计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(10):67−72 [doi: 10.15888/j.cnki.csa.005977]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Intel SGX 的 Ansible 安全增强①

杨　骁[1,2], 于佳耕[1], 武延军[1]

[1](中国科学院 软件研究所 基础软件国家工程研究中心, 北京 100190)
[2](中国科学院大学, 北京 100049)

摘　要: Ansible 是当今最主流的云平台自动化运维工具之一, 通常拥有大量集群的管理员账号信息来批量执行运维任务, 这些账号信息一般明文存储在配置文件中. 但在云计算环境中这种机制具有较大的安全隐患, 因为 Ansible 的配置信息的机密性和完整性依赖云服务厂商提供的基础软件的安全性. 因此, 加强 Ansible 配置信息管理机制的安全性至关重要. 本文基于 Intel 近年推出的安全机制 SGX(Software Guard eXtensions) 开发了 Ansible 的配置管理模块, 它可以独立地在可信的执行环境 (Trusted Execution Environment, TEE) 内管理 Ansible 的配置信息, 从而保证了 Ansible 配置信息不被外界读取和修改, 同时其安全性不再依赖于底层基础软件. 实验评估显示, 本文方案更加安全可靠, 性能的额外开销也在接受范围之内, 而且可以进一步扩展成通用的配置管理组件集成到 OpenStack 等云平台中.

关键词: Intel SGX; 云计算; 安全; 运维; Ansible

引用格式: 杨骁,于佳耕,武延军.基于 Intel SGX 的 Ansible 安全增强.计算机系统应用,2017,26(10):67–72. http://www.c-s-a.org.cn/1003-3254/5977.html

Security Enhancement of Ansible Based on Intel SGX

YANG Xiao[1,2], YU Jia-Geng[1], WU Yan-Jun[1]

[1](National Engineering Research Center for Fundamental Software, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: As one of the most popular tools for automatic operation and maintenance in cloud platforms, Ansible usually stores a lot of administrator accounts information in a configuration file for batching executions. The configuration file is usually stored in the disk in plain text. However, it is not safe in the cloud because the confidentiality and integrity of configuration of Ansible depend on the security of the underlying software. Therefore, it is crucial to reinforce the security of configuration management mechanism of Ansible. In this paper, we implement a configuration management component for Ansible based on SGX (Software Guard eXtensions) proposed by Intel in recent years, which can manage the configuration information of Ansible in a trusted execution environment (TEE) independently. With this component, the configuration information cannot be read or written from outside and its security doesn't depend on the underlying software. The experiments show that our solution is more reliable, and the extra overhead is also acceptable. It is possible to extend our application to a general component for configuration protection in cloud platforms such as OpenStack.

Key words: Intel SGX; cloud computing; security; operation and maintenance; Ansible

自动化运维工具在现在的云平台上使用非常广泛, 它大大方便了用户对服务器集群的管理和配置, 降低了集群的运维成本. 主流的自动化运维工具有 Ansible[1]、Puppet、Chef 等等, 其中 Ansible 诞生较晚, 但是目前

① 基金项目: 中国科学院先导专项 (XDA06010600); 国家自然科学基金青年基金 (61402451)
收稿时间: 2017-01-04; 采用时间: 2017-02-13

看来最为活跃, 市场占有率持续增长, 很有潜力成为日后最为主流的自动化运维工具.

Ansible 管理着服务器集群的登录信息, 这些信息都会被保存在 Ansible 的配置文件中. 一旦 Ansible 的配置文件的信息被泄露, 相应的集群中服务器的管理权限也将被窃取, 所以对 Ansible 的配置信息的保护至关重要. 但是目前 Ansible 的配置信息一般是明文存储在磁盘上, 读取磁盘的权限一旦被恶意程序获取, 将导致大量服务器的管员权限被窃取. 针对这个问题, 除了在磁盘上明文存储配置信息之外, Ansible 可以用其内置的 vault 机制先将配置文件加密, 再将加密后的信息存储在磁盘, 但是这需要用户每次使用 Ansible 时都提供解密配置文件的密钥, 这将极大的降低 Ansible 本身的便捷性. 除此之外, 目前没有其他方法来保护 Ansbile 的配置信息. 其主要原因在于单纯地依赖软件来保护配置文件是不够的. 现有软件层面的保护手段需要将一些庞大的底层模块纳入 TCB(Trusted Computing Base), TCB 过大将会降低系统的安全性, 一旦 TCB 中的任何组件出现漏洞, 就意味着其他程序的安全性无法保障. 另一方面, 如果攻击者可以从物理上访问设备, 直接控制软件本身, 软件是无法抵御的. 而且从软件层面是无法防御像内存嗅探这种物理攻击的.

因此, 本文引入了 Intel SGX[2-4]硬件保护机制, Intel SGX 可以将 Ansible 的配置信息和相应的关键代码运行在加密内存中, 在程序运行的过程中, 外部无法窥探加密内存. 当需要将配置信息持久化存储时, 程序会先在加密内存中用外部无法获知的密钥先将配置信息加密后再写入到磁盘. 因此即使恶意程序获取了整个磁盘和内存的访问权限, 也无法窃取 Ansible 的配置信息.

本文工作的主要难点在于: 1) 对 Intel SGX 接口的更高层封装; 2) 设计和开发独立的配置管理模块与 Ansible 结合. 虽然 Intel SGX 官方 SDK 已经提供了一些较为底层的接口, 但是使用它保护配置信息还需要根据需求进行相应的设计和实现, 本文针对配置信息保护这一类需求开发了更便捷的接口. 目前 Intel SGX 官方的 SDK 仅支持 C 和 C++接口. 本文实现的高层封装支持 python 语言, 基于此架构添加对其他语言的支持也非常简单. 另一方面, Intel SGX 官方 SDK 目前只支持将 enclave 以动态链接库的形式加载到其他进程中, 这样导致在不同的进程之间共享 enclave 中的数据较为麻烦, 而且和其他程序的集成代价较大. 本文针对此问题设计和开发了独立的配置管理模块, 它可以不依赖其他进程单独地运行, 支持多个进程同时访问机密数据, 任何程序都可以通过 web 接口直接和配置管理模块交互, 这种方式大大降低了在原有程序内集成 Intel SGX 来保护机密信息的难度.

1 相关工作

对应用机密信息的保护一直是一个热门的研究课题, 其关键在于防止特权代码对用户机密信息的访问. 其研究方向主要分为基于软件层面的保护和基于硬件层面的保护.

首先探讨基于软件层面的研究. 微软早期就提出了 NGSCB[5], 它在硬件之上使用了称之为 isolation kernel 的隔离层, 然后再在隔离层之上同时运行了可信和不可信的操作系统, 最终将需要保护的应用运行在可信的操作系统中. 它通过隔离层和另一个可信的操作系统阻止了来自不可信操作系统的访问, 但是其 TCB 包含了隔离层和可信的操作系统, 是较为庞大的. 之后 Richard Ta-Min 等人提出的 Proxos[6]基于 xen 实现了类似于 NGSCB 的相似机制, 在不可信的系统运行的同时也运行另一个可信系统来保护目标应用. 之后 Xiaoxin Chen 等人提出的 Overshadow[7]是基于 VMware 的 VMM 上实现了对指定的 Guest OS 的内存的加密, 从而防止其他的系统对该系统内存的访问. 这些从软件层面防护的方法都需要在操作系统下层有一个可信的虚拟化层, 而且受保护的应用所在的 Guest OS 也必须是可信的. 这两个组件本身就非常复杂, 做到完全可信比较困难. 而且如果攻击者可以从物理上访问系统, 一旦该虚拟层被控制或者攻击者从物理上进行内存嗅探, 这些软件保护措施都无法应对.

由于仅依赖软件层面保护应用有一定的局限性, 所以也出现了很多从硬件层面保护应用的手段. 除了本文将要使用的 Intel SGX 之外, 当前最为主流的手段是 TPM(trusted platform modules)[8]和 ARM TrustZone[9].

TPM 提供一些特定的功能来保证系统的安全性. Flicker[10]实现了基于 TPM 的安全模块, 但是因为 TPM 的性能问题, 安全模式和非安全模式的切换会导致较高的性能开销. 针对 TPM 的性能问题, CloudVisor[11]为了避免频繁使用 TPM, 它将整个 hypervisor 也包括在 TCB 中, 而 TPM 主要负责 hypervisor 的启动完整性校验. 显然, 这扩大了 TCB.

ARM TrustZone 主要应用在移动设备应用安全, 它将整个系统分为普通区域和安全区域, 然后从硬件上保障普通区域的组件无法访问安全区域, 将需要保护的代码和数据运行在安全区域. 因为 TrustZone 的安

全区域只能有一个, 所以它需要在安全区域内运行单独的可信的系统来进行管理, 这部分系统是包含在 TCB 中的.

Intel SGX 也是一种基于硬件的软件防护机制. 相比 TPM, 它的模式切换并不需要较大代价, 所以它没有 TPM 的性能问题, 也不需要将 hypervisor 这种庞大的系统纳入 TCB. 相比 ARM TrustZone, 它可以同时运行多个互不干扰的安全环境, 每个安全环境仅仅包含应用程序本身, 不需要另外的可信系统进行单独的管理. 同时, Intel SGX 还可以防御物理内存嗅探, 这一点是 TPM 和 ARM TrustZone 都无法做到的. 所以本文采取 Intel SGX 技术对 Ansible 进行安全增强, 避免了 TPM 和 ARM TrustZone 的上述缺陷.

2 技术背景

2.1 Ansible

Ansible 是一个开源的跨平台自动化运维工具, 它的诞生时间相对于 Puppet、Chef 等等同类工具较晚, 但是后来居上. 在 github 上的 star、fork 和 commit 数量远远超过了其他同类工具, 这说明它的社区活跃度非常高. 根据 OpenStack 的 2016 年 4 月份的用户调查[12], 使用 Ansible 部署和配置 OpenStack 集群的用户比例已经和位于第一的 Puppet 相差在 1% 以内. 而在去年同期, Ansible 的用户比例落后 Puppet 14% 左右. 这说明 Ansible 在 OpenStack 的部署和配置方面的应用非常广泛, 而且还在持续地增长. 根据 The New Stack 的调查[13], Ansible 在容器编排方面的应用已经远远超过了 Puppet 和 Chef.

Ansible 可以方便地管理 Linux 和 Windows 系统. 相比其他的自动化运维工具, Ansible 的优势在于它的配置和部署简单, 它不需要在被管理的系统上安装任何 agent 程序. 只需要在 Ansible 的配置文件中设置好登录信息, Ansible 就可以根据这些信息登录相应的系统进行预设的管理操作. 而 Ansible 的配置信息目前是明文地存储在系统磁盘上的, 一旦磁盘的访问权限被攻击者非法获取, 这将意味着这些机密信息将完全被攻击者掌握, 攻击者将得到相应的系统的管理权限, 这将造成巨大的安全隐患.

Ansible 针对配置文件中机密信息保护的问题提出了相应功能, 即 Vault. Vault 会根据用户提供的密码将配置文件加密, 达到了配置文件中的机密信息不会明文出现在磁盘中的目的. 但是这样做也有很大的缺陷. 首先, 如果需要修改配置文件, 那么必须对整个文件重新加密. 其次, 用户每次读取配置文件时, 用户需要提供解密配置文件的密码. 如果用户每次使用 Ansible 都需要手动输入密码, 其便捷性将大大降低; 如果用户将加密配置信息的密码保存成另外一个配置文件, 这又会让问题回到原点, 这个存储密码的配置文件又明文出现在了磁盘上. 本文将针对这个问题使用 Intel SGX 来加固 Ansible 的配置安全.

2.2 Intel SGX

SGX 是 Intel 开发的新的处理器扩展指令集, 全称为 Intel Software Guard Extensions, 用于增强软件安全性. SGX 能将安全应用依赖的 TCB 减小到仅包含 CPU 和安全应用本身, 将不可信的其他所有因素都排除在安全边界之外.

SGX 可以在计算平台上提供一个可信的空间, 将用户软件的安全操作封装在一个 enclave(飞地) 中, 保障用户关键代码和数据的机密性和完整性, 甚至外部的特权软件 (比如操作系统或者 Hypervisor) 都无法访问 enclave 中的代码和数据. Enclave 的代码和数据位于一段称之为 EPC(enclave page cache) 的受保护的物理内存中, 内置在 CPU 中的内存加密引擎 (memory encryption engine, MME) 会在存取 DRAM 中的数据和指令时进行加密解密.

SGX 提供了一套密钥供给机制. 在同一个处理器上, enclave 中的程序可以生成一套和该 enclave 唯一对应的密钥. 这个密钥只有在同一个 enclave 中才能获取和使用, 整个过程对外界完全是加密的, 而且其他的 enclave 中也无法获取该密钥. 基于这套密钥供给机制, SGX 实现了密封机制. 密封可以在 enclave 中的数据持久化存储之前将其加密, 同时保证这些加密数据只有在相同的 enclave 中能解密. Enclave 首先获取专属于自己的密钥, 然后将 enclave 中的机密信息加密后导出到磁盘上持久保存. 当 enclave 需要再次解密原来的数据时, 只需要再次获取专属于自己的密钥来解密数据. 因为其他任何程序都无法获取该密钥, 所以可以保证这些信息的机密性.

3 系统整体架构

Ansible 默认把配置信息明文存储在磁盘的配置文件中. 显然, 这样很容易造成 Ansible 的配置信息泄露, 而且 Ansible 配置文件中的信息包含了其管理的系统的账号密码, 泄露这些信息的危害非常大. 因此必须将 Ansible 的配置信息保护起来. 为了保护 Ansible 的配置信息, 文本采取的方式是将 Ansible 的配置管理功

能独立成一个模块, 再基于 Intel SGX 对此模块进行安全增强.

系统的整体架构如图 1 所示, 整个系统主要包括 Ansible、配置管理模块和持久化存储三个部分. 配置管理模块运行在支持 Intel SGX 的平台上, 它将配置信息和管理配置信息的关键代码运行在 enclave 中, 确保里面的数据和代码的机密性和完整性. 然后在 enclave 外层进行封装, 提供配置信息增删改查的接口, Ansible 只能通过这些接口来获取配置信息. 模块的另一重要功能是持久化存储配置信息. 显然, 在将配置文件写入到磁盘之前必须将数据加密. 配置管理模块会利用 SGX 的密封机制保证数据的机密性.

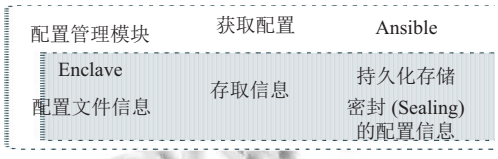

图 1 系统整体架构图

4 系统实现

4.1 设计配置管理模块

配置管理模块运行在开启了 Intel SGX 功能的平台上, 它负责 Ansible 的配置信息的管理和存取, 该模块的关键代码和相关数据运行在 enclave 中, Intel SGX 技术保证了 enclave 中的内容无法被外部非法窥探, 用户仅可以通过指定的 API 查询和修改数据.

配置管理模块的设计如图 2. 最底层的关键代码使用 C++编写, 这些代码会被编译成一个 enclave.signed.so 动态链接库文件, 运行时会以 enclave 的形式运行在加密内存之中, 只有 CPU 内部能获取解密后的数据和指令. Enclave.signed.so 以边界函数 (Edge-routines) 的形式对外部程序开放特定的接口, 这些接口是 enclave 和外部程序唯一的交互方式. ECALLS 表示外部程序通过边界函数调用 enclave 内部程序的接口; OCALLS 代表 enclave 内部程序通过边界函数调用外部程序的接口.

因为 ECALLS 只能被 C/C++代码直接调用, 为了让各种语言编写的程序都可以调用 enclave 的接口, sgxdb.so 对 ECALLS 再次做了一层封装, 向上提供了能适配各种语言的纯 C 的接口.

最外层的非核心业务逻辑和网络通信功能使用 python 实现. sgxdblib.py 模块对 sgxdb.so 提供的 C 接口做了进一步的 python 封装, 任何 python 程序都可以便捷地通过它来调用 enclave 提供的接口. 最终配置管理模块以 Web 接口的形式向外提供服务.

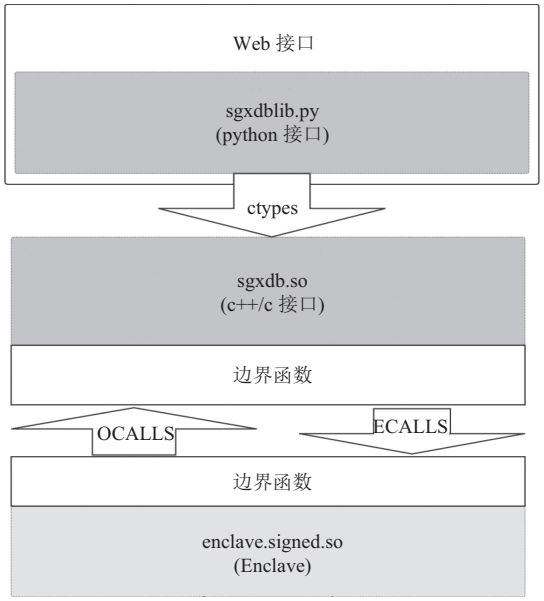

图 2 配置管理模块设计结构图

4.2 持久化存储配置信息

程序的停止和重启是任何应用程序生命周期中必然会发生的环节, 配置管理模块也不例外. 配置管理模块的配置信息存储在 enclave 中, 必须有相关的机制在程序停止前持久化存储 enclave 中的状态和数据, 并能够在下次程序重启时进行恢复. 在模块的运行过程中, enclave 内部的数据和代码无法被外部任何程序窥探, 所以对 enclave 内状态和数据的保存和恢复必须在 enclave 内部完成. 为了保证数据的机密性, 最终保存在磁盘上的数据必须是进行过加密的, 而且必须保证这些数据无法被任何第三方解密.

持久化存储配置信息流程如图 3 所示. 当配置管理模块需要将 enclave 中的状态和数据持久化存储时, 需要获得和当前 enclave 唯一对应的密钥, 然后用该密钥在 enclave 内部将需要导出的数据加密, 最终将加密后的信息导出到 enclave 外部的持久化存储中.

首先, 程序会在 enclave 内部使用 EGETKEY 指令, 该指令会收集能唯一标识该 enclave 的信息和嵌入在 CPU 内部的 SGX master Derivation Key 这两个数据作为 AES-CMAC[14]算法的输入, 然后生成 Provisioning Key. 只要这两个输入的值确定, 最后生成的 Provisioning Key 就是确定的. 每个支持 SGX 的 CPU 都有唯一的 SGX master Derivation Key, 为了保证它的机密性, 用

户仅仅可以在使用 EGETKEY 指令时让 CPU 自动读取它, 但是无法直接获取它的值. 唯一标识该 enclave 的信息有两种选择, 调用 EGETKEY 指令时可以选择 enclave 的度量 (enclave measurement) 或者 enclave 的作者的身份信息 (即 enclave 初始化时 MRSIGNER 寄存器中的值) 来标识一个 Enclave. 如果使用的是 enclave 的度量信息作为 AES-CMAC 算法的输入, 下次相同代码和初始数据的 enclave 才能生成相同的 Provisioning Key; 如果使用 enclave 作者的信息, 即使代码和初始数据有不同, 只要是相同的作者创建的 enclave, 也会生成相同的 Provisioning Key. 本文中使用 enclave 的度量作为标识信息. 最终, 用生成的 Provisioning Key 来加密 enclave 中的状态和数据, 将加密的数据导出到持久化存储中后, 配置管理模块就可以正常关闭了.

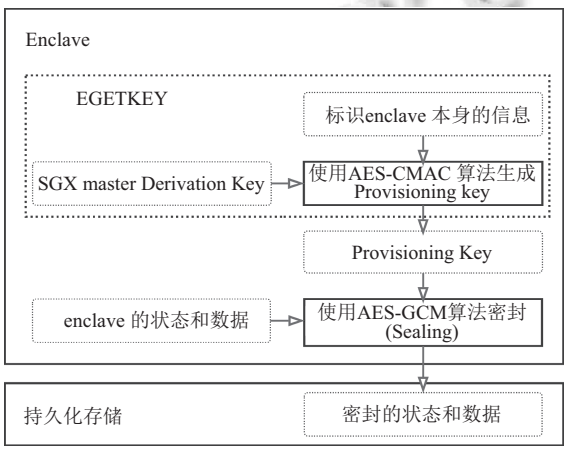

图 3 持久化存储配置信息流程

当配置管理模块再次启动需要恢复之前的状态和数据时, 仍然用相同的方式调用 EGETKEY. 在同一个处理器上的相同的 enclave 可以保证 SGX master Derivation Key 和标识 enclave 本身的信息相同, 从而可以得到相同的 Provisioning Key. 生成了相同的密钥后, 我们就可以再次在 enclave 中解密数据来恢复程序关闭之前的状态.

5 评估

5.1 安全增强评估

TCB 是保障系统安全所需的软硬件的集合, 保持较小规模的 TCB 可以减少系统产生漏洞的概率.

本系统的 TCB 的软件部分的代码都包含在配置管理模块中, 其代码近似行数 (lines of code, LoC) 见表 1. 移植到 Enclave 内的 C++标准库占了代码量的绝大比重, 而且这一部分代码量是较多的. Intel 在 SGX SDK 中包含了 C++标准库的移植, 在编译阶段整个 C++标准库会被静态链接到最终生成的 enclave.signed.so 库函数中. 相比其他的库函数和用户自己编写的代码, C++标准库函数经过了大量使用和检验, 其本身的可靠性较高, 将它移植到 Enclave 中并不会对它的可靠性造成太大影响, 所以这部分代码是较为可靠的. 其他包含在 TCB 中的代码在 4000 行左右, 总体来看, 软件规模是较小的, 系统是相对安全可控的.

表 1 代码量统计

组件	代码量(LoC)	TCB
Enclave内C++标准库	57400	√
Enclave内其他库函数	3500	√
配置管理模块(Enclave内)	500	√
配置管理模块(Enclave外)	900	×

除了软件部分, 本系统还依赖于 Intel SGX CPU 的安全性. Intel SGX 假设攻击者可以控制包括操作系统在内的所有软件, 对系统中传输的任何数据进行任意修改, 对物理内存进行任意读取, 但是无法从物理上破解带有 Intel SGX 功能的 CPU. 只要符合该假设, Intel SGX 就可以保障 enclave 内部数据和代码的机密性和完整性. 对于攻击者来说, 攻击 CPU 硬件是极其困难的, 而且 Intel 也做了相当多的工作来保障 CPU 的安全, 因此硬件方面的可靠性是相当高的.

本系统主要关注配置管理模块本身的安全性, 当配置信息从配置管理模块被取出后, 其机密性不受本系统保障. 但是 Ansible 取出配置信息后, 其他仍在模块内的配置信息的机密性不受影响, 而且这些取出的配置信息仅仅会短时间存在于内存中.

5.2 性能评估

本节设计了实验来评估加入配置管理模块后对 Ansible 日常操作性能的影响. 本文使用了戴尔 Inspiron i7359 笔记本电脑, CPU 型号为 Intel(R) Core(TM) i7-6500U CPU @ 2.50 GHz, 内存大小为 8 G, 操作系统版本为 Ubuntu 14.04 LTS, 本文选用的 linux sgx driver 的版本为 1.7. 配置管理模块和 Ansible 都运行在 Inspiron i7359 上, 被管理的服务器是一台装有 Ubuntu 14.04 LTS 的虚拟机, 其 CPU 为 2 核, 内存大小为 4 G. 本文使用 TP-LINK WR842N 搭建局域网作为实验网络环境.

实验中, Ansible 执行了服务器管理操作并对相应操作计时. 实验根据 Ansible 是否使用配置管理模块分为两组, 每组实验执行 100 次指定的操作, 然后对比两

组实验的每次操作平均其耗时. 实验中选用的操作一共包含三种: 1) 获取根目录下的文件列表; 2) 安装 Nginx; 3) 重启 Mysql 服务. 实验将根据是否使用配置管理模块对比这三种操作的平均耗时. 因为一般程序首次启动时间会比之后启动时间稍长, 为了避免这一不稳定因素, 每组实验的最初的 5 次操作时间不会被计算到最终的操作平均耗时.

如果使用配置管理模块, Ansible 执行服务器管理操作时, 会先从配置管理模块查询相应的服务器配置信息. 否则, Ansible 将直接从磁盘读取配置信息. 根据图 4 的结果, 从配置管理模块读取配置信息比直接从磁盘读取配置信息平均耗时多 179~227 ms, 占总耗时的 4%~13%. 在保证安全性的条件下, 这个性能损耗是可以接受的. 根据配置管理模块的统计, 每次从配置管理模块读取配置信息的耗时大约在 200 ms 左右, 其中执行查询操作时间消耗在 1 ms 左右, 剩下的时间主要由网络通讯和外部的 web 框架本身的开销组成. 因为执行的查询操作并非计算密集型, 这部分耗时较少是符合预期的. 如果仍然觉得配置管理模块的开销较大, 可以直接将配置管理模块以动态链接库的方式嵌入在 Ansible 中来消除相应的网络开销. 这种方式虽然可以达到和原生 Ansible 一样的性能, 但是与将配置管理模块以单独的服务运行相比, 它会增加系统的耦合性.

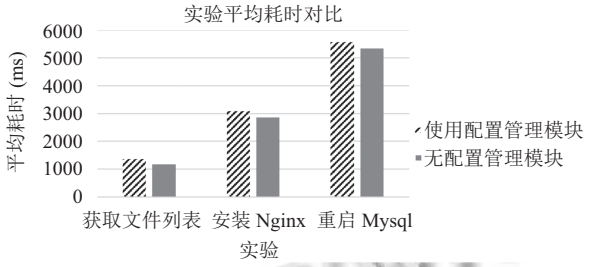

图 4 实验平均耗时对比

6 总结与展望

本文对基于 Intel SGX 技术实现了对 Ansible 的配置信息的保护, 相比原来 Ansible 直接将包含敏感信息的配置文件明文存储在磁盘上, 本文达到了配置文件在内存和磁盘中都处于被加密状态的目标, 实现了对 Ansible 的安全增强. 从实验评估结果可以看到, 本文提出的安全机制虽然会带来一些额外性能开销, 但其性能损耗是可以接受的. 本文配置管理模块的代码已经开源[15].

本文目前仅仅是将配置管理的安全增强应用在了 Ansible 中, 但在未来还可以将其扩展成通用组件, 作为通用的配置管理安全增强模块集成到例如 OpenStack 等云平台中.

参考文献

1 https://github.com/ansible/ansible.
2 INTEL CORP. Intel® software guard extensions programming reference, Ref.#.329298-002US. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf. [2014-10].
3 INTEL CORP. Intel® software guard extensions (Intel® SGX), Reference Number: 332680-002. https://software.intel.com/sites/default/files/332680-002.pdf. [2015-06].
4 Costan V, Devadas S. Intel sgx explained. Cryptology ePrint archive: Report 2016/086. https:// eprint.iacr.org/2016/086. [2017-02-20].
5 Peinado M, Chen YQ, England P, *et al*. NGSCB: A trusted open system. Australasian Conference on Information Security and Privacy. Berlin Heidelberg, Germany. 2004. 86–97.
6 Ta-min R, Litty L, Lie D. Splitting interfaces: Making trust between applications and operating systems configurable. Proc. of the 7th USENIX Symposium on Operating Systems Design and Implementation. Seattle, WA, USA. 2006. 279–292.
7 Chen XX, Garfinkel T, Lewis EC, *et al*. Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. Proc. of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems. Seattle, WA, USA. 2008. 2–13.
8 ARM Limited. ARM security technology: Building a secure system using trustZone technology. White Paper PRD29-GENC-009492C, 2009.
9 Trusted Computing Group. TPM Main Part 1 Design Principles, Specification. version 1.2, revision 103, 2007.
10 Mccune JM, Parno BJ, Perrig A, *et al*. Flicker: An execution infrastructure for TCB minimization. Proc. of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems. Glasgow, Scotland UK. 2008. 315–328.
11 Zhang FZ, Chen J, Chen HB, *et al*. CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. Proc. of the 23rd ACM Symposium on Operating Systems Principles. 2011. 203–216.
12 Openstack user survey: A snapshot of openstack users' attitudes and deployments. https://www. openstack.org/assets/survey/April-2016-User-Survey-Report.pdf. [2016-04].
13 Hecht L. TNS research: Is ansible really leading chef and puppet for container orchestration? http://thenewstack.io/ansible-leading-chef-puppet/. [2016-06-17].
14 Dworkin M. Recommendation for block cipher modes of operation: The CMAC mode for authentication NIST Special Publication 800-38B, NIST, 2005.
15 https://github.com/you-n-g/sgxdb.