软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(8):3637−3658 [doi: 10.13328/j.cnki.jos.006612]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

可信执行环境访问控制建模与安全性分析*

苗新亮[1,2], 常 瑞[2,3], 潘少平[2], 赵永望[2], 蒋烈辉[1]

[1](数学工程与先进计算国家重点实验室 (战略支援部队信息工程大学), 河南 郑州 450002)

[2](浙江大学 计算机科学与技术学院, 浙江 杭州 310027)

[3](浙江省区块链与网络空间治理重点实验室, 浙江 杭州 310027)

通信作者: 常瑞, E-mail: crix1021@zju.edu.cn

摘 要: 可信执行环境 (TEE) 的安全问题一直受到国内外学者的关注. 利用内存标签技术可以在可信执行环境中实现更细粒度的内存隔离和访问控制机制, 但已有方案往往依赖于测试或者经验分析表明其有效性, 缺乏严格的正确性和安全性保证. 针对内存标签实现的访问控制提出通用的形式化模型框架, 并提出一种基于模型检测的访问控制安全性分析方法. 首先, 利用形式化方法构建基于内存标签的可信执行环境访问控制通用模型框架, 给出访问控制实体的形式化定义, 定义的规则包括访问控制规则和标签更新规则; 然后利用形式化语言 B 以递增的方式设计并实现该框架的抽象机模型, 通过不变式约束形式化描述模型的基本性质; 再次以可信执行环境的一个具体实现 TIMBER-V 为应用实例, 通过实例化抽象机模型构建 TIMBER-V 访问控制模型, 添加安全性质规约并运用模型检测验证模型的功能正确性和安全性; 最后模拟具体攻击场景并实现攻击检测, 评估结果表明提出的安全性分析方法的有效性.

关键词: 内存标签; 可信执行环境; 访问控制; 模型检测; 安全性分析

中图法分类号: TP311

中文引用格式: 苗新亮, 常瑞, 潘少平, 赵永望, 蒋烈辉. 可信执行环境访问控制建模与安全性分析. 软件学报, 2023, 34(8): 3637–3658. http://www.jos.org.cn/1000-9825/6612.htm

英文引用格式: Miao XL, Chang R, Pan SP, Zhao YW, Jiang LH. Modeling and Security Analysis of Access Control in Trusted Execution Environment. Ruan Jian Xue Bao/Journal of Software, 2023, 34(8): 3637–3658 (in Chinese). http://www.jos.org.cn/1000-9825/6612.htm

Modeling and Security Analysis of Access Control in Trusted Execution Environment

MIAO Xin-Liang[1,2], CHANG Rui[2,3], PAN Shao-Ping[2], ZHAO Yong-Wang[2], JIANG Lie-Hui[1]

[1](State Key Laboratory of Mathematical Engineering and Advanced Computing (Strategic Support Force Information Engineering University), Zhengzhou 450002, China)

[2](College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)

[3](Key Laboratory of Blockchain and Cyberspace Governance of Zhejiang Province, Hangzhou 310027, China)

Abstract: The security of the trusted execution environment (TEE) has been concerned by Chinese and foreign researchers. Memory tag technology utilized in TEE helps to achieve finer-grained memory isolation and access control mechanisms. Nevertheless, prior works often rely on testing or empirical analysis to show their effectiveness, which fails to strongly guarantee the functional correctness and security properties. This study proposes a general formal model framework for memory tag-based access control and introduces a security analysis method in access control based on model checking. First, a general model framework for the access control model of TEE based on memory tag is constructed through a formal method, and those access control entities are formally defined. The defined rules include

* 基金项目: 国家自然科学基金 (61872016, 62132014)
本文由“形式化方法与应用”专题特约编辑陈立前副教授、孙猛教授推荐.
收稿时间: 2021-08-30; 修改时间: 2021-10-14; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28
CNKI 网络首发时间: 2023-01-19

access control rules and tag update rules. Then abstract machines under the framework are incrementally designed and implemented with formal language B. In addition, the basic properties of the machines are formalized through invariant constraints. Next, a TEE implementation called TIMBER-V is used as an application case. The TIMBER-V access control model is constructed by instantiating these abstract machines, and the security properties are formally specified. Furthermore, the functional correctness and security of the models are verified based on model checking. Finally, this study simulates the specific attack scenarios, and these attacks are successfully detected. The evaluation results have proved the effectiveness of the security analysis method.

Key words: memory tag; trusted execution environment (TEE); access control; model checking; security analysis

可信执行环境 (trusted execution environment, TEE) 是利用软硬件隔离机制提供的一块可信区域, 通过对加载到该区域的数据和代码提供机密性和完整性保护, 防御来自恶意系统的攻击. 近几年, TEE 技术在工业界和学术界均受到持续关注, 工业界的主流 TEE 实现方案包括 ARM TrustZone[1]、Intel SGX[2]、AMD SEV[3]等, 学术界已有的典型方案包括 TIMBER-V[4]、Sanctum[5]、Sanctuary[6]、Keystone[7]等. 这些 TEE 实现方案广泛应用于云端和设备端来托管数字产权管理[8]、移动支付[9–11]和生物特征识别[12]等安全功能. 然而, 随着 TEE 中曝出越来越多的漏洞[13–18], 其安全性正面临着严重的挑战.

为了提高 TEE 自身的安全性, 研究者们提出了一些增强内存隔离性的方案. 这些方案或通过在不可信区域构建隔离域以支持可信应用运行[6,19–21], 或通过对可信区域进行进一步划分以增加隔离区域数量[7,22,23]. 尽管这些方案有效减小了 TEE 系统的攻击面, 但由于这些内存隔离仍然是区域 (region) 级的, 即保护的内存在连续的地址空间中, 因此无法支持更细粒度的安全保护策略. 同时, 隔离区域的增加也意味着更大的性能损耗. 为此, 内存标签 (memory tag) 技术逐渐应用于 TEE 的内存保护中. 内存标签的核心是在每个内存字上划分若干位作为标签位, 用于标识内存对象上的某种属性, 可以提供细粒度、灵活的隔离边界. AMD 上的 TEE 实现方案如 SEV[3], SEV-ES[24], SEV-SNP[25]利用物理地址上的 43–46 位标记不同的虚拟机, 当通过页表访问虚拟机内存 (可信区域) 时, 根据虚拟机物理地址携带的标签判断访问哪个虚拟机数据. 此外, TIMBER-V[4]在内存字上使用 2 位标签位标记不同的隔离区域并根据标签判断访问是否允许, 而 PENFLAI[26]利用所有权位图来标记每个物理页的状态, 区分属于可信区域还是不可信区域. 这些方案均实现了页表级的内存隔离以支持更细粒度的访问控制策略, 有效地防御了某些攻击. 然而, 这些利用内存标签的方案都是以测试或经验分析的方式对访问控制机制的有效性进行评估, 一直缺乏严格的功能正确性和安全性分析, 因此数据泄露[16]、任意代码执行[27]、提权[28,29]等攻击仍时有发生.

为解决上述问题, 本文利用形式化方法对 TEE 访问控制进行研究, 形式化方法是借助数学方法建立数学模型并进行模型验证的方法, 可以对模型满足的性质提供严格的保证[30]. 本文的研究工作主要包括两个方面.

(1) 对基于内存标签的 TEE 访问控制进行形式化建模. 针对基于内存标签的 TEE 访问控制机制, 本文提出通用的模型框架, 该框架基于 TEE 提供的隔离区域建立标签域, 然后利用标签在较高的抽象层次上定义了主体属性、客体属性和权限操作, 并构建了安全规则, 包括访问控制规则和标签更新规则. 结合 TEE 系统运行过程, 分析了访问控制中的状态迁移行为, 并描述了基本操作和需要满足的基本安全性质. 本文利用形式化语言 B 设计并实现了该框架的访问控制抽象机模型, 包括标签、主体、客体和策略 4 个基本实体抽象机和访问控制执行抽象机. 同时抽象机模型使用形式化的规范描述了作用于模型操作中的不变式约束以保证无二义性. 基于创建的标签域, 该框架和抽象机模型适用于 ARM TrustZone 系列、AMD SEV 系列以及基于 RISC-V 的 PENGLAI, Keystone, TIMBER-V 等 TEE 方案.

(2) 实现基于模型检测的 TEE 访问控制安全性分析. 为实现基于内存标签的 TEE 访问控制安全性分析, 本文提出一种基于模型检测的安全性分析方法. 通过分析具体的访问控制规则、标签更新规则以及系统中的安全操作, 以不变式约束的形式对访问控制安全性质进行规约, 并利用模型检测工具 ProB 检测构建的模型是否符合安全性质, 实现模型功能正确性和安全性的验证. 同时, 结合威胁模型的定义, 构建威胁操作以模拟攻击场景, 并通过模型检测验证威胁模型是否与安全性质冲突以实现攻击检测. 本文以 TIMBER-V 中的访问控制为应用案例, 结合 TIMBER-V 的具体实现, 通过实例化通用模型构建了 TIMBER-V 基于内存标签的访问控制模型, 然后利用提出的安全性分析方法实现了 TIMBER-V 访问控制安全性分析.

本文的主要贡献如下.

(1) 提出一种基于内存标签的 TEE 访问控制通用模型框架 MTF: 通过构建 TEE 标签域, 将标签属性赋给访问控制实体并给出各实体的形式化定义, 框架还分析系统访问控制状态迁移关系, 描述访问控制基本操作和性质, 同时利用形式化 B 方法实现该框架的抽象机模型, 重点设计 4 个基本实体抽象机并递增构建了基于内存标签的 TEE 访问控制模型 MTAC, 形式化描述了基本实体操作, 支持模型扩展和改进.

(2) 模型实例化: 为验证 (1) 提出的模型框架的通用性, 以 TEE 的一个典型实现 TIMBER-V 为应用案例, 实现模型实例化, 结合系统实现方式分别对基本实体抽象机和模型 MTAC 进行了修改和扩展, 构建 TIMBER-V 访问控制模型 MTAC_TIV, 包含 42 个实体操作以及 16 个安全不变式 (框架和实例化模型开源在 https://github.com/mxlaga/Memory-tag-based-access-control-models.git).

(3) 提出一种基于模型检测的访问控制安全性分析方法: 该方法结合具体实现的访问控制规则和标签更新规则, 以不变式约束形式化描述访问控制规范, 验证模型满足的安全性质, 对 (2) 建立的模型进行模型检测, 自动化遍历了 14 699 098 个状态和 37 834 011 个状态转移, 评估结果表明无死锁及不变式冲突; 同时根据威胁模型模拟了 2 个具体的攻击场景, 模型检测结果可有效检测出非法操作及违反的安全性质.

本文第 1 节介绍用到的一些基础知识, 包括 B 方法的基本语法和符号, 以及可信执行环境相关架构知识. 第 2 节分析威胁模型, 给出安全假设以及威胁模型的形式化定义. 第 3 节提出基于内存标签的 TEE 访问控制模型框架. 第 4 节描述构建的抽象机模型, 包括模型结构和具体操作. 第 5 节重点阐述模型添加的变量及操作, 给出实例化模型. 第 6 节利用模型检测进行安全性分析并展示检测结果. 第 7 节讨论针对不同 TEE 方案如何实例化模型并给出未来工作展望. 第 8 节讨论相关工作. 第 9 节总结全文.

1 预备知识

1.1 B 方法

B 方法[18]是一种面向模型的形式化方法, 基于一阶逻辑和集合论基础, 其提供的形式化语言称为 B 语言. 该方法的基本思想是从一些已知的简单抽象数学对象出发, 构造目标系统的状态特征和行为特征模型, 其过程覆盖了从规范说明到代码生成的整个系统开发周期, 完全证明后的形式化模型能够转换为实现的可执行代码. B 方法描述系统的基本单元是抽象机, 每个抽象机从关键字 MACHINE 开始到 END 结束. 关键字 CONSTANTS 声明常量, PROPERTIES 声明常量类型, VARIABLES 声明变量, INVARIANT 以逻辑公式的方式描述变量的不变式关系, INITIALISATION 声明所有变量的初始值, OPERATIONS 描述抽象机操作, 包括输入、输出以及对抽象机状态的影响. 由于 B 方法是基于集合论的, 其支持的数据类型均为集合, 如$\mathbb{Z}$表示整数集合, $\mathbb{N}$表示自然数集合, INT 和 NAT 分别表示系统可实现的整数集合和自然数集合, 以及 BOOL 表示布尔类型, 其基本符号如表 1 所示. 同时, B 方法通过 INCLUDES, IMPORTS 以及 SEES 等连接来描述不同抽象机之间的关联, 如 INCLUDES M1 表明该抽象机包含抽象机 M1, 用于从简单的抽象机组合出更为复杂的抽象机; IMPORTS M1 表明该抽象机和抽象机 M1 的实例之间的关系, 用于创建 M1 的一个具体实例并完全占用 M1 中的服务; SEES M1 表明该抽象机参考抽象机 M1 实例, 可以以只读的方式引用抽象机 M1 中的变量. 支持 B 方法的形式化验证工具包括 Atelier B 和 ProB. Atelier B[19]是一个定理证明工具, 在对构造的抽象机进行类型检查后, 它自动生成证明义务并支持交互式定理证明. ProB[20]是一个约束求解器, 支持对 B 方法构造的抽象机进行模型检测, 同时可以将规约的状态空间图形化. 由于 B 方法基于严格定义的数学概念和语言, 保证了规范描述简明、精确和无歧义性, 且开发工具较为成熟, 目前已经应用于诸多重要安全攸关的领域如无人驾驶系统、交通调度等.

1.2 可信执行环境

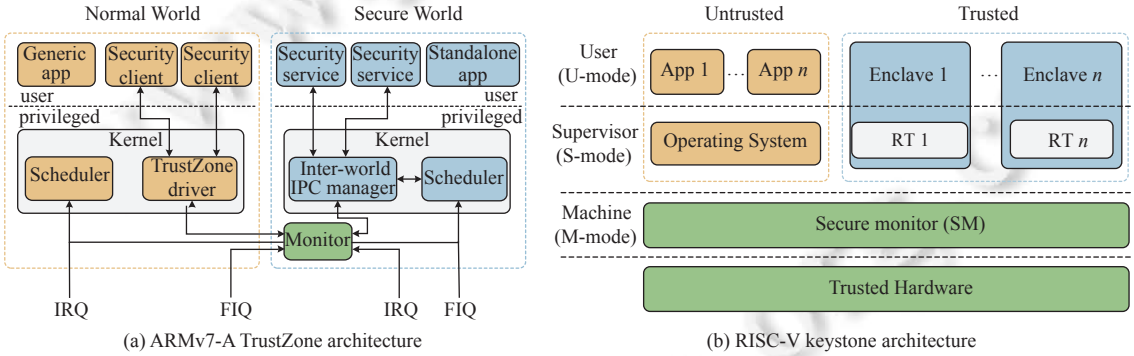
可信执行环境 (TEE) 是一个隔离的区域, 可以在其中加载需要保护的敏感代码和数据. 它提供隔离的 CPU、内存和外设访问, 以及隔离执行、机密性和完整性保护等基于硬件的安全功能. 该隔离机制通常在逻辑上将执行环境划分为 TEE 和富执行环境 (rich execution environment, REE), 分别称为安全世界 (secure world) 和普通世界

(normal world). 安全世界中的程序可以访问普通世界中的资源, 但反之则不行.

ARM TrustZone 是目前最主流的 TEE 之一, 广泛应用于嵌入式设备中以提供高效的系统级安全保护. TrustZone 将安全敏感的代码和数据存储在安全世界中, 包括可信操作系统和可信应用 (TA). 普通世界中运行普通的操作系统和客户端应用 (CA), 它们只能访问普通世界中的资源. 如图 1(a) 所示, TrustZone 有 3 种处理器模式: 安全世界、普通世界和用于在两个世界之间切换的监视器模式 (monitor mode). 其中普通世界中运行 Linux、Android 等普通操作系统, 安全世界中运行安全子系统或定制内核. 此外, 安全世界中的系统内核可以利用内存管理单元将安全世界内存空间划分成多个用户空间沙箱. 只要安全内核是正确实现的, 来自不同厂商的 TA 就可以在不需要互相信任的情况下安全地并发执行. TrustZone 通过 SMC 指令、IRQ、FIQ 或者异常进入 Monitor Mode, 实现不同世界的切换.

表 1 B 符号

$\mathbb{N}_1$	非0自然数集合
$x \mapsto y$	有序对 (x,y)
$S_1 \times S_2$	$\{(x,y) \mid x \in S_1 \wedge y \in S_2\}$
$\tilde{p}$	$\{(y,x) \mid x \in S_1 \wedge y \in S_2 \wedge (x,y) \in p\}$
$dom(p)$	$\{x \mid x \in S_1 \wedge \exists y.(y \in S_2 \wedge (x,y) \in p)\}$
$ran(p)$	$dom(\tilde{p})$
$s \lhd p$	$\{(x,y) \mid (x,y) \in p \wedge x \in s\}$
$s \lhd\!\!\!- p$	$\{(x,y) \mid (x,y) \in p \wedge x \notin s\}$
$s \nrightarrow t$	部分函数
$s \rightarrow t$	全函数



(a) ARMv7-A TrustZone architecture (b) RISC-V keystone architecture

图 1 不同处理器架构下的 TEE 实现

Keystone 是一个开源 TEE 框架, 它利用物理内存保护 (PMP) 在 RISC-V 之上构建了一个高度可定制的 TEE. 如图 1(b) 所示, Keystone 使用了 RISC-V 提供的 3 种处理器模式: 用户模式 (U-mode)、监管者模式 (S-mode) 和机器模式 (M-mode). 用户应用程序运行在 U-mode, 操作系统运行在 S-mode. 安全监视器 (SM) 运行在 M-mode, 可以直接访问物理资源 (如中断、内存、外设等), 它利用硬件原语为 TEE 提供安全保证. 每个用户程序可以创建对应的安全区 (enclave), 在 enclave 中加载安全敏感的程序和数据. 每个 enclave 都运行在隔离的物理内存区域中, 并且有自己的运行时 (RT). RT 运行在 S-mode, 提供用于系统调用的接口、标准 libc 支持、enclave 内虚拟内存管理等功能. 安全监视器还可以利用可配置硬件来提供增强的安全机制.

2 威胁模型和安全假设

介于普通世界中操作系统的高复杂度, 假设攻击者可以通过系统漏洞实施内核攻击以获取该操作系统的所有

权 (如图 2 所示), 因此攻击者不仅可以任意读写普通应用程序内存、篡改应用程序调用的返回值, 还可以利用系统安全功能创建恶意的安全区 (enclave) 来破坏其他良好的安全区, 从而泄露、篡改敏感数据以及执行任意代码. 本文假设安全世界中的内核、安全监视器 (SM) 以及硬件是可信的, 同时加密原语是安全的. 针对 TEE 的硬件攻击和侧信道攻击不在考虑范围内.

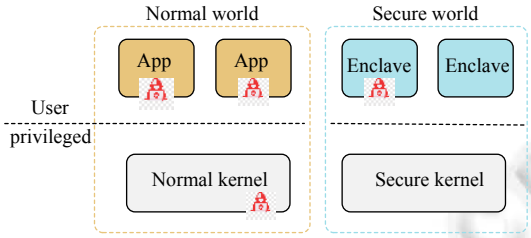


图 2 敌手模型

本文同时对访问控制系统模型和该系统的威胁模型形式化定义如下.

定义 1. 访问控制系统. 访问控制系统是一个四元组 $S=\langle P,P_0,F,T\rangle$, 其中, P 是系统状态集合, $P_0\subseteq P$ 是初始状态集合, F 是操作集合, 当 $f\in F$ 时, $f(p)$ 表示状态 p 在执行操作 f 后到达的一个系统状态, $T=\left\{\left(p_i,f,p_j\right)|f(p_i)=p_j,p_i\in P,p_j\in P,f\in F\right\}$ 表示状态迁移关系的集合.

定义 2. 访问控制系统威胁模型. 访问控制系统威胁模型定义为一个五元组 $S'=\langle P,P_0,F,T,\Phi\rangle$, $\Phi=\{\phi_1,\phi_2,\ldots,\phi_n\}$ 为包含访问控制规则的安全约束集合且集合 T 中存在攻击转移行为 t'使得访问控制系统 $S\models\Phi$ 不成立, 即系统不满足安全约束 Φ, 其中 t'由执行威胁操作 f'得到的状态迁移.

3 基于内存标签的访问控制模型框架

3.1 模型整体框架

访问控制模型中, 请求者 (主体)、被访问者 (客体) 以及权限操作是一个访问请求中必需的 3 个要素. 访问控制执行模块以这 3 个要素作为输入, 通过检查是否满足访问控制规则给出请求允许或拒绝操作. 在此基础上, 基于内存标签的访问控制将内存标签作为一个重要属性参与访问请求的判断过程. 同时, 内存标签通常可以进行更新操作以满足内存管理的灵活性. 结合可信执行环境提供多域隔离的特点, 如图 3 所示, 本文提出的基于内存标签的访问控制模型主要有以下组成部分: 标签域、主体、客体、权限操作、规则和访问控制执行模块.

内存标签通过对内存字预留的比特位赋值来对该内存地址空间进行标记. 由于 TEE 技术将运行环境隔离为非可信域 (non-trusted domains) 和可信域 (trusted domains), 同时 CPU 运行模式一般分为用户模式 (user mode) 和监管者模式 (supervisor mode). 因此, TEE 实现的隔离域可分为 NU 域、NS 域、TU 域和 TS 域, 分别代表不可信用户空间、不可信内核空间、可信用户空间和可信内核空间. 本文提出的模型框架定义标签集合 $TAG=\{$NU, NS, TU, TS$\}$, 这些利用标签进行标记的隔离域称为标签域. 利用标签域, 可分别描述主体和客体所在的执行空间.

定义 3. 主体. 主体作为发起访问的个体, 可以是用户, 也可以认为是进程、服务、程序等对象, 主体有主体集合 SUB, 同时有主体标识集合 SID. 每个主体 sub 有自己的标识以及标记主体所在隔离域的标签, 这些组成主体属性. 主体属性是一个四元组: $SA=\langle sid,stag,exe\_state,ac\_state\rangle$, 其中, $sid\in SID$, $stag\in TAG$, $exe\_state\in sub\rightarrow EXE\_STATE$, $ac\_state\in sub\rightarrow AC\_STATE$, EXE_STATE 和 AC_STATE 均为枚举集合, 表示主体的执行状态和访问状态.

主体的唯一性通过主体标识属性和主体标签属性确定. 同时, 在可信执行环境中, 隔离域可通过定义的系统调用、中断、异常以及特殊指令等方式实现域间的切换. 因此, 主体标签属性在执行相关域切换操作后可动态修改.

定义 4. 客体. 客体是被请求访问的资源, 在基于内存标签的访问控制模型框架中, 客体认为是内存资源, 客体

有客体集合 OBJ, 也有客体标识集合 OID. 通过客体标识以及客体所在的标签域, 保证每个客体的唯一性. 客体属性是一个三元组: $OA=\langle oid, otag, status\rangle$, 其中, $oid\in OID$, $otag\in TAG$, $status\in Status$, OID 是自然数集合, $Status$ 是一个枚举集合, 表明该客体资源是否可用.

内存标签通常支持动态更新以提高内存管理的灵活性. 标签更新操作在以主体属性、客体属性为参数的更新规则下执行. 同时在访问控制过程中, 客体的可用状态是影响访问结果的重要因素, 当客体正在被访问时, 其资源状态应改为不可用以防止冲突, 同时在访问结束后, 其状态更改为可用以备其他主体访问.

定义 5. 权限操作. 权限操作是定义主体行为一类对象, 定义主体对客体执行访问操作的类型. 用 OP 表示权限操作集合, 为突出模型的通用性, 该集合中仅定义读 (Read)、写 (Write) 和执行 (Exc) 操作, 表示为 OP={Read, Write, Exc}. 其他操作可在实例验证过程中根据真实环境进行添加.

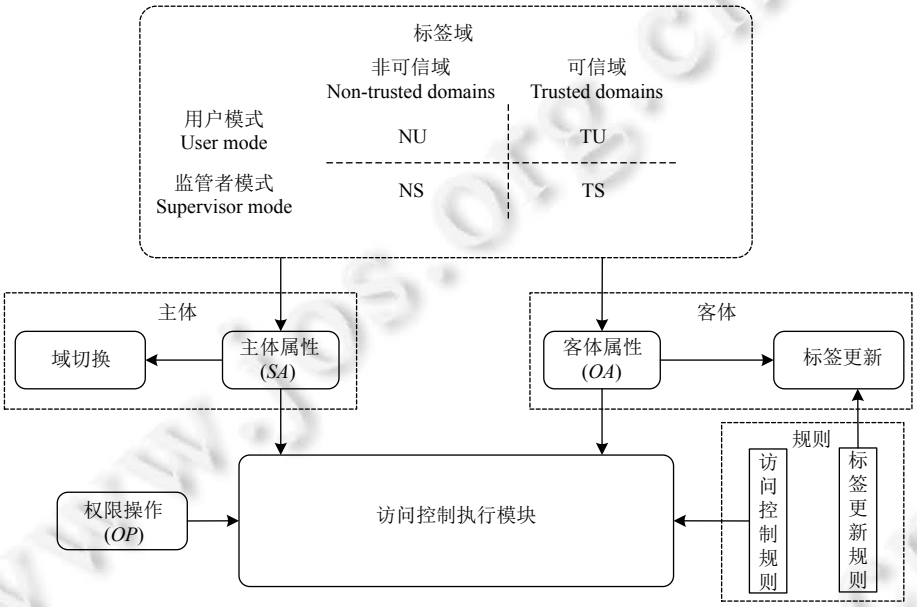


图 3 MTF: 基于内存标签的 TEE 访问控制模型框架

3.2 规则定义

访问控制执行模块的一个核心是检查主体对客体的访问是否满足访问控制规则. 该模型框架中基于内存标签的访问控制规则如表 2 所示. 依据定义 5 简写权限给出访问控制规则, 其中 r 表示读 (Read)、w 表示写 (Write)、x 表示执行 (Exc). 当主体和客体在同一个标签域时, 主体对客体具有操作的所有权. 为提高内存保护, 管理模式执行保护 (SMEP) 技术广泛应用于处理器中以禁止内核执行用户空间的代码, 因此模型中其他标签域对 NU 域和 TU 域的访问仅限于读写操作. 标签更新操作用于更改客体资源所处的标签域, 比如在一个非可信用户空间创建一个隔离空间来保护敏感数据和代码时, 则该隔离空间从 NU 更新为 TU, 当执行完毕并销毁该空间后, 其标签重新标记为 NU. 如表 3 所示, 标签仅允许在同等或低于本身安全等级的标签域中更新以防止权限提升.

表 2 访问控制规则

	NU	NS	TU	TS
NU	rwx	—	—	—
NS	rw-	rwx	—	—
TU	rw-	rw-	rwx	—
TS	rw-	rw-	rw-	rwx

表 3 标签更新规则

	NU	NS	TU	TS
NU	√	×	×	×
NS	√	√	×	×
TU	√	√	√	×
TS	√	√	√	√

3.3 状态迁移分析

操作系统访问控制过程中, 访问操作的执行状态记录主体执行访问时所处的状态, 该状态是影响访问请求能否响应的关键因素之一. 同时访问控制执行模块在响应过程中会产生对应的访问状态. 访问状态是对访问执行过程中各个阶段的抽象, 并最终反馈访问控制执行结果, 是模型中必要的状态描述. 因此, 该模型框架抽象定义主体访问客体过程中的执行状态 (*EXE_STATE*) 和访问状态 (*AC_STATE*) 并附加为主体属性. 执行状态包括为就绪 (Ready)、运行 (Run)、阻塞 (Blocked) 和最终状态 (Final). Ready 状态是主体在创建后就进入的状态, 此时主体准备或正在发起访问请求; Run 状态表示当前主体正在访问客体资源; Blocked 状态表示被请求的客体资源由于被占用而当前不可用, 访问执行操作被阻塞; Final 状态表示访问操作结束. 访问状态包括访问前 (Pre)、访问中 (Acing)、访问等待 (Wait) 以及访问结束 (Post). 在初始化时, 执行状态初始化为 Ready, 访问状态初始化为 Pre. 模型框架中执行状态和访问状态的迁移关系如图 4 所示.

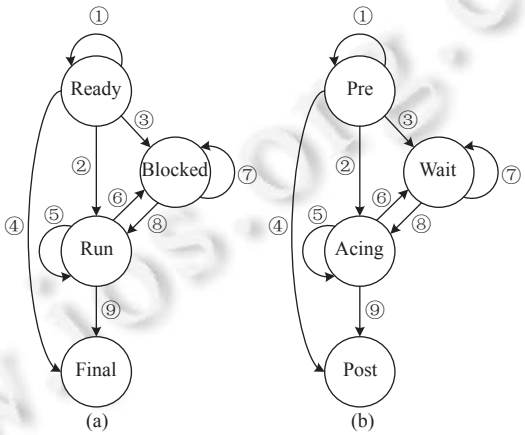


图 4 访问控制模型执行状态和访问状态迁移关系图

① 主体创建后进入的状态, 准备或正在发起访问请求, 执行状态和访问状态保持不变;

② 主体请求访问客体, 客体资源可用, 且访问请求满足访问控制策略, 则执行状态进入 Run 状态, 且访问状态进入 Acing 状态;

③ 主体请求访问客体, 访问请求满足访问控制策略但客体资源不可用, 则执行状态进入 Blocked 状态, 访问状态进入 Wait 状态;

④ 主体请求访问客体, 访问请求未满足访问控制策略, 则执行状态变为 Final, 且访问状态变为 Post, 表示访问操作结束;

⑤ 在访问过程中, 主体访问其他客体资源, 访问操作满足访问控制策略且客体资源此时可用, 则执行状态和访问状态均保持不变;

⑥ 在访问过程中, 主体访问其他客体资源, 满足访问控制策略但被访问资源不可用, 则执行状态进入 Blocked 状态, 访问状态进入 Wait 状态;

⑦ 主体访问的客体资源一直处于不可用状态, 则执行状态和访问状态保持不变;

⑧ 主体等到被访问的客体资源可用, 则执行状态进入 Run 状态, 同时访问状态进入 Acing 状态;

⑨ 在访问过程中, 主体对其他客体资源的访问请求不满足访问控制策略, 或者无其他客体资源需要访问, 则执行状态变为 Final, 访问状态变为 Post, 表示访问结束.

3.4 基本操作和性质描述

访问控制模型通过定义各个实体操作以描述系统访问控制过程中的具体行为. 对于主体实体, 除了定义主体的创建和删除操作, 还应结合可信执行环境定义主体标签域切换操作. 为提高模型的通用性, 这些操作不做任何安

全判断, 仅为执行模块提供操作接口, 安全判断在执行模块中实施. 同理, 客体实体中除了定义客体资源的初始化、创建、删除操作, 也定义了客体资源状态的更改操作以及客体标签更新操作. 规则实体中除了定义权限操作类型, 同时根据主客体的标签属性创建了访问控制策略集合和标签更新策略集合. 为提高系统安全性, 访问控制规则和标签更新规则往往不允许进行任何修改, 因此该实体没有规则的添加、删除和修改操作. 在对真实系统进行验证时, 依据系统策略文件描述这些规则即可.

访问控制执行模块描述系统执行访问控制操作的整个过程, 首先应该对系统环境进行初始化. 当处理访问请求时, 先判断请求是否满足访问控制策略, 如果满足, 则根据客体资源可用状态实施相应的操作, 并对主体的执行状态和访问状态进行更改. 此外, 执行模块还应描述系统运行过程中由于系统调用和中断等事件导致的标签域切换操作以及标签更新操作. 通过这些基本操作, 实现对系统访问过程中状态迁移的描述.

访问控制执行模块在执行实体操作进行状态转移的过程中应始终满足一些安全性质. 如: (1) 非可信域中的主体只能访问非可信域中的客体; (2) 可信域中的主体可以访问非可信域中的客体; (3) 可信域中的主体只能访问与之匹配的客体; (4) 主体只能访问处于可用状态下的客体; (5) 主体只有在运行状态时才可以访问客体等. 这些安全性质是整个运行过程中都要满足的. 通过检测这些性质的满足性, 可实现对模型功能正确性和安全性分析.

4 抽象机建模

4.1 模型基本结构

模型框架描述了各个组成部分的基本定义和主要功能, 在此基础上, 本文采用 B 方法以抽象机方式对该框架进行建模. 构造的抽象机模型包括标签抽象机 Tag、主体抽象机 Sub、客体抽象机 Obj、策略抽象机 Policy 和访问控制抽象机 MTAC. 抽象机中的关键变量和抽象机间的结构关系如图 5 所示.

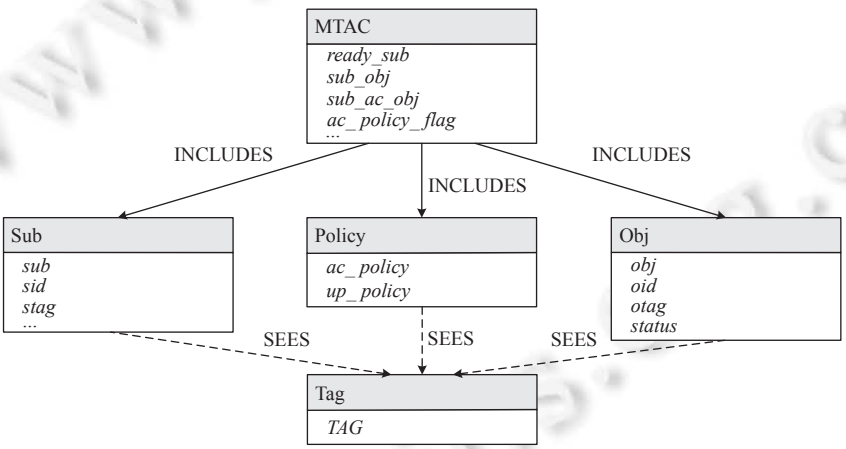


图 5 B 方法构建的访问控制模型结构

标签抽象机 Tag 描述 TEE 中的标签域集合, 标签抽象机 Sub 描述访问请求的主体, 包括主体属性和主体基本操作, 同理, 抽象机 Obj 描述访问请求的客体. 抽象机 Policy 利用主客体标签属性描述了基于标签的访问控制规则和标签更新规则. 抽象机 Sub、Obj 和 Policy 都需要引用抽象机 Tag 中定义的集合, 因此与抽象机 Tag 是 SEES 关系. 访问控制模型 MTAC 包含 (INCLUDES) Sub、Obj 和 Policy 这 3 个抽象机, 通过调用基本实体抽象机中的变量和操作形式化描述了环境初始化、隔离域切换、访问执行等具体行为.

4.2 基本实体抽象机

基本实体抽象机描述各实体组件, 定义基本的实体属性以及可调用的实体操作. 在构建的访问控制 B 模型中,

基本实体抽象机包括 Tag 抽象机、Sub 抽象机、Obj 抽象机和 Policy 抽象机, 分别描述标签域、主体、客体和策略实体对象. 其中, Tag 抽象机根据框架中描述的隔离域定义标签域集合 $TAG = \{NU, NS, TU, TS\}$, 该抽象机极为简单, 这里不再赘述, 接下来对其他 3 个抽象机模型详细阐述.

抽象机 Sub 首先定义了主体集合 (*SUB*) 以声明所有的主体, 然后描述主体属性以及基本操作 (如表 4 所示). 其中, 主体属性包括主体标识 (*sid*)、主体标签 (*stag*)、执行状态 (*exe_state*) 和访问状态 (*ac_state*), 主体操作包括创建 (*Create_Sub*)、终止 (*Delete_Sub*)、状态切换 (*Change_Sub_State*) 和标签转移 (*Change_Sub_Tag*).

表 4 主体抽象机

抽象机	主体 (Sub)
SEES	Tag
SETS	SUB, $EXE\_STATE = \{Ready, Run, Blocked, Final\}$, $AC\_STATE = \{Pre, Acing, Wait, Post\}$
VARIABLES	sub, SID, sid, $stag$, exe_state, ac_state
INVARIANT	$sub \subseteq SUB \wedge SID \subseteq \mathbb{N}_1 \wedge sid \in sub \rightarrow SID \wedge stag \in sub \rightarrow TAG \wedge$ $exe\_state \in sub \rightarrow EXE\_STATE \wedge ac\_state \in sub \rightarrow AC\_STATE$
OPERATIONS	*Create_Sub*, *Delete_Sub*, *Change_Sub_State*, *Change_Sub_Tag*

Create_Sub 操作创建新的主体并赋值主体属性, 其执行状态和访问状态分别为 Ready 和 Pre. *Delete_Sub* 操作将该主体从主体集合 *SUB* 中删除, 其映射的属性也随之删除. *Change_Sub_State* 操作根据传入的状态参数更改指定主体的执行状态和访问状态, 为访问控制模型抽象机提供操作接口. 同理, *Change_Sub_Tag* 操作更改指定主体所在的标签域并提供操作接口.

抽象机 Obj 定义客体集合 (*OBJ*) 以声明所有的客体资源, 如表 5 所示. 在访问控制过程中, 涉及的客体属性包括客体标识符 (*oid*)、客体标签 (*otag*) 和客体资源可用状态 (*status*). 对客体资源的操作包括定义初始客体资源 (*Init_Obj*)、添加客体资源 (*Add_Obj*)、删除客体资源 (*Del_Obj*)、更改客体资源可用状态 (*Change_Obj_Status*) 以及客体标签更新操作 (*Update_Obj_Tag*).

表 5 客体抽象机

抽象机	客体 (Obj)
SEES	Tag
SETS	OBJ, $STATUS = \{Free, Full\}$
VARIABLES	obj, oid, $otag$, $status$
INVARIANT	$obj \subseteq OBJ \wedge oid \in obj \rightarrow \mathbb{N}_1 \wedge otag \in obj \rightarrow TAG \wedge status \in obj \rightarrow STATUS$
OPERATIONS	*Init_Obj*, *Add_Obj*, *Del_Obj*, *Change_Obj_Status*, *Update_Obj_Tag*

客体资源可用状态包括 Free 状态和 Full 状态, 当该内存资源未被占用时, 其状态为 Free, 反之, 状态为 Full. *Init_Obj* 操作描述了系统初始化时的客体资源, 模型默认此时的客体资源包括 NU 标签、NS 标签、TU 标签和 TS 标签下的资源. *Add_Obj* 操作添加客体资源并赋值客体属性, *Del_Obj* 操作删除指定的客体资源. *Change_Obj_Status* 操作根据传入的状态参数更改客体资源可用状态, 为访问控制模型抽象机提供操作接口. 同理, *Update_Obj_Tag* 执行对指定客体资源的标签更新操作.

抽象机 Policy 描述主体对客体的访问控制规则以及标签更新规则, 通过 SEES 关系对抽象机 Tag 中的集合和变量以只读的方式引用 (如表 6 所示). 在定义权限操作集合 *OP* 时, 除了框架中描述的读、写、执行操作, 还定义 *Nop* 操作, 表示主体完成对客体的所有访问操作后, 无其他访问操作. 抽象机 Policy 定义操作 *Create_Ac_Policy* 构建访问控制策略集, 该集合包含以主体标签、客体标签以及权限操作构成的有序对, 如 $TS \mapsto TU \mapsto Read$ 表示主体在 TS 域时, 对 TU 标签域的客体允许读操作.

此外, 抽象机 Policy 也描述了框架中描述的标签更新规则. 该规则明确主体在不同标签域下对不同客体标签进行更改的权限. *Create_Up_Policy* 操作构建标签更新策略集, 该集合包含主体标签和客体标签构成的有序对, 如 TS ↦ NU 表示主体在 TS 域下, 可将某客体的标签从 TU 更新为其他标签.

表 6 策略抽象机

抽象机	策略 (Policy)
SEES	Tag
SETS	OP = {Read, Write, Exc, Nop}
VARIABLES	*ac_policy*, *up_policy*
INVARIANT	$ac\_policy \subseteq TAG \times TAG \times OP$, $up\_policy \subseteq TAG \times TAG$
OPERATIONS	*Create_Ac_Policy*, *Create_Up_Policy*

4.3 访问控制抽象机 MTAC

访问控制抽象机模型 MTAC 描述主体请求客体资源、规则检查、资源访问成功与失败的完整过程. 如图 6 所示, 抽象机 MTAC 以 INCLUDES 方式包含 Sub, Obj 和 Policy 这 3 个基本实体抽象机, 实现对这 3 个抽象机中定义的变量和封装的操作的调用.

MACHINE MTAC

INCLUDES Sub, Obj, Policy

SETS

DT={SYS, SYS_EXIT, TSYS, TSYS_EXIT, TUENTER, TULEAVE, TSENTER, TSLEAVE, NULL}

VARIABLES

ready_sub, *run_sub*, *blocked_sub*, *sub_obj*, *sub_ac_obj*, *ac_flag*, *ac_policy_flag*, *sub_next_obj*, *next_obj*

ac_event, *dtrans*,…

INVARIANT

$ready\_sub \subseteq sub \wedge run\_sub \subseteq sub \wedge blocked\_sub \subseteq sub \wedge sub\_obj \in sub \leftrightarrow obj \wedge$

$sub\_ac\_obj \in sub \nrightarrow obj \wedge ac\_flag \in 0..1 \wedge ac\_policy\_flag \in 0..1 \wedge sub\_next\_obj \in sub \nrightarrow obj \wedge$

$ext\_obj \in sub \nrightarrow obj \wedge ac\_event \in OP \wedge dtrans \in DT \wedge \ldots$

INITIALISATION

ready_sub := {}

ac_flag,*ac_policy_flag* := 0,0||*proc_next_mem*:={}||*ac_event* := *Nop*||*dtrans* := NULL||…

OPERATIONS

Init_Env(*ss*, *sid*1, *oo*1, *oid*1, *oo*2, *oid*2, *oo*3, *oid*3, *oo*4, *oid*4); *Domain_Trans*(*ss*, *dt*);

Update_Tag(*ss*, *oo*, *tt*); *Judge_Ac_Policy*(*ss*, *oo*, *ac*); *Judge_Status*(*ss*,*oo*); *Sub_Ac_Obj*(*ss*);

Quit_Ac_Obj(*ss*);

图 6 MTAC 抽象机

模型 MTAC 定义隔离域切换事件集合 *DT*, 其中, SYS 表示从 NU 切换到 NS 的可能事件如系统调用、中断、异常等, SYS_EXIT 表示从 NS 切换回 NU. 同理, TSYS 和 TSYS_EXIT 表示发生从 TU 切换到 TS 以及返回的事件. TUENTER 和 TULEAVE 表示从 NU 域进入 TU 域以及从 TU 域进入 NU 域的事件, TSENTER 和 TSLEAVE 分别表示从 NS 进入 TS 和从 TS 进入 NS 的事件, NULL 表示无隔离域切换事件. 为记录主体的执行状态, 该模型定义主体状态集合, 包括就绪集合 (*ready_sub*)、阻塞集合 (*blocked_sub*)、以及正在运行的主体的集合 (*run_sub*),

其中 *blocked_sub* 记录的是因得不到请求的资源而无法执行访问操作的主体. *sub_obj* 记录与主体绑定的客体, *sub_ac_obj* 和 *sub_next_obj* 分别表示进程正在进行访问和下一个已准备访问的内存资源, *next_obj* 表示下一个请求访问的内存资源. *ac_policy_flag* 表示当前主体对客体的访问是否满足访问控制策略, 而 *ac_flag* 表示主体对客体的访问是否成功.

为描述系统访问控制的完整过程, 模型 MTAC 定义的操作主要分为 4 类: 初始化操作、域切换操作、标签更新操作以及访问执行操作. 初始化操作是对系统环境的初始化 (*Init_env*), 包括调用抽象机 Sub 中的 *Create_Sub* 操作创建初始客体、调用抽象机 Obj 中的 *Init_Obj* 操作进行客体资源初始化、调用 Policy 中的 *Create_Ac_Policy* 和 *Create_Up_Policy* 声明访问控制规则和标签更新规则等. 域切换操作 (*Domain_Trans*) 根据输入参数 *dt* 表示的域切换事件执行相应的切换, 标签更新操作 (*Update_Tag*) 根据标签更新规则决定是否将客体 *oo* 的标签更新为 *tt* .

在执行访问控制过程中, 当主体请求访问某一客体资源时, 首先根据 *Judge_Ac_Policy* 操作检查是否满足访问控制策略, 包括是否符合访问控制规则且客体与主体绑定, 如果满足, 变量 *ac_policy_flag* 置 1, 且将该客体资源添加到 *sub_next_obj* 有序对中表明将要对其访问; 否则, 主体的执行状态直接进入 Final 状态, 访问状态进入 Post 状态, 表示访问结束. 在通过访问策略检查后, 执行 *Judge_Status* 操作, 该操作检查被请求的客体资源是否可用并进行相应的操作 (如图 7 所示). 该操作传入的参数 *ss* 和 *oo* 分别表示访问的主体和客体. PRE 是操作执行的前置条件, THEN 描述操作行为. 前置条件保证主体和客体资源在模型中有定义且该客体正是主体要请求访问的资源, 同时, 访问请求满足访问控制规则. 若客体资源可用, 则主体进入运行集合 *run_sub*, 变量 *ac_flag* 置 1 表示主体正在访问该客体资源, 主体的执行状态变为 Run 且访问状态变为 Acing, 该主体请求的资源变为已准备状态; 否则, 主体进入阻塞集合, 执行状态和访问状态分别为 Blocked 和 Wait, 同时将该进程正在访问和接下来要访问的内存资源从队列中清除. 操作 *sub_ac_obj* 和 *quit_ac_obj* 分别描述访问过程中客体资源可用状态的改变以及访问结束后相应的行为.

Judge_Status(*ss*, *oo*)=
PRE
ss∈ *sub* ∧*oo* ∈ *obj* ∧*ss* ↦*oo* ∈ *next_obj* ∧
exe_state(*ss*)∈{Ready, Run, Blocked}∧*ac_flag*=0∧*ac_policy_flag*=1
THEN
IF
status(*oo*)=Free
THEN
ready_sub:=*ready_sub*−{*ss*}||*run_sub*:=*run_sub*∪{*ss*}||
blocked_sub :=*blocked_sub*−{*ss*}||*ac_flag* :=1||
Change_Sub_State(*ss*, Run, Acing)||*sub_next_obj*(*ss*) :=*next_obj*(*ss*)||
next_obj :={}
ELSE
ready_sub := *ready_sub* −{*ss*}||*run_sub* := *run_sub*−{*ss*}||
blocked_sub :=*blocked_sub*∪{*ss*}||*Change_Sub_State*(*ss*, Blocked, Wait)||
sub_next_obj：={*pp*}◂*sub_next_obj*||*sub_ac_obj*：={*pp*}◂*sub_ac_obj*
END
END;

图 7 MTAC 中定义的 *Judge_Status* 操作

此外, 模型 MTAC 还通过描述不变式约束对访问控制过程中必须要满足的基本性质进行规约. 这些基本性质是系统运行在任何状态时都要保持的. 针对 TEE 中基于内存标签的访问控制机制, 本文主要定义以下性质.

性质 1. 已建立的访问关系, 必须是访问控制规则中明确说明的:

$$\forall(ss,oo).(ss \in sub \wedge oo \in obj \wedge oo = sub\_ac\_obj(ss) \Rightarrow stag(ss) \mapsto otag(oo) \in dom(ac\_policy)).$$

其中, $policy \subseteq sub \times obj \times op$, 表示访问规则允许 sub 对 obj 执行 op 访问操作.

性质 2. 主体只能访问上下文匹配的客体资源:

$$\forall ss.(ss \in sub \Rightarrow sub\_ac\_obj(ss) \in ran(\{ss\} sub\_obj)).$$

性质 3. 主体只能访问未被占用的客体资源:

$$status[sub\_next\_obj[run\_sub]] \subseteq \{\text{Free}\}.$$

性质 4. 访问规则表中的主体、客体标签以及权限操作, 必须与系统保持一致:

$$dom(dom(ac\_policy)) \subseteq TAG \wedge ran(dom(ac\_policy)) \subseteq TAG \wedge ran(ac\_policy) \subseteq OP.$$

5 模型实例化

利用构建的抽象机模型, 本文对 TIMBER-V 中的访问控制过程进行形式化描述并验证. TIMBER-V 是一个标签化的开源内存架构, 利用标签设计访问控制规则实现对代码和数据灵活、有效的隔离保护, 同时在 RISC-V 模拟器上进行了概念验证. 结合 TIMBER-V 的具体实现, 本文在提出的抽象机模型上进行扩展, 包括增加新的变量和操作, 给出 TIMBER-V 访问控制模型实例.

5.1 基本实体抽象机

在 TIMBER-V 的访问控制模型实例中, 进程作为发起访问请求的主体, 内存作为客体资源, 因此本文将模型中的抽象机 Sub 和 Obj 分别修改为进程抽象机 Proc 和内存抽象机 Mem. 此时, 主体标签表示进程运行在的特权模式 (或特权级), 客体标签表示划分的内存虚拟地址空间范围. 由于主体标签和客体标签不一致, 为简化模型结构, 本文不再定义标签抽象机, 将标签属性各自在抽象机 Proc 和 Mem 中描述. 因此, 模型实例中的基本实体抽象机包括 Proc 抽象机、Mem 抽象机和 Policy 抽象机.

抽象机 Proc 的描述和抽象机 Sub 基本相同, 同样定义进程集合 (*PROC*) 以及进程的属性, 此外, 定义进程运行的特权模式集合 $MODE = \{\text{NU},\text{NS},\text{TU},\text{TS},\text{MM}\}$. TIMBER-V 是针对 RISC-V 提出的标签隔离架构, RISC-V 架构定义了 3 种特权模式, 即用户模式 (U)、监管者模式 (S) 和机器模式 (M). 其中, M 模式具有最高特权级, 该模式下的所有操作都认为是可信的; U 模式是最低级别, 主要执行用户应用程序; S 模式介于两者之间, 运行操作系统. TIMBER-V 在此基础上划分了 4 种运行模式: N 域模式、TU 模式、TS 模式以及 M 模式. TIMBER-V 认为用户应用程序和操作系统是不可信的, 因此把它们划分到 N 域, 该域支持传统的 U 模式和 S 模式的分离. 在用户内存空间创建的可信隔离域 (即 enclaves) 运行在可信用户模式 (TU 模式) 下用于保护敏感数据和代码. 同时, TIMBER-V 创建一个可信管理器 TagRoot, TagRoot 运行在可信监管者模式 (TS 模式) 下为 enclaves 和不可信的操作系统提供可信服务. M 模式和 RISC-V 架构的机器模式保持一致. 抽象机 Proc 将进程所处的运行模式抽象为 NU 模式、NS 模式、TU 模式、TS 模式以及 MM 模式, NU 和 NS 分别表示 N 域中的用户模式和监管者模式, MM 模式表示 TIMBER-V 中的 M 模式. 同时, 进程操作也包括创建 (*Create_Proc*)、终止 (*Delete_Proc*)、状态切换 (*Change_Proc_State*) 和模式切换 (*Change_Proc_Mode*).

抽象机 Mem 是对抽象机 Obj 的实例化, 该抽象机定义内存资源集合 (*MEM*) 和内存属性包括资源标识符 (*mid*)、内存标签 (*mtag*) 和资源可用状态 (*mstatus*). 由于 TIMBER-V 在每 32 位内存字上使用 2 位作为标签位来定义 4 类内存标签, 即 N 标签、TU 标签、TS 标签和 TC 标签, 分别标记不可信内存地址空间 (N 域)、可信用户内存地址空间 (TU 域)、可信监管者内存地址空间 (TS 域) 和进入 TU 域或 TS 域的安全入口点. 因此抽象机 Mem 定义内存标签集合 $TAG = \{N\_tag, TC\_tag, TU\_tag, TS\_tag\}$. 同时, 对内存资源的操作也包括定义初始内存 (*Init_Mem*)、添加内存资源 (*Add_Mem*)、删除内存资源 (*Del_Mem*)、更改资源可用状态 (*Change_Mem_Status*) 以及标签更新操作 (*Tag_Updating*).

抽象机 Policy 通过 SEES 关系对抽象机 Proc 和 Mem 中的集合和变量以只读的方式引用, 描述进程对内存资

源的访问控制规则以及标签更新规则 (如表 7 所示). 除了读、写、执行操作, 该抽象机定义了 *Enter* 和 *Leave* 操作. TIMBER-V 中, 运行在 N 域的进程通过 TC 标签入口点进入 TU 模式或者 TS 模式, 因此进程在 N 域模式下对安全入口点具有 *Enter* 权限操作. 同理, 进程在 TU 模式下和 TS 模式下可直接对 N 标签内存执行权限操作 *Leave* 实现到 N 域模式的切换. 该抽象机定义的操作与原策略抽象机类似, *Create_Ac_Policy* 操作描述 TIMBER-V 中实现的访问控制策略集, *Create_Up_Policy* 操作描述 TIMBER-V 中实现的标签更新策略集, 同样以有序对进行描述.

表 7 TIMBER-V 策略抽象机

抽象机	Policy
SEES	Proc, Mem
SETS	OP = {Read, Write, Exc, Enter, Leave, Nop}
VARIABLES	ac_policy, up_policy
INVARIANT	$ac\_policy \subseteq MODE \times TAG \times OP$, $up\_policy \subseteq MODE \times TAG$
OPERATIONS	*Create_Ac_Policy*, *Create_Up_Policy*

5.2 访问控制模型

访问控制模型 MTAC_TIV 形式化描述 TIMBER-V 系统中进程对内存资源的具体访问行为. MTAC_TIV 同样以 INCLUDES 方式包含 Proc, Mem 和 Policy 这 3 个基本实体抽象机. 同时, MTAC_TIV 模型中还定义了进程队列状态、enclave 控制块 (ECB) 信息、共享内存 (shm) 信息等变量以及进程访问相关操作.

模型 MTAC_TIV 除了实例化抽象机 MTAC 中定义的变量和操作, 如进程状态队列变量、记录进程访问内存信息的变量以及环境初始化、访问执行等操作外, 还扩展了新的变量和操作以描述 TIMBER-V 中具体的访问控制过程. 其中一些关键变量和操作如表 8 所示. TIMBER-V 中, 可信区域 (TU 域和 TS 域) 由称为 enclave 控制块 (ECB) 的数据结构标识, 该数据结构保存在 TS 标签内存中, 因此在模型中定义变量 *ecb_mem* 表示 ECB 对应的内存信息. 同时, ECB 数据结构中用于唯一识别可信区域的 EID 和记录 enclave 运行状态的 state 分别用变量 *ecb_eid* 和 *ecb_state* 表示. 此外, TIMBER-V 支持共享内存 (shm) 作为一种快速灵活的 enclave 间通信方法. 共享内存的提供者和接受者通过该内存的 EID 标识唯一确认, 因此模型中定义 *shm_ecb_mem* 表示共享内存区域资源, 定义 *shm_ecb_eid* 表示接收该共享内存区域的目标 enclave 的标识. 此外, TIMBER-V 利用内存保护单元 (memory protection unit, MPU) 实现同一个可信区域内进程的隔离, 在模型 MTAC_TIV 中, 通过定义变量 *proc_mem* 表示每个进程在 MPU 中装载的内存资源.

结合 TIMBER-V 系统实际运行情况, 模型 MTAC_TIV 定义的操作除了包含抽象机 MTAC 中描述的初始化操作、域切换操作、标签更新操作以及访问执行操作, 还扩展了可信服务操作. 如前文所述, TIMBER-V 开发了一个运行在 TS 模式下的可信管理器 TagRoot, enclave 由操作系统在 TagRoot 的帮助下在普通用户进程中创建和加载. TagRoot 提供可信的操作系统服务以及可信的 enclave 服务. ECB 通过 *Create_Enclave* 操作创建, 当创建一个新的 enclave 后, 操作系统调用 *Add_Region* 向 enclave 中添加内存区域. 同时通过 *Add_Data* 和 *Add_Entries* 操作添加数据和安全入口点. 当执行初始化操作 *Init_Enclave* 后, enclave 的状态标记为可运行, 当 enclave 即将运行时, 调用 *Load_Enclave* 操作将 enclave 进行装载. 运行结束后, *Destroy_Enclave* 和 *Resume* 分别执行 enclave 销毁和上下文恢复操作, 该模型通过定义这些操作描述了 TIMBER-V 中 enclave 的整个生命周期过程. 同时, TIMBER-V 支持 enclave 间共享内存, 模型 MTAC_TIV 定义的 *Shm_offer* 、 *Shm_accept* 、 *Shm_release* 操作同样描述了这一过程. 值得注意的是, 模型 MTAC_TIV 在进行环境初始化时, 初始的内存资源不包括 TU 标签 (即 enclave) 内存, 该内存资源在执行上述操作创建 enclave 后才存在. 同时, 该模型将 MTAC 中描述域切换的操作 *Domain_Trans* 进行了扩展, 包括定义进入 MM 模式以及从 MM 模式退出的操作.

表 8 抽象机 MTAC_TIV 中扩展的关键变量及操作

类型	名字	描述
变量	*cur_mode*	进程当前运行在的特权模式
	ecb	enclave控制块集合
	eid	enclave控制块标识集合
	ecb_eid	enclave控制块中的标识, 以⟨*ecb*,*eid*⟩二元组表示
	ecb_mem	enclave控制块描述的内存, 以⟨*ecb*,*mem*⟩二元组表示
	ecb_state	enclave控制块描述的资源状态
	shm_ecb	enclave控制块中记录共享内存区域信息的数据结构的集合
	shm_ecb_mem	enclave控制块中描述的共享内存区域, 以⟨*shm_ecb*,*mem*⟩二元组表示
	shm_ecb_eid	enclave控制块中描述的接收共享内存的目标enclave的标识
操作	*Create_Enclave*	创建enclave控制块
	Add_Region	向enclave添加内存区域 (连续的内存地址)
	Add_Data	存放enclave数据和应用程序数据
	Add_Entries	为创建的enclave添加安全入口点
	Init_Enclave	初始化enclave
	Load_Enclave	装载enclave
	Destroy_Enclave	卸载enclave, 释放enclave区域并清除控制块内存
	Resume	恢复上下文
	Shm_Offer	enclave向目标enclave提供部分共享内存区域的服务
	Shm_Accept	目标enclave接收共享内存区域
	Shm_Release	目标enclave释放共享内存区域

6 模型检测与安全性分析

本节针对 TIMBER-V 的访问控制机制, 提出基于模型检测的安全性分析方法. 该以不变式规约安全性质并在模型检测工具 ProB 中验证功能正确性和安全性, 然后模拟攻击场景, 根据模型检测结果实现对 TIMBER-V 访问控制机制的安全性分析.

6.1 安全性质描述

模型检测是通过遍历系统的状态空间验证模型是否满足给定的性质. 在构建访问控制模型后, 本文分析 TIMBER-V 中访问控制过程中需要满足的安全性质, 并通过不变式约束描述安全性质. 在检测过程中, 除了检测死锁和不变式间冲突的发生, 还会检查每个系统状态下是否满足不变式约束. 除了第 4.3 节描述的基本性质, 本文针对 TIMBER-V 添加的主要不变式关系如下.

① 所有进程对内存的访问都必须是访问控制规则允许的.

$$\forall(pp,mm).(pp\in dom(proc\_ac\_mem)\wedge mm\in mem\wedge mm=proc\_ac\_mem(pp)\Rightarrow pmode(pp)\mapsto mtag(mm)\in dom(ac\_policy)).$$

② 对任意一个正在访问且运行在 NU 或 TU 模式下的进程, 其访问的内存一定在该进程分配的资源中.

$$\forall pp.(pp\in run\_proc\wedge pmode(pp)\in\{\mathrm{NU},\mathrm{TU}\}\Rightarrow(proc\_ac\_mem(pp)\in ran(\{pp\}\triangleleft proc\_mem))).$$

③ 对任意一个正在访问且运行在 NU 模式下的进程, 其访问的内存一定属于 N 标签.

$$\forall pp.(pp\in run\_proc\wedge pmode(pp)=\mathrm{NU}\Rightarrow(mtag[proc\_ac\_mem[pp]]\subseteq N\_tag)).$$

④ 对任意一个正在访问且运行在 NS 模式下的进程, 其访问的内存一定属于 N 标签.

$$\forall pp.(pp\in run\_proc\wedge pmode(pp)=\mathrm{NS}\Rightarrow(mtag[proc\_ac\_mem[pp]]\subseteq\{N\_tag\})).$$

⑤ 对任意一个正在访问且运行在 TU 模式下的进程, 其访问的内存一定属于 TC, TU 和 N 组成的标签集合.

$$\forall pp.(pp \in run\_proc \wedge pmode(pp) = \text{TU} \Rightarrow (mtag[proc\_ac\_mem[pp]] \subseteq \{TC\_tag, TU\_tag, N\_tag\})).$$

⑥ 对任意一个正在访问且运行在 TS 模式下的进程, 其访问的内存一定属于 TC, TS, TU 和 N 组成的标签集合.

$$\forall pp.(pp \in run\_proc \wedge pmode(pp) = \text{TS} \Rightarrow (mtag[proc\_ac\_mem[pp]] \subseteq \{TC\_tag, TS\_tag, N\_tag, TU\_tag\})) \ .$$

⑦ 正在被访问的内存资源, 其当前可用状态一定为被占用 (Full).

$$mstatus[proc\_ac\_mem[run\_proc]] \subseteq \{\text{Full}\}.$$

⑧ 进程一定且只能处于就绪、阻塞、运行队列中的一个.

$$ready\_procs \cap run\_proc = \varnothing \wedge ready\_procs \cap blocked\_proc = \varnothing \wedge ready\_procs \cup run\_proc \cup blocked\_proc = proc.$$

⑨ 在当前的访问关系表中, 进程状态均为“访问中”.

$$dom(proc\_ac\_mem) \subseteq ac\_state^{-1}[\{Acing\}],$$

其中, *ready_procs*, *run_proc*, *blocked_proc* 分别是进程就绪队列、运行队列以及阻塞队列的集合, *proc_mem* 描述进程分配的内存资源, *proc_ac_mem* 描述进程正在访问的内存资源, *pmode* 描述进程运行在的特权模式, *mtag* 描述内存的标签属性, *mstatus* 描述内存可用状态, *ac_state* 描述进程访问状态.

6.2 模型检测结果分析

本文利用 B 方法集成工具 ProB 对 TIMBER-V 访问控制模型进行模型检测. 实验所用硬件平台为 Intel Xeon E5-2678 v3, 32 GB DDR4, 软件环境为 Ubuntu-16.04.7, ProB-1.10.0. 在载入模型后, 对规模较小的进程、内存资源和策略基本实体抽象机采用混合的深度优先和广度优先算法 (mixed depth first and breadth first, Mixed DF/BF) 搜索状态空间. 抽象机 MTAC 由于包含了其他两个抽象机, 且定义的变量和转移操作较多, 因此状态空间规模较大. 为了在状态空间中遍历系统模型的原定义操作, 对抽象机 MTAC_TIV 采用广度优先算法保证覆盖所有操作. 各抽象机模型的检测结果如表 9 所示.

表 9 TIMBER-V 访问控制模型检测结果

抽象机	独立状态数	总迁移数量	检测时间 (s)
Proc	102 618	689 415	84
Mem	29 682	488 132	57
Policy	134 849	2 639 221	333
MTAC_TIV	14 699 098	37 834 011	39 504

在每个抽象机中, 所有变量的当前值决定一个独立状态, 当任何变量的值发生改变时, 则状态也发生改变. 状态根据相关操作改变到另一个状态的过程称为状态迁移. ProB 对模型中所有的抽象机进行了检测, 在设置抽象集合元素个数以及最大操作数量后, 可自动遍历独立状态和迁移关系. 结果表明, 所有的抽象机均通过了检测, 无死锁和不变式冲突发生.

在这些抽象机中, MTAC_TIV 抽象机规模最大, 定义的操作最多, 利用广度优先算法进行模型检测, 得到部分状态空间图 (生成的状态图均为矢量图, 文件较大, 请在 GitHub 上查阅 https://github.com/mxlaga/Memory-tag-based-access-control-models.git). 由于规模限制, MTAC_TIV 生成的状态空间图中包含 12 730 个独立状态和 32 763 个迁移, 其中已检查的节点显示为绿色. 由于 MTAC_TIV 中变量多, 因此模型中的状态节点规模较大, 同时广度遍历的访问、域切换等操作多次改变模型变量值, 造成节点间迁移复杂.

6.3 攻击场景模拟与安全性分析

为实现访问控制模型进一步地安全性分析, 本文构建威胁模型并进行攻击场景模拟, 验证访问控制模型的安全性. 本文分别描述了两个敌手行为, 第 1 个场景是攻击者利用不可信的应用程序对其划分的 enclave 区域进行访问, 执行的步骤如图 8 所示.

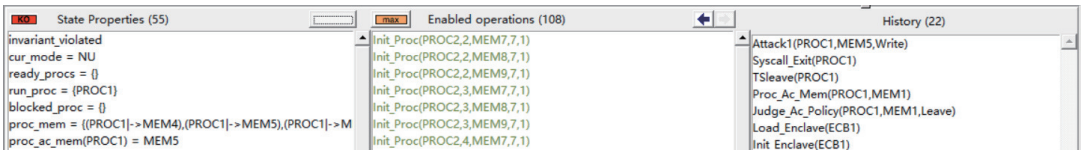


图 8　攻击场景 1 的状态及历史操作

图 8 中右栏记录了系统执行的历史操作, 中间栏表示当前可执行的操作, 左栏表示当前的系统状态. 从图中可以看出, 当系统模型在执行环境初始化、可信区域 enclave 创建等操作后, 此时的进程 PROC1 运行在 NU 特权模式下, 且内存 MEM5 的标签为 TU. 当执行 Attack1 操作即 PROC1 对 MEM5 进行写操作时, 检测发生终止, 同时图的左上角变为红色, 表示有冲突发生.

如图 9 所示, 对于第 1 个攻击操作, 模型检测结果表明其违反了第 6.1 节中的第 1 个安全性质约束, 该访问操作无法获得设计的访问控制规则的允许. 其生成的状态空间图如图 10 所示 (可放大查阅, 下同), 当执行第 22 个操作即 Attack1 操作时, 系统进入坏死的状态 (红点所示), 这进一步说明了该模型使场景一的攻击操作失效.

```
!pp.(pp : PROC1 => !mm.(mm : mem & mm = proc_ac_mem(pp) => pmode(pp) |-> mtag(mm) : dom(ac_policy)))
  == false
```

图 9　攻击场景 1 检测的冲突结果

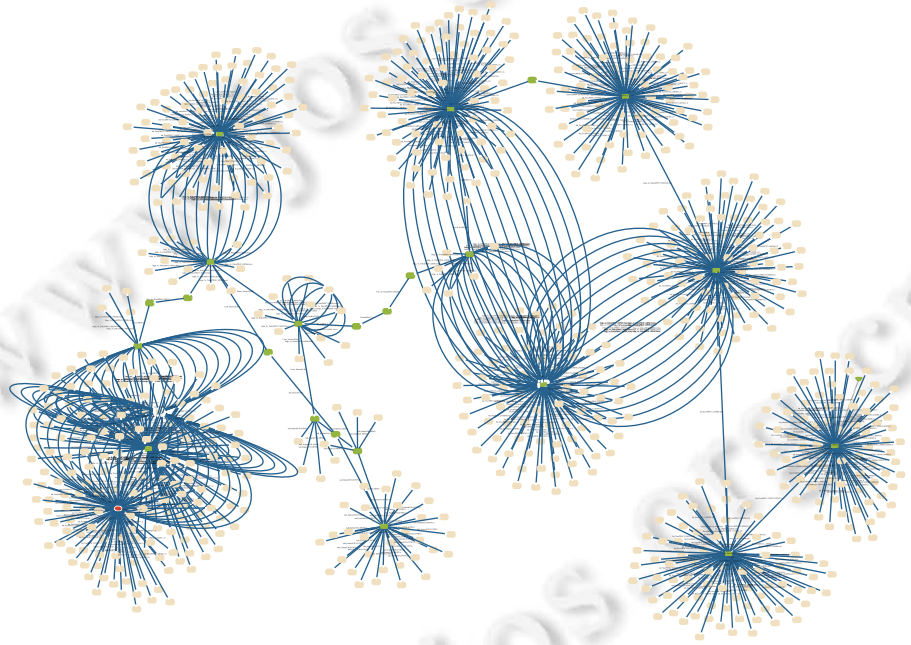


图 10　攻击场景 1 的当前状态空间

第 2 个攻击场景是假设攻击者对操作系统具有所有权, 通过构造有害的 enclave 来破坏良好的 enclave. 此时, 有害的 enclave 和良好的 enclave 属于不同的进程空间. 模拟执行的步骤如图 11 所示.

该攻击模拟执行了 45 个操作, 在经过前期的初始化、enclave 创建等操作后, 内存 MEM8 分配给进程 PROC2 且标签为 TU, 表示 PROC2 下的 enclave 内存区域, 进程 PROC1 运行在 TU 特权模式下. 当执行 Attack2 操作后, 即 PROC1 对 MEM8 执行访问操作, 检测发生终止.

如图 12 所示, 对于第 2 个攻击操作, 模型检测结果表明其违反了第 4.3 节中描述的性质 2, 即主体只能访问上下文匹配的客体资源. 生成的状态空间如图 13 所示, 在执行 44 个操作后, 当执行 Attack2 操作时, 由于其违反不变式约束, 系统进入坏死的状态.

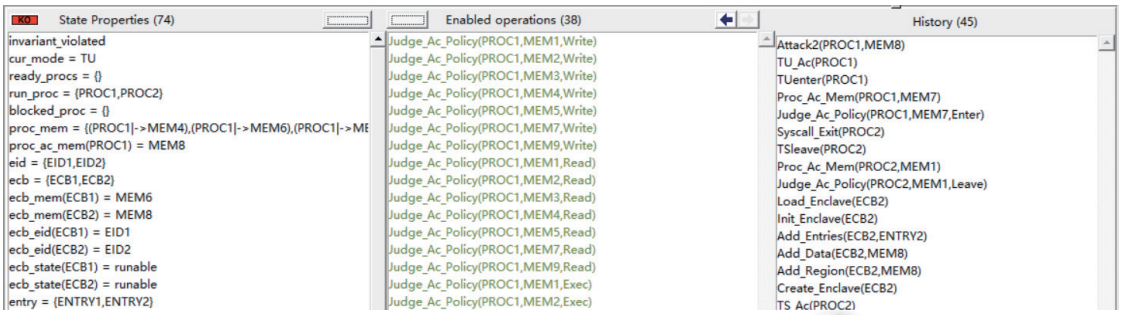


图 11　攻击场景 2 的状态及历史操作

```
!pp.(pp = PROC1 => proc_ac_mem(pp) : ran({pp} <| proc_mem))
 == false
```

图 12　攻击场景 2 检测的冲突结果

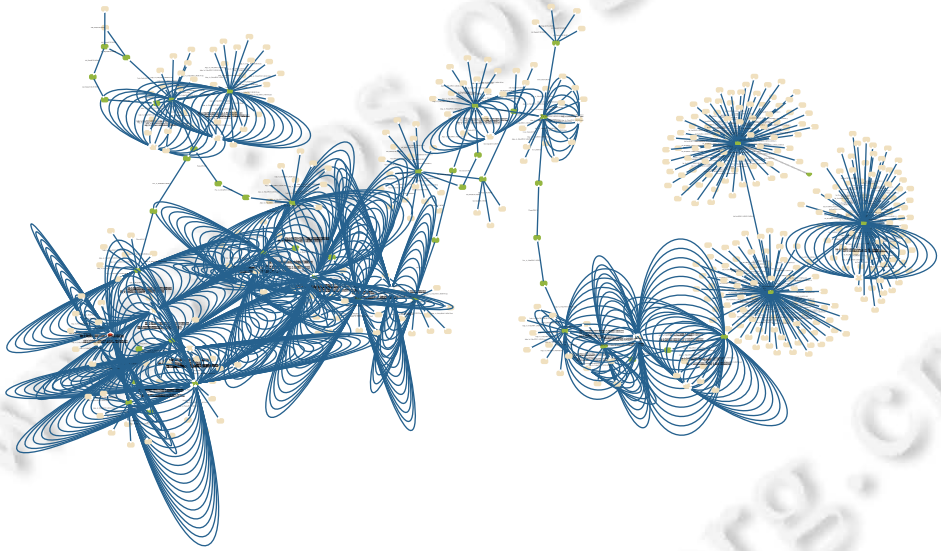


图 13　攻击场景 2 的当前状态空间

通过对分析模拟的攻击场景检测结果可知, 构建的 TIMBER-V 访问控制模型可以成功检测出攻击操作违反的不变式约束. 这证明 TIMBER-V 实现的基于内存标签的访问控制机制可保证良好的 enclave 免遭来自不可信应用程序和其他 enclave 的攻击, 提高了 enclave 中敏感数据和代码的安全保护.

7　讨论及未来工作

本文提出基于内存标签的 TEE 访问控制通用模型框架 MTF, 通过抽象机建模构建访问控制模型 MTAC, 并以 TIMBER-V 为应用案例进行模型实例化, 利用模型检测实现安全性分析. 本文采用形式化 B 方法完成访问控制建模和安全性分析, 相比于其他模型检测工具如 NuSMV 和 SPIN, B 方法主要有以下特点: (1) 规约语言可读性高, 操作规约可通过伪代码表示, 支持描述包含更多细节的具体行为, 更接近具体实现, 表达能力强; (2) 其结构在规约中支持模块化, B 方法以抽象机形式将系统分为各个子系统进行规约, 可构建不同层次的模型, 同时更细粒度表示模块间复杂的关系; (3) 集成工具功能丰富, B 方法模型检测工具 ProB 不仅支持 B 语言不变式描述, 也支持 LTL、CTL、CSP-M 表达式形式规约, 此外, 其模型检测结果生成的状态空间图可更直观地分析检测路径, 可追溯性强,

同时给出检查的状态节点数以及状态转移数, 以提供严格的检测结果分析.

模型框架 MTF 和抽象机模型 MTAC 可适用于不同架构下的 TEE 方案, 除了文中针对 TIMBER-V 实现的实例化模型, 其他如 ARM TrustZone 系列、AMD SEV 系列以及基于 RISC-V 的 PENGLAI, Keystone 等 TEE 方案均支持实例化. 在模型实例化时, 需要根据实例化对象的特征对抽象机模型进行修改和扩展, 通常需要修改初始化环境、标签更新等操作, 并扩展隔离域切换等系统操作. 如针对 TrustZone 进行实例化时, 其内存标签可以直接采用通用模型中的分类, 即 NU、NS、TU、TS 分别表示非可信应用、普通世界内核、可信应用和安全世界内核, 环境初始化和标签更新需要根据 TrustZone 具体实现进行描述, 此外, 应扩展中断和切换指令引起的安全域切换操作. 针对 Keystone 进行实例化时, 主体内存标签需要进一步扩展以表示机器模式下的安全监控器 (security monitor, SM), 其他操作如安全域切换操作和可信 OS 服务操作类似于 TIMBER-V 进行相应的修改和扩展, 以满足 Keystone 具体实现的访问控制模型构建. 此外, 本文提出的框架和模型可以用于大规模访问控制系统的验证, 但由于系统性能的限制, 通常扩展到大规模访问控制系统验证时, 在基于标签的访问控制系统建模过程中不会对主客体每个具体资源设计唯一标签, 而是将所有资源划分为若干不同的安全级别以添加标签属性. 当利用模型框架 MTF 和抽象机模型 MTAC 对该访问控制系统进行实例化时, 仅需要关注不同标签域下的访问控制机制并扩展相关安全操作, 极大缩减了模型检测的状态空间. 未来的工作主要在两个方面: 一是对模型结构进一步优化并设计合理的检测算法, 以支持更丰富的操作和更多状态变量下的模型的检测; 二是继续扩展模型, 实现对更多场景下如网络多隔离域基于标签的访问控制的建模与安全性分析.

8 相关工作

8.1 内存标签

内存标签技术在信息流跟踪[31–33]、数据隔离[34]和访问控制[35–38]中的研究由来已久. 如 Loki[38]设计字级别的内存标签机制, 通过对物理内存实现更细粒度的访问控制, 可以在内核受损的情况下对应用程序进行安全保护, CHERI[37]利用内存标签保护内存中存储的功能信息, 从而实现了细粒度的内存保护和高度可扩展的软件分区. 近年来, 随着 TEE 可信区域大小及数量受限等问题的日益突出, 利用内存标签实现更细粒度的内存隔离成为有效的缓解途径. AMD SEV-SNP[25]通过在 guest 页表中预留 1 比特位来标记内存页是否共享实现页表级的内存隔离. Weiser 等人[4]提出的 TIMBER-V 利用每个内存字上的最高 2 位设置标签实现页表级细粒度、灵活的隔离边界, 同时利用 MPU 实现进程隔离, 当执行内存访问时, 只有标签检查和 MPU 隔离检查均通过才允许. 夏虞斌老师团队[26]同样基于 RISC-V 架构提出 PENGLAI 可信执行环境, 为实现细粒度、灵活的内存隔离, 他们利用所有权位图记录每个物理页的状态, 并引入硬件原语 guarded page table (GPT) 实现所有权检查, 最终实现页表级内存隔离. 这些基于内存标签的访问控制方案都是通过测试或者经验主义的分析来证明其有效性, 缺乏严格、强力的保证. 本文利用形式化方法研究基于内存标签的 TEE 访问控制机制, 构建访问控制抽象机模型并利用基于严格数学证明的模型检测工具进行安全性分析, 对实现的访问控制机制的正确性和安全性提供严格的保证.

8.2 访问控制形式化分析

利用形式化方法对应用系统或其中实现的某个机制进行建模和安全性分析成为保证系统安全的一种重要手段[39–45]. 袁春阳等人[46]利用 Z 语言提出基于状态的 RBAC 形式化模型, 通过描述应满足的约束条件并证明, 实现对 RBAC 系统的验证. Cheng 等人[47]提出一种结合 UML 和模型检测技术对安全策略模型进行形式化验证的方法, 该方法利用状态机图和类图描述访问控制模型并验证了其对于安全需求的满足性. Hughes 等人[39]提出一种可自动化验证访问控制策略性质的方法, 该方法通过提取基于可扩展访问控制标记语言 (XACML) 描述的访问控制策略语义, 将其转换为数学模型, 通过将定义的访问控制策略间的偏序关系转化为布尔可满足性问题并利用 SAT 求解器实现自动化验证. Zahoor 等人[48]基于事件演算提出基于属性的访问控制形式化模型, 并利用该模型对 AWS 中的身份和访问管理策略进行形式化描述, 通过验证策略内及策略间的冲突实现安全性分析.

随着多隔离环境的广泛应用, 越来越多的工作对其中的访问控制进行形式化描述与验证[49–52]. 曹进等人[53]针

对 SaaS 系统数据模型安全性低且数据隔离差的特点, 提出基于安全标签的访问控制模型, 该模型给访问主体和客体分配安全标签并给出形式化定义, 通过实验分析了其对租户之间以及租户内部数据的强隔离性. Unal 等人[54–56]针对多域移动网络环境提出一种变异的基于角色的访问控制模型 FPM-RBAC, 该模型支持对访问所需的移动性和位置约束进行形式化描述, 同时利用提出的时空模型检测算法分别对移动性和位置约束的满足性进行形式化验证. Alam 等人[57]提出一个跨租户的访问控制形式化模型, 该模型设计 4 个算法处理访问请求激活, 并通过 Z 语言形式化分析了算法的正确性. Ren 等人[58]针对 ARM TrustZone 隔离环境提出可扩展的访问控制 B 模型, 并通过定理证明和模型检测相结合的方式分析了其安全性. 和这些工作一样, 本文提出的模型框架和实现的抽象机模型也是针对多隔离域环境下的访问控制, 但本文更关注可信执行环境下的内存访问以及基于内存标签实现的隔离机制, 同时该模型框架的通用性更高, 适用于 ARM TrustZone、AMD SEV 以及基于 RISC-V 的 TEE 等实现方案. 此外, 本文还通过模拟攻击场景实现了一个应用案例的安全性分析.

9 总 结

为增强可信执行环境本身的安全, 本文采用形式化方法对基于内存标签的 TEE 访问控制进行建模和安全性分析. 首先根据建立的 TEE 标签域提出通用的访问控制模型框架, 在较高的抽象层次给出访问控制实体的形式化定义, 在模型框架的基础上, 利用 B 方法构建了抽象机模型, 包括标签、主体、客体和策略 4 个基本实体抽象机和递增完成的访问控制抽象机模型. 同时在抽象机中描述了不变式约束和基本操作. 接着, 本文在此基础上提出基于模型检测的安全性分析方法, 通过形式化描述访问控制规约以及攻击操作, 在 ProB 中进行模型检测以验证访问控制机制功能正确性并进行安全性分析. 本文以 RISC-V 架构上的 TIMBER-V 为分析实例, 结合其具体实现对原抽象机模型进行修改, 并添加了可信服务的相关变量及操作, 实现模型的实例化, 并利用提出的方法分析了其实现的访问控制的安全性.

References:

[1] ARMs TrustZone. 2021. https://developer.arm.com/ip-products/security-ip/trustzone

[2] McKeen F, Alexandrovich I, Berenzon A, Rozas CV, Shafi H, Shanbhogue V, Savagaonkar UR. Innovative instructions and software model for isolated execution. In: Proc. of the 2nd Int'l Workshop on Hardware and Architectural Support for Security and PrivacyHasp@isca. Tel-Aviv: ACM, 2013. 10. [doi: 10.1145/2487726.2488368]

[3] Kaplan D, Powell J, Woller T. AMD memory encryption. White Paper, 2016. https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf

[4] Weiser S, Werner M, Brasser F, Malenko M, Mangard S, Sadeghi A-R. TIMBER-V: Tag-isolated memory bringing fine-grained enclaves to RISC-V. In: Proc. of the 26th Annual Network and Distributed System Security Symp. San Diego: NDSS, 2019.

[5] Costan V, Lebedev I, Devadas S. Sanctum: Minimal hardware extensions for strong software isolation. In: Proc. of the 25th USENIX Conf. on Security Symp. Austin: USENIX Association (USENIX Security 16), 2016. 857–874.

[6] Brasser F, Gens D, Jauernig P, Sadeghi AR, Stapf E. SANCTUARY: Arming trustzone with user-space enclaves. In: Proc. of the 26th Annual Network and Distributed System Security Symp. San Diego: NDSS, 2019.

[7] Lee D, Kohlbrenner D, Shinde S, Asanović K, Song D. Keystone: An open framework for architecting trusted execution environments. In: Proc. of the 15th European Conf. on Computer Systems. Heraklion: ACM, 2020. 1–1638. [doi: 10.1145/3342195.3387532]

[8] McGillion B, Dettenborn T, Nyman T, Asokan N. Open-tee–an open virtual trusted execution environment. In: Proc. of the 2015 IEEE Trustcom/BigDataSE/ISPA. Helsinki: IEEE, 2015, 1. 400–407. [doi: 10.1109/Trustcom.2015.400]

[9] Pirker M, Slamanig D. A framework for privacy-preserving mobile payment on security enhanced arm trustzone platforms. In: Proc. of the 11th IEEE Int'l Conf. on Trust, Security and Privacy in Computing and Communications. Liverpool: IEEE, 2012. 1155–1160. [doi: 10.1109/TrustCom.2012.28]

[10] Santos N, Raj H, Saroiu S, Wolman A. Using ARM trustzone to build a trusted language runtime for mobile applications. In: Proc. of the 19th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Salt Lake City: ACM, 2014. 67–80. [doi: 10.1145/2541940.2541949]

[11] Zheng XY, Yang LL, Ma JG, Shi G, Meng D. TrustPAY: Trusted mobile payment on security enhanced ARM trustzone platforms. In: Proc. of the 2016 IEEE Symp. on Computers and Communication (ISCC). Messina: IEEE, 2016. 456–462. [doi: 10.1109/ISCC.2016.

7543781]

[12] Bianchi A, Fratantonio Y, Machiry A, Krüegel C, Vigna G, Chung SP, Lee W. Broken fingers: On the usage of the fingerprint API in android. In: Proc. of the 25th Annual Network and Distributed System Security Symp. San Diego: NDSS, 2018.

[13] Lipp M, Gruss D, Spreitzer R, Maurice C, Mangard S. ARMageddon: Cache attacks on mobile devices. In: Proc. of the 25th USENIX Conf. on Security Symp. Austin: USENIX Association (USENIX Security 16), 2016. 549–564.

[14] Wang J, Fan CY, Cheng YQ, Zhao B, Wei T, Yan F, Zhang HG, Ma J. Analysis and research on SGX technology. Ruan Jian Xue Bao/Journal of Software, 2018, 29(9): 2778–2798 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5594.htm [doi: 10.13328/j.cnki.jos.005594]

[15] Ryan K. Hardware-backed heist: Extracting ECDSA keys from qualcomm's TrustZone. In: Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security. London: ACM, 2019. 181–194. [doi: 10.1145/3319535.3354197]

[16] Lee D, Jung D, Fang IT, Tsai CC, Popa RA. An off-chip attack on hardware enclaves via the memory bus. In: Proc. of the 29th USENIX Conf. on Security Symp. USENIX Association (USENIX Security 20), 2020. 28.

[17] Cloosters T, Rodler M, Davi L. TEEREX: discovery and exploitation of memory corruption vulnerabilities in SGX enclaves. In: Proc. of the 29th USENIX Conf. on Security Symp. USENIX Association (USENIX Security 20), 2020. 841–858.

[18] Cerdeira D, Santos N, Fonseca P, Pinto S. SoK: Understanding the prevailing security vulnerabilities in TrustZone-assisted TEE systems. In: Proc. of the 2020 IEEE Symp. on Security and Privacy (SP). San Francisco: IEEE, 2020. 1416–1432. [doi: 10.1109/SP40000.2020.00061]

[19] Cho Y, Shin J, Kwon D, Ham M, Kim Y, Paek Y. Hardware-assisted on-demand hypervisor activation for efficient security critical code execution on mobile devices. In: Proc. of the 2016 USENIX Conf. on Usenix Annual Technical Conf. Denver: USENIX Association (USENIX ATC 16), 2016. 565–578.

[20] Jang J, Choi C, Lee J, Kwak N, Lee S, Choi Y, Kang BB. PrivateZone: Providing a private execution environment using ARM trustzone. IEEE Trans. on Dependable and Secure Computing. IEEE, 2016, 15(5): 797–810. [doi: 10.1109/TDSC.2016.2622261]

[21] Hua ZC, Gu JY, Xia YB, Chen HB, Zang BY, Guan HB. vTZ: Virtualizing ARM trustzone. In: Proc. of the 26th USENIX Conf. on Security Symp. (USENIX Security 17), Vancouver: USENIX Association, 2017. 541–556.

[22] Li WH, Xia YB, Lu L, Chen HB, Zang BY. TEEv: Virtualizing trusted execution environments on mobile platforms. In: Proc. of the 15th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. Providence: ACM, 2019. 2–16. [doi: 10.1145/3313808.3313810]

[23] Kwon D, Seo J, Cho Y, Lee B, Paek Y. PrOS: Light-weight privatized Se cure OSes in ARM TrustZone. IEEE Trans. on Mobile Computing, 2019, 19(6): 1434–1447. [doi: 10.1109/TMC.2019.2910861]

[24] Kaplan D. Protecting vm register state with sev-es. White Paper, 2017. https://www.amd.com/system/files/TechDocs/Protecting%20VM%20Register%20State%20with%20SEV-ES.pdf

[25] SEV-SNP A. Strengthening vm isolation with integrity protection and more. White Paper, 2020. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf

[26] Feng EH, Lu X, Du D, Yang BC, Jiang XQ, Xia YB, Zang BY, Chen HB. Scalable memory protection in the penglai enclave. In: Proc. of the 15th USENIX Symp. on Operating Systems Design and Implementation (OSDI 21). OSDI, 2021. 275–294.

[27] Joffrey G. Attacking the ARM's TrustZone. 2021. https://blog.quarkslab.com/attacking-the-arms-trustzone.html

[28] Brandon A, Trimarchi M. Trusted display and input using screen overlays. In: Proc. of the 2017 Int'l Conf. on ReConFigurable Computing and FPGAs (ReConFig). Cancun: IEEE, 2017. 1–6. [doi: 10.1109/RECONFIG.2017.8279826]

[29] Suciu D, McLaughlin S, Simon L, Sion R. Horizontal privilege escalation in trusted applications. In: Proc. of the 29th USENIX Conf. on Security Symp. USENIX Association (USENIX Security 20), 2020. 47.

[30] Wing JM. A specifier's introduction to formal methods. Computer, 1990, 23(9): 8–22. [doi: 10.1109/2.58215]

[31] Dhawan U, Hritcu C, Rubin R, Vasilakis N, Chiricescu S, Smith JM, Knight Jr TF, Pierce BC, DeHon A. Architectural support for software-defined metadata processing. In: Proc. of the 20th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Istanbul: ACM, 2015. 487–502. [doi: 10.1145/2694344.2694383]

[32] Suh GE, Lee JW, Zhang D, Devadas S. Secure program execution via dynamic information flow tracking. ACM SIGPLAN Notices, 2004, 39(11): 85–96. [doi: 10.1145/1037187.1024404]

[33] Dalton M, Kannan H, Kozyrakis C. Raksha: A flexible information flow architecture for software security. ACM SIGARCH Computer Architecture News, 2007, 35(2): 482–493. [doi: 10.1145/1273440.1250722]

[34] Song CY, Moon H, Alam M, Yun I, Lee B, Kim T, Lee W, Paek Y. HDFI: Hardware-assisted data-flow isolation. In: Proc. of the 2016 IEEE Symp. on Security and Privacy (SP). San Jose: IEEE, 2016. 1–17. [doi: 10.1109/SP.2016.9]

[35] Witchel E, Cates J, Asanović K. Mondrian memory protection. In: Proc. of the 10th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. ACM, 2002. 304–316. [doi: 10.1145/605397.605429]

[36] Xia HY, Woodruff J, Barral H, Esswood L, Joannou A, Kovacsics R, Chisnall D, Roe M, Davis B, Napierala E, Baldwin J, Gudka K, Neumann PG, Richardson A, Moore SW, Watson RNM. CheriRTOS: A capability model for embedded devices. In: Proc. of the 36th IEEE Int'l Conf. on Computer Design (ICCD). Orlando: IEEE, 2018. 92–99. [doi: 10.1109/ICCD.2018.00023]

[37] Markettos AT, Baldwin J, Bukin R, Neumann PG, Moore SW, Watson RNM. Position paper: Defending direct memory access with CHERI capabilities. In: Proc. of the 2020 Hardware and Architectural Support for Security and Privacy. ACM, 2020. 7.

[38] Zeldovich N, Kannan H, Dalton M, Kozyrakis C. Hardware enforcement of application security policies using tagged memory. In: Proc. of the 8th USENIX Conf. on Operating Systems Design and Implementation. San Diego: USENIX Association. 2008, 8. 225–240.

[39] Hughes G, Bultan T. Automated verification of access control policies using a sat solver. Int'l Journal on Software Tools for Technology Transfer. Springer, 2008, 10(6): 503–520. [doi: 10.1007/s10009-008-0087-9]

[40] Wu XG, Wen YJ, Chen LQ, Dong W, Wang J. Data race detection for interrupt-driven programs via bounded model checking. In: Proc. of the 7th IEEE Int'l Conf. on Software Security and Reliability Companion. Gaithersburg: IEEE, 2013. 204–210. [doi: 10.1109/SERE-C.2013.33]

[41] Wang B, Bai XY, He F, Song XY. Survey on modeling and verification techniques of composable embedded software. Ruan Jian Xue Bao/Journal of Software, 2014, 25(2): 234–253 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4533.htm [doi: 10.13328/j.cnki.jos.004533]

[42] Xu FW, Fu M, Feng XY, Zhang XR, Zhang H, Li ZH. A practical verification framework for preemptive os kernels. In: Proc. of the 28th Int'l Conf. on Computer Aided Verification. Toronto: Springer, 2016. 59–79. [doi: 10.1007/978-3-319-41540-6_4]

[43] Abomhara M, Yang HH, Køien GM, Lazreg MB. Work-based access control model for cooperative healthcare environments: Formal specification and verification. Journal of Healthcare Informatics Research. Springer, 2017, 1(1): 19–51. [doi: 10.1007/s41666-017-0004-7]

[44] Zhang YD, Song F. Model-checking for heterogeneous multi-agent systems. Ruan Jian Xue Bao/Journal of Software, 2018, 29(6): 1582–1594 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5462.htm [doi: 10.13328/j.cnki.jos.005462]

[45] Rivera V. Formal verification of access control model for my health record system. In: Proc. of the 25th Int'l Conf. on Engineering of Complex Computer Systems (ICECCS). Singapore: IEEE, 2020. 21–30. [doi: 10.1109/ICECCS51672.2020.00010]

[46] Yuan CY, He YP, He JB, Zhou ZY. Formal specification and proof of the RBAC model with constraints of conflicts. Journal of Computer Research and Development, 2006, 43(S2): 498–508 (in Chinese with English abstract).

[47] Cheng L, Zhang Y. A verification method of security model based on uml and model checking. Chinese Journal of Computers, 2009, 32(4): 699–708 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2009.00699]

[48] Zahoor E, Asma Z, Perrin O. A formal approach for the verification of AWS IAM access control policies. In: Proc. of the 6th European Conf. on Service-oriented and Cloud Computing. Oslo: Springer, 2017. 59–74. [doi: 10.1007/978-3-319-67262-5_5]

[49] Lu JF, Li RX, Lu ZD, Li B. Integrating trust and role for secure interoperation in multi-domain environment. In: Proc. of the 2008 Int'l Conf. on Information Security and Assurance (ISA 2008). Busan: IEEE, 2008. 77–82. [doi: 10.1109/ISA.2008.30]

[50] Tang Z, Zhang SH, Li KL, Feng BM. Security analysis and validation for access control in multi-domain environment based on risk. In: Proc. of the 6th Int'l Conf. on Information Security Practice and Experience. Seoul: Springer, 2010. 201–216. [doi: 10.1007/978-3-642-12827-1_15]

[51] Gouglidis A, Mavridis I, Hu VC. Verification of secure inter-operation properties in multi-domain RBAC systems. In: Proc. of the 7th IEEE Int'l Conf. on Software Security and Reliability Companion. Gaithersburg: IEEE, 2013. 35–44. [doi: 10.1109/SERE-C.2013.25]

[52] Nazerian F, Motameni H, Nematzadeh H. Secure access control in multidomain environments and formal analysis of model specifications. Turkish Journal of Electrical Engineering & and Computer Sciences, 2018, 26(5): 2525–2540. [doi: 10.3906/elk-1802-55]

[53] Cao J, Li PF, Zhu QM, Qian PD. SECURITY tag-based multi-domain secure access control model. Computer Applications and Software, 2015, 32(1): 297–302 (in Chinese with English abstract). [doi: 10.3969/j.issn.1000-386x.2015.01.075]

[54] Unal D, Çağlayan MU. A formal role-based access control model for security policies in multi-domain mobile networks. Computer Networks. Elsevier, 2013, 57(1): 330–350. [doi: 10.1016/j.comnet.2012.09.018]

[55] Unal D, Akar O, Caglayan MU. Model checking of location and mobility related security policy specifications in ambient calculus. In: Proc. of the 5th Int'l Conf. on Mathematical Methods, Models, and Architectures for Computer Network Security. St. Petersburg: Springer, 2010. 155–168. [doi: 10.1007/978-3-642-14706-7_12]

[56] Ünal D, Çağlayan MU. Spatiotemporal model checking of location and mobility related security policy specifications. Turkish Journal of Electrical Engineering and Computer Sciences, 2013, 21(1): 144–173. [doi: 10.3906/elk-1105-54]

[57] Alam Q, Malik SUR, Akhunzada A, Choo K-KR, Tabbasum S, Alam M. A cross tenant access control (CTAC) model for cloud

computing: formal specification and verification. IEEE Trans. on Information Forensics and Security, 2016, 12(6): 1259−1268. [doi: 10.1109/TIFS.2016.2646639]

[58] Ren L, Chang R, Yin Q, Wang W. Using the B method to formalize access control mechanism with TrustZone hardware isolation (short paper). In: Proc. of the 13th Int'l Conf. on Information Security Practice and Experience. Melbourne: Springer, 2017. 759–769. [doi: 10.1007/978-3-319-72359-4_47]

附中文参考文献:

[14] 王鹃, 樊成阳, 程越强, 赵波, 韦韬, 严飞, 张焕国, 马婧. SGX技术的分析和研究. 软件学报, 2018, 29(9): 2778−2798. http://www.jos.org.cn/1000-9825/5594.htm [doi: 10.13328/j.cnki.jos.005594]

[41] 王博, 白晓颖, 贺飞, Song XY. 可组合嵌入式软件建模与验证技术研究综述. 软件学报, 2014, 25(2): 234−253. http://www.jos.org.cn/1000-9825/4533.htm [doi: 10.13328/j.cnki.jos.004533]

[44] 张业迪, 宋富. 异构多智能体系统模型检查. 软件学报, 2018, 29(6): 1582−1594. http://www.jos.org.cn/1000-9825/5462.htm [doi: 10.13328/j.cnki.jos.005462]

[46] 袁春阳, 贺也平, 何建波, 周洲仪. 具有冲突约束的RBAC模型的形式化规范与证明. 计算机研究与发展, 2006, 43(S2): 498–508.

[47] 程亮, 张阳. 基于UML和模型检测的安全模型验证方法. 计算机学报, 2009, 32(4): 699–708. [doi: 10.3724/SP.J.1016.2009.00699]

[53] 曹进, 李培峰, 朱巧明, 钱培德. 基于安全标签的多域安全访问控制模型. 计算机应用与软件, 2015, 32(1): 297–302. [doi: 10.3969/j.issn.1000-386x.2015.01.075]

苗新亮(1994－), 男, 博士生, 主要研究领域为形式化方法, 嵌入式系统安全.

赵永望(1979－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为形式化方法, 操作系统与安全, 安全认证.

常瑞(1981－), 女, 博士, 副教授, 博士生导师, CCF 高级会员, 主要研究领域为嵌入式系统安全, 形式化分析与验证.

蒋烈辉(1967－), 男, 博士, 教授, 博士生导师, 主要研究领域为计算机体系结构, 逆向工程与安全.

潘少平(1994－), 男, 硕士生, CCF 学生会员, 主要研究领域为形式化方法, 可信执行环境.