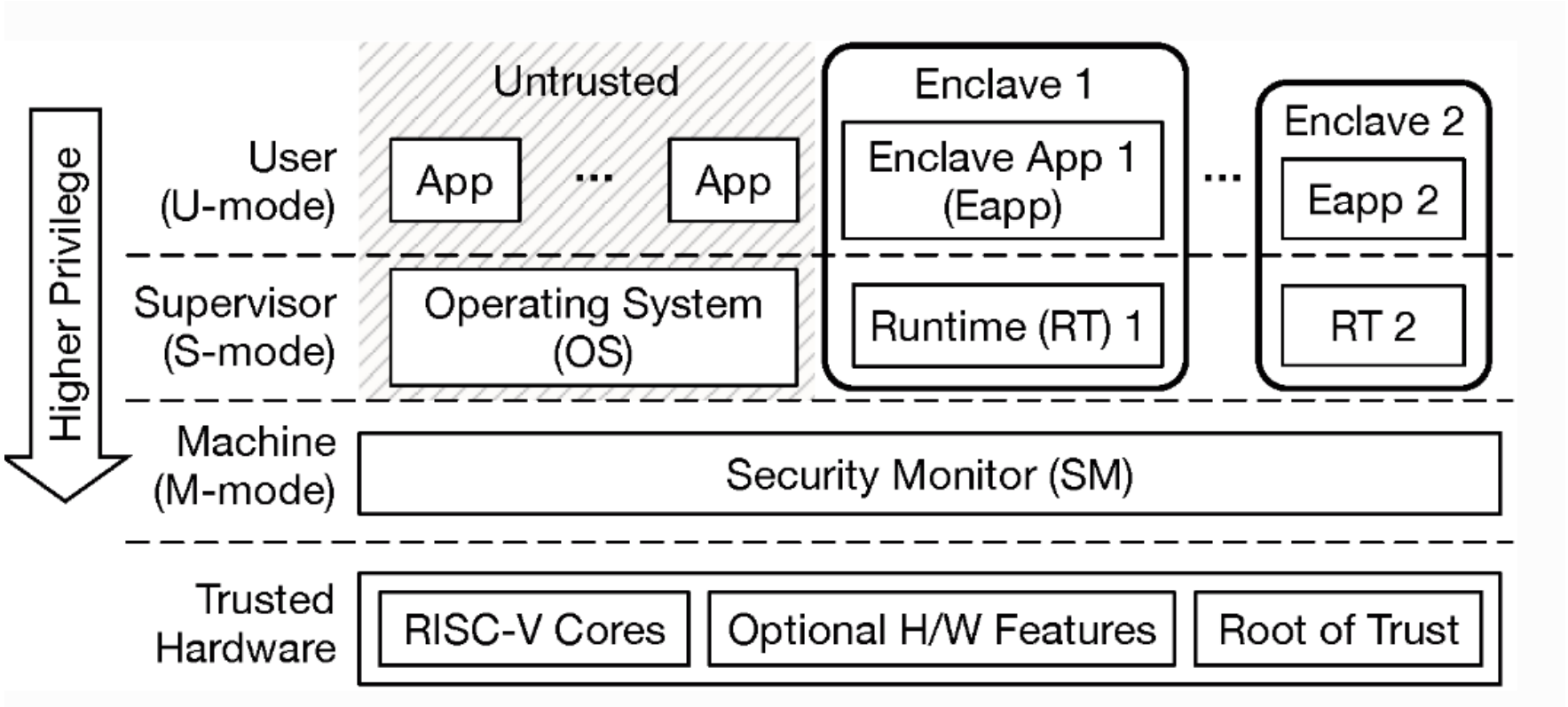
Enclave模式的Rust-TEEOS

孙宇涛 安一帆 赖瀚宇

/01

系统运行逻辑

整体运行框架


Higher Privilege
User
(U-mode)
Supervisor
(S-mode)
Machine
(M-mode)
Trusted
Hardware
Untrusted
App
...
App
Operating System
(OS)
Enclave 1
Enclave App 1
(Eapp)
Runtime (RT) 1
...
Enclave 2
Eapp 2
RT 2
Security Monitor (SM)
RISC-V Cores
Optional H/W Features
Root of Trust

创建流程

- 创建Enclave
  - Host App计算Enclave所需的内存空间，Host OS在内存中开辟一块连续的内存提供给Enclave使用；
- 创建共享内存
  - 与私有内存相似；
- 确认创建完成
  - Host OS将若干参数传递给sbi层，sbi保存相应的页表地址与runtime、app启动地址，以便移交控制权时runtime可以有效访问内存；
- 删除Enclave
  - 释放私有内存与共享内存，清除保存的相应数据结构，流程结束。

运行流程

- Host App发出运行信号，向下传递到sbi层，sbi层经过处理之后交给runtime运行；
- Runtime设置时钟中断，中断后暂停Enclave的运行，将控制权转交给sbi；
- Sbi再将控制权转交给Host OS，执行Host上的其他程序；
- 时间片轮转到Host App时，恢复Enclave的运行；

用户态交互

- 由于Runtime不是一个完整的操作系统，Enclave App与Host的交互是不可避免的；
- EApp依靠Edge Call在Host环境下执行相应的函数；
- EApp在共享内存中写入调用信息之后移交控制权给Host，Host检查共享内存之后执行调用；
- 调用方法为维护一个字典，其中保存了函数id与相应的函数地址，函数id在EApp开始执行时是已知的；

/02

系统实现细节

系统调用

- 在原有的Linux实现中，利用到了module的特性；
- 在基于zCore的实现中，类似地，将所有的操作封装为一个文件句柄，通过mmap和ioctl来进行实际的系统调用；
- 出于简单考虑，所有的Enclave系统调用均通过一个句柄，写死到系统中，fd=666；
- mmap的实现以及整个内存管理模式与zCore强耦合，相比于Linux上的实现做了较多修改；

内存管理

- 私有内存与共享内存需要分配连续的内存块，方便sbi和runtime进行处理；
- 最初的做法是，在初始化时，分配一个连续内存的vmo，在mmap时通过create_child进行虚拟地址的分配；
  - 但是，在zicron标准中对于contiguous的vmo进行create_child是禁止的！
  - 尝试修改这块代码使其变的可能，发现create_child嵌套的很深，修改很困难；
- 在初始化时，alloc若干个连续的物理页，在实际需要时（mmap时）再对其分配虚拟地址，修改页表；
  - 也配套修改了创建vmo的过程，但是这步是完全兼容的；
  - 在TEE需要的范围内，并不需要维护父子关系；

内核数据结构

- 内核需要管理一系列的Enclave的参数，以及id的分配：
  - 参考zCore的其他栈式分配系统，利用lazy_static创建一个全局管理器；
  - 由于需要频繁修改全局管理器所保存的参数，因此参考rCore的函数式编程来对其进行修改；
- 私有内存与共享内存的参数，相比于Linux上的实现做了较大修改：
  - 保存舒适化分配的若干PhysFrame组成的Vector，需要的时候对其进行切片；
  - 由于分配内存的时机不同，没有保存若干虚拟地址；

总结

- 由于缺乏系统级的开发经验，在环境配置、编译选项等环节踩了不少坑；
- TEE的实现与运行流程较为复杂，调试困难，因此后期进程缓慢，没有完成后续的目标；
- 整体上更加偏向工程，对于其他TEE的实现缺乏了解，论文阅读效率很低；
- 可以看出Keystone在实现与设计上的一些缺陷，如果未来有机会继续这项工作，可能的规划为：
  - 将runtime自己实现一遍；
  - 改进Host OS和Host App的交互机制；
  - 改进用户态交互的机制，增强or修改runtime的能力；
  - ……
- 感谢陈渝、向勇老师的指导与建议，感谢贾越凯、王润基助教和洛佳的答疑！