软件学报 ISSN 1000-9825, CODEN RUXUEW
2024,35(11):4949−4972 [doi: 10.13328/j.cnki.jos.007001] [CSTR: 32375.14.jos.007001]

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

基于远程证明的数据服务完整性验证方法*

张 溯[1], 张 颖[2], 张 伟[3], 黄 罡[1]

[1](高可信软件技术教育部重点实验室 (北京大学), 北京 100871)

[2](北京大学 软件工程国家工程研究中心, 北京 100871)

[3](神旗数码有限公司, 北京 100085)

通信作者: 张颖, E-mail:zhang.ying@pku.edu.cn

摘 要: 数据作为一种新型生产要素, 需要在不同主体间流通以发挥价值. 在这一过程中, 数据需要确保其完整性, 避免受到未经授权的篡改, 否则可能导致极为严重的后果. 现有工作通过将分布式账本与数据加密、校验技术结合实现数据存证以证明待流通数据在传输、存储等环节中未受篡改, 保障数据的完整性. 然而, 此类工作难以确认数据供方所提供数据本身的完整性, 一旦数据供方主动或被动提供了伪造数据, 后续完整性保障工作将失去意义. 为此, 提出一种基于远程证明的数据服务完整性验证方法, 所提方法以可信执行环境作为信任锚, 对特定数据服务静态代码、执行过程和执行结果的完整性进行多维度量与验证, 并通过程序切片优化对特定数据服务的完整性验证, 从而将数据完整性保障的范围延伸至数据供方提供数据的环节. 通过在 3 个真实 Java 信息系统中 25 个数据服务上的一系列实验验证了所提出方法的有效性.

关键词: 数据服务; 数据完整性; 远程证明; 控制流证明; 可信执行环境

中图法分类号: TP311

中文引用格式: 张溯, 张颖, 张伟, 黄罡. 基于远程证明的数据服务完整性验证方法. 软件学报, 2024, 35(11): 4949–4972. http://www.jos.org.cn/1000-9825/7001.htm

英文引用格式: Zhang S, Zhang Y, Zhang W, Huang G. Method of Data Service Integrity Verification Based on Remote Attestation. Ruan Jian Xue Bao/Journal of Software, 2024, 35(11): 4949–4972 (in Chinese). http://www.jos.org.cn/1000-9825/7001.htm

Method of Data Service Integrity Verification Based on Remote Attestation

ZHANG Su[1], ZHANG Ying[2], ZHANG Wei[3], HUANG Gang[1]

[1](Key Laboratory of High-confidence Software Technology of Ministry of Education (Peking University), Beijing 100871, China)

[2](National Engineering Research Center for Software Engineering, Peking University, Beijing 100871, China)

[3](Internetware Technologies Corporation, Beijing 100085, China)

Abstract: As an important production factor, data need to be exchanged between different entities to create value. In this process, data integrity needs to be ensured, or in other words, data cannot be tampered without authorization, or otherwise, it may lead to extremely serious consequences. The existing work realizes data evidence preservation by combining distributed ledger with data encryption and verification technology to ensure the integrity of data to be exchanged in transmission, storage, and other related data processing phrases. However, such work is difficult to confirm the integrity of the data provided by the data supplier. Once the data supplier provides forged data, all subsequent integrity assurance will be meaningless. Therefore, this study proposes a method for verifying the integrity of data services based on remote attestation. By using the trusted execution environment as the trust anchor, this method can measure and verify the integrity of the static code, execution process, and execution result of a specific data service. It also optimizes the integrity verification of a specific data service through program slicing, thus extending the scope of data integrity assurance to the time point when the data supplier provides data. A series of experiments are carried out on 25 data services of three real Java information systems to validate the

* 基金项目: 国家重点研发计划 (2021YFF1201103)
收稿时间: 2023-02-26; 修改时间: 2023-04-20; 采用时间: 2023-07-14; jos 在线出版时间: 2023-11-08
CNKI 网络首发时间: 2023-11-10

proposed method.

Key words: data service; data integrity; remote attestation (RA); control flow attestation; trusted execution environment (TEE)

如今, 数据已经成为一种引导各类资源发挥作用、推动生产力发展的新型生产要素[1]. 为了充分发挥其要素价值, 数据需要在不同主体间进行高效且合规的流通, 从而通过对数据的深度融合与挖掘提升数据驱动的决策能力, 解决业务挑战. 例如, 智慧政务需要整合各级职能部门的各类政务数据, 以通过监测、分析和智能响应加强职能监管并提升政府的管理效率和服务水平; 智能制造需要打通产业链上下游、企业各部门、工厂各生产设备的数据, 以实现生产制造全流程的精准监测和控制; 传染病防控需要对交通出行、移动通信、病毒检测等数据进行融合分析, 以实现精准高效的人员管控、医疗救治、复工复产.

在数据流通过程中, 数据的完整性 (integrity) 不受破坏是一项基本的数据安全要求, 它要求保证只能由某种特定的、已授权的方式来修改数据, 即防止数据受到篡改. 现有工作通常采用基于分布式账本的数据存证来保障数据流通过程中的数据完整性[2]. 分布式账本 (distributed ledger)[3]是一种在网络成员之间共享、复制和同步的数据库, 可以对存储在其中的数据提供极强的完整性保障. 基于此, 将数据本身或其完整性校验信息 (如散列值) 存储在分布式账本中, 即可避免数据受到篡改或发现数据所受到的篡改, 从而实现数据存证. 由于数据存证技术仅能保障存证后数据的完整性, 因此其在数据流通场景下的有效性依赖于如下假设: 数据供方所提供的待存证数据本身是未经篡改的. 然而, 这一假设面临失效的风险. 随着数据技术与市场的发展, 越来越多的中小企业甚至个人开始参与数据流通, 数据供方的身份愈加鱼龙混杂. 与此同时, 数据供方提供伪造数据的动机也逐渐增强, 例如, 在以数据使用量计费的数据交易模式下伪造数据 (也可视为篡改数据) 可以获得更多利益, 在排污统计等场景下篡改数据可以避免追责[4]. 此外, 数据供方信息系统还可能由于受到黑客攻击而影响对数据请求的响应过程, 返回受到篡改的伪造数据. 数据流通是一个多环节的链路, 一旦数据供方可信的假设失效, 现有关注于后续流通环节中数据完整性保障的工作都将失去意义. 因此, 应当更多地考虑数据供方不可信的情况, 设计相关技术方案将完整性保障的范围延伸至数据供方提供数据的环节, 即数据供给过程.

当数据供方希望通过互联网与数据需方可控地流通数据时, 通常会发布数据服务[5]以提供对各类数据的访问. 网络应用程序接口 (Web API) 是数据服务最常见的形式, 在此类接口式数据服务中, 数据需方通过发起 HTTP GET 或 POST 请求的方式对数据服务进行调用, 并通过对 HTTP 请求返回结果的解析获得所需数据. 为了实现对数据供给过程的完整性保障, 应当通过某种方式向数据需方证明其调用数据服务所获得的数据是该服务如约执行所产生的正确数据, 而非受到过篡改的伪造数据. 考虑到基于数据服务的数据供给是一个软件执行的过程, 可以通过对该过程中所涉及软件的完整性[6,7]进行度量与验证的方式, 实现上述证明. 远程证明 (remote attestation, RA) 是一种用于验证在他方设备上所运行软件的完整性的技术, 其利用证明者环境中存在的可信组件对待证明对象进行度量, 并在验证者环境进行验证. 静态远程证明通过对特定程序二进制文件的完整性进行度量与验证, 保障其程序逻辑没有受到静态篡改; 动态远程证明则通过对特定程序运行时执行路径的完整性进行度量与验证, 保障其程序逻辑没有受到运行时的动态篡改 (如控制流攻击). 虽然此类静态与动态远程证明的结合理论上可以对目标信息系统的可执行程序本身以及运行时行为的完整性进行较好的保障, 但由于动态远程证明方法需要对目标信息系统的所有合法控制流路径进行分析, 复杂度极高, 因而现有工作主要面向简单的小型程序. 然而, 对外提供数据服务的数据供方信息系统多为较复杂的业务处理程序, 如企业资源计划 (ERP) 系统[8], 直接应用此类方法将面临控制流路径状态爆炸的问题, 并且动态远程证明的过程也可能对目标系统中其他业务功能的正常使用产生影响.

针对上述问题, 本文提出了一种新型的数据服务完整性验证方法. 如前文所述, 在数据流通场景下需从数据产生的源头保障其完整性, 否则无法真正防止数据受到篡改. 因此, 本文将产生待流通数据的数据服务作为研究对象, 面向通过数据服务进行数据流通的典型流程, 提出基于远程证明的多维度量与验证方法以保障数据来源真实有效. 此外, 本文还并重点关注动态远程证明方法应用于大型程序时面临的控制流合法路径分析状态爆炸问题, 提出基于程序切片的局部验证方法来解决这一问题, 从而使得所提出的多维度量与验证方法能够有效应用于如企业资源管理系统的大型程序. 主要步骤如下: 首先, 在基于远程证明的完整性验证方法设计上, 提出了一个数据服务

完整性多维度量策略, 度量对象包括目标数据服务的静态代码、执行过程以及执行结果; 设计了一个运行在可信执行环境内的度量引擎, 以此对经过事先程序插桩的特定数据服务进行上述度量并构造证明报告, 实现度量过程的可信执行; 通过将验签后的度量结果与事先离线计算获得的合法度量值进行比对, 验证数据服务的完整性. 其次, 在基于程序切片的完整性验证过程优化上, 设计了一个用于描述程序行为的运行时行为模型, 通过对模型的构造与分析定位目标数据服务所对应的程序片段, 缩小待验证范围, 实现度量对象的裁剪; 将该程序片段切片为一段与其余业务功能解耦的等效可执行代码, 并根据数据依赖关系为其恢复执行所需的上下文环境, 从而以对该段代码而非对完整原始程序插桩度量逻辑的方式, 实现度量过程隔离且轻量化的数据服务完整性验证. 通过在 3 个真实 Java 信息系统中 25 个数据服务上的一系列实验对所提方法的有效性和性能进行了评估. 实验结果表明, 该方法能够有效地对数据服务完整性进行度量和验证, 其引入的额外开销较低, 在数据流通场景下是可接受的, 其中采用的基于程序切片的优化策略能够有效降低代码膨胀率以及度量和验证过程的开销.

本文的主要贡献如下.

● 针对现有工作难以验证数据服务返回结果是否真实有效的问题, 提出了一种基于远程证明的数据服务完整性验证方法, 首次将远程证明应用到数据服务的复杂场景中, 通过对数据服务的静态代码、执行过程和执行结果进行多维度量与验证, 保障特定数据服务的完整性.

● 设计了新型的远程证明开销优化方法, 通过针对特定数据服务的程序切片, 缩小待度量与验证对象的范围, 避免由于控制流图过于复杂而导致的状态爆炸, 并使得度量执行过程与相应业务功能隔离, 实现轻量化的数据服务完整性验证.

● 在 3 个真实 Java 信息系统中的 25 个数据服务上进行了一系列实验, 结果表明所提方法能够对目标数据服务的完整性进行有效验证, 其引入的额外开销较低, 在数据流通场景下是可接受的, 其中采用的基于程序切片的优化策略能够有效降低代码膨胀率以及度量和验证过程的开销.

本文第 1 节介绍所需的基础知识以及本文的相关工作. 第 2 节介绍本文设计的数据服务完整性多维度量与验证方法. 第 3 节进一步介绍本文设计的基于程序切片的数据服务完整性验证优化策略. 第 4 节通过对 3 个真实应用中 25 个数据服务的一系列实验验证了本文所提出方法的有效性, 并对方法的安全性、可行性与局限性进行了分析. 最后, 第 5 节总结全文.

1 基础知识与相关工作

本节介绍所需的基础知识以及与本文研究内容相关的工作.

1.1 远程证明

远程证明是一种验证远程设备上软件完整性的方法. 通常, 它被实现为一种质询-响应协议, 在该协议中不受信任的设备 (称为证明者, prover) 可以向远程受信任方 (称为验证者, verifier) 证明在设备上运行的软件是合法的并且没有受到恶意破坏. 典型的远程证明过程始于验证者向证明者发起的一个请求, 然后证明者安全地对目标软件进行度量以构造所需的证据, 最后将包含证据的报告发送给验证者加以验证 (如图 1(a) 所示). 在本文中, 我们基于远程证明对数据服务的完整性进行验证, 即以数据供方作为证明者、数据需方作为验证者、数据服务作为待度量软件, 在数据供方通过数据服务提供待流通数据的同时, 以远程证明的方式向调用方提供相关证据以供其验证该数据服务的完整性 (如图 1(b) 所示).

远程证明的标准信任假设要求证明者环境中存在一个不能被攻击者 (包括证明者自身在内) 破坏的可信组件 (称为信任锚) 来建立基础信任. 信任锚有许多实现, 例如可信平台模块 (TPM)[9]、可信执行环境 (TEE)[10], 当前主流的 Intel、AMD、ARM 计算平台均提供对 TPM 与 TEE 的支持[11−13].

大多数远程证明方案使用静态方法来度量程序的状态. 在此类方案中, 证明者收集程序代码 (通常为二进制文件) 的静态度量值 (通常为散列值), 并计算加密签名来进行证明. 近年来, 关于静态远程证明的研究工作主要关注于对实时程序友好的证明[14]、瞬态恶意软件检测[15]等方面.

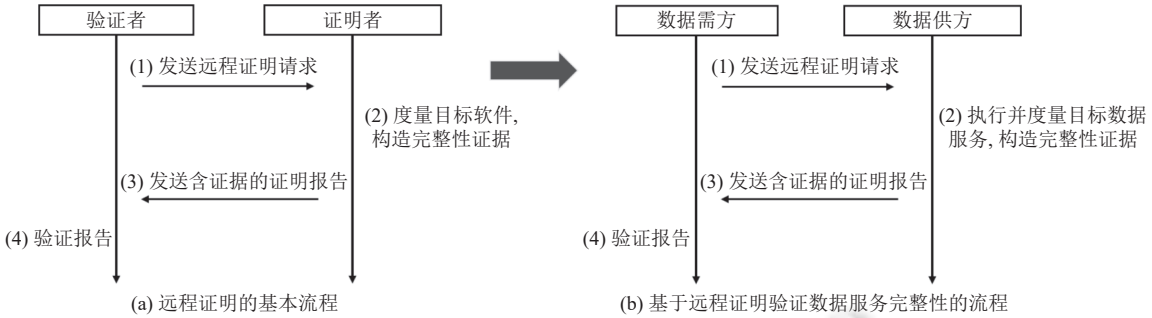


图 1 经典远程证明与本文数据服务完整性验证的流程

静态证明虽然有效, 但只能确保二进制文件的完整性, 不能确保其执行的完整性. 例如, 静态证明无法检测到控制流攻击, 包括面向返回的编程 (return-oriented programming)[16]和面向跳转的编程 (jump-oriented programming)[17]. 此类控制流攻击使用代码复用技术, 破坏控制数据以将控制流引导到特定位置并使程序执行一些恶意代码而不需要注入恶意指令, 因此程序的静态度量值总是保持不变, 验证者无法检测到对证明者的攻击. 针对此类控制流攻击问题, 一些工作进一步提出了称为控制流证明 (control-flow attestation) 的动态验证方案[18,19], 通过在运行时跟踪并比对控制流执行路径来检查控制流攻击.

1.2 控制流证明

控制流证明 (control-flow attestation, CFA) 是一种动态远程证明技术, 通过在程序执行期间安全地记录每个控制流更改指令 (例如跳转、分支、返回) 的目的地信息, 帮助验证者确认特定程序的指令是否以特定的预期顺序执行.

C-FLAT[18]提出了首个控制流证明方案. 在证明前, C-FLAT 基于控制流图将设备的程序划分为稳定的程序块. 在证明阶段, C-FLAT 使用软件工具和可信执行环境来跟踪程序的控制流执行, 并对控制流路径中每个程序块的 ID 进行累计散列计算以获得证明结果, 验证者可以通过该累计散列值与预先存储的散列值进行比较来验证设备的控制流完整性. LO-FAT[19]在 C-FLAT 的基础上利用硬件提高了方案的效率. LO-FAT 利用现有的处理器功能在硬件中记录控制流而无需软件检测, 从而减轻主处理器的负担并提高并行计算的能力. ReCFA[20]提出了一种无需对所有合法控制流路径进行离线分析的弹性控制流证明策略以提高控制流证明的可扩展性. ReCFA 通过过滤可跳过的调用节点、折叠与程序结构相关的控制流事件和贪婪压缩等方式来尽可能地减少执行控制流证明时处理的运行时控制流事件的数量. Papamartzivanos 等人[21]提出了一种软件辅助的动态远程证明方法, 通过对 eBPF 和 IntelPT 探测器进行编程, 拦截系统内部操作以生成运行时控制流路径, 而无需对待证明程序进行程序插桩.

上述工作都是基于 C-FLAT 提出的累计散列值比对方式进行控制流证明的, 这一方式需要通过静态程序分析穷举所有合法控制流路径所对应的散列值. 由于上述工作是面向小型嵌入式设备程序 (程序大小通常仅为 KB 级) 进行设计与实现的, 因此穷举所带来的开销是可以接受的. 然而当将控制流证明应用于如 ERP 等运行在桌面 PC、服务器中的大型程序时 (程序大小通常在 MB 级及以上), 穷举过程将面临严重的状态爆炸问题.

Liu 等人[22]提出了一种基于日志的动态远程证明方法, 通过完整记录程序执行路径来避免传统控制流证明方法在验证时可能存在的状态爆炸问题. 其中, 日志的安全存储是基于片上 SRAM 物理不可克隆功能 (PUF) 的轻量级信任根实现的. 然而, 此类基于日志的方法应用于大型程序时将产生日志规模爆炸以及在线验证时间大幅增加的新问题.

为了将控制流证明有效应用于大型程序, 本文提出了一种基于程序切片的优化策略, 通过静态与动态程序分析确定需要进行控制流证明的关键程序片段, 仅对关键程序片段而非完整程序进行控制流证明, 从而在继承传统控制流证明方法度量结果规模较小且在线验证效率极高的优势的同时, 避免合法控制流路径分析状态爆炸问题.

1.3 预言机

预言机 (Oracle) 是一种为分布式账本[3]从外部数据源获取数据并将其包含在分布式账本隔离执行环境中的技术[23]. 由于预言机并不拥有原生区块链协议的安全属性, 为了确保分布式账本上事务 (也称为交易) 执行的一致性和有效性, 预言机需要通过完整性证明机制来证明数据未受篡改[24].

在集中式预言机中, Provable[25]预言机在获取链外数据的同时为获取过程构建完整性证据 (例如 TLS-Notary[26]) 以证明数据访问代码已经在链外按预期执行, 保障从数据源获取的数据未被篡改. Town Crier[27]预言机则将访问数据的核心逻辑作为在英特尔 SGX 可信执行环境的飞地 (Enclave) 中的可信代码进行执行, 并为数据结果附加基于可信执行环境的完整性证据. 在去中心化预言机中, Chainlink[28]预言机设计了一个去中心化的预言机网络, 其中多个预言机使用 K-out-of-M 阈值签名来就要接受的数据结果达成共识, 并嵌入信誉智能合约以激励和惩罚参与报告的预言机. Augur[29]预言机则提出了一个去中心化预测市场形式的预言机网络, 拥有声誉代币的用户可以质押代币并参与对数据结果的投票并在被采纳后收取结算费用. 数据结果产生后, 不同意该结果的用户可以质押代币并提出异议. 最后, 激励合约根据最终投票结果对相关用户进行奖惩.

尽管预言机采用了基于软件或硬件的完整性证明机制来证明数据未受篡改, 但其仍可能因信任单一第三方数据源而出现数据完整性问题[24]. 一旦单一第三方数据源向预言机提供了伪造数据, 那么预言机向分布式账本提供的数据也必然是伪造数据. 这一问题正是本文工作希望解决的问题. 因此, 在预言机技术的视角下, 可以将本文工作视作是对现有预言机工作在数据完整性验证方面的补充.

2 基于远程证明的数据服务完整性多维度量与验证

要实现对数据服务的完整性多维度量与验证, 首先需要确定待度量的对象. 为此, 本文分析数据供方基于数据服务提供数据的流程, 确定其中易使数据受到篡改的 3 个薄弱方面: 静态代码、执行过程与执行结果, 并据此设计待度量的对象. 其次, 需要确保度量过程与结果是可信的. 为此, 本文设计一个能够对特定程序进行上述完整性度量并构造证明报告的运行在可信执行环境内的可信度量引擎, 并利用程序插桩在待证明程序中插入用于获得运行时信息并与可信度量引擎交互的特定处理逻辑, 从而实现可信的度量执行. 最后, 需要对所获得的度量结果进行验证. 为此, 本文事先为待证明程序离线计算可能的合法度量值, 然后在数据需方获取数据与度量结果时验证度量结果的签名并将各度量值与合法度量值进行比对, 从而完成对度量结果的验证.

2.1 多维度量的对象设计

在调用数据服务获取数据的过程中, 调用者无法获悉数据供方在接收到其调用请求后的具体响应过程, 只能获得其返回的调用结果. 也就是说, 即使数据供方未按照事先约定正确执行数据服务以进行响应, 而是直接执行了其他任意自定义的业务逻辑, 返回了本不应返回的数据, 数据需方也无法对此进行鉴别. 在数据流通场景下, 数据供方是有一定动机进行上述恶意行为的. 例如, 在以数据使用量计费的数据交易模式下伪造数据 (也可视为篡改数据) 可以获得更多利益, 在排污统计等场景下篡改数据可以避免追责. 此外, 数据供方信息系统还有可能由于受到黑客攻击而影响对数据请求的响应过程, 返回受到篡改的伪造数据. 在本文中, 我们将试图篡改数据的恶意方统称为攻击者. 因此, 为了避免上述问题, 数据服务应当在返回数据的同时给出一定的证据供数据需方进行验证, 以证明数据供方响应数据请求过程的完整性未受破坏, 进而帮助数据需方确认所调用数据服务的完整性. 图 2 展示了数据供方基于数据服务提供数据的流程. 在这一过程中, 存在以下 3 个薄弱方面, 使得数据可能由于供方的恶意行为或信息系统遭受攻击而受到篡改.

1) 数据服务的静态代码: 在目标数据服务执行前, 通过对其可执行文件的篡改, 使其在后续被执行时加载已被篡改的恶意代码, 进而改变其运行时行为以实现数据篡改.

2) 数据服务的执行过程: 在目标数据服务执行时, 利用软件漏洞修改控制数据 (如代码指针) 或关键约束数据来操纵设备的控制流, 从而动态改变其运行时行为以实现数据篡改.

3) 数据服务的执行结果: 在目标数据服务执行后, 通过代理劫持等方式进行中间人攻击以修改其返回至数据

需方的执行结果 (即所需数据), 以实现数据篡改.

也就是说, 数据服务返回数据结果时附带的证据应当能够支撑对以上 3 个方面的完整性进行验证. 我们假设能够确认数据供方信息系统在某一时刻是安全的, 并且能够以某种方式记录下此时该信息系统的相关信息. 这样, 响应数据请求时通过某种机制来构造上述证据后, 服务调用方即可通过对证据以及与先前所记录的信息进行比对和分析, 来确认数据供方信息系统确实执行了应执行的数据服务, 并且其执行过程和结果都是安全可信的. 为此, 需要在数据供方信息系统执行数据服务的同时, 利用其环境中的信任锚对数据服务静态代码、执行过程和执行结果的完整性分别进行度量.

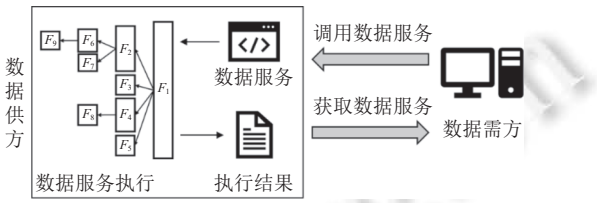


图 2 通过数据服务获得所需数据的过程

在现代计算机设备中, 程序的执行受到操作系统和硬件的数据执行保护 (data execution prevention, DEP)[30]机制的保护, 其能够利用内存页面保护机制防止恶意代码被注入正在运行的程序中, 使得正在运行的程序中加载到内存的代码在该次执行过程中无法被修改. 因此, 对目标数据服务已被加载的代码段进行散列值的计算, 并将计算结果与已知的标准合法散列值进行比较, 即可验证目标数据服务的静态代码是否受到了篡改. 在本文中, 我们将这一度量结果称为静态代码度量值.

程序的执行过程不同于其静态代码, 具有动态性. 要对其进行度量, 首先需要为其设计一种适当的表示方法. 基本块 (basic block, BB)[31]是指一段不包含任何跳转类指令或跳转目标的顺序执行的代码. 一个基本块中只存在一个入口和出口, 即程序无法绕开入口直接跳转至基本块中的非入口位置, 且只有最后一条指令能进入其他基本块. 控制流图 (control flow graph, CFG)[32], 也叫控制流程图, 是一个过程或程序的抽象表现, 代表了一个程序执行过程中会遍历到的所有路径. 它用图的形式表示一个过程内所有基本块执行的可能流向, 也能反映一个过程的实时执行过程. 在控制流图中, 图中的每个节点代表一个基本块, 有向边则用于表示控制流中的跳转. 一段程序的一次执行就是一系列基本块的执行, 可以表示为控制流图中从入口节点至出口节点的一条路径. 如果为每个基本块分配一个唯一的编号 (ID), 那么就可以用一个基本块 ID 的序列来表示该段程序的一次执行过程. 我们将这样一个基本块 ID 序列称之为一个控制流图执行路径. 考虑到一次程序执行的过程中可能包含数量极其巨大的基本块执行, 例如循环语句会导致循环内的基本块被反复执行, 直接记录完整的基本块 ID 序列可能带来较大的存储和传输开销. 为此, 可以通过对基本块 ID 进行累计的散列值计算, 使用累计结果来表示该序列.

图 3 展示了上述累计散列计算的过程. 该累计散列值的计算公式为 $h_{cur} = H(h_{prev}, id_{cur})$, 其中 h_{cur} 为当前的累计散列值, h_{prev} 为先前的累计散列值, id_{cur} 为当前基本块 ID. 基本块 ID 是预先分配的, 其与基本块为一一对应关系, 每个基本块具有唯一的 ID. 可以看到, 对于图 3 中的基本块关系, 存在两种可能的 CFG 执行路径, 分别为 <BB_1, BB_2, BB_3, BB_6> 和<BB_1, BB_4, BB_5, BB_6>, 其累计散列计算结果分别为 $h_6 = H(h_3, id_6)$ 和 $H(h_5, id_6)$. 由于基本块 ID 具有唯一性, 因此在不存在散列冲突的情况下, 不同 CFG 执行路径所对应的累计散列计算结果也是唯一的, 可以使用该结果来代表一个 CFG 执行路径. 在本文中, 我们将这样代表一段程序的一次执行过程的度量结果称为执行过程度量值.

采用上述方式可以极大地简化对 CFG 执行路径的度量过程, 但同时这也要求在对该度量值进行验证时需要事先求得所有可能的 CFG 执行路径的合法度量值, 然后再将待验证度量值与其进行比对以完成验证. 考虑到当目标程序中存在递归或循环逻辑时, 可能存在无穷的不同合法 CFG 执行路径, 这会导致无法计算出所有合法度量值, 进而无法完成对度量结果的验证. 因此, 本文采用 Abera 等人提出的子过程策略[18]对递归与循环逻辑进行处理来避免此类问题的发生. 具体来说, 每当进入递归或循环逻辑时, 将其视为是一段独立的子过程采用上述方式进行

度量; 对于一个递归或循环逻辑中多次度量得到的相同度量值, 记录其出现次数; 对于嵌套的子过程, 采用嵌套的方式进行记录. 通过这种方式, 可以将无穷的合法 CFG 执行路径度量值转换为有穷的子过程度量值和子过程迭代次数的记录.

最后, 对于执行结果的度量则只需由度量引擎在其产生后但返回前对其进行散列计算即可, 其难点更多的在于如何定位特定数据服务所对应的程序片段范围以获取上述执行结果, 我们将在第 3.1.2 节进行介绍. 我们将这一度量结果称为执行结果度量值.

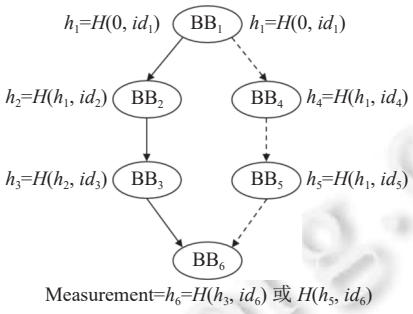


图 3 基于累计散列计算的控制流图执行路径表示

图 4 展示了对数据服务完整性进行多维度量与验证的流程. 首先, 首次执行前, 证明端 (数据供方) 对目标数据服务所在程序进行度量逻辑插桩 (图 4 ①); 其次, 验证端 (数据需方) 发送对目标数据服务的调用请求 (图 4 ②), 证明端收到请求后执行目标数据服务 (图 4 ③); 然后, 证明端在目标数据服务的执行过程中, 对其静态代码、执行过程以及执行结果进行度量 (图 4 ④); 最后, 验证端对相关度量结果进行验证以保障所调用数据服务的完整性 (图 4 ⑤).

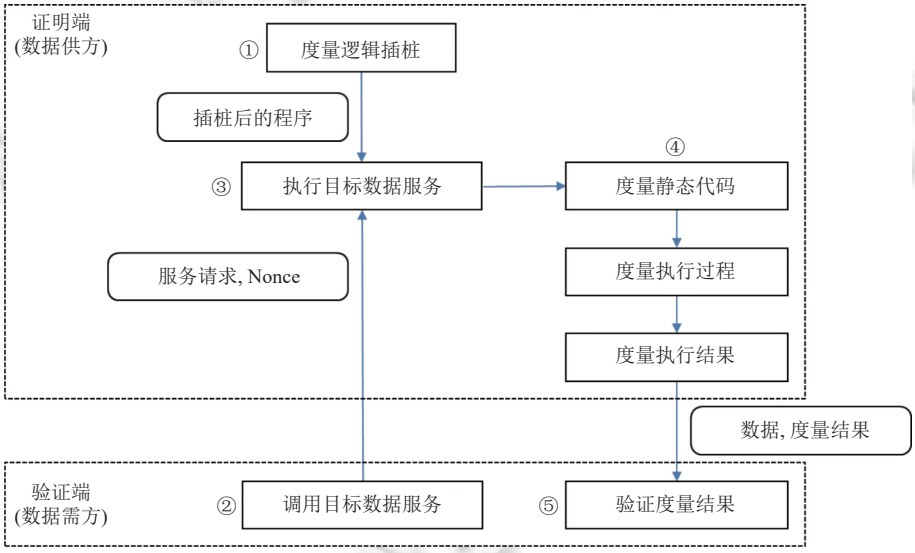


图 4 数据服务完整性多维度量的流程图

需要说明的是, 静态代码度量值的度量对象是插桩后的待度量程序而非原始程序, 即需要同时对插桩到待度量程序中的代码与待度量程序本身的代码进行完整性验证, 以防止插桩的代码受到篡改而影响度量过程的安全性.

2.2 多维度量的可信执行

为了确保度量过程与结果是可信的, 需要基于某个可信的基础组件 (也称为信任锚) 来实现对目标系统的相

关状态进行监测与记录 (即度量). 在本文中, 我们采用可信执行环境 (trusted execution environment, TEE) 作为信任锚以在数据供方信息系统中构造可信的远程证明报告. 我们在可信执行环境内设计了一个能够对特定程序静态代码、执行过程和执行结果进行度量并构造证明报告 (即完整性证据) 的软件, 称为可信度量引擎.

可信执行环境是一个独立的安全运行环境, 该运行环境与数据供方信息系统所处的富执行环境 (rich execution environment, REE)[33]逻辑隔离. 可信执行环境为可信度量引擎提供了安全的执行环境, 保证了可信度量引擎内相关资源和数据的机密性和完整性. 我们假设可信执行环境可以提供其宣称的安全计算能力, 即运行于其中的程序不会由于物理破坏以外的方式而受到攻击. 换言之, 在可信执行环境内执行的包括散列计算、数据加解密在内的所有操作均是可信的, 采用可信度量引擎进行度量得到的度量结果也是可信的. 此外, 利用可信执行环境中受硬件保护的私钥, 可信度量引擎可以通过对度量结果进行签名以保障结果本身的完整性. 目前, 大部分主流的 Intel、AMD 与 ARM 计算平台均已内置可信执行环境[11−13], 因此我们可以假设运行数据供方信息系统的计算机内存在一个可用的可信执行环境. 构建在可信执行环境内的可信度量引擎能够在目标数据服务运行时对其静态代码、执行过程和执行结果进行度量并使用受可信执行环境硬件保护的私钥对度量结果进行签名. 需要说明的是, 对于诸如物理攻击以及在人工录入数据时直接录入伪造数据等数据篡改方式, 由于并非发生在软件层面, 本文将不予以关注.

上述可信度量引擎需要与待证明目标程序进行交互以完成完整性多维度量, 因此我们采用程序插桩[34]的方法在数据供方信息系统中插入用于获得程序运行时信息并与可信度量引擎交互的特定处理逻辑, 从而为数据供方生成用于证明其数据数据完整性的证据, 过程如下: 对于目标数据服务, 首先我们确定其所对应的程序片段范围, 然后在其入口、出口以及其中每个基本块入口处进行度量逻辑的插桩. 这里, 我们假设已经确定了数据服务所对应程序片段的范围, 我们将在第 3.1 节中进一步介绍如何对此进行确定. 在此基础上, 对于目标数据服务入口和出口位置, 只需通过简单的程序分析即可确定; 而基本块入口的位置则可以在遍历可执行程序时通过简单的条件判定进行确定, 函数第一行、跳转语句下一行以及跳转标号所在行即为基本块入口.

图 5 展示了利用可信度量引擎对目标数据服务进行完整性度量的过程, 其中左侧部分为需要进行度量的目标数据服务, 右侧部分为运行于可信执行环境内的度量引擎, 其具体过程如下.

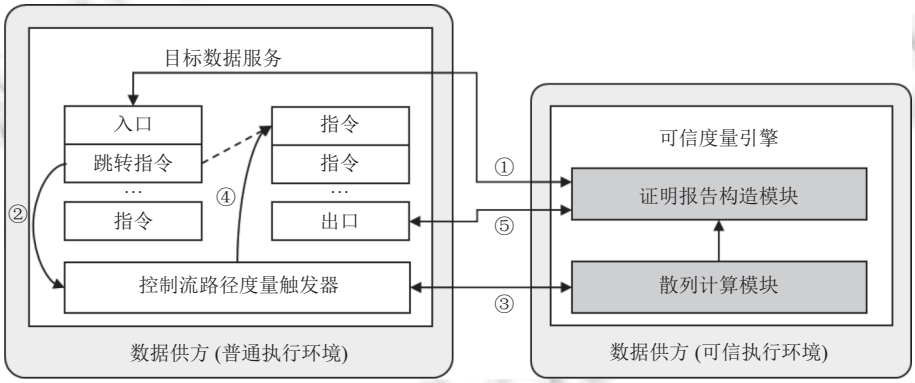


图 5　利用可信度量引擎对数据服务进行完整性度量的过程

1) 当目标数据服务开始执行时, 执行插桩代码, 向度量引擎发出“Hello”消息 (图 5 ①); 度量引擎接收“Hello”消息后, 对度量状态进行初始化, 并对目标数据服务的静态代码进行度量, 形成静态代码度量值.

2) 每当目标数据服务执行进入一个新的基本块时 (图 5 ②), 执行插桩代码, 通过度量触发器向度量引擎发送当前基本块对应的唯一基本块 ID(图 5 ③), 随后继续执行后续指令 (图 5 ④); 度量引擎接收基本块 ID 后, 根据前述执行过程度量方法进行累计散列计算.

3) 在目标数据服务执行结束前, 执行插桩代码, 向度量引擎发出“Goodbye”消息、将要返回的执行结果以及用于避免重放攻击[35]的仅使用一次的随机数 (Nonce)(图 5 ⑤); 度量引擎接收“Goodbye”消息后, 结束累计散列计算并形成执行过程度量值, 然后对执行结果进行散列计算以形成执行结果度量值; 最后, 度量引擎对静态

代码、执行过程、执行结果度量值以及 Nonce 进行散列计算, 并对计算结果进行签名, 构造数据服务完整性度量结果.

上述过程是对不包含递归或循环逻辑的目标数据服务进行度量的过程. 当目标数据服务中存在递归或循环逻辑导致的子过程时, 通过静态分析找到这些子过程的起始节点, 然后在其中通过插桩代码向度量引擎发送“Subroutine <ID> Begin”, 其中 ID 为该节点的基本块 ID; 在循环中所有指向该起始节点的节点处通过插桩代码向度量引擎发送“Subroutine <ID> End”消息, 其中 ID 为该起始节点的基本块 ID; 度量引擎接收后, 将会对期间接收到的基本块 ID 进行独立的累计散列计算并统计其执行次数, 以嵌套的方式对相关结果进行记录.

为了避免由于多线程并发而干扰对执行过程的度量, 所有发送到度量引擎的消息中都会附带一个线程 ID 用以区分不同线程的执行过程; 度量引擎在对执行过程进行度量时会为每个线程维护一系列的当前度量状态, 根据线程 ID 在相对应的度量状态上进行上述初始化、度量、签名等操作.

2.3 多维度量的结果验证

数据需方在获得所请求的数据和对应的数据服务完整性度量结果后, 将对其进行验证. 数据服务完整性度量结果中包括静态代码度量值、执行过程度量值、执行结果度量值以及签名这 4 个组成部分. 其中, 对静态代码度量值与执行过程度量值进行验证前需要事先通过离线静态程序分析确定可能的合法度量值.

对于合法静态代码度量值, 可以通过计算目标数据服务所对应程序文件的散列值得到. 该散列值应当以进行了程序插桩后的可执行文件作为计算目标, 以保障其与程序运行时由度量引擎度量得到的静态代码度量值一致, 同时也可以使得所插桩注入的代码段的完整性同样受到保障. 对于合法执行过程度量值, 算法 1 展示了对目标数据服务的控制流图进行离线静态分析以对其进行计算的算法. 执行算法前, 通过对插桩后的目标数据服务进行静态分析获得其控制流图, 并标记其中各子过程 (循环或递归) 的起始节点与终止节点. 然后, 将目标数据服务整体视作第一个子过程, 以深度优先的方式搜索控制流图 (第 2–4 行). 在搜索的过程中不断计算累计散列值, 并在搜索至当前子过程对应终止节点时, 将当前累计散列值记录为该子过程的一个合法控制流度量值 (第 8–11 行); 在搜索过程中, 每当遇到一个新的子过程起始节点时, 将会递归开始一次新的子过程搜索, 并重新计算累积散列值 (第 14–16 行). 这样, 可以为每个子过程都计算得到一个代表其中合法执行路径的累计散列值的集合, 本文中将这些集合统称为合法执行过程度量值集合. 在进行在线验证时, 每个子过程中的执行过程度量值都只与该子过程对应的合法执行过程度量值集合进行比对.

算法 1. 离线计算合法执行过程度量值.

输入: *cfg*;
输出: *measurements*.

```
1. function CALCUMEASUREMENTS(cfg)
2.     measurements ← empty map
3.     curHash ← 0
4.     TraverseCFG(cfg.entryNode, cfg.entryNode, cfg.endNode, curHash, measurements)
5.     return measurements
6. end function
7. function TraverseCFG(curNode, entryNode, exitNode, &curHash, &measurements)
8.     prevHash ← curHash
9.     curHash ← Hash(curHash, curNode.id)
10.    if curNode == exitNode then
11.        measurements.insert(curNode, curHash)
12.    else
```

```
13.      for all CFG node c in C.children do
14.        if IsSubroutineEntry(c) then
15.          curHash' ← 0
16.          TraverseCFG(c, c, c.endNode, curHash', measurements)
17.        else
18.          TraverseCFG(c, entryNode, exitNode, curHash, measurements)
19.        end if
20.      end for
21.    end if
22.    curHash ← prevHash
23. end function
```

数据需方对数据服务完整性度量结果的具体验证过程如下.

1) 使用可信度量引擎的公钥对签名进行解签得到一个散列值, 再使用数据需方保存的本次请求的 Nonce、数据服务完整性度量结果中的静态代码度量值、执行过程度量值、执行结果度量值进行散列计算得到另一个散列值; 如果二者相同, 则代表度量结果本身是真实有效的, 反之则意味着度量结果本身就存在错误, 不必进行后续验证. 这样, 可以避免攻击者利用旧度量结果进行重放攻击或冒充度量引擎提供伪造度量结果等情况.

2) 将度量结果中的静态代码度量值与预先计算得到的合法静态代码度量值进行比对以验证数据服务静态代码的完整性; 如果二者相同, 则代表目标数据服务的静态代码并未受到篡改, 否则代表受到篡改. 这样, 可以避免在目标数据服务执行前, 攻击者通过对其可执行文件进行篡改的方式, 改变其后续行为进而篡改数据的情况.

3) 将度量结果中的执行过程度量值与预先计算得到的合法执行度量值集合进行比对以验证数据服务执行过程的完整性; 如果执行过程度量值与在集合中的任一合法度量值相同, 则代表目标数据服务的执行过程并未受到篡改, 否则代表受到篡改. 这样, 可以避免攻击者在目标数据服务执行时, 利用软件漏洞修改控制数据或关键约束数据来改变其运行时行为以实现数据篡改的情况.

4) 将所接收的数据与执行结果度量值进行比对以验证数据服务执行结果的完整性; 如果二者相同, 则代表在数据供方信息系统发送执行结果到数据需方接收到执行结果的过程中, 执行结果其未受到篡改, 否则代表受到篡改. 这样, 可以避免攻击者在目标数据服务执行后, 通过 HTTP 劫持等方式修改其返回结果以实现数据篡改的情况.

在对数据服务完整性度量结果进行验证的过程中, 上述离线静态分析得到的合法度量值可以被反复使用, 无需多次计算. 然而, 当数据供方信息系统的程序规模较大时, 其控制流图可能极为复杂, 进行上述离线静态分析时可能面临状态爆炸的问题, 下文将进一步优化解决.

3 基于程序切片的数据服务完整性验证优化

根据前述数据服务完整性多维度量策略, 需要在目标数据服务的特定程序位置上插桩用于记录运行时控制流信息并与可信度量引擎交互的处理逻辑. 在现有的动态远程证明方案中, 此类插桩通常是针对目标系统的完整程序进行的, 这种方式会带来较大的代码膨胀率和额外开销. 一方面, 复杂业务处理程序的控制流图通常极为复杂, 对其完整的可执行程序进行静态程序分析来获得合法度量值易导致状态爆炸, 带来极大的验证开销; 另一方面, 由于模块化复用是此类系统设计中的一种常见做法, 上述被插桩了度量逻辑的程序模块可能会在与该数据服务无关的其他业务功能执行时被调用, 这可能会导致不必要的额外开销并对正在构建的数据服务完整性度量结果产生干扰.

为了解决上述问题, 本文对目标数据服务所在原始程序进行程序切片, 仅对目标数据服务所对应程序片段而非完整原始程序进行前述完整性验证. 具体来说, 首先, 我们执行目标数据服务并对原始程序的执行进行监测, 结合动态与静态分析定位目标数据服务所对应程序片段, 以此对需要度量的对象进行裁剪, 缩小完整性验证涉及的

程序范围以降低开销; 其次, 我们进一步对上述定位到的程序片段进行切片, 构造一个与该数据服务业务逻辑一致但不会被其他业务功能复用的等效可执行代码片段来进行前述多维度量, 以此将包含完整性度量的数据服务执行过程与其余业务执行过程隔离, 避免代码复用导致的干扰. 图 6 展示了上述流程.

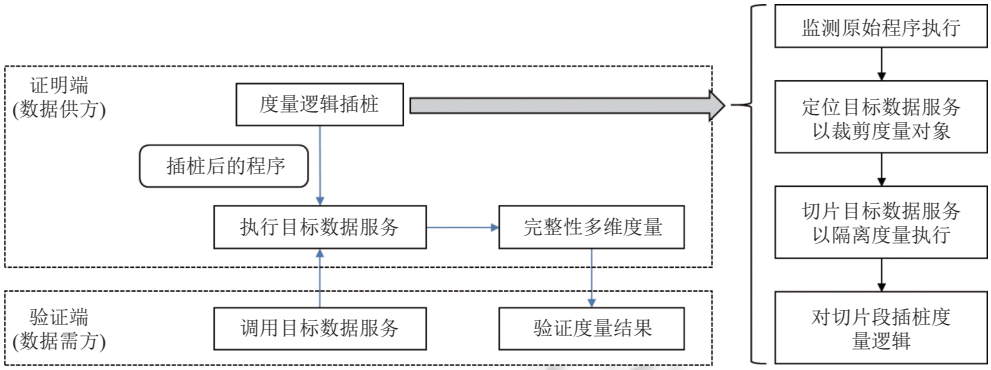


图 6 优化后的度量逻辑插桩流程

3.1 基于运行时行为模型的度量对象裁剪

考虑到真实系统多为没有源代码、文档和开发商支持的“黑盒”系统, 为了提高本文技术的适用性, 需要在缺少相关文档资料的情况下对目标系统进行程序分析, 确定和目标数据服务所对应的程序片段, 从而完成对度量对象的裁剪. 因此, 难以直接通过静态程序分析确定该片段, 而是需要首先进行动态分析, 执行目标数据服务并对其执行过程进行监测, 然后再基于监测结果进行进一步分析. 需要说明的是, 在这一动态分析的过程中, 也需要对目标程序进行一定的插桩, 但这一插桩是临时性的, 仅会在对目标数据服务程序片段进行定位时生效. 而前文所介绍的用于数据服务完整性度量逻辑的插桩则是永久性的, 将在完成数据服务程序片段定位与切片后, 作用于切片片段并长期生效.

为了进行精准高效的程序切片, 本文首先定义一个运行时行为模型 (runtime behavior model, *RBM*) 来描述一个应用程序在一段时间内的程序行为, 然后设计运行时行为模型的构造方法, 并基于所构造的模型进行目标数据服务片段的定位, 最后对定位的程序片段进行切片.

3.1.1 运行时行为模型的定义与构造

当应用程序运行时, 代码段的执行会引起其他内存区域和寄存器的变化. 应用程序的运行时行为模型应该能够在一段时间内从两个方面对应用程序做出反应: 应用程序的执行行为及其执行所需的数据. 应用程序的运行时控制流代表其运行时执行程序语句的顺序, 是其静态控制流图 (control flow graph, CFG)[32]的一条或多条 (多线程时) 路径; 而运行时数据流则是其运行时数据变量的值随程序执行的变化过程, 可以理解为数据流图 (data flow diagram, DFD)[36]中的一组数据在程序运行不同时刻的值. 对于一段程序在一定时间内的执行行为, 可以通过运行时控制流对其进行描述; 对于该执行过程所需的数据, 则可以通过运行时数据流进行描述. 一般来说, 对于运行时控制流可以做到完全记录, 而对于运行时数据流的记录由于所涉及数据变量较多且其值的变化可能极为频繁, 因此通常只能选择一些关键数据变量和关键执行节点进行记录.

因此, 在本文提出的运行时行为模型中, 我们对目标程序的运行时控制流和运行时数据流进行描述以反应应用程序的执行行为及其执行所需的数据, 其中对控制流的描述粒度是基本块级 (basic block, BB)[31]或函数级的, 对数据流的描述粒度是函数级的, 即在函数入口和出口处记录函数的调用参数和返回值以及部分其他变量 (通常为重要的全局或静态变量) 在该时刻的值, 图 7 展示了运行时行为模型的结构示意图. 下面给出运行时行为模型的形式化定义.

定义 1. 运行时行为模型是对应用程序运行时行为的抽象, 由一组基本块 B、一组数据变量状态 D、一个表

示两个基本块之间执行先后顺序的关系 ε 以及一个表示基本块和数据对象状态之间关联的关系 σ 组成.

$$RBM=(B,D,\varepsilon,\sigma).$$

- $B=\{bb_i\}$ 是基本块的集合, 其中 bb_i 表示一个由一组顺序执行的语句组成的基本块.
- $D=\{(id,address,data)_i\}$ 是数据变量运行时状态的集合, 其中 $(id,address,data)_i$ 表示变量的编号、变量在内存中的地址和变量的记录值.
- $\varepsilon\subseteq B\times B$ 表示两个基本块之间的执行顺序关系.
- $\sigma\subseteq B\times D$ 表示一个基本块与相关数据变量状态的对应关系.

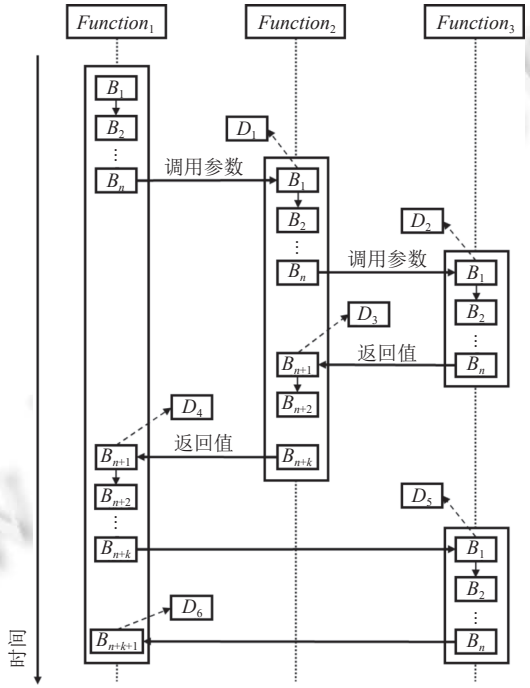


图 7 运行时行为模型的结构示意图

运行时行为模型的构建过程主要需要解决以下 3 个挑战: 1) 应用程序的执行过程较为复杂, 如何获得相关的运行时信息; 2) 应用通常使用多线程来提供更好的用户体验, 如何记录不同线程的控制流之间的依赖关系; 3) 根据定义, 运行时行为模型涵盖了一段时间内执行行为的所有变化以及应用程序所需的数据, 建模时如何解决状态爆炸问题.

为了克服上述挑战, 我们采取以下措施.

1) 实现了一个可以在运行时对目标程序进行按需动态插桩的监测器以记录目标程序运行时的相关控制流和数据流信息. 对于控制流信息, 以基本块为粒度进行插桩以记录所执行的基本块序列; 对于数据流信息, 在函数调用和函数返回的语句前进行插桩以记录函数调用参数和函数返回值. 我们还插桩了一些系统方法, 其中包括将系统消息发送给应用中控件的方法, 这意味着用户和应用程序之间的交互也能够包含在运行时行为模型中, 图 8 展示了上述记录的结构示意图.

2) 通常, 一个线程对应于一个控制流记录, 不同线程所对应的控制流之间的依赖关系可以分为两类: 同步依赖和通信依赖. 同步依赖表示由于线程间同步而产生的依赖关系, 通信依赖则表示由于线程间通信而引起的依赖关系. 通常, 同步依赖与函数调用和函数返回指令相关联, 而通信依赖则与特定通信接口的访问相关联, 依赖关系可以在运行时信息记录过程中进行推断, 并在运行时行为模型中由 ε 关系进行表示.

3) 考虑到对目标程序的运行时控制流和数据流进行完整记录的时间和空间复杂性过高, 因此我们设计了一个

多级过滤器来缓解状态爆炸问题, 其中包括交互级过滤器、类级过滤器和方法级过滤器. 交互级过滤器通过控件的类型和标题 (例如按钮, 文本框) 过滤与特定功能无关的行为. 如前所述, 表示用户和应用程序之间交互的系统消息会被包含在运行时行为模型中. 当用户与应用程序交互以触发特定功能时, 诸如鼠标点击或按键键入的事件将转换为系统消息并传递到应用程序的某些特定控件中. 然后, 这些消息会触发控件的某些特定响应过程的执行, 从而触发目标程序的特定业务功能. 因此, 可以通过分析用户行为与应用程序控件响应之间的相关性, 将运行时行为模型的规模缩小到特定业务功能. 此外, 类级过滤器可以通过类名的黑白名单过滤开发人员不关注的类 (如工具类); 方法级过滤器则可以通过方法名的黑白名单过滤开发人员不关注的方法. 开发人员可以按需配置多级过滤器, 在不同情况下对目标程序进行适当范围和精度的监测以构造运行时行为模型. 一方面, 对于与目标功能无关的行为, 例如对网络连接的常规检查, 可以在监测过程中对其进行过滤; 另一方面, 可以首先对目标程序进行粗粒度监测与分析, 缩小目标数据服务程序片段所在的范围后, 再对该范围进行细粒度监测与分析.

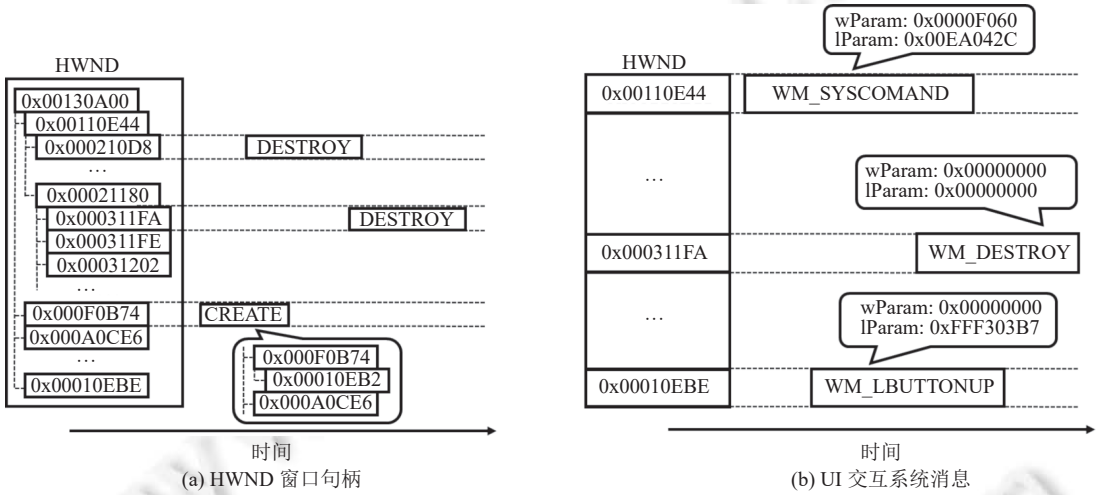


图 8 UI 交互记录的结构示意图

3.1.2 基于模型分析定位程序片段

数据供方通过对运行时行为模型的构造和分析定位特定数据服务所对应的程序片段, 虽然运行时行为模型已经对目标程序的执行过程进行了较好的抽象表示, 但是由于大型信息系统的复杂性, 其运行时行为模型仍较为复杂. 此外, 由于编译过程的信息丢失和可能存在的代码混淆, 目标程序可执行文件中的函数名、变量名等信息通常存在语义信息缺失的问题, 进而导致运行时行为模型中的相关部分也存在语义缺失, 这进一步增加了定位目标程序片段的难度.

针对模型结构复杂和语义信息缺失的问题, 我们提出了一种基于运行时信息的数据服务程序片段定位算法, 其核心思想是利用数据供方提供的与特定业务功能执行相关的业务数据 (例如, 执行业务功能的过程中键入的内容或服务调用参数、业务功能执行过程中或结束后人机交互界面显示的内容或服务返回结果), 通过将模型中记录的运行时数据流信息与其进行相似度匹配确定关键数据流位置, 并据此对控制流进行进一步的静态分析, 最终确定特定数据服务所对应的代码片段.

算法 2 展示了该程序片段定位算法. 首先, 对目标程序进行用于运行时行为模型建模的插桩, 然后执行目标数据服务以构建模型, 并获得数据服务的执行结果 (第 4–7 行). 然后, 根据该次数据服务执行的调用参数和返回结果, 与运行时行为模型中的数据流信息进行匹配, 确定目标数据服务的入口基本块和出口基本块 (第 8–13 行). 最后, 在目标程序的静态控制流图中, 分别以入口基本块和出口基本块为起点, 通过两次深度优先搜索寻找所有从入口可以到达的基本块以及所有可以到达出口的基本块, 两次搜索结果的交集即为所有以入口基本块为起点、以出口基本块为终点的路径所对应的程序片段, 也就是需要定位的目标数据服务所对应程序

片段.

算法 2. 定位数据服务的程序片段.

输入: *program, input*;
输出: *targetSlice*.

```
1. function LocateDataAPISlice(program, input)
2.    targetSlice ← empty set
3.    for all basic block bb in program do
4.       InstrumentForLocating(bb)
5.    end for
6.    RBM, output ←ExecuteDataAPI(program, input)
7.    for all variable record d in RBM.D do
8.       if entry == null and Match(d.data, input) then
9.          entry ← RBM.σ[d.id]
10.      end if
11.      if Match(d.data, output) then
12.         exit ← RBM.σ[d.id]
13.      end if
14.   end for
15.   cfg ←GetControlFlowGraph(program)
16.   slice1 ← GetReachableBBs(cfg, entry)
17.   slice2 ← GetReverseReachableBBs(cfg, exit)
18.   targetSlice ← slice1 ∩ slice2
19.   targetSlice.insert(entry)
20.   targetSlice.insert(exit)
21.   return targetSlice
22. end function
```

3.2 基于等效代码片段的度量执行隔离

在确定了目标数据服务所对应的程序片段之后, 我们需要对其进行程序切片, 通过在目标程序中构造一段与目标数据服务具有相同执行效果但与其他业务功能解耦的代码片段, 进而以该片段而非原始程序作为数据服务执行过程的度量对象的方式实现度量执行过程的隔离, 避免度量执行过程与复用了被度量服务中部分代码的其他业务功能间的不必要干扰. 为此, 首先需要构造出与其他业务功能解耦的代码片段, 然后再为其恢复执行所需的上下文环境以使其可执行.

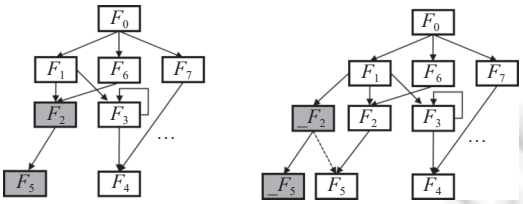
3.2.1 构造等效代码片段

首先, 对目标数据服务的程序片段中的全部基本块进行复制并插桩到目标程序中. 具体来说, 对于上述基本块, 我们首先确定其所在函数的集合, 然后将这些函数复制为以_cfa_前缀的新函数, 从新函数中删除多余不在目标数据服务片段中的基本块, 并将新函数插桩到原函数所在的类中.

由于对函数的复制仅是对函数本身进行的复制, 新函数的调用关系仍指向原始函数调用关系对应的函数, 即原始函数和复制后的函数仍然指向相同的函数. 因此, 需要对复制后的函数之间的调用关系进行重构, 从而避免执行复制后的程序片段时出现控制流错误而影响度量结果. 为此, 根据目标数据服务所对应原始程序片段的静态函数调用图中的函数间调用关系, 遍历新程序片段中的所有用于定义函数调用关系的指令, 找到表示该原始函数间

用于定义函数调用的指令, 并将该指令中的原始函数更换为相应的原始函数经过复制得到的函数.

图 9 对上述复制与重构的步骤进行了示意. 当复制 F_2、F_5 得到 $\_F_2$ 和 $\_F_5$ 后, 此时 $\_F_2$ 与 F_5 之间仍存在调用关系, $\_F_2$ 和 $\_F_5$ 之间不存在调用关系, 需要将 $\_F_2$ 对 F_5 的调用重构为 $\_F_2$ 对 $\_F_5$ 的调用. 此外, F_1 和新程序片段的入口函数 $\_F_2$ 之间也并不存在调用关系, 需要为其添加调用关系以使得当需要进行数据服务执行情况的度量时, F_1 能够对 $\_F_2$ 进行调用.



(a) 重构调用关系前 (b) 复制并重构调用关系后

图 9 函数调用关系重构的示意图

此外, 我们还需要对新函数内基本块之间的跳转关系也进行重构, 具体重构过程与函数间调用关系的重构类似, 即参照原跳转关系将新函数内基本块中跳转语句的目标地址进行替换, 对此不再进行赘述. 这样, 通过对这段新构建的等效代码片段而不是目标数据服务的原始代码片段进行用于构造完整性度量结果的程序插桩, 就可以避免由于完整性度量与其他耦合业务功能之间互相干扰的问题.

3.2.2 恢复执行所需上下文

通过前述的目标数据服务程序片段复制和调用/跳转关系重构, 我们获得了一个与原始数据服务程序片段具有相同执行效果的新代码片段. 为了确保生成的等效代码片段的可执行性, 还需要正确构建代码片段执行所依赖的上下文环境, 即为代码片段中各个变量构建正确的数据依赖关系.

数据依赖表示程序中对某变量进行引用的语句对定义该变量语句的依赖, 即一种“定义-引用”依赖关系. 存在数据依赖的变量可以分为 3 种类型: 函数内的局部变量、类的非静态成员变量和全局变量/类的静态成员变量 (一些编程语言中不存在全局变量, 但可以以静态成员变量的形式实现全局变量的效果). 其中, 对于类的非静态成员变量和全局变量/类的静态成员变量, 由于进行等效代码片段构造时将各基本块复制到了其所在的类中, 因此对这些类型变量的访问仍可以正常进行. 对于函数内的局部变量, 由于进行代码片段复制时可能丢失对其的声明和赋值, 因此需要为其构建正确的数据依赖关系. 数据依赖图 (data dependence graph, *DDG*) 是一种表示程序语句或基本块间数据依赖关系的有向图. 通过对目标程序的静态分析可以构建数据依赖图 $DDG=(V, E)$, 其中 V 表示程序中所有语句对应节点的集合, E 表示语句之间数据依赖关系的集合, 进而据此为局部变量恢复丢失的相关指令以构建正确的数据依赖关系. 这样, 可以使得所得到的新代码片段在执行时具有正确的上下文环境, 从而保障其可执行性.

通过对上述程序切片得到的等效可执行代码片段而非原始数据服务进行用于度量的程序插桩, 即可将对该数据服务进行度量的执行过程与其余业务功能隔离, 避免代码复用导致的干扰.

4 实验分析

本节通过一系列的实验对所提出方法的有效性和性能进行评估.

4.1 原型系统实现

Java 是 GitHub 中 Web 服务相关项目中被采用最多的开发语言[37], 因此我们针对 Java 语言环境实现了原型系统 DataAttest. 图 10 展示了 DataAttest 的系统架构概览. 其中, 度量引擎实现为一个运行在 Intel SGX 飞地 (Enclave)[11]内的 C++程序, 其中使用密码散列算法 BLAKE2s[38]进行散列计算和累计散列计算; 程序监测器和程序重构器基于 Java 字节码操纵框架 ASM[39]进行实现, 其中通过 JNI (Java native interface) 与可信度量引擎交互, 通过 JVM Attach

和 JavaAgent 机制进行字节码插桩 (用于实现程序监测和运行时度量) 和运行时字节码获取 (用作被度量的静态代码); 数据服务完整性验证器基于 Java 静态分析框架 Soot[40]进行合法度量值的离线静态分析与在线验证.

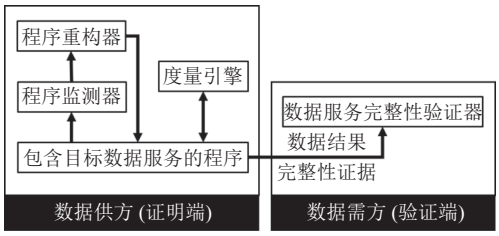


图 10 DataAttest 的系统架构概览

4.2 实验方法

实验使用了 3 个真实的 Java 应用, 对其中总计 25 个数据服务应用了 DataAttest, 应用和数据服务的具体信息如表 1 所示. 第 1 个应用是某省财政管理一体化信息系统, 它主要对国库财政项目的支出申请进行录入、送审、审核等操作, 其中包括与上述环节相关的 18 个数据服务, 用于向上级部门提供业务数据以进行审计. 第 2 个应用是某自治州事业单位登记系统, 它主要对事业单位设立、变更、注销等事项进行管理, 其中包括与上述环节相关的 6 个数据服务, 用于向政务“一门式”服务平台提供数据以打通业务流程. 第 3 个应用是某企业 ERP 系统, 它主要对企业采购计划进行管理, 其中仅包括采购应付单查询一个数据服务, 用于向企业内另一管理系统提供数据以打通业务流程. 上述 3 个应用的运行环境均为 Java 6.

表 1 实验所用的 Java 应用和所涉及的数据服务

应用名称	数据服务名称	数据服务功能
某省财政管理一体化信息系统	finance/directPay/getPlanList	获取直接支付用款计划列表
	finance/directPay/getJjflIdList	获取直接支付经济分类列表
	finance/directPay/getJsfsIdList	获取直接支付结算方式列表
	finance/directPay/getSendRequestList	获取直接支付待送审申请列表
	finance/directPay/getAuditRequestList	获取直接支付待审核申请列表
	finance/accreditPay/getPlanList	获取授权支付用款计划列表
	finance/accreditPay/getJjflIdList	获取授权支付经济分类列表
	finance/accreditPay/getJsfsIdList	获取授权支付结算方式列表
	finance/accreditPay/getSendRequestList	获取授权支付待送审申请列表
	finance/accreditPay/getAuditRequestList	获取授权支付待审核申请列表
	finance/paycard/getCardIdList	获取公务卡报销公务卡列表
	finance/paycard/getSubmitRecordList	获取公务卡报销待确认消费明细列表
	finance/paycard/getRecordList	获取公务卡报销已确认消费明细列表
	finance/paycard/getPlanList	获取公务卡报销用款计划列表
	finance/paycard/getJjflIdList	获取公务卡报销经济分类列表
	finance/paycard/getSendRequestList	获取公务卡报销待送审申请列表
	finance/paycard/getAuditRequestList	获取公务卡报销待审核申请列表
	finance/paycard/getCertificateList	获取公务卡报销待生成凭证列表
某自治州事业单位登记系统	register/shelidengji/getList	获取设立登记列表
	register/shelidengji/getAttachDetail	获取设立登记附件详情
	register/biangengdengji/getList	获取变更登记列表
	register/biangengdengji/getAttachDetail	获取变更登记附件详情
	register/zhuxiaodengji/getList	获取注销登记列表
	register/zhuxiaodengji/getAttachDetail	获取注销登记附件详情
某企业ERP系统	erp/getBillList	查询应付账单列表

上述实验使用一台桌面个人计算机作为数据服务完整性验证端. 它的 CPU 是 Intel Core i5-9400F 2.90 GHz, 内存 16 GB, 运行的操作系统是 Ubuntu 16.04 (AMD64), JDK 版本是 1.8. 实验以 3 个应用项目合作方各自的内网服务器作为证明端 (即数据供方信息系统所在环境), 其操作系统为不同版本的 Windows Server (2008 年和 2012 年), 所运行的 JDK 版本均为 1.6. 由于目标应用均为带有严格权限限制的企业或政府部门内网应用, 因此实验过程中无法统一其运行环境. 由于后续实验主要关注于对 DataAttest 所引入额外开销相对于原始开销的比例而非绝对值, 对于同一目标应用的与完整性度量相关的多次实验会始终在同一台服务器中进行, 因此服务器配置的不同不会对实验结果产生较大影响. 实验中验证端通过 HTTPS 协议访问证明端对外提供的数据服务.

4.3 实验结果与分析

为了评估 DataAttest 在数据服务完整性验证方面的有效性与性能, 我们对以下几个方面进行进一步的实验.

- RQ1: DataAttest 是否可以应用于真实的数据服务场景并有效发现各类攻击?
- RQ2: 应用 DataAttest 会带来多少额外开销?
- RQ3: DataAttest 中采用的基于程序切片的优化策略是否能够有效降低额外开销?

4.3.1 RQ1: DataAttest 是否可以应用于真实的数据服务场景并有效发现各类攻击?

我们以某省财政管理一体化信息系统中的一个用于获取当前待送审支付申请信息列表的数据服务为例, 展示 DataAttest 的工作流程并验证其检测各类攻击的有效性. 表 2 展示了该数据服务的具体信息. 显然, 对于此类财政系统的数据, 无论是用于财务审核还是其他业务环节, 一旦其数据存在完整性问题, 带来的后果都是十分严重的.

表 2 finance/accreditPay/getSendRequestList 数据服务的详情

参数名	参数类型	参数描述
voucher_no	String	申请单号
create_date	String	申请日期
create_user_name	String	申请人
en_code	String	预算单位编码
en_name	String	预算单位名称
bs_code	String	支出功能分类编码
bs_name	String	支出功能分类名称
bsi_name	String	经济分类
pay_money	String	申请金额
pay_summary_name	String	摘要
payee_account_name	String	收款方全称
payee_account_bank	String	收款方开户行
payee_account_no	String	收款方账号
bo_name	String	预算类型
mk_name	String	资金性质
bl_name	String	指标来源
mb_name	String	业务处(科)室
pk_name	String	支付方式

首先, 通过程序监测器对目标系统进行用于运行时信息记录的临时性程序插桩, 该程序插桩的内容在监测完成后将不会存在, 不会对目标系统后续的使用产生任何影响. 然后, 由业务人员执行该数据服务, 在这一过程中程序监测器对目标系统的执行情况进行监测, 并基于运行时行为模型记录其动态控制流和数据流信息. 完成监测后, 通过程序重构器对监测结果进行分析以定位目标数据服务所对应的程序片段, 并对其进行切片和度量逻辑的插桩. 在这一过程中, 需要业务人员提供进行监测时对该数据服务进行调用的参数以及最终得到的数据结果, 以供程序重构器完成对目标数据服务程序片段的定位. 完成重构后, 再次执行该数据服务, 可信度量引擎就会在数据服务

运行时与其交互并为其生成对应于该次执行的数据服务完整性度量结果, 该度量结果会随数据服务的执行结果一同返回至数据服务的调用方. 图 11 展示了该度量结果, 其中 data 字段是数据服务所返回的数据结果, DAIR 字段是完整性多维度量结果, 包括静态代码度量值 CM、执行过程度量值 EM、执行结果度量值 RM 以及可信度量引擎对上述度量结果的签名.

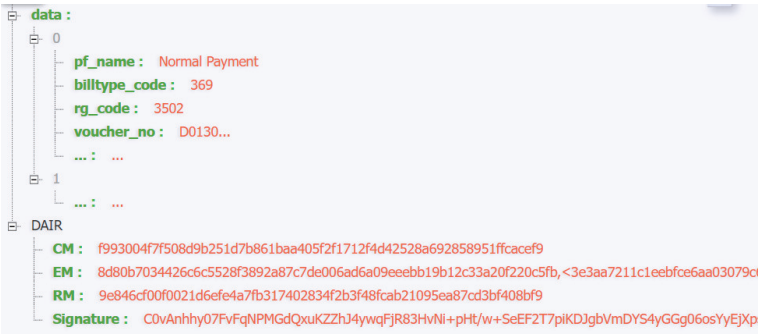


图 11 数据服务调用结果和完整性度量结果的示例

在得到数据服务的执行结果以及完整性度量结果后, 通过数据服务完整性验证器基于该度量结果中的内容对所调用数据服务的完整性进行验证. 需要注意的是, 在首次对一个数据服务的完整性进行验证前, 需要对该数据服务执行静态程序分析以确定验证所需的合法度量值. 为了验证 DataAttest 应对各类攻击的有效性, 我们模拟了 3 种能够篡改数据的攻击. 根据 Kuang 等人[41]的调研, 远程证明中的对抗模型包括静态软件攻击、运行时软件攻击、物理攻击、拒绝服务攻击以及瞬态恶意软件攻击这 5 方面. 其中, 物理攻击通常通过物理手段进行抵御, 不在本文对抗模型范围内; 拒绝服务攻击与软件完整性无关; 瞬态恶意软件攻击则天然的可以被动态远程证明抵御. 因此, 本文只需针对静态软件攻击与运行时软件攻击进行测试. 静态软件攻击主要通过代码注入, 即替换或添加恶意软件片段的方式篡改程序静态代码, 以实现攻击; 动态软件攻击则主要通过以面向返回编程 (ROP) 攻击和面向跳转编程 (JOP) 攻击为代表的代码重用攻击 (CRA) 篡改程序执行过程. 在本文中, 我们以模拟攻击 1 代表静态软件攻击, 以模拟攻击 2 代表动态软件攻击 (ROP 与 JOP 攻击均依赖于控制流篡改), 以模拟攻击 3 代表数据服务场景下可能额外面临的网络劫持攻击, 从而较为全面的验证本文方法的有效性, 攻击内容和验证结果如表 3 所示.

表 3 3 种模拟攻击和验证结果

攻击对象	攻击方式	攻击内容	验证结果
静态代码	程序篡改	在数据服务运行前修改其可执行程序, 使其返回伪造的数据	检测到静态代码度量值与合法值不同, 表明代码受到修改
执行过程	控制流攻击	在数据服务运行时修改一个跳转语句的目标地址, 改变其控制流	检测到执行过程度量值不属于合法度量值集合, 表明执行过程受到攻击
执行结果	HTTPS劫持	通过代理劫持HTTPS请求, 并修改数据服务的返回数据	检测到执行结果度量值与返回数据的散列值不同, 表明返回结果受到篡改

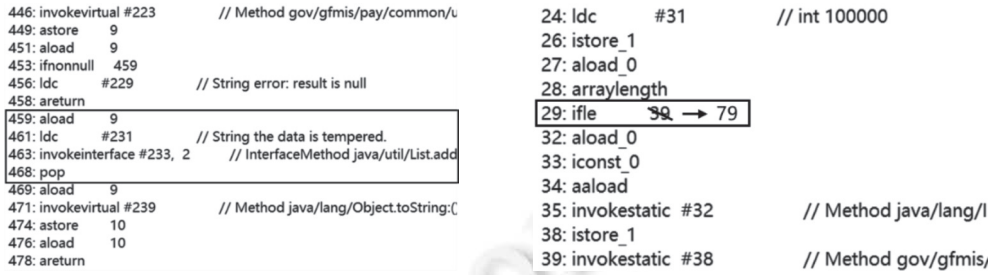
表 3 中, 攻击 1 是对目标数据服务静态代码完整性的攻击, 图 12(a) 展示了其具体攻击内容, 它直接修改了目标数据服务的可执行程序, 在其返回的数据中注入了一段字符串“the data is tempered.”. 对此, DataAttest 在验证数据服务完整性度量结果时检测到静态代码度量值与合法值不同, 这表明数据服务的静态代码可能受到过恶意修改, 成功发现到了攻击 1 导致的完整性问题.

攻击 2 是对目标数据服务执行过程完整性的攻击, 图 12(b) 展示了其具体攻击内容, 它在运行时修改了一条 ifle 跳转指令的目的地从而改变了数据服务的控制流, 跳过了部分参数的初始化, 使得数据服务仅返回了部分数据. 由于此类控制流攻击是发生在运行时的, 并没有修改目标数据服务的静态代码, 因此其能够通过对静态代码完

整性的验证. 但是, DataAttest 检测到完整性度量结果中执行过程度量值不属于合法执行过程度量值集合, 这表明数据服务执行了非法的控制流路径, 成功发现到了攻击 2 导致的完整性问题.

攻击 3 是对目标数据服务执行结果完整性的攻击, 它基于网络调试器 Fiddler[42]的 bpafter 命令模拟了 HTTPS 代理劫持来修改数据服务返回的数据. 此类攻击能够通过对静态代码完整性和执行过程完整性的验证, 但是 DataAttest 检测到完整性度量结果中执行结果度量值与接收到的数据结果的散列值不同, 这表明数据结果与数据服务执行完成时产生的结果不同, 成功发现到了攻击 3 导致的完整性问题.

上述实验结果表明, DataAttest 能够应用于真实世界的数据源 Java 应用并保障其对外提供数据过程的完整性.



(a) 对静态代码完整性的攻击内容 (b) 对执行过程完整性的攻击内容

图 12 对目标数据服务进行攻击的具体内容

4.3.2 RQ2: 应用 DataAttest 会带来多少额外开销?

我们将 DataAttest 应用于 3 个真实的数据供方 Java 应用, 并记录了代码膨胀率和时间开销. 需要说明的是, 由于 DataAttest 所引入的额外内存开销仅为 KB 级, 这对于服务器端环境而言是可以忽略的, 因此我们没有对其进行展开讨论.

表 4 展示了目标应用重构前后的应用大小、字节码数. 可以看到, 对于仅重构了一个数据服务的某企业 ERP 系统, 其应用膨胀率仅为 0.03%, 字节码数膨胀率仅为 0.30%; 对于重构了 6 个数据服务的事业单位登记系统, 其应用膨胀率和字节码数膨胀率为 0.18% 和 2.30%; 对于重构了 18 个数据服务的财政支付管理系统, 其应用膨胀率和字节码数膨胀率为 0.33% 和 3.87%. 由于此类系统的 jar 文件中通常包含大量非字节码 (即 class 文件) 的资源文件, 因此重构后其应用膨胀率通常远低于字节码数膨胀率. 可以看到, 重构所带来的代码膨胀率整体上是极低的, 即使在重构了 18 个数据服务的系统中也仅为 3.87%, 并且单个数据服务重构引入字节码膨胀率均小于 0.4%.

表 4 DataAttest 应用于数据供方 Java 应用的代码膨胀率

度量项	财政支付管理系统	事业单位登记系统	某企业ERP系统
原始应用全部jar文件的大小 (KB)	50 790	24 488	38 974
重构后应用全部jar文件的大小 (KB)	50 957	24 533	38 985
重构后应用的大小增长 (KB)	167	45	11
重构后的应用膨胀率 (%)	0.33	0.18	0.03
原始应用全部类方法所对应的字节码数	999 542	466 882	842 608
重构后应用全部类方法所对应的字节码数	1 038 224	477 605	845 156
重构后字节码数的增长	38 682	10 723	2 548
重构后的字节码数膨胀率 (%)	3.87	2.30	0.30
重构的数据服务数	18	6	1

需要说明的是, 由于 DataAttest 中采用了基于程序切片的重构, 因此重构所带来的额外字节码数除了来自与可信度量引擎交互以实现相关度量的逻辑以外, 还来自程序切片所构建的独立可执行代码片段. 以财政支付管理系统中的 Accredit_sendRequest_getRequestList 数据服务为例, 表 5 展示了其重构具体细节. 在重构导致的 2 397 行字节码增长中, 由程序切片片段导致的增长数为 1 829, 主要来自对目标数据服务原始程序片段的拷贝; 由度量逻

辑导致的增长数为 568, 主要来自用于与度量引擎交互的类的代码以及在 125 个基本块中插桩的相关度量逻辑. 由于 DataAttest 仅对精确定位的目标数据服务程序片段进行切片与插桩, 因此其对原始系统的修改是轻量级的.

表 5 DataAttest 重构数据服务的具体细节

度量项	finance/accreditPay/getSendRequestList
重构导致的字节码数增长	2 397
程序切片片段导致的字节码数增长	1 829
度量逻辑导致的字节码数增长	568
目标数据服务片段中的基本块数	125

此外, 我们还在 3 个应用中各选择了一个数据服务对 DataAttest 引入的额外时间开销进行了评估, 结果如表 6 所示. 对于 3 个数据服务, 数据服务完整性度量所引入的额外时间开销分别为 35 ms (19.13%)、24 ms (17.52%) 和 42 ms (18.18%). 我们使用 t_{code}、t_{path}、t_{result} 和 t_{sign} 分别代表计算一次静态代码散列值的时间、计算一次控制流执行路径散列值的时间、计算一次执行结果散列值的时间和计算一次签名的时间, 则上述额外时间开销 $t_{attest}=t_{code}+n\times t_{path}+t_{result}+t_{sign}$, 其中 n 为控制流事件 (例如跳转、函数调用、函数返回) 的数量. 因此, 完整性度量的开销主要取决于目标数据服务执行时的控制流事件数量, 即当存在大量分支、循环、函数调用逻辑时该开销也会随之较大. 更具体的, 每个控制流事件都会导致度量触发器的调用、与可信执行环境的上下文切换和度量引擎中度量算法的执行, 三者的开销占比约为 10%、20% 和 70%, 大部分开销来自度量算法本身.

表 6 DataAttest 引入的额外时间开销

度量项	finance/accreditPay/getSendRequestList	register/shelidengji/getList	erp/getBillList
原始数据服务执行的时间开销 (ms)	183	137	231
重构后数据服务执行时间开销 (ms)	218	161	273
重构后引入的额外时间开销占比 (%)	19.13	17.52	18.18
合法度量值离线分析时间开销 (s)	4.87	3.72	4.23
在线验证的时间开销 (ms)	45	42	47

在合法度量值离线静态分析的时间开销方面, 3 个数据服务分别为 4.87 s、3.72 s 和 4.23 s, 此开销主要由静态代码散列值计算、静态控制流图构建以及合法控制流路径散列值计算这 3 部分组成, 其中静态代码散列值计算的时间复杂度为常数级, 静态控制流图构建和合法控制流路径散列值计算的时间复杂度均与目标程序片段中基本块数量正相关. 由于采用确定入口与出口的深度优先搜索遍历所有合法执行路径, 并且控制流图中的环被视为子过程独立度量, 因此合法控制流路径散列值的计算时间开销与目标程序片段中的基本块数量成正比.

在在线验证的时间开销方面, 3 个数据服务分别为 45 ms、42 ms 和 47 ms, 此开销主要来自签名验证、对所获得的数据结果的散列值计算以及对合法度量值的搜索比对. 其中, 前二者的时间复杂度为常数级, 对合法度量值的搜索比对的时间复杂度则为 $O(n\times\log n)$, n 为合法控制流路径度量值的数量.

整体来看, 3 个数据服务中的额外开销占比均在 20% 以内, 完整性度量和在线验证的总开销均在 100 ms 以内, 这对于时延相对不敏感的数据流通场景来说是完全可以接受的.

4.3.3 RQ3: DataAttest 中采用的基于程序切片的优化策略是否能够有效降低额外开销?

为了验证 DataAttest 中采用的基于程序切片的优化策略的有效性, 我们以某企业 ERP 系统为例, 以采用和不采用该策略两种方式对 QueryBill 数据服务进行了实验. 图 13 展示了采用该优化策略后定位并切片得到的部分代码片段, 极大地缩小了进行完整性验证的程序范围, 并通过函数间调用关系重构与上下文环境恢复使度量过程得到有效隔离.

图 14 展示了实验结果. 可以看到, 在字节码膨胀率方面, 采用该策略进行局部验证后从 12.37% 降低至 0.30%, 减少了 97.57%. 在同一应用中重构的数据服务数量增加时, 局部验证所引入的字节码膨胀率也会随之增

加, 但根据前文实验中的结果来看, 整体上仍会远低于不采用基于程序切片的优化策略时的字节码膨胀率. 在目标数据服务执行时引入的额外时间开销占比方面, 采用该策略后从 19.04% 降低至 18.18%, 减少了 4.52%. 这是因为在目标数据服务的执行期间, 该 ERP 系统中同时有其他线程正在运行, 这些线程也触发了插桩注入的度量逻辑, 从而影响了度量引擎的响应度量请求的效率, 增加了目标数据服务的执行时间. 需要说明的是, 虽然是否采用基于程序切片的优化策略只对目标数据服务的执行时间带来了较小的优化, 但其余线程所触发的度量逻辑极有可能会导致执行过程度量出错, 如果不采用该策略将需要进行较为复杂的设计和处理才能规避.

```
Class<?> PaginationQueryVO_class =
  Class.forName("nc.vo.pubapp.pagination.PaginationQueryVO", true, provisionClassLoader);
Method getAllpks_method =
  PaginationQueryVO_class.getDeclaredMethod("_cfa_getAllpks", new Class[0]);
getAllpks_method.setAccessible(true);
String[] allPks = (String[])getAllpks_method.invoke(queryVO, new Object[0]);
Class<?> IArapBillQueryBaseService_class =
  Class.forName("nc.itf.arap.bill.IArapBillQueryBaseService", true, provisionClassLoader);
Method queryBillByPKForPageQry_method =
  IArapBillQueryBaseService_class.getDeclaredMethod("_cfa_queryBillByPKForPageQry", new Class[] { String[]
queryBillByPKForPageQry_method.setAccessible(true);
Object[] rs = (Object[])queryBillByPKForPageQry_method.invoke(IArapPayableBillQueryService_instance, new
Class<?> AggPayableBillVO_class =
  Class.forName("nc.vo.arap.payable.AggPayableBillVO", true, provisionClassLoader);
Method getParent_method =
  AggPayableBillVO_class.getMethod("_cfa_getParent", new Class[0]);
getParent_method.setAccessible(true); byte b; int i; Object[] arrayOfObject;
for (i = arrayOfObject = rs.length, b = 0; b < i; ) { Object AggPayableBillVO_instance = arrayOfObject[b]
  String parentStr = getParent_method.invoke(AggPayableBillVO_instance, new Object[0]).toString();
  ret.add(parentStr.replaceAll("\r\n", ",")); b++; }
```

图 13 对某企业 ERP 系统切片得到的部分代码片段

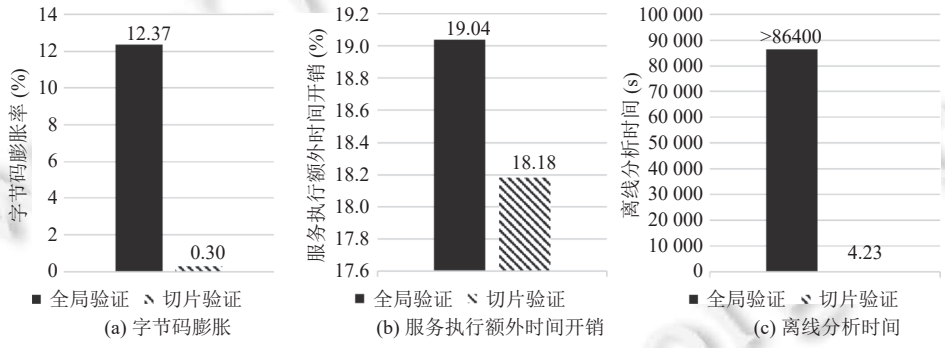


图 14 是否采用基于程序切片的优化策略带来的影响

此外, 在进行对合法度量值的离线分析时, 全局验证还将面临状态爆炸的问题. 这是由于在全局验证的方式下, 我们并不确定目标程序进行一次执行期间的起始基本块和终止基本块, 因此需要预先计算任意两个基本块之间可能的执行路径所对应的度量值才能确保进行验证时不会产生遗漏. 虽然这种方式能够在需要为多个数据服务进行计算时一次性得到结果, 但其复杂度将从对于单个确定入口和出口的数据服务程序片段进行计算的 $O(n)$ 提高到 $O(n^3)$, 其中 n 为基本块数量. 对于本实验中的某企业 ERP 系统而言, 其单个数据服务内的基本块数量为 142, 其整体的基本块数量则超过了 50 000, 我们在运行了 24 h 后仍未得到全局验证模式下完整的离线分析结果. 通过基于程序切片的优化策略, DataAttest 有效降低了将动态远程证明应用于数据服务完整性验证的额外开销.

4.4 安全性分析

1) 对完整性度量结果进行验证的过程的完整性. 为了有效保障数据服务的完整性, 还需要保证对完整性度量结果进行验证的过程的完整性. 与数据供方信息系统不同, 完整性度量结果验证的业务逻辑和环境依赖都较为简单, 因此我们可以直接利用现有的安全计算技术来保障其过程的完整性. 例如, 在本文的原型系统实现中, 我们将

对完整性度量结果进行验证的过程实现为了一段可以直接在可信执行环境内执行的 C++程序.

2) 完整性度量结果验证过程中数据和目标程序的机密性. 完整性度量结果的验证过程还需要保障数据的机密性, 不能在验证过程中泄露数据. 类似的, 可以利用现有的隐私保护计算技术来保障这一过程. 例如, 在本文的原型系统实现中, 我们在证明端使用可信执行环境中受硬件保护的私钥所对应的公钥对验证所需的数据和度量结果进行加密后发送至对应的可信执行环境, 这样即可保证明文的数据和度量结果仅在可信执行环境内才可被访问. 此外, 如果需要保护目标程序的机密性 (即避免程序文件的泄露), 对于目标程序的预处理静态分析也可以以类似的方式进行.

3) 对复杂数据服务执行过程的完整性验证. 在本文中, 我们以单个信息系统直接对外提供数据服务的形式为例对完整性度量和验证方法进行了介绍. 在真实场景中, 数据供方信息系统可能是由多个子系统构成的复杂系统, 数据在其中可能会经过多个环节的流转, 最终经由对外提供数据服务的子系统传出. 在这种情况下, 应当在每个对数据进行了修改的子系统中应用本文所提出的度量和验证方法, 产生多个度量结果并分别对其进行验证以保障数据服务执行过程的完整性. 如果数据在流转过程中并未发生变化, 那么只需要对最初产生数据的子系统进行度量即可.

4.5 可行性分析

1) DataAttest 中数据服务完整性度量的可信执行要求数据供方设备硬件中存在可用的可信执行环境. 考虑到当前主流的 Intel、AMD、ARM 计算平台均提供对 TPM 与 TEE 的支持[11−13], 这一环境要求是易满足的.

2) DataAttest 需要对数据供方信息系统进行一定的程序插桩形成证明报告以对数据服务完整性进行度量. 考虑到一方面, 随着数据要素市场的成熟, 数据完整性将日益成为数据可信流通的重要支撑和基本要求; 另一方面, 数据流通能够为数据供方带来可观的收益, 数据供方在证明自身提供数据过程的完整性方面是具有主动性的; 再一方面, 该程序插桩过程是自动化的, 且不会改变系统原本的业务处理逻辑. 因此, 本文中提出的远程证明方法对数据供方来说是可接受且易实施的.

3) DataAttest 需要数据需方调用数据服务后执行额外的验证过程 (部分场景下, 这一验证方是数据平台). 考虑到验证过程是自动化的, 在线验证过程引入的开销是极低的, 并且数据需方确实有验证数据服务完整性的诉求, 这一要求对于数据需方来说是可接受且易实施的.

4.6 局限性分析

目前, DataAttest 仍然存在以下两方面的局限性有待进一步完善.

1) 虽然 DataAttest 将数据完整性保障提前到了数据供方信息系统执行数据服务的环节, 但当数据篡改发生在诸如人工数据录入等数据服务执行过程之外的环节时, DataAttest 无法发现此类数据篡改.

2) 目前, DataAttest 尚未将 Java 虚拟机 (JVM) 纳入证明的范围, 也就是说当攻击发生在 JVM 对 Java 字节码的即时编译/解释执行过程中时, DataAttest 无法发现此类数据篡改.

5 总 结

数据作为新型生产要素, 需要在不同主体间流通以创造价值. 期间, 需要防止数据受到篡改, 保障数据的完整性. 针对现有工作难以保障数据供方所提供数据本身完整性的问题, 本文提出了一种基于远程证明的数据服务完整性验证方法, 利用构造在可信执行环境中的度量引擎, 在数据供方 (证明端) 相关数据服务运行时对其静态代码、执行过程和执行结果的完整性分别进行度量以构造数据服务完整性证据, 使得数据需方 (验证端) 可以在获取数据服务结果后通过基于该证据对数据服务执行过程的完整性进行验证, 从而将数据完整性保障的范围延伸至数据供方对外提供数据的环节. 考虑到真实系统规模庞大, 其复杂的控制流图会导致离线分析状态爆炸, 并且代码的模块化复用会导致度量过程和其他业务功能互相干扰, 设计了基于程序切片的完整性验证优化策略. 该策略基于运行时行为模型监测并定位目标数据服务所对应的程序片段, 对其进行切片并构造等效可执行程序片段, 然后对该切片片段而非原始数据服务进行与业务功能隔离的完整性度量, 从而在复杂度可控的前提下实现上述数据服务完整性验证. 通过在 3 个真实 Java 信息系统中 25 个数据服务上的一系列实验对 DataAttest 的有效性和性能进行了

评估. 实验结果表明, DataAttest 能够有效地对数据服务的完整性进行度量和验证, 其引入的额外开销较低, 在数据流通场景下是可接受的, 其中采用的基于程序切片的优化策略能够有效降低代码膨胀率以及度量和验证过程的开销.

References:

[1] Opinions on building a more perfect system and mechanism for market based allocation of factors. 2020 (in Chinese). http://www.gov.cn/xinwen/2020-04/10/content_5500740.htm

[2] Mei H. On Data Governance. Beijing: China Renmin University Press, 2022 (in Chinese).

[3] Walport M. Distributed ledger technology: Beyond block chain. UK Government Office for Science, 2016. 1: 1–88.

[4] Typical cases of ecological environment law enforcement. 2022 (in Chinese). https://www.mee.gov.cn/ywdt/xwfb/202206/t20220609_985021.shtml

[5] Carey MJ, Onose N, Petropoulos M. Data services. Communications of the ACM, 2012, 55(6): 86–97. [doi: 10.1145/2184319.2184340]

[6] Abadi M, Budiu M, Erlingsson U, Ligatti J. Control-flow integrity principles, implementations, and applications. ACM Trans. on Information and System Security, 2009, 13(1): 4. [doi: 10.1145/1609956.1609960]

[7] Kuznetzov V, Szekeres L, Payer M, Candea G, Sekar R, Song D. Code-pointer integrity. In: The Continuing Arms Race: Code-reuse Attacks and Defenses. Association for Computing Machinery and Morgan & Claypool, 2018. 81–116. [doi: 10.1145/3129743.3129748]

[8] Enterprise Resource Planning. 2023. http://en.wikipedia.org/wiki/Enterprise_resource_planning

[9] Trusted Computing Group. Trusted Platform Module (TPM). https://trustedcomputinggroup.org/work-groups/trusted-platform-module/

[10] GlobalPlatform. TEE System Architecture v1.3. 2023. https://globalplatform.org/specs-library/tee-system-architecture/

[11] Intel® Software Guard Extensions (SGX). 2023. https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html

[12] AMD Secure Encrypted Virtualization (SEV). 2023. https://developer.amd.com/sev/

[13] ARM. TrustZone. 2023. https://developer.arm.com/ip-products/security-ip/trustzone

[14] Surminski S, Niesler C, Brasser F, Davi L, Sadeghi AR. RealSWATT: Remote software-based attestation for embedded devices under realtime constraints. In: Proc. of the 2021 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2021. 2890–2905. [doi: 10.1145/3460120.3484788]

[15] De Oliveira Nunes I, Jakkamsetti S, Rattanavipanon N, Tsudik G. On the TOCTOU problem in remote attestation. In: Proc. of the 2021 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2021. 2921–2936. [doi: 10.1145/3460120.3484532]

[16] Buchanan E, Roemer R, Shacham H, Savage S. When good instructions go bad: Generalizing return-oriented programming to RISC. In: Proc. of the 15th ACM Conf. on Computer and Communications Security. Alexandria: ACM, 2008. 27–38. [doi: 10.1145/1455770.1455776]

[17] Bletsch T, Jiang XX, Freeh VW, Liang ZK. Jump-oriented programming: A new class of code-reuse attack. In: Proc. of the 6th ACM Symp. on Information, Computer and Communications Security. Hong Kong: ACM, 2011. 30–40. [doi: 10.1145/1966913.1966919]

[18] Abera T, Asokan N, Davi L, Ekberg JE, Nyman T, Paverd A, Sadeghi AR, Tsudik G. C-FLAT: Control-flow attestation for embedded systems software. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. Vienna: ACM, 2016. 743–754. [doi: 10.1145/2976749.2978358]

[19] Dessouky G, Zeitouni S, Nyman T, Paverd A, Davi L, Koeberl P, Asokan N, Sadeghi AR. Lo-fat: Low-overhead control flow attestation in hardware. In: Proc. of the 54th Annual Design Automation Conf. Austin: ACM, 2017. 1–6. [doi: 10.1145/3061639.3062276]

[20] Zhang YM, Liu XZ, Sun C, Zeng DR, Tan G, Kan X, Ma SQ. ReCFA: Resilient control-flow attestation. In: Proc. of the 2021 Annual Computer Security Applications Conf. ACM, 2021. 311–322. [doi: 10.1145/3485832.3485900]

[21] Papamartzivanos D, Menesidou SA, Gouvas P, Giannetsos T. Towards efficient control-flow attestation with software-assisted multi-level execution tracing. In: Proc. of the 2021 Int'l Mediterranean Conf. on Communications and Networking. Athens: IEEE, 2021. 512–518. [doi: 10.1109/MeditCom49071.2021.9647635]

[22] Liu JB, Yu Q, Liu W, Zhao SJ, Feng DG, Luo WF. Log-based control flow attestation for embedded devices. In: Proc. of the 11th Int'l Symp. on Cyberspace Safety and Security. Guangzhou: Springer, 2019. 117–132. [doi: 10.1007/978-3-030-37337-5_10]

[23] Xu XW, Weber I, Staples M. Architecture for Blockchain Applications. Switzerland: Springer, 2019.

[24] Pasdar A, Lee YC, Dong ZL. Connect API with blockchain: A survey on blockchain oracle implementation. ACM Computing Surveys, 2023, 55(10): 208. [doi: 10.1145/3567582]

[25] Provable documentation. 2023. https://docs.provable.xyz

[26] TLS-Notary. 2023. https://tlsnotary.org/
[27] Zhang F, Cecchetti E, Croman K, Juels A, Shi E. Town Crier: An authenticated data feed for smart contracts. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. Vienna: ACM, 2016. 270–282. [doi: 10.1145/2976749.2978326]
[28] Breidenbach L, Cachin C, Chan B, Coventry A, Ellis S, Juels A, Koushanfar F, Miller A, Magauran B, Moroz D, Nazarov S, Topliceanu A, Tramèr F, Zhang F. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. 2021.
[29] Peterson J, Krug J, Zoltu M, Williams AK, Alexander S. Augur: A decentralized oracle and prediction market platform. arXiv: 1501.01042, 2020.
[30] Stojanovski N, Gusev M, Gligoroski D, Knapskog SJ. Bypassing data execution prevention on microsoftwindows XP SP2. In: Proc. of the 2nd Int'l Conf. on Availability, Reliability and Security. Washington: IEEE, 2007. 1222–1226. [doi: 10.1109/ARES.2007.54]
[31] Basic blocks. 2023. https://gcc.gnu.org/onlinedocs/gccint/Basic-Blocks.html
[32] Allen FE. Control flow analysis. ACM SIGPLAN Notices, 1970, 5(7): 1–19. [doi: 10.1145/390013.808479]
[33] Global Platform™. Introduction to trusted execution environments. 2018. https://globalplatform.org/resource-publication/introduction-to-trusted-execution-environments/
[34] Huang JC. Program instrumentation and software testing. Computer, 1978, 11(4): 25–32. [doi: 10.1109/C-M.1978.218134]
[35] Replay attack. 2023. http://en.wikipedia.org/wiki/Replay_attack
[36] DeMarco T. Structure analysis and system specification. In: Pioneers and Their Contributions to Software Engineering. Bonn: Springer, 1979. 255–288. [doi: 10.1007/978-3-642-48354-7_9]
[37] GitHub projects related to Web service. 2023. https://github.com/search?q=web+service
[38] Blake2. 2023. https://www.blake2.net/
[39] ASM. 2023. https://asm.ow2.io/
[40] Soot. 2023. http://soot-oss.github.io/soot/
[41] Kuang BY, Fu AM, Susilo W, Yu S, Gao YS. A survey of remote attestation in Internet of Things: Attacks, countermeasures, and prospects. Computers & Security, 2022, 112: 102498. [doi: 10.1016/j.cose.2021.102498]
[42] Fiddler. 2023. https://www.telerik.com/fiddler

附中文参考文献:

[1] 关于构建更加完善的要素市场化配置体制机制的意见. 2020. http://www.gov.cn/xinwen/2020-04/10/content_5500740.htm
[2] 梅宏. 数据治理之法. 北京: 中国人民大学出版社, 2022.
[4] 生态环境执法典型案例. 2022. https://www.mee.gov.cn/ywdt/xwfb/202206/t20220609_985021.shtml

张溯(1995－), 男, 博士, 主要研究领域为数据安全和隐私保护, 大数据互操作.

张伟(1989－), 男, 硕士, 主要研究领域为系统软件, 大数据互操作.

张颖(1983－), 男, 博士, 副研究员, CCF 专业会员, 主要研究领域为大数据互操作, 软件构件化开发.

黄罡(1975－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为系统软件, 软件自适应, 数联网.