T-Counter 阅读报告

一、论文研究背景、动机与主要贡献

1.1 研究背景

云服务应用越来越广泛，云服务提供商（CSP）通常使用按量付费的计费方案，根据资源的使用情况计费。对消费者来说这种计费方案也更经济。然而，从消费者的角度来看，CSP 可能是不可信任的，因为他们完全控制着整个操作系统，有动机为了获得更多利益来通过各种手段虚报账单。因此，如何提供一个可以跟踪和验证的解决方案来衡量使用者的资源使用情况是一个具有挑战性的问题。

1.2 研究动机

目前已经有过多次构建可信任的解决方案来衡量CPU使用情况的尝试，但它们都有严重的局限性。

> 飞地——SGX enclave，本论文中可以认为在该空间中运行的程序和变量等受到较高的保护，几乎不会被 CSP 篡改，或者篡改的成本较高。

(1)安全需求

并不是所有的应用程序都有很强的安全性要求，可能没有必要使用 SGX。

2)性能开销

构建自我度量解决方案的一个关键障碍是性能开销。**整个应用程序在飞地中**运行会增加飞地的内存访问，导致频繁ecalls/ocalls。飞地内存访问将带来过高的不可接受的性能开销。

3) TCB的大小

旧版应用程序完全运行在飞地内将导致大型 TCB 过大，这不仅会带来**高性能的开销**，还会使飞地中的应用程序面临**更多的安全风险**。

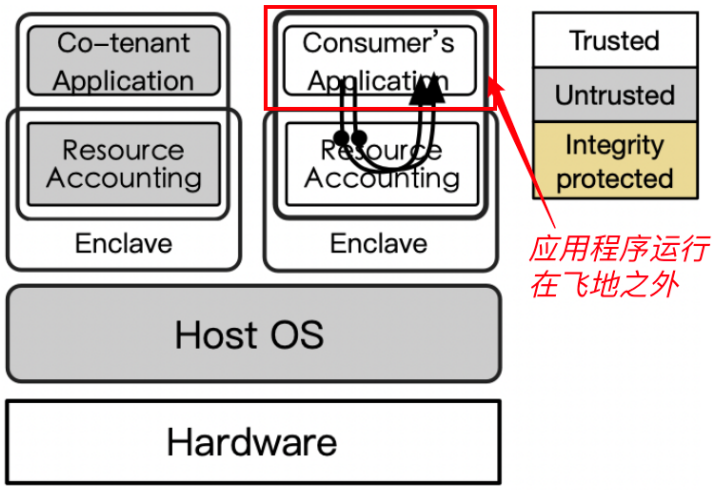
4)应用程序修改

需要将应用程序**重写为SGX应用程序**，并**添加测量代码**，这需要消费者有相关的开发经验。

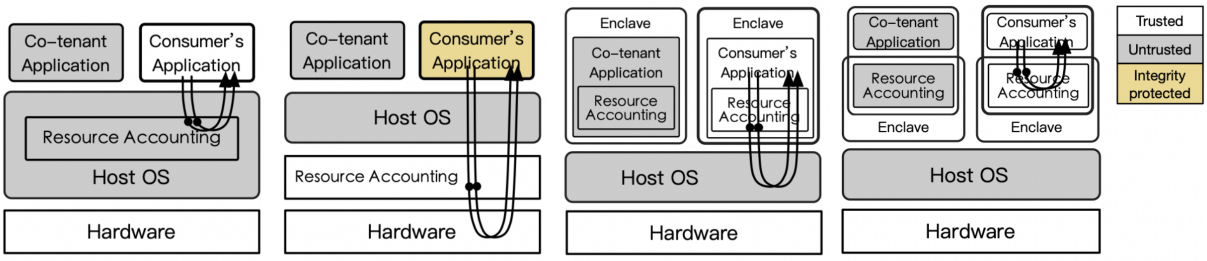
该论文希望构建一个应用程序进行自我度量解决方案，它具有以下特点

1. **飞地之外运行**：应用程序运行在飞地之外，**不依赖系统服务**而是令应用程序自身来度量 CPU 使用情况；

2. **保密性**：可以保证结果的保密性，可以防止CSP （或者对手） 操作应用程序或**数据流图**来篡改计算结果；
3. **高效指令计数**：使用了 SGX SDK提供的ecall和无调用切换，实现了高效的指令计数。



(d) T-Counter



(a) Current approach (b) Observer approach (c) Self-Measurement approach (d) T-Counter

Fig. 1: Four approaches: a) Current approach, b) Observer approach, c) Self-measurement approach, and d) T-Counter.

1.3 主要贡献

论文提出了 T-Counter 。T-Counter 可以通过**应用程序本身**对CPU使用率进行度量。具体的实现方法是对在飞地中使用**计数器**对应用程序执行的指令进行计数，应用程序中由于飞地是一个可新空间，所以这个计数是可信的。主要贡献可以总结如下：

- ***通用的解决方案***：T-Counter 可以测量运行在飞地之外的应用程序的 CPU 使用情况，采用了如下设计
  - **安全性**：用户应用程序在飞地外运行，而计数器在飞地内运行；
  - **低开销**：选择性地在CFG(控制流图)中设置计数器，尽量减少 SGX 的 ecall 使用数，从而降低开销，达到最佳性能；
  - **TCB最小化**：应用程序运行在enclave之外，只有可信组件(如可信计数

器)运行在enclave内；
  - **易用性**：在T-Counter的帮助下，用户可以轻松地在程序中设置计数器，并自动在飞地中建立测量代码
- *基于**选择性策略**和优化策略设置计数器从而降低开销*：论文提出的算法可以在尽可能少的块内设置计数器并实现指令技术。论文设计了两种算法分别为 Base-Ins 和 Opti-Ins 算法，除此之外还提出了一系列优化策略来进一步优化解决方案
- *T-Counter 理论分析*：论文对 T-Counter 架构和算法的正确性、安全性、性能开销和准确性进行了理论分析。结果表明：
  - (1) CFG中计数正确；
  - (2) 可信计数器（飞地中的计数器）能够防御攻击
  - (3) 所使用的算法和策略可以实现最小延迟和降低开销
- *T-Counter 的实现和评估*： 论文对 LLVM 进行了扩展并与Intel Linux SGX SDK 进行了集成，实现了 T-Counter， 并且进行了实验来测试用例在不同参数（运行时间、消息传递频率）下的性能。还与多种其他文献中的方案（S-FaaS和VeriCount）进行对比。结果表明，T-Counter与理论分析相符，对 CPU 使用率的计算的准确度和效率均较高。

二、论文问题描述或定义

论文建立了一个威胁模型，该模型设定了问题的场景，并陈述了架构和算法设计的目标

2.1 威胁模型 （**Threat Model**）

在该模型中，进行如下假设

- 消费者只能信任云提供的飞地（enclave）
- CSP 完全控制云中的所有资源，可以利用一切手段来收取更多的费用
- CSP 是理性的，攻击的开销不会超过他通过攻击可以得到的额外费用
- CSP是谨慎的，他的攻击不会引起消费者注意
- CSP 无法对处理器进行破坏和篡改
- 不讨论 DoS 攻击等外部恶意操作，这超出了本篇论文讨论范围

假设 CSP 有如下能力

1. 直接修改票据
2. 篡改资源使用日志、操纵服务时间、修改计算环境等从而做出虚假报告

3. 可以暗中篡改用户程序（**飞地之外**），但是运行时不会篡改

2.2 方案目标

在威胁模型基础上，提出的解决方案应该尽量满足以下目标

C1: 可信和可验证的资源计费

即任何攻击者都无法干扰计费机制，计费结果必须对用户可信，并且可以验证;

C2: 计费完整性和保密性

即防止CSP对计费过程的篡改。在该 T-Counter 架构中，**计费结果在飞地中得到保护**。但是，带有计数器的程序是在飞地之外，对手可以操纵计费过程;

> 该框架无法保障，在没有其他技术的情况下，在飞地之外保护计费的完整性，这仍然是一个未解决的挑战。

C3: 执行完整性

CSP 可能会篡改消费者程序的控制流。**执行完整性**是为了确保消费者能够检测到对程序控制流的攻击。这项任务比计费完整性更具有挑战性，因为消费者程序是在没有任何保护的飞地之外运行的;

C4: 低性能开销和最小的修改

该目标挑战是在相对**较低的性能开销**和**降低迁移成本**的前提下实现可验证的资源计费。

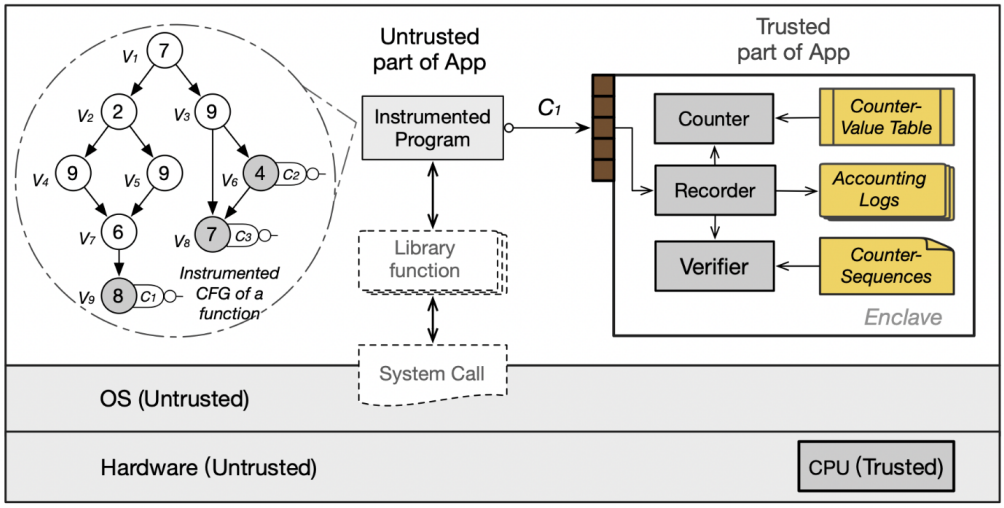


Fig. 3: Architecture Overview.

三、论文提出的新思路、新理论、或新方法

该论文提出的应用程序 CPU 使用情况自我度量解决方案可以实现**应用程序在飞地外运行**的同时进行安全的计费。

论文构建的架构扩展了 LLVM 在程序的基本块中使用计数器，并在可信部分（飞地中）添加了是三个组件来计数指令和防御恶意的 CSP 操作。

此外，为了降低开销，论文还设计了两种算法和多种针对不同情景的策略，可以减少计数器数量。

四、论文方法的理论分析或实验评估方法与效果

4.1 理论分析

论文首先对 T-Counter 的**正确性**和**安全性**进行了分析，之后定量分析 T-Counter 在一些用户用例中的**延迟**、**效率**和精确度。理论分析和评估表明，T-Counter可以有效地测量 CPU 使用率，防御恶意的 CSP 操作。

4.1.1 正确性

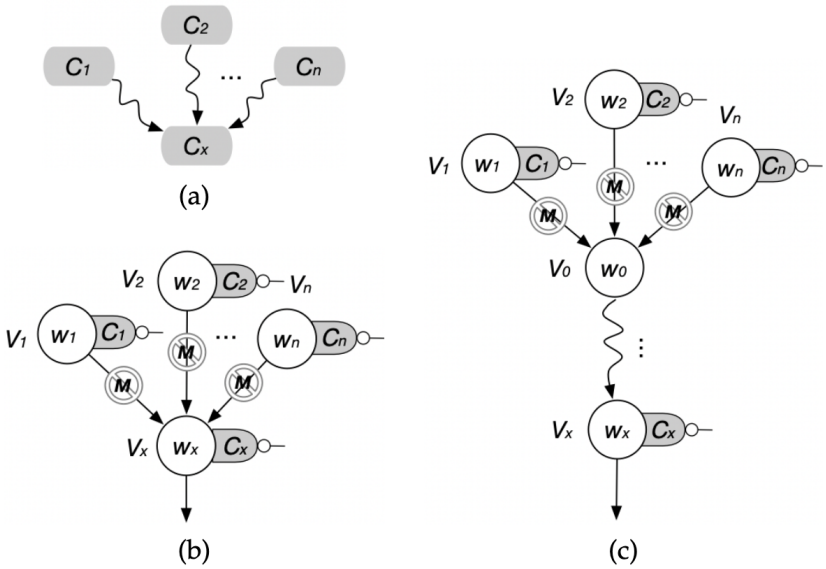


Fig. 11: Proof of Base Instrumentation Algorithm

论文证明：在 CFG 中，计数器 C 和与他直接连接的计数器之间的 Pathlet 的权值相同。

4.1.2 安全性分析

论文分析表明 T-Counter 满足设计目标中的

1. 可验证
2. 计费机密性
3. 执行完整性

4.1.3 延迟和效率

论文定量评估了 6 个方案中 7 个应用程序的平均延迟和开销

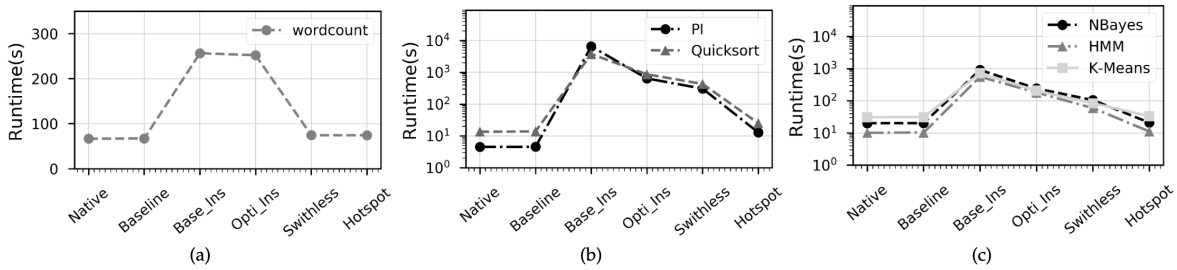


Fig. 12: Runtime comparisons of six schemes of six applications..

6 个方案分别为：native, baseline, base-ins, opti-ins, switchless, hotpots

4.1.4 准确性

由于CPU的内部设计，**指令的数量不能完全等于CPU周期开销**。不同的指令需要不同的执行周期。但是实际上，**即使完全按照执行周期对指令开销进行计算也无法得到更准确的结果。**因此，该论文直接使用所有指令的平均开销作为权重，将权重与指令数量相乘的结果作为最终的 CPU 开销。这种策略可以降低路径分析的难度，并且可以保证准确性

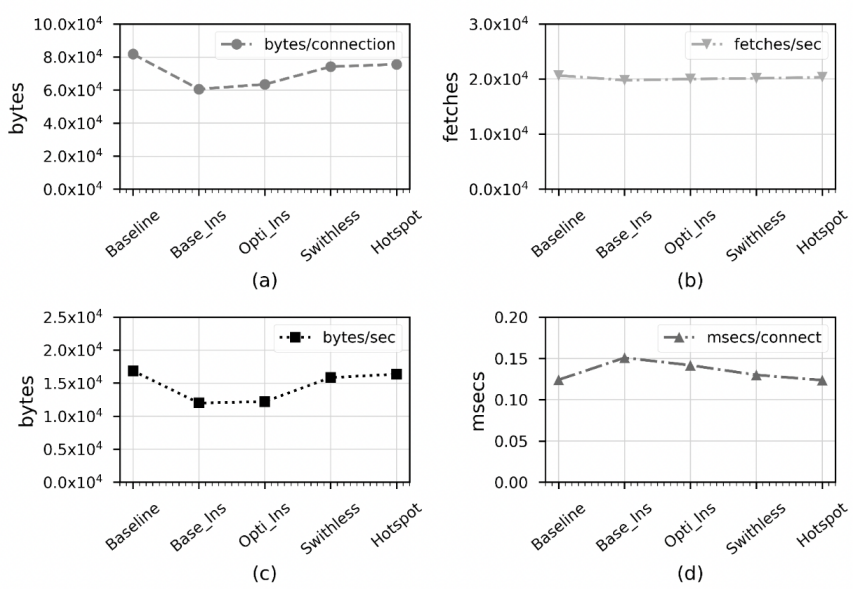
4.2 实验评估

4.2.1 评估方法

首先进行 T-Counter 框架的实现——对 LLVM 进行扩展，实现语言是 C 语言（850行）。然后，在计算机上对 SGX 应用程序进行基准测试。电脑 CPU 支持 SGX 其他参数此处省略。

论文使用 C/C++ 实现了六个代表性的用例

1. 3 个微基准:单词计数，PI和快速排序;
2. 3 个机器学习应用:朴素贝叶斯(NBayes)， K-均值和隐藏马尔可夫模型(HMM);
3. 以及一个 web 服务器应用：Thttpd.



4.2.2 评估结果

两种检测算法的平均运行时开销分别为3.82 (wordcount) - 1441.62(PI)和3.75 (wordcount) -141.33 (PI)。2种优化方案的平均优化率分别为50.32% (Quicksort) - 70.45 (wordcount)%(随机数为1000)和0 (wordcount) - 95.82% (PI)。

五、总结

5.1 优点

1. 该论文提出了一个通用的 CPU 使用率度量框架，可以在保证安全性和可信性的前提下对应用程序的 CPU 使用情况进行度量
2. **应用程序可以运行在可信空间（飞地）之外**
3. 提出了针对热点的优化策略
4. 可以使得攻击面最小化，攻击方的攻击代价大于攻击收益

5.2 缺点

- 虽然已经可以使得攻击面最小化，但是 T-Counter 对手仍然可以对资源度量造成一定损害——可信空间外的部分仍然可能被攻击；
- 无法对 CPU 外的其他许多重要资源进行度量，比如内存、网络、磁盘等

5.3 后续研究思路

5.3.1 安全性增强

该论文提出的解决方案使攻击方可以使用的攻击面最小化，保证了我们 CPU 使用情况计算的有效性和可靠性。但是它并不能完美地抵御所有攻击，而且对手仍然可以对资源度量造成一些损害。我们需要抵御更多的攻击和更有效的**反序列验证方法。**

5.3.2 其他资源度量

除了CPU资源外，还需要度量其他许多重要的资源，如内存、网络、磁盘等。过去的工作有如下方法

1. 通过函数或解释器使用的malloc、realloc和free函数来测量内存开销；
2. 使用线性内存大小来记录工作负载所消耗的内存；
3. I/O开销：使用附加的计数器来累积通过I/O功能流入和从飞地流出多少个字节

> 但这些方案只能测量飞地内部的内存成本。实现对飞地之外的资源使用的信任度量是一个具有挑战性的问题。