全句柄以及与TEE相关的安全上下文。

tee\_client\_close\_session()：该函数向TEE发送请求，执行特定的应用程序或服务功能。请求通常包括输入参数和输出参数，并使用TEE会话中的安全句柄进行安全通信。

tee\_client\_close\_context()：该函数关闭TEE客户端上下文并释放与之关联的资源。这可能包括TEE运行时环境的状态和配置信息以及与TEE相关的安全上下文。

流程图：



### **3.3** 关键数据结构

#### 3.3.1 主要常量

关键数据度量中主要包含以下数据常量

#define MAX\_SHM\_SIZE 128\*1024

#define MAX\_HASH\_NUM (MAX\_SHM\_SIZE - 4) / 32 //4095

#define TA\_HELLO\_WORLD\_CMD\_INC\_VALUE 0

#define TA\_HELLO\_WORLD\_CMD\_DEC\_VALUE 1

#define TEE\_DATABASE\_ADD\_BASELINE 0x1

#define TEE\_DATABASE\_DEL\_BASELINE 0x2

#define TEE\_DATABASE\_READ\_BASELINE 0x4

#define TEE\_DATABASE\_ELF 0x1

#define TEE\_DATABASE\_SO 0x2

#define TEE\_DATABASE\_KO 0x4

#define TEE\_DATABASE\_MODULE\_TEXT 0x8

#define TEE\_DATABASE\_KERNEL\_TEXT 0x10

#define TEE\_DATABASE\_TASK\_TEXT 0x20

#define TEE\_DATABASE\_MEM\_CRITICAL\_DATA 0x40

#define TEE\_DATABASE\_OP\_SUCCESS 0x1

#define TEE\_DATABASE\_OP\_FAILED 0x2

#### 3.3.2 主要数据结构

通信代理包含以下几个主要数据结构

**struct SM3\_Baseline：**SM3摘要值缓存的数据结构

struct SM3\_Baseline{

u8 Hash[32];

};

其中各成员表示：

u8 Hash[32] 表示一个哈希值

**struct TEE\_Message：**TEE通信报文内同的数据结构

struct TEE\_Message{

u8 operator\_type;

u8 baseline\_type;

int length;

int result;

struct SM3\_Baseline SM3\_array[MAX\_HASH\_NUM];

};其中各成员表示：

u8 operator\_type 表示操作类型

u8 baseline\_type 表示摘要值类型

int length; 表示此次报文摘要值个数

int result; 表示TEE验证结果

struct SM3\_Baseline SM3\_array[MAX\_HASH\_NUM] 表示此次度量计算的摘要值

#### 3.3.3 外部接口

##### 向TEE发送报文

|  |  |
| --- | --- |
| 名称/编号 | 报文交换函数 |
| 接口定义 | void check\_k\_text(struct TEE\_Message \*mess) |
| 参数 | mess为全局变量struct TEE\_Message指针 |
| 返回值 | 返回发送结果 |
| 功能描述 | 在ree内核空间开辟一块共享内存，并向TEE侧发送通信报文 |