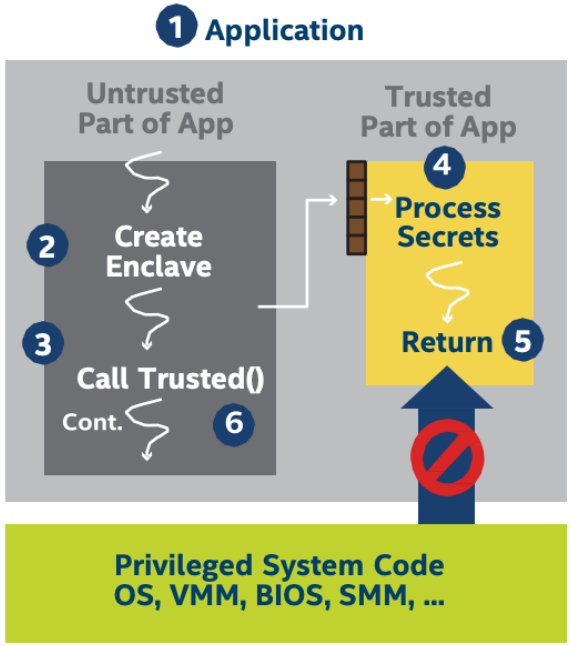
SGXLock: Towards Efficiently Establishing Mutual Distrust Between Host Application and Enclave for SGX

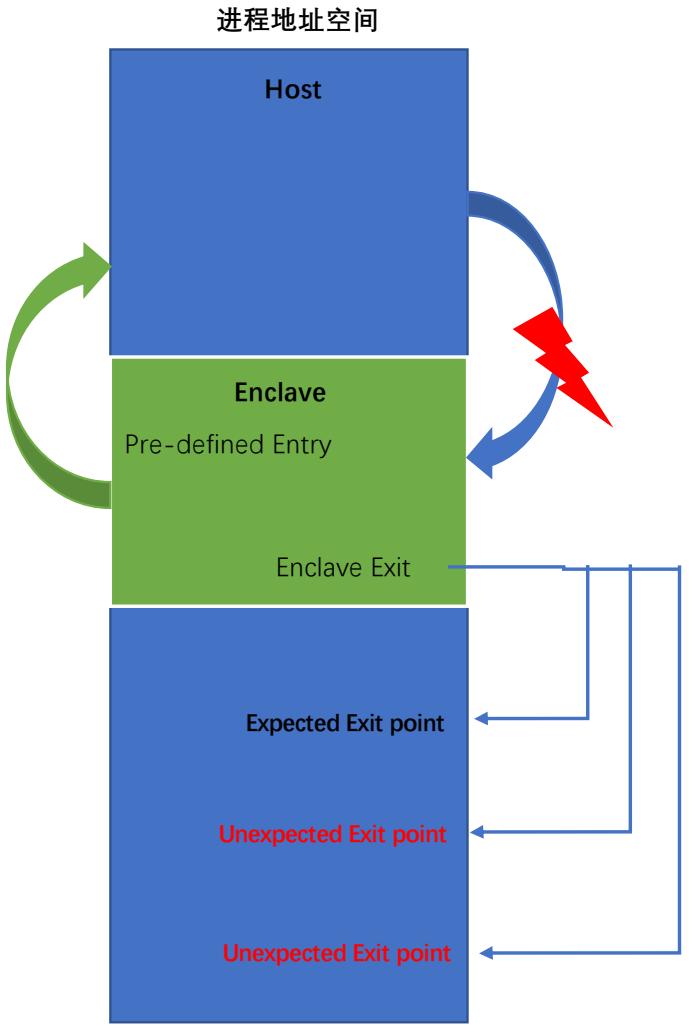
背景介绍

- Enclave：存放敏感代码/数据
  - 以类似动态库的形式内嵌在Host Application地址空间
- Host Application：用户应用
  - 调用Enclave完成敏感操作
  - 帮助Enclave完成与外界交互
- SGX硬件设计的威胁模型
  - Enclave可信
  - Host Application不可信



Enclave-Host Asymmetry问题

- 现实场景
  - Enclave，Host Application互不信任
- Enclave-Host Asymmetry问题
  - 原因：现实场景与SGX硬件设计威胁模型的错配
  - Data access asymmetry
    - Enclave可以访问Host内存，反之不行
  - Control flow asymmetry
    - Enclave可以跳转Host任意位置，反之不行
      - via EEXIT指令



解决方案：SGXLock

- Data Access Asymmetry Elimination
  - 利用硬件特性：Intel MPK (Memory Protection Key)
  - Enclave与Host内存分属不同MPK key (i.e. E, H)
  - 参数buffer：分属E
  - CPU MPK访问权限 (i.e. PKRU寄存器)
    - 进入Enclave：{H, E} -> {E}
    - 退出Enclave： {E} -> {H, E}
- Control Flow Asymmetry Elimination
  - 利用硬件特性：单步模式
  - 退出Enclave：产生单步调试异常
- Enclave/Host交互流程（如右图）

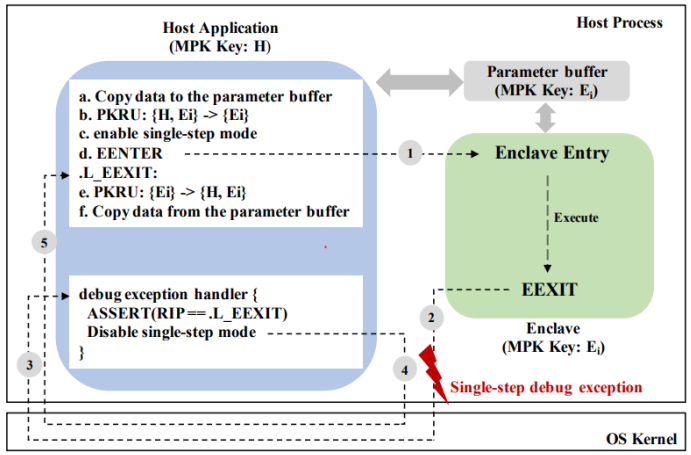
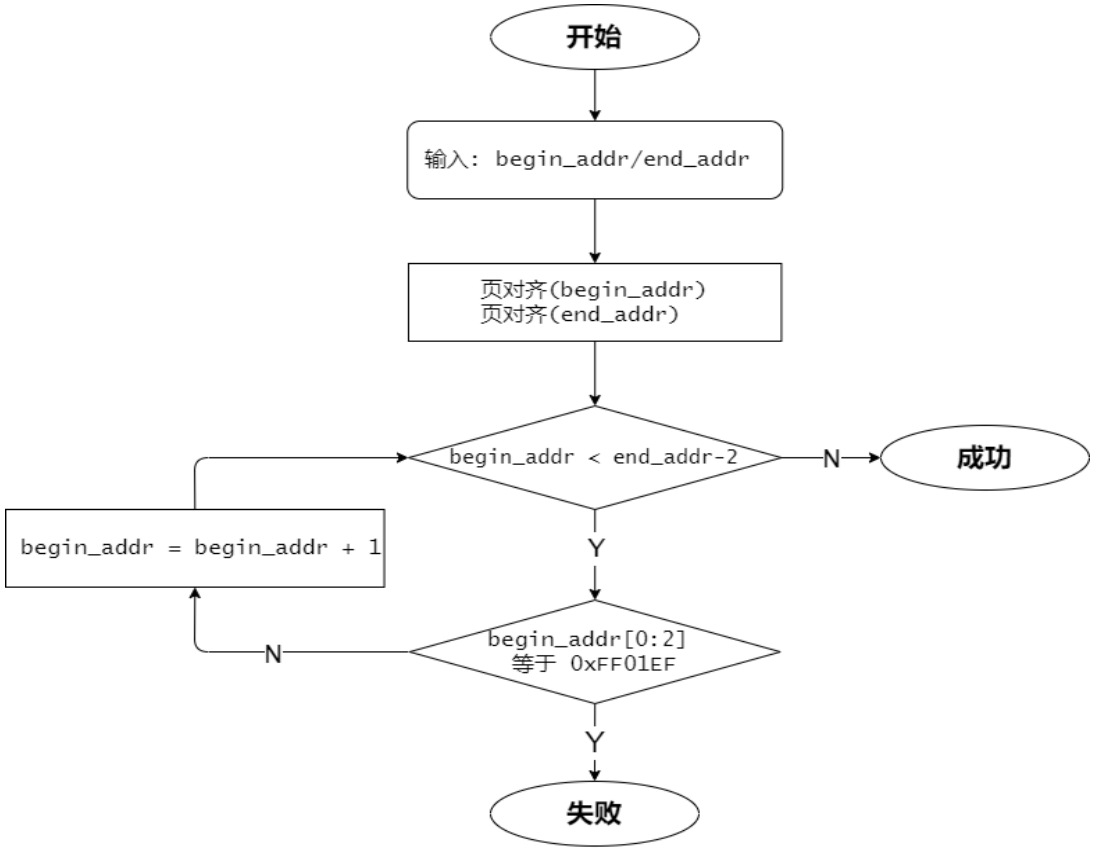


Figure 1: The overall system architecture.

挑战与解决方案

- 防止Enclave内对PKRU寄存器的更新
  - XSAVE/XRSTOR指令：将PKRU寄存器作为处理器执行状态保存到或者从内存读取
    - 解决方案：disable the bit 9 of Enclave ‘s XFRM field
  - WRPKRU指令：可以更新PKRU寄存器
    - 解决方案：代码审计以避免Enclave内出现WRPKRU指令
      - 静态审计：针对明文Enclave代码，Enclave创建时检查
      - 动态审计：针对Enclave内动态生成或加载的代码，Enclave运行时检查
        - W xor X => Host监测到Enclave内动态生成或加载的代码
        - 内嵌审计逻辑 => 在不泄露代码内容前提下，Host实现代码审计
- Host栈指针保护
  - HMAC-based Host栈指针完整性验证

内嵌审计逻辑


开始
输入: begin_addr/end_addr
页对齐(begin_addr)
页对齐(end_addr)
begin_addr < end_addr-2
N
成功
Y
begin_addr = begin_addr + 1
begin_addr[0:2]
等于 0xFF01EF
N
Y
失败

实现与实验配置

- Intel SGX SDK (v2.9.1) for Linux
- SGX Driver v2.6
- Ubuntu 18.04 (Kernel v5.4.28)
- Intel i7-10700F CPU (2.90 GHz)

Micro-benchmark

- Raw ECALL/OCALL

Table 1: The evaluation result of the raw ECALL/OCALL latency.

	Original	SGXLock	SGXLock *[2]
ECALL	7,636	11,662 (52.7%)	9,288 (21.6%)
OCALL	5,908	9,588 (62.3%)	7,303 (23.6%)

[1] Time is measured in CPU cycles.
[2] SGXLock * represents our prototype with single-step mode disabled.

- 参数传递

Table 3: The evaluation result of parameter passing. Time is measured in milliseconds.

	ECALL			OCALL		
	[in]	[out]	[in,out]	[in]	[out]	[in,out]
Original	28.0	145.5	45.6	17.9	157.0	45.7
SGXLock	32.9	150.9	55.7	22.7	161.1	54.4
Additional Time	4.9	5.4	10.1	4.8	3.1	8.7
Overhead	17.6%	3.7%	22.3%	26.4%	1.9%	18.9%

Real-world Application

- Machine Learning Inference
  - Highly optimized for SGX Enclave
- Database (SQLite)
  - 原来：13092 OCALL/s & 1186 ECALL/s
  - SGXLock: 12787 OCALL/s & 1159 ECALL/s
- HTTP Server
  - Apache HTTP benchmark tool
  - High-frequent OCALL
    - ~742k OCALLs for 1000 requests

Table 4: The execution time (milliseconds) and overhead of the machine learning service using our system.

	Original	SGXLock	Overhead
Enclave creation	283.80	283.91	0.04%
Model loading	43.59	54.00	23.90%
Inference	1,962.10	1,970.85	0.45%
Total	2,289.49	2,308.76	0.84%

Table 5: The execution time (milliseconds) and overhead of database operations.

	Original	SGXLock	Overhead
INSERT	1,856,643	1,867,392	0.58%
SELECT	8,428	8,629	2.38%
UPDATE	183,271	186,740	1.89%
DELETE	182,417	182,763	0.19%

Table 6: The evaluation result of a HTTP web server.

	Transfer Rate (kb/s)		Overhead	# of OCALLs
	Original	SGXLock		
1	5.94	5.69	4.21%	~742,000
2	5.93	5.7	3.88%	~742,270
4	5.9	5.7	3.39%	~742,810
8	5.93	5.67	4.38%	~743,890
16	5.92	5.68	4.05%	~744,960

动态审计性能开销

- 9.29 us/Page
  - ~26941 cycles for the CPU used in the evaluation

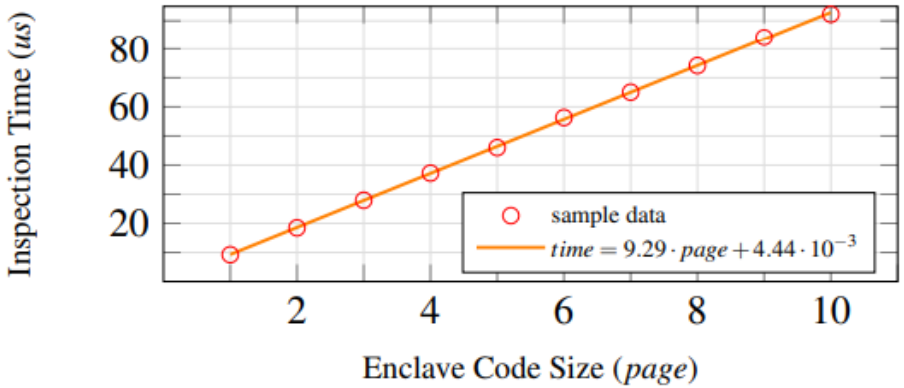


Figure 3: The dynamic inspection time of different enclave code size.

Thank You