HiveAttacker:一个针对 Hive 数据仓库的两阶段安全性检测方案

李文超[1,2,3,4], 李 丰[1,2,3], 薄德芳[1,2,3,4], 周建华[1,2,3,4], 霍 玮[1,2,3,4]

[1]中国科学院信息工程研究所, 北京 中国 100093
[2]中国科学院网络测评技术重点实验室, 北京 中国 100093
[3]网络安全防护技术北京市重点实验室, 北京 中国 100093
[4]中国科学院大学网络空间安全学院, 北京 中国 100049

摘要 大数据所蕴藏的巨大价值, 使其成为当前网络攻击的重点目标之一。然而, 长期以来, 以 Hive 为代表的数据仓库及大数据处理引擎, 以及其所依托的分布式处理平台, 普遍重视服务的高可用性、高扩展性, 未充分考虑安全性, 导致在大数据的存储、处理过程中存在安全风险。本文以 Hadoop 平台上的 Hive 数据仓库及查询引擎为切入点, 归纳了 Hive 在查询解析过程中, 以及在与 Hadoop 平台或其他第三方组件交互过程中面临的两个主要攻击面, 并针对性地设计了一个两阶段安全性检测方案。方案的第一阶段针对 Hive 因接收、解析用户查询所引入的攻击面, 对传统模糊测试技术进行定制化扩展, 重点挖掘 Hive 自身代码中存在的可能造成提权、授权绕过等利用效果的漏洞; 第二阶段针对 Hive 因与其他组件交互引入的攻击面, 重点检测可能通过组件间交互触发的漏洞, 并进行预警。基于上述方案实现的原型工具 HiveAttacker, 在 Hive 两个历史版本及最新版本中共挖掘出 8 个漏洞, 其中包含 2 个最新版本中尚未修复的漏洞, 并在搭建的真实 Hive 运行环境中检测出因组件交互引入的安全威胁 7 处, 验证了方案的有效性。

关键词 Apache Hive; 模糊测试; 漏洞检测
中图法分类号 TP311 DOI 号 10.19363/J.cnki.cn10-1380/tn.2026.01.01

HiveAttacker: A Two-stage Security Detecting Approach for Apache Hive

LI Wenchao[1,2,3,4], LI Feng[1,2,3], BO Defang[1,2,3,4], ZHOU Jianhua[1,2,3,4], HUO Wei[1,2,3,4]

[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2]Key Laboratory of Network Assessment Technology, Chinese Academy of Sciences, Beijing 100093, China
[3]Beijing Key Laboratory of Network Security and Protection Technology, Beijing 100093, China
[4]School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract The enormous value that big data holds has made it one of the prime targets for network attack today. However, data warehouses and big data processing engines, represented by Hive, as well as the distributed processing platforms on which they rely, have long focused on high availability and scalability of services, without paying enough attention to security, leading to security risks during the storage and processing of big data. In this paper, we take the Hive data warehouse and query engine on the Hadoop platform as the starting point, summarize the two main attack surfaces that Hive faces during query parsing and the interaction process with the Hadoop platform or other third-party components, and design a targeted two-stage security detecting solution. The first stage of the solution targets the attack surface introduced by Hive's reception and parsing of user queries. We customize and extend traditional fuzz testing techniques to focus on vulnerabilities in Hive's own code that may cause exploitation effects such as privilege escalation and authorization bypass. The second stage of the solution targets the attack surface introduced by Hive's interaction with other components, with a focus on detecting vulnerabilities that may be triggered through inter-component interactions and issuing warnings accordingly. Based on this solution, we implemented a prototype tool named HiveAttacker and applied it on two of the historical revisions as well as the latest revision of Hive. The tool has detected a total of eight vulnerabilities, including two vulnerabilities that have not yet been fixed in the latest versions of Hive. It has also identified seven security threats introduced by component interactions in a real Hive operating environment, thus verifying the effectiveness of the proposed approach.

Key words Apache Hive; fuzzing test; vulnerability detecting

通讯作者: 李丰, 博士, 副研究员, lifeng@iie.ac.cn。
本课题得到国家自然科学基金(No. U1836209, No. 62032010, No. 61802394), 中国科学院战略性先导科技专项(No. XDC02040100)资助。
收稿日期: 2020-12-07; 修改日期: 2021-02-09; 定稿日期: 2023-02-20

1 引言

信息化时代, 数据已经成为重要的生产要素和社会财富。数据体量以及对处理效率的需求促进了大数据分析平台及相关应用的发展, 而大数据所蕴藏的巨大价值, 也使其成为当前网络攻击的重点目标之一[1]。

Hadoop[2]是当前主流的分布式处理平台之一, 依托其上的数据仓库及查询引擎 Hive[3]可支持规模化数据的分析处理。上述平台及应用普遍重视服务的高可用性和高扩展性, 在设计时充分考虑了容错、冗余备份等要素。但由于其最初的应用假设是面向可信内网集群下由可信用户相互协作处理大规模数据的场景, 故未充分考虑用户及数据的安全性[4]。以 Hadoop 为例, 如图 1 所示, 迄今为止在公共漏洞数据库(Common Vulnerability Exposure)中公开的历史漏洞共 20 余个[5], 利用效果包括信息泄露、容器逃逸、授权机制绕过、认证机制绕过、命令注入(SQL 注入、外部实体注入)等。Hive 已公开的历史漏洞共 10 个[6], 如图 2 所示, 利用效果包括授权机制绕过、认证机制绕过和命令注入等。虽然陆续引入了用户身份认证(如 Kerberos[7])、授权访问控制等安全机制, 以及为集群提供统一安全管理的独立组件(如 Ranger[8]、Sentry[9]), 但如图 3 所示, 近年来 Hadoop 和 Hive 仍持续曝出漏洞或安全相关的 JIRA 事务[10-11]。利用这些漏洞轻则造成信息泄露, 重则导致权限提升等攻击效果, 严重影响数据仓库用户以及数据处理过程的安全性。

学术界围绕分布式处理平台及相关组件的缺陷检测、异常检测[12-13]提出了一系列技术及原型, 但上述工作大都侧重于关注可能对分布式系统或应用的可靠性及可用性造成影响的缺陷, 并未考虑对漏洞或可能影响安全性的缺陷的检测。以模糊测试为代表的传统漏洞挖掘技术[14-18], 虽然能够有效检测操作系统内核、协议、开源库等传统关键基础软件中的漏洞, 但其变异[19]、监控[20-21]策略更适合于挖掘内存破坏类漏洞, 对信息泄露、越权访问、认证机制绕过等逻辑漏洞的触发和监控能力不足。此外, 现有的漏洞挖掘技术及工具, 并未考虑对分布式处理平台、数据仓库及查询引擎中特有攻击面的支持。

本文以 Hadoop 平台上的 Hive 数据仓库及查询引擎为切入点, 通过对 Hive 工作原理及已公开漏洞的分析, 归纳 Hive 在查询解析和执行过程中可能存在的主要攻击面。Hive 的基本工作流程是将用户使用 SQL 语法子集书写的查询解析、转化成可以在分布式处理平台上执行的一系列查询任务, 根据配置调用 MapReduce、Tez[22]、Spark[23]等组件执行查询任务并返回查询分析结果。在上述处理过程中, 涉及

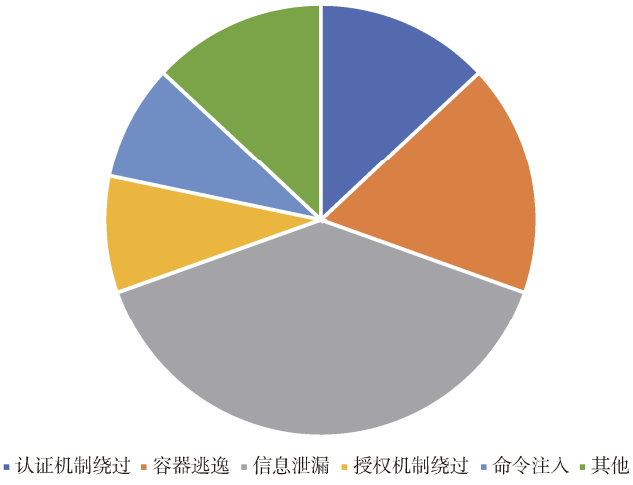


图 1 Hadoop 已知 CVE 漏洞利用效果分类
Figure 1 Classification of Hadoop CVE vulnerability exploitation effect

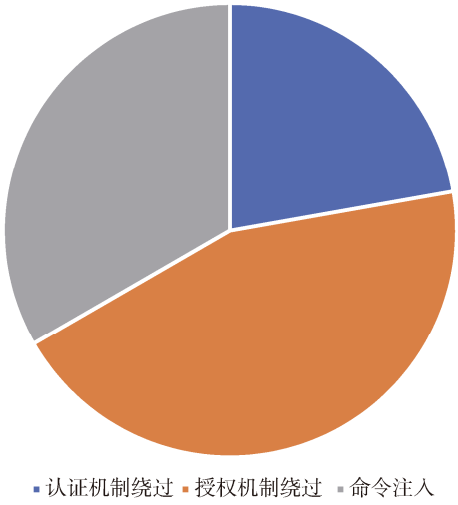


图 2 Hive 已知 CVE 漏洞利用效果分类
Figure 2 Classification of Hive CVE vulnerability exploitation effect

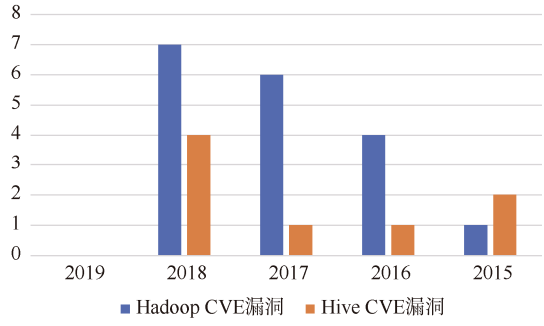


图 3 Hadoop 和 Hive 近五年漏洞分布
Figure 3 Vulnerability distribution of Hadoop and Hive in recent five years

外部交互的位置可以大致归为两类: 一类是接收用户输入的查询语句的位置, 另一类是 Hive 在查询解析及执行过程中与分布式处理平台中其他组件进行交互的位置, 比如, 向安全管理组件读取用户访问权限、向资源调度组件 Yarn[24]提交查询任务、向 HDFS[25]写入查询结果等位置。攻击者可以利用上述交互位置构造恶意输入, 触发 Hive 及相关组件中存在的漏洞或安全缺陷, 影响 Hive 及其他组件的安全性。下文将上述位置分别称为 Hive 的“查询攻击面”和“交互攻击面”(详见 2.2 节)。

为预防来自上述两个攻击面的安全威胁, 本文设计了一个系统性的两阶段安全性检测方案, 分别针对 Hive 的查询攻击面和交互攻击面进行漏洞挖掘与威胁检测。

方案的第一阶段针对因接收、解析用户查询所引入的攻击面, 设计定制化的模糊测试技术, 通过对传统模糊测试技术在输入生成、异常监控等方面的定制化扩展, 支持对 Hive 自身代码中存在的可能导致权限提升、授权机制绕过等安全威胁的漏洞的自动化挖掘。为进一步提高模糊测试的效率和效果, 本文在该阶段进行了预处理, 基于历史漏洞总结出的漏洞代码模式, 提前筛选出包含疑似漏洞的函数, 从而指导模糊测试优先生成可能覆盖疑似漏洞函数的输入。

方案的第二阶段针对因组件交互引入的攻击面, 分别从通过其他组件中的漏洞对 Hive 造成安全威胁, 以及通过 Hive 触发其他组件中的漏洞从而对后者造成安全威胁这两个角度进行检测。交互攻击面隐蔽性强、输入格式多样、输入空间大, 模糊测试难以触发。本阶段借助静态程序分析技术, 检测与 Hive 存在交互的 Ranger、Yarn、HDFS 等组件中的漏洞, 包括历史漏洞以及新发现的漏洞, 是否具备通过与 Hive 间的交互接口触发的条件, 并对潜在威胁进行预警。

基于以上两阶段方案, 本文实现了原型工具 HiveAttacker, 支持对数据仓库 Hive 自身漏洞以及其所依赖的其他组件中可能引发权限违反、信息泄露等安全威胁的系统性测试。实验表明, HiveAttacker 在 Hive 的两个历史版本及最新版本中共挖掘出 8 个漏洞, 其中包含 2 个最新版本中尚未修复的漏洞; 在搭建的真实 Hive 运行环境中检测出因组件交互引入的安全威胁 7 处, 经人工触发验证, 可能影响 Hive 自身以及 Hadoop 平台的安全性, 利用效果包括信息泄漏、授权机制绕过、文件提权等。

本文的主要贡献如下:

1) 分析归纳了数据仓库及查询引擎 Hive 因用户查询、组件交互引入的两个主要攻击面。

2) 提出了针对 Hive 的两阶段安全性检测方案, 分阶段进行针对性的测试, 支持对 Hive 自身代码中的提权、权限绕过漏洞, 以及组件交互引起的安全威胁的系统性测试与检测。

3) 基于上述方案实现的原型工具 HiveAttacker, 挖掘了 Hive 历史版本及最新版本中的 8 个漏洞, 其中包括 2 个最新版本中尚未修复的漏洞, 并辅助验证了因组件交互引入的 7 个安全威胁。

本文接下来的行文结构如下: 第 2 节介绍 Hive 的工作原理、历史漏洞成因, 并归纳了 Hive 的两个主要攻击面; 第 3 节介绍本文针对两个主要攻击面设计的两阶段安全性检测方案的原理与基本流程; 第 4 节和第 5 节依次介绍方案第一阶段和第二阶段的技术及实现细节; 第 6 节介绍实验设计和数据分析; 第 7 节介绍相关工作; 第 8 节总结全文。

2 Hive 原理和攻击面

2.1 Hive 原理

Hive 是 Facebook 于 2008 年启动的一个大数据分析、解决方案, 目前由 Apache 开源社区维护, 是 Hadoop 的子项目。Hive 的基本工作流程是将用户使用 SQL 语法子集书写的查询解析、转化为可以在分布式处理平台上执行的一系列查询任务, 根据配置调用 HDFS、Yarn、MapReduce、Tez、Spark 等组件执行查询任务并返回查询分析结果。Hive 自身代码以及除 Spark 外的组件主要由 Java 语言实现。

Hive 在代码结构上由用户接口、元数据存储(MetaStore)、Hive 驱动(Driver)以及运行时系统四部分构成。其中, 用户接口包括命令行接口、网页图形接口、基于 Thrift Server 的远程跨系统查询接口(包含 JDBC、ODBC)等多种形式。元数据存储支持本地内置数据库(derby)、分布式数据库, 以及远程数据库(如 MySQL、Oracle)等多种存储形式及数据格式。Hive 驱动是 Hive 查询引擎的重要组成部分之一, 分为解释器(Parser)、优化器(Optimizer)和执行器(Executor)。其中, 解释器对通过用户接口接收到的 SQL 查询进行词法检查和语法解析, 并将其转化成抽象语法树(AST)的形式; 优化器从元数据存储中读取诸如表名、列名、类型等元数据信息, 对抽象语法树进行类型检查, 并将其转化为由一系列逻辑查询操作构成的有向无环图, 也称为逻辑查询计划; 再根据默认的优化规则, 对逻辑查询计划进行优化, 并为优化后的查询树中各个逻辑查询操作选择一个

较优的物理查询算法, 生成物理查询计划。执行器根据所配置的运行时系统, 将物理查询计划划分成对应的分布式查询任务, 交由运行时系统执行, 待执行结果返回后, 将其传递到用户接口, 最终呈现给用户。

2.2 Hive 威胁模型

OWASP 组织指出, 软件攻击面是进出系统的所有数据和命令的路径总和与保护这些路径的代码(例如认证、授权、编码、数据校验等)[26]。在上述工作流程中, 与 Hive 存在外部数据和命令交互的位置可以大致归为两类: 1)外部用户通过命令行接口、网页接口、基于 Thrift Server 的远程跨系统查询接口等用户接口与 Hive 进行的命令输入交互; 2)Hive 在查询解析和查询执行过程中与分布式处理平台其他组件进行的数据或代码交互。这两类交互位置构成 Hive 的两个主要攻击面(如图 4 所示)。

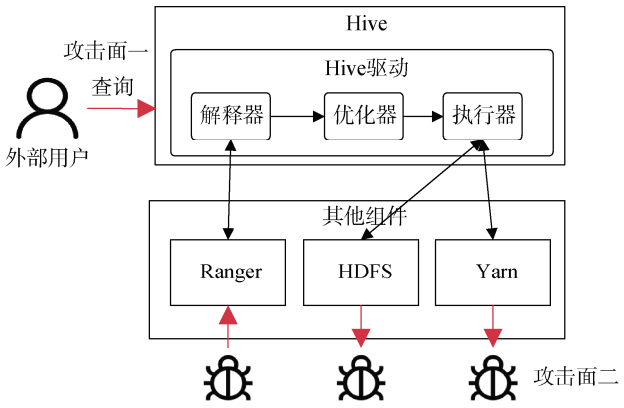


图 4 Hive 工作原理与两个攻击面示意

Figure 4 Hive working principle and two attack surfaces

第一个攻击面是由 Hive 与外部用户的查询命令交互引入的(以下简称“查询攻击面”)。用户提交的用 SQL 语言子集书写的查询, 经由语法解析、语义分析、用户访问权限检查等一系列处理后, 转化成分布式查询任务。在上述处理的代码执行流中, 起到保护作用的有用户访问认证、授权、输入校验与过滤等模块, 如果未对查询中的转义符等特殊字符进行有效过滤(如 CVE-2018-1282[27])或对用户的访问控制权限进行合理的检查(如 CVE-2015-7521[28]), 可能破坏上述模块的正常工作流程, 引入 SQL 注入、未授权访问等漏洞, 影响 Hive 数据仓库及查询引擎自身的安全性, 导致 Hive 暴露于查询攻击面之中。在已公开的 Hive 历史漏洞中(图 2、图 3), 占比最高且近年来仍频繁出现的是与授权机制相关的漏洞。此类漏洞通常不会导致程序崩溃等容易捕获到的异常现象, 存在隐蔽性强、不易被传统漏洞挖掘技术发现等特点。本文在设计方案时也将重点关注 Hive 查询攻击面中与授权机制相关的漏洞。

第二个攻击面是 Hive 在查询解析以及查询执行过程中与分布式处理平台中其他组件进行数据或代码交互引入的(以下简称“交互攻击面”)。交互数据流包括 Hive 到其他组件和其他组件到 Hive 两个方向, 也构成了两类潜在的攻击路径。具体而言, 其他组件到 Hive 的交互数据流主要包括: 在查询解析过程中, 从 Ranger、Sentry 等安全组件中处理用户请求权限并返回评估结果; Hive 到其他组件的交互数据流主要包括: 查询任务生成后向 Hadoop 平台的任务调度组件 Yarn 提交任务, 以及查询任务执行过程中从分布式文件系统组件 HDFS 读取数据或写入数据等。上述的组件间交互, 是 Hive 执行查询过程中的必要环节, 因此交互攻击面的暴露范围不可忽视。由于 Hive 的每个版本都可以与一定版本范围内的组件进行交互, 攻击者可以利用存在交互的组件中的已知漏洞, 对 Hive 发起攻击, 或利用 Hive 作为攻击入口对存在漏洞的组件发起攻击, 影响 Hive 乃至 Hadoop 平台其他组件的安全性。

2.3 技术挑战

模糊测试是当前工业界和学术界最为通用的软件漏洞挖掘技术之一。它通过向目标程序提供大量的经过特殊构造的测试用例, 并在程序运行过程中监控程序的执行行为, 以程序是否发生异常行为为标志, 来发现目标程序可能存在的安全漏洞[29]。目前的工作通过对传统模糊测试技术的优化和定制, 能够有效发现操作系统内核、协议、开源库等关键基础软件中的漏洞。

但现有的模糊测试技术及工具, 并未考虑对数据仓库及查询引擎中特有攻击面的支持。以用户查询为例, 通过随机变异生成的查询输入可能存在语法、语义不正确的问题, 影响模糊测试效率。据文献[30]统计, 以传统数据库 MySQL 为例, 目前最常用的基于变异的模糊测试工具 AFL[31]在 24 h 内生成的 2 万个输入中, 能够通过语法正确性检查的占 30%, 而能够通过语义正确性检查的仅占 4%。以 Peach[32]、booFuzz[33]为代表的基于生成的模糊测试虽然支持用户编写用于生成输入的模板文件或脚本, 但一方面, Hive 支持的查询语法只是传统 SQL 语法的子集, 如果不对传统 SQL 语法进行裁剪, 可能生成大量无效的输入, 影响模糊测试效率; 另一方面, 组件间交互对应的输入往往形式多样、格式各异, 人工书写模板难度大、耗时长且可能不完善。

此外, Hive 自身代码以及其所依赖的 Hadoop 平台组件主要由 Java 语言实现。以 AFL、LibFuzzer[34] 为代表的模糊测试工具虽然能够有效监控 C/C++编写的程序中的缓冲区溢出、空指针解引用等内存破坏类漏洞, 但对 Java 程序的支持有限, 且未考虑对 Hive 查询攻击面中常见的提权、授权绕过等漏洞触发效果的监控。而交互攻击面由于涉及的组件多、输入空间大, 难以通过查询攻击面的查询命令交互构造触发其他组件漏洞的输入, 且漏洞类型多样, 缺少可归纳的通用漏洞特征, 对模糊测试的输入构造、追踪和监控均存在挑战, 需要设计更加契合的检测方案。

3 针对 Hive 的两阶段安全性检测方案

本文针对数据仓库及查询引擎 Hive 处理查询分析的过程中存在的两个主要攻击面, 设计并实现了一个针对性的两阶段安全性检测方案, 服务于 Hive 及相关组件的漏洞挖掘与威胁检测。方案的第一阶段旨在挖掘 Hive 自身代码实现中包含的授权机制相关漏洞, 第二阶段侧重检测 Hive 在与 Hadoop 平台其他组件的交互过程中引入的安全威胁。方案的设计原理如图 5 所示, 主要设计思路如下。

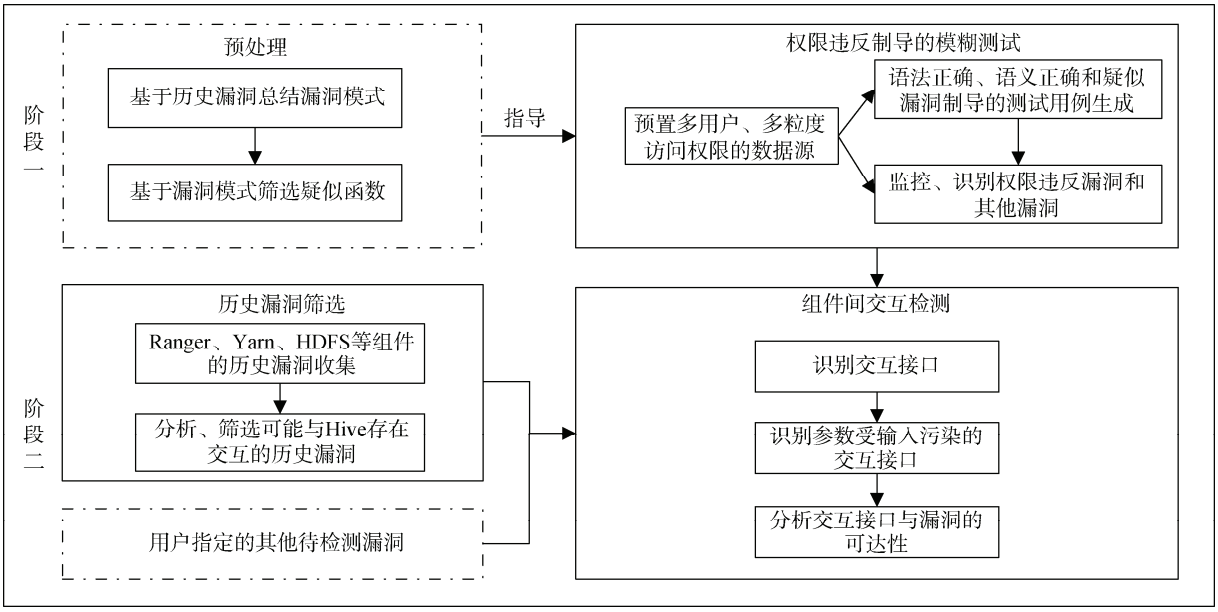


图 5 HiveAttacker 系统原理图

Figure 5 Principle of HiveAttacker system

阶段一: 定制化模糊测试。方案的第一阶段针对因接收、解析用户查询所引入的攻击面, 对传统模糊测试技术进行定制化扩展, 重点挖掘 Hive 自身代码中存在可能导致权限提升、授权机制绕过等利用效果的漏洞。该阶段包含预处理和权限违反制导的模糊测试两个步骤。其中, 预处理步骤为可选步骤。

步骤 1: 预处理。本步骤首先根据历史漏洞人工总结出含有漏洞的代码模式, 随后通过代码相似性检测找出其他可能包含漏洞的疑似函数。该步骤检测出的所有疑似漏洞函数, 将最终被用于指导模糊测试优先生成可能覆盖这些函数的输入。

步骤 2: 权限违反制导的模糊测试。本步骤对传统模糊测试技术的输入生成策略和异常监控策略进行定制, 提高用户权限相关漏洞被触发的概率。定制方案如下: 在生成输入时, 一方面, 参考预先设置的包含多用户、多粒度访问权限的数据源, 构造语法、语义正确的查询; 另一方面, 在构造查询的过程中, 为可能覆盖疑似漏洞函数的查询操作, 赋予更高的优先级。

在异常监控时, 除监控传统模糊测试关注的进程崩溃、异常挂起现象外, 还对 Hive 在执行查询任返回的用户权限信息与执行该查询实际需要的权限进行比较, 如果出现不一致, 则认为发生了权限违反务前读取用户访问权限的接口进行监控, 并将接口异常。

阶段二: 组件间安全威胁检测。方案的第二阶段针对 Hive 因与其他组件交互引入的攻击面, 分别从通过其他组件中的漏洞对 Hive 造成安全威胁, 以及通过 Hive 触发其他组件中的漏洞从而对后者造成安全威胁这两个角度进行检测。该阶段由历史漏洞筛选和组件间交互检测两个步骤构成。

步骤 1: 历史漏洞筛选。本步骤首先爬取公共漏

洞数据库中与 Hive 存在交互的 Ranger、Yarn、HDFS 等组件的历史漏洞, 根据漏洞描述自动检索 JIRA 上对应的事务, 再参照事务中提供的漏洞原理、修补位置等信息筛选出可能涉及 Hive 交互的历史漏洞。

步骤 2: 组件间交互检测。本步骤首先根据被测 Hive 所依赖的组件版本, 查询步骤 1 中满足版本范围的已知漏洞, 并与第一阶段新发现的漏洞, 以及用户指定的其他潜在漏洞, 共同构成后续检测的对象集合。然后, 通过静态分析识别 Hive 中与第三方组件进行交互的接口。最后通过分析待检测漏洞与交互接口间是否存在可达路径、是否输入可控等漏洞触发的必要因素, 筛选出具备触发条件的漏洞, 生成威胁预警。安全分析人员可根据预警信息对漏洞触发的影响进行确认。

本文的第 4、5 节将分别介绍以上两个阶段的技术及实现细节。

4 阶段一: 定制化模糊测试

本章介绍两阶段安全性检测方案中阶段一的技术细节。本阶段的处理流程如图 6 所示, 分为预处理和权限违反制导的模糊测试两步。其中, 预处理为可选步骤。

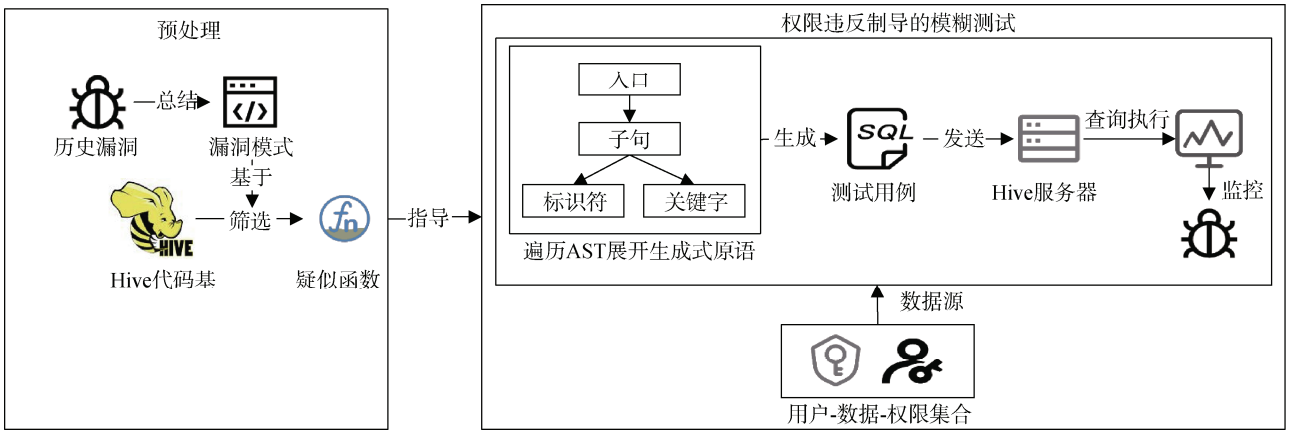


图 6 定制化模糊测试阶段的处理流程图

Figure 6 Process flow of customized fuzzing test stage

4.1 预处理

预处理步骤作为模糊测试开始前的准备步骤, 旨在检测 Hive 的查询解析代码中缺少权限检查或权限检查不完善的疑似漏洞函数, 为后续的模糊测试步骤提供导向信息。检测疑似漏洞函数的依据是从 Hive 已公开历史漏洞中归纳的代码模式。

Hive 对用户是否具备某一查询的执行权限的检查发生在生成抽象语法树后的语义分析阶段。Hive 驱动会针对不同类型的语句, 调用不同的语义分析器进行处理。图 7 所示为 Hive 对抽象语法树形式的查询语句或子句进行权限检查的过程: 1) 首先, 根据语句的不同特点从抽象语法树上提取语义特征, 例如表的名字、列的名字、视图的名字等; 2) 然后对提取到的语义特征进行简单分析后创建每个任务的描述, 描述中包含执行任务需要的属性和运行时信息; 3) 如果某些语义特征需要对用户的读写请求进行约束, 语义分析器会提取出相应的读写实体, 例如 drop table 语句需要请求数据库类型的读实体和写实体, 在后续进行授权检查时判断实体和用户所授予的权限是否相容; 4) 基于描述和读写实体构建任务对象, 最终提交到查询任务列表。

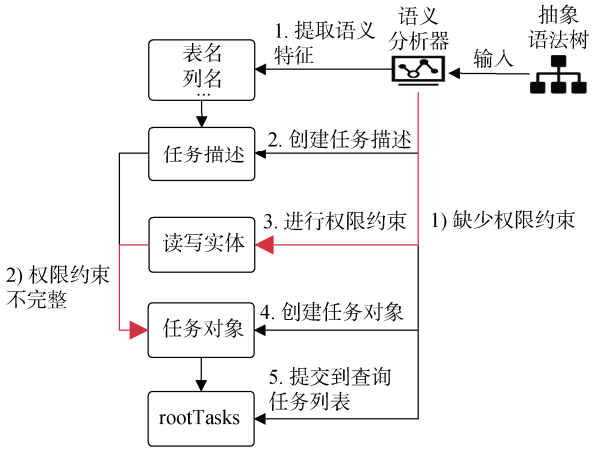


图 7 Hive 语义分析器执行流程和两类历史漏洞模式

Figure 7 Process of Hive Semantic Analyzer execution and two types of historical vulnerability

Hive 历史漏洞的代码模式可以归纳为两种情况: 1) 在某个查询语句的语义分析器执行过程中, 缺少对语义特征中有风险的读写实体的约束, 导致创建的查询任务在后续权限检查时恒通过, 造成潜在的

授权漏洞; 2) 语义分析器中并不缺少对读写实体的约束, 但可能约束并不完善, 或由于其他原因(如行列过滤)最终造成授权漏洞的发生。前者相对而言具有较为明显的代码模式特征, 适合作为静态检测的基准。因此, 我们基于前者的代码模式进行静态相似性比对, 筛选疑似函数。

算法 1. 疑似函数和对应语句类型识别算法

输入: 调用图 *CG*

输出: 疑似函数对应文法标识符的列表 *L*

```
1:FUNCTION recognize_candidates(CG)
2:    FOR 函数f IN CG
3:        IF 当前函数f中有查询任务提交
4:            FOR 函数F IN 函数f及其在
                  CG上的后继节点集合
5:                IF 函数F内无读写实体识别
6:                    candidates.add(f)
7:                ENDIF
8:            END FOR
9:        ENDIF
10:   END FOR
11:   FOR 函数f IN candidates
12:       FOR 函数p IN 函数f的前驱节点集合
13:           IF 函数p == 语义分析入口方法
14:               BREAK
15:           ENDIF
16:           IF 从函数p调用点处对应的条件
                  语句中提取到的 token 非空
17:               L.add(token)
18:           ENDIF
19:       END FOR
20:   END FOR
21:RETURN L
22:END FUCNTION
```

识别疑似漏洞函数及其对应的子句类型的伪码如算法 1 所示。算法的输入为分析 Hive 源码得到的调用图 *CG*, 输出为疑似函数对应文法标识符的列表 *L*。算法 1 的第 2~10 行负责疑似漏洞函数的识别: 首先遍历调用图 *CG* 找到内部有查询任务提交的函数 *f*, 然后检查其自身或后继节点 *F* 中是否有权限约束相关代码, 如果没有则标记 *f* 为疑似漏洞函数。第 11~20 行负责将疑似漏洞函数映射到文法标识符: 遍历每一个识别出的疑似函数 *f*, 沿着调用图分析前驱节点直到语义分析器的入口方法, 提取出分支条件中的相应文法标识符 *token*, 加入输出列表 *L*。

4.2 权限违反制导的模糊测试

算法 2 为本文设计的权限违反制导的模糊测试算法流程。算法 2 的输入为被测 Hive 应用、预处理阶段生成的疑似漏洞函数对应文法标识符的列表 *L*, 以及一个预先设置的包含多粒度访问权限的数据源 *D*, 输出为模糊测试过程中监控到的异常信息及触发该异常的输入构成的二元组列表 *E*。其中, 数据源 *D* 由用户、数据、权限 3 种元素构成。数据粒度包括表、视图、列, 权限类型包括增删改查以及表的创建、销毁。为了保证测试用户对具有相同结构的表上的不同粒度数据持有的权限被设置为正交关系, 数据源中相同结构的数据表的数量为两个或两个以上。

算法 2. 权限违反制导的模糊测试算法流程

输入: 被测 Hive 应用源码 *Prog*, *L*, *D*

输出: 监控到的异常信息及触发该异常的输入构成的二元组列表 E

```
1:FUNCTION main_fuzzing_loop(Prog, L, D)
2:    generators = create_generators(Prog)
3:    WHILE TRUE
4:        IF 用户中断等中止条件
5:            BREAK
6:        ENDIF
7:        IF random(0,1) > δ
8:            token = choice(L)
9:            generator = get_generator(token,
                                  generators)
10:       ELSE
11:           generator = choice(generators)
12:       ENDIF
13:       input,entities=gen_sql(D,generator)
14:       trace = execute_query(Prog,input)
15:       fc = cal_formal_check(entities, D)
16:       rc = parse_real_check(trace)
17:       IF fc 和 rc 中存在不一致
18:           exception = gen_exception(fc, rc)
19:           E.add((input, exception))
20:       ENDIF
21:   END WHILE
22:RETURN E
23:END FUNCTION
24:FUNCTION gen_sql(D, generator)
25:   WHILE 深度优先展开 generator 未结束
26:       IF 当前是标识符生成式
27:           基于数据源 D 生成相应标识符
28:           添加到授权实体 entities
29:       ELSE IF 当前是关键字生成式
30:           生成关键字
31:       ELSE IF 当前是查询语句生成式
32:           gen_sql(D,当前生成式)
33:       ENDIF
34:   END WHILE
```

35:RETURN *input, entitles*
36:END FUNCTION

算法 2 的每轮模糊测试循环(第 3~21 行)包括输入生成阶段和异常监控阶段, 其中加粗的伪码是在现有基于生成的模糊测试基础上的扩展, 包括对输入生成阶段的扩展(第 7~12 行)以及对异常监控阶段的扩展(第 14~20 行), 算法的终止条件为用户中断(第 4~6 行)。

在模糊测试循环开始前, 算法会基于Hive QL的文法创建大量生成式原语, 包括标识符生成式、关键字生成式、入口生成式和查询语句生成式(第 2 行)。预处理阶段首先取 0~1 之间的随机值与 δ 比较, 这里 δ 取 0.5(第 7 行)。如果随机值较大, 则从预处理结果中选择疑似漏洞函数对应的文法标识符(第 8 行), 然后获取文法标识符对应的查询语句生成式(第 9 行); 否则, 从所有查询语句生成式中随机选择一个查询语句生成式(第 10~12 行)。输入生成阶段由 gen_sql 子函数生成查询语句(第 13 行), 然后发送查询请求并插桩追踪服务器的执行结果(第 14 行)。gen_sql 子函数从 Hive QL 的 AST 根节点(即入口生成式原语, 包含 Hive 支持的所有类型的查询语句生成式)开始, 沿着深度优先方向遍历 AST, 递归展开每个生成式原语(第 25~34 行)。如果遇到关键字生成式, 则直接生成对应关键字, 例如文法符号 SELECT(第 29~30 行); 如果遇到查询语句生成式, 则继续调用 gen_sql 展开对应的查询语句, 例如 FROM 从句和 CREATE TABLE 语句(第 31~33 行); 如果遇到表名、视图名、列名等标识符生成式原语, 则基于从数据源中获取的知识, 生成符合语义上下文的标识符, 同时还会提取出授权实体辅助异常监控(第 26~28 行)。例如, 保证 SELECT 语句选择的列在 FROM 从句决定的表上。最终, 生成的 HIVE QL 查询可以正常通过词法、语法和语义分析阶段, 且容易触发权限违反的逻辑漏洞。

在异常监控阶段, 算法会基于数据源 *D*, 结合权限传递的公认知识(比如表和视图之间的行列过滤传递关系), 分析查询所请求的授权实体 entities 与用户权限的关系, 判断哪些表、视图、列等实体对象需要被检查或过滤(第 15 行)。然后根据插桩追踪得到的用户授权相关信息, 识别哪些表、列等实体对象在集群运行时实际被检查或过滤(第 16 行)。如果需要被检查或过滤的实体在实际运行时没有被检查或过滤, 则判断触发了异常; 如果需要被检查的实体在实际运行时检查结果与数据源中的授权上下文不一致, 且用户原有权限低于请求权限, 也会报告为异常(第 17~20 行)。

5 阶段二: 组件间安全威胁检测

本章介绍两阶段安全性测试方案中阶段二的技术细节。如前所述, 阶段二旨在对 Hive 在与 Hadoop 平台其他组件的交互过程中引入或触发的安全威胁进行预警, 以提高 Hive 自身以及相关组件的安全性。对于预先收集的交互组件中的历史漏洞、第一阶段新发现的漏洞, 以及用户指定其他潜在漏洞, 阶段二基于对 Hive 与该组件间的交互接口与漏洞之间的可达性、输入可控性的分析, 判断该漏洞是否具备跨组件触发的必要条件。对满足条件的漏洞进行威胁预警, 辅助安全分析人员进行安全威胁的确认与防护。

5.1 历史漏洞筛选

历史漏洞筛选主要采取人工方式, 筛选步骤及原则如下: 1) 组件筛选: 选择与 Hive 存在交互的组件(如表 1 所示), 例如安全管理组件 Ranger, 在开启 Ranger 的 Hive 集群上, 恶意用户通过发送精心构造的恶意查询可以触发权限处理逻辑中的漏洞, 造成安全风险。2) 漏洞关联: 从公开漏洞数据库或组件官方上爬取上述组件的历史漏洞信息, 包括 CVE 编号、漏洞类型、影响的版本、漏洞描述以及相关链接。部分漏洞的相关链接中已包含 JIRA 上对应事务的链接或补丁的链接, 其余漏洞需根据爬取到的漏洞描述中的关键字检索 JIRA 上对应的事务及补丁链接。3) 功能筛选: 对能够关联到事务及补丁链接的历史漏洞, 根据事务描述及补丁所在的功能模块, 判断该漏洞是否涉及所在组件与 Hive 间的交互功能。以 Ranger 为例, 其官网共公开了 13 个 CVE[35],

表 1 Hive 交互组件表

Table 1 Hive interactive component table

组件	与 Hive 间的功能交互	交互接口举例
Ranger	集中管理授权策略和处理用户的访问请求	org.apache.ranger.authorization.hive.authorizer.RangerHiveAuthorizerBase.checkPrivileges
Sentry	集中管理授权策略和处理用户的访问请求	org.apache.sentry.binding.hive.v2.authorizer.DefaultSentryValidator.checkPrivileges
HDFS	向分布式系统读取和写入数据	org.apache.hadoop.fs.FileSystem.copyToLocalFile
YARN	向分布式系统提交查询任务等	org.apache.hadoop.mapred.JobClient.submitJob

其中 10 个都是在负责策略管理的控制页面上发现的 Web 漏洞。Ranger 的控制页面不涉及与 Hive 查询解析或查询执行流程间的功能交互, 因此, 这 10 个漏洞不会对 Hive 造成安全威胁或通过 Hive 触发, 应当被筛除。其余位于 Ranger 的策略评估引擎中的漏洞则可能通过 Hive 的访问权限查询接口, 返回错误的权限检查结果, 影响 Hive 的授权机制, 故应记录并作为下一步分析的输入之一。

经上述步骤筛选得到的历史漏洞及其所在模块、影响的版本、原理描述等信息, 可用于进一步分析漏洞触发路径。另外, 部分与历史漏洞关联的事务链接中详细描述了漏洞的触发方式。本文对这些历史漏洞进行了人工复现。表 2 所示为本文经此步骤筛选及复现的历史漏洞数量。其中, HDFS 虽然存在与 Hive 之间的功能交互, 但从目前已公开的历史漏洞中, 未筛选出与交互功能相关的漏洞。

表 2 交互组件历史漏洞表

Table 2 Historical vulnerabilities in components

组件	漏洞数量	漏洞影响
Ranger	3	权限提升
Sentry	3	权限提升、代码执行、信息泄漏
HDFS	0	无
YARN	5	信息泄漏, 路径穿越, 文件提权

5.2 组件间交互检测

本步骤首先根据被测 Hive 应用所依赖的组件版本, 从上一步骤筛选得到的历史漏洞集合查询对应版本中包含的历史漏洞, 与第一阶段新发现的漏洞, 以及用户指定其他潜在漏洞, 共同构成检测对象集合。对每个待检测的漏洞, 判断其是否能够跨组件触发需要满足两个必要条件: 1) 漏洞触发路径上至少包含 1 个漏洞所在组件与 Hive 之间的交互接口; 2) 漏洞触发的输入可由用户控制。如前所示, Hive 的交互攻击面在攻击方向上包含通过触发其他组件中的漏洞对 Hive 造成安全威胁, 以及通过 Hive 触发其他组件中的漏洞从而对后者造成安全威胁两个方向。因此, 条件 2)中的输入位置需要同时考虑 Hive 及历史漏洞所在组件中接收输入的位置。

算法 3. 组件间交互检测算法

输入: *HiveICFG*、*moduleICFGs*、*vuls*、*inputs*

输出: 漏洞位置 *vul*、交互位置 *cs*、影响交互位置的输入 *I* 构成的三元组列表 *List*

```
1:FUNCTION detect_interaction(HiveICFG,
                              moduleICFGs, vuls, inputs)
2:   callsites = get_inter_callsites(HiveICFG)
3:   FOR cs IN callsites
4:     FOR m IN moduleICFGs
5:       IF (find_method(m, cs) == f ) != FALSE
6:         inters[cs] = <m,f>
7:       ENDIF
8:     END FOR
9:   END FOR
10:  FOR cs IN inters
11:    FOR I IN inputs
12:      IF tainted_by_input(I, cs)
13:        m_vuls = get_vuls(vuls, m)
14:        FOR vul in m_vuls
15:          IF reachable(inters[cs][f], vul)
16:            List.add((vul,cs,I))
17:          ENDIF
18:        END FOR
19:      ENDIF
20:    END FOR
21: END FOR
22:RETURN List
23:END FUNCTION
```

算法 3 为检测指定漏洞或漏洞集合是否具备跨组件触发条件的算法。算法的输入包括被测 Hive 应用的过程间控制流图(ICFG)*HiveICFG*, 交互组件的 ICFG 列表 *moduleICFGs*, 漏洞位置 *vuls* 以及外部输入位置列表 *inputs*。算法的输出为具备触发条件的漏洞信息列表, 即潜在的安全威胁列表。列表中的每个元素是一个由漏洞位置/编号、交互接口、输入位置构成的三元组。算法首先遍历并收集 *HiveICFG* 上缺少函数体的函数调用点 *cs*(第 2 行), 并在组件 ICFG 上查找对应的函数。如果在组件 *m* 中找到对应的函数实现 *f*, 则将该函数调用点 *cs* 记录为一个交互位置, 并记录其与<*m, f*>二元组间的对应关系(第 3~6 行)。借助静态污点分析, 从所有交互位置中, 筛选出参数受输入污染的位置(第 10~12 行)。对每个参数受输入污染的交互位置, 分析其在组件 *m* 中对应的函数 *f*, 与 *m* 组件在漏洞列表中的漏洞位置 *vul* 之间, 是否存在可达路径。如果存在, 则将由该漏洞的位置 *vul*、交互位置 *cs*、影响交互位置的输入 *I* 构成的三元组加入输出列表 *List*(第 13~16 行)。

6 实验与分析

本文基于第 3、4、5 节章介绍的原理与技术细节, 实现了针对 Hive 数据仓库的两阶段安全性检测原型系统 HiveAttacker。其中, 阶段一的预处理功能以及阶段二均基于 WALA 静态分析框架[36]实现; 模糊测试的异常监控基于 X-Trace[37]和 Javassist[38]实现, 通

过字节码插桩, 对授权相关接口进行监控; 模糊测试的输入生成基于 Grammarinator[39]和 ANTLR[40]实现, 共包含 2194 行 Java 代码和 1833 行 Python 代码。

为验证 HiveAttacker 的漏洞挖掘与检测效果, 本章选择了 Hive 2.x 系列的最新版本(2.3.7)及包含已公开漏洞的两个历史版本(1.2.1 和 2.3.0)为实验对象, 并配合不同版本的 Hadoop(2.5.1、2.8.2), 以及 Ranger(0.5.3、0.7.0)或 Sentry(1.6.0、2.0.0)组件, 搭建了用于实验的目标集群。集群配置中组件版本的搭配关系如表 3 所示。

表 3 目标集群配置

Table 3 Configuration for target Hive cluster

	Hive	Hadoop	Ranger	Sentry
集群 1	2.3.7	2.8.2	0.7.0	2.0.0
集群 2	2.3.0	2.8.2	0.7.0	2.0.0
集群 3	1.2.1	2.5.1	0.5.3	1.6.0

实验均运行在 8G 内存、4 核 Intel Xeon Gold 6161 CPU@2.20GHz 的 Ubuntu 18.04.3 LTS 系统宿主机上, 宿主机上部署了包含一个主节点和两个从节点的完全分布式集群, 其中每个节点都运行在一个 Docker 容器中。Hive 服务器、Ranger 服务器和 HiveAttacker 模糊测试主模块均运行在主节点上。

6.1 实验一: 定制化模糊测试

本节使用 HiveAttacker 第一阶段的定制化模糊测试功能对表 3 所列的 3 个目标集群分别进行模糊测试。实验过程中, 将 HiveAttacker 原型系统部署在各集群的主节点上, 被测目标集群上的 Hive 开启 Ranger 授权管理。为了避免随机因素的影响, 每个集群执行 5 轮测试, 每轮的时间上限为 8 h。

表 4 所示为 HiveAttacker 在各集群安装的 Hive 版本中挖掘出的漏洞。HiveAttacker 一共发现了 8 个与访问权限相关漏洞, 其中有 6 个是已知漏洞, 有 2 个是在最新的 2.3.7 版本中发现的漏洞。已知漏洞中除 3 个被授予了 CVE 编号的漏洞外, 其余 3 个 JIRA 事务因缺少或绕过权限检查, 同样可能对 Hive 的安全性造成影响。而在最新版本中, 2 个漏洞 Hive-23461[41]和 HIVE-12495[42]均已被开源社区确认, 但尚未修复。6.3 节将介绍 Hive-12495 的原理及挖掘过程。

通过分析上述 8 个漏洞所在的代码及触发原理, 我们发现, 这些漏洞都是由于权限检查缺失或不完善引起的。其中 5 个在 HiveAttacker 的预处理阶段被识别为疑似漏洞函数, 从而为模糊测试的输入生成提供导向。触发上述漏洞的输入所涉及的 SQL 语法, 包括语义解析语句(EXPLAIN)、查询语句(SELECT)、插入语句(INSERT)、导入语句(IMPORT)等, 以及分区表、视图、备份等高级特性, 说明 HiveAttacker 的定制化模糊测试策略能够在预处理的指导下, 生成丰富的语法现象组合及对应的权限违反。

图 8 进一步统计了 HiveAttacker 在定制化模糊测试阶段的时间开销分布: 预处理阶段, 对于大小为 33M 的 Hive 内核代码 Jar 包, 系统平均需要执行 228.89 s 得到疑似漏洞函数和 token 的键值对, 但其中 99.6%的时间都消耗在 WALA 构建类层次图和调用图的过程中; 权限违反制导的模糊测试阶段, 开销主要集中在 Hive 查询引擎进行查询编译和查询执行的过程中, 采用字节码插桩和动态追踪技术动态分析 Hive 和 Ranger, 仅会额外带来 13.68%的开销。

6.2 实验二: 组件间安全威胁检测

本节使用 HiveAttacker 第二阶段的组件间安全检测功能对表 3 所列的 3 个目标集群分别进行威胁检测, 共检测出的安全威胁共计 8 个, 检测结果如表 5 所示。经人工触发验证, 除 CVE-2018-8009[43]之外, 其余漏洞均可在对应集群中触发。其中, CVE-2018-

表 4 HiveAttacker 检测到的授权漏洞

Table 4 Discovered authorization-related vulnerabilities by HiveAttacker in Apache Hive

集群编号	Hive 版本	漏洞编号	状态	是否预处理标定	描述
集群 1	2.3.7	HIVE-23461	已确认	是	explainrewrite 语句缺少权限检查, 存在信息泄漏
集群 1	2.3.7	HIVE-12495	已确认	是	表并发锁缺少权限检查, 存在提权风险
集群 2	2.3.0	HIVE-17208	已知	是	集群备份操作缺少权限检查, 可能导致信息泄漏
集群 2	2.3.0	CVE-2018-1314	已知	是	explain 语句存在信息泄漏漏洞
集群 2	2.3.0	CVE-2017-12625	已知	否	列过滤机制在视图上存在信息泄漏
集群 3	1.2.1	CVE-2015-7521	已知	否	分区表的查询存在提权风险
集群 3	1.2.1	HIVE-12367	已知	是	数据库并发锁缺少权限检查, 存在提权风险
集群 3	1.2.1	HIVE-11988	已知	否	可以用 import 绕过 create 权限检查

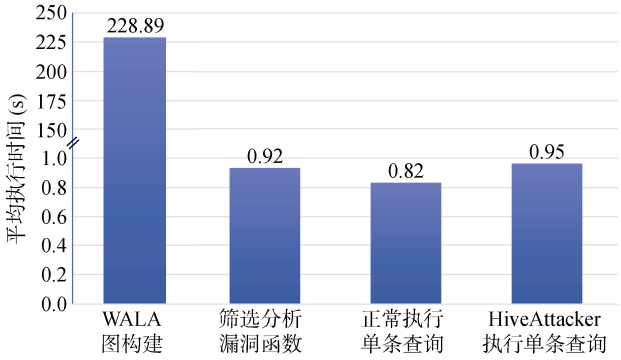


图 8　实验一开销分布

Figure 8　Overhead distribution of preprocessing

8009 的触发前提是向分布式缓存中上传 zip 格式的文件, 而 Hive 只在 MapJoin 查询中使用到分布式缓存, 并只支持上传 tar.gz 格式的文件, 触发条件不满足。在可触发的 7 个漏洞中, 5 个属于触发其他组件中的漏洞对 Hive 造成安全威胁, 2 个属于可通过 Hive 触发其他组件中的漏洞从而对后者造成安全威胁。6.3 节将从 3 个存在威胁的交互组件中各选择一个漏洞, 介绍检测过程及触发影响。通过对 HiveAttacker 在组件间安全威胁检测阶段的时间开销的进一步分析, 本阶段的检测开销主要花费在 WALA 构建类层次图、调用图以及程序依赖图的过程中。上述开销占本阶段总时间开销的 98.1%

6.3　案例分析

本节将通过 4 个案例, 介绍 HiveAttacker 挖掘的最新版本中的漏洞, 以及检测和辅助验证的组件间安全威胁的原理。

6.3.1　Hive 提权漏洞(HIVE-12495)

HiveAttacker 的定制化模糊测试阶段检测到 Hive 最新版本 2.3.7 中有一个权限提升漏洞。该漏洞所在的函数被阶段一的预处理模块识别为疑似漏洞函数, 后经模糊测试生成触发该漏洞的输入。此漏洞位于 Hive 用于解析 DDL 查询的 DDLSemanticAnalyzer 类中。漏洞函数在解析形如 lock/unlock table 的查询语句的过程中, 缺少对表类型读写实体的检查, 导致存在潜在的提权风险。在开启并发支持的多用户 Hive 集群中, 恶意用户可以对其他用户主动施加在的某个表上的锁进行解锁, 这一方面可能会导致并发的数据竞争, 影响集群的稳定性; 另一方面, 恶意用户可能借助该漏洞实现表中的数据泄漏和其他利用效果, 影响集群的安全性。

表 5　威胁检测实验结果表

Table 5　Result of threat detection evaluation

	检出威胁总数	已验证威胁总数	已验证漏洞编号	威胁来源	影响方向	触发效果
集群 1	6	5	CVE-2017-7676	Ranger	Ranger->Hive	权限提升
			CVE-2017-7677	Ranger	Ranger->Hive	权限提升
			CVE-2018-8028	Sentry	Sentry->Hive	权限提升
			SENTRY-2533	Sentry	Sentry->Hive	信息泄漏
			CVE-2017-15713	Yarn	Hive->Yarn	信息泄漏
集群 2	6	5	CVE-2017-7676	Ranger	Ranger->Hive	权限提升
			CVE-2017-7677	Ranger	Ranger->Hive	权限提升
			CVE-2018-8028	Sentry	Sentry->Hive	权限提升
			SENTRY-2533	Sentry	Sentry->Hive	信息泄漏
			CVE-2017-15713	Yarn	Hive->Yarn	信息泄漏
集群 3	3	3	CVE-2017-7677	Ranger	Ranger->Hive	权限提升
			CVE-2016-0760	Sentry	Sentry->Hive	代码执行
			CVE-2014-3627	Yarn	Hive->Yarn	文件提权

6.3.2　Sentry 授权管理威胁(SENTRY-2533)

HiveAttacker 的组件间安全威胁检测阶段检测到 Hive 最新版本 2.3.7 可能受到 Sentry 已知漏洞 SENTRY-2533 的影响。该漏洞对应的代码片段允许用户在查询语句中使用 in_file 函数。当 Hive 使用包含该漏洞的 Sentry 版本进行授权管理时, 普通用户可以通过执行包含 in_file 内建函数的恶意查询, 远程获取 Hive 服务器所在节点上的本地文件信息, 破坏分布式系统的安全边界。经验证, 该漏洞可在集群 1 中触发, 造成远程信息泄漏等安全风险, 属于攻击面二中由于 Hive 所依赖第三方组件中存在的漏洞影响 Hive 安全性的情况。

6.3.3　Ranger 授权管理威胁(CVE-2017-7677)

HiveAttacker 的组件间安全威胁检测阶段检测到

Hive 最新版本 2.3.7 可能受到 Ranger 已知漏洞 CVE-2017-7677[44]的影响。该漏洞对应的代码片段忽略了对外部位置的权限检查。当 Hive 使用包含该漏洞的 Ranger 版本进行授权管理时, 恶意用户可以通过创建或查询外部数据表绕过权限检查。经验证, 在集群 1 中, 攻击者可以利用此漏洞造成权限提升的利用效果, 属于攻击面二中由于 Hive 所依赖第三方组件中存在的漏洞影响 Hive 安全性的情况。

6.3.4 Yarn 信息泄漏威胁(CVE-2017-15713)

HiveAttacker的组件间安全威胁检测阶段检测到 Hive 最新版本 2.3.7 可能受到 Hadoop Yarn 已知漏洞 CVE-2017-15713[45]的影响。该漏洞存在于 Hadoop 2.8.3 之前版本的历史作业服务器(Job History Server)代码中。在开启历史作业服务器的漏洞 Hadoop 集群中, 攻击者可以利用该漏洞, 在 Hive 提交给 Yarn 的作业配置文件中构造恶意的 XML 实体引用并提交, 待执行完毕后会从前端泄漏出历史作业服务器本地目录文件的内容。经验证, 在集群 1 中, 攻击者可以利用此漏洞造成远程信息泄漏的利用效果, 属于攻击面二中通过执行 Hive 查询影响到 Hadoop 平台上用户的安全性的情况。

7 相关工作

模糊测试是目前通用的软件漏洞挖掘技术之一。相关研究工作通过对传统模糊测试技术的优化和定制, 能够有效发现操作系统内核、协议、开源库等传统目标程序中的漏洞。近年来, 通用模糊测试领域取得了蓬勃的发展。Angora[18]认为以 AFL 为代表的灰盒模糊测试技术的变异策略缺少数据流信息, 难以覆盖包含复杂约束的路径。Angora 引入动态污点分析技术计算输入字节偏移与分支约束的数据流依赖, 然后结合类型推断技术进行细粒度的变异, 按照“梯度下降”的原则求解约束, 获取较高的程序覆盖率和漏洞挖掘效率。QSYM[17]着重解决符号执行技术的性能瓶颈, 设计并实现了一个高效的混合执行(concolic)引擎用于通用模糊测试技术。其核心思想是用动态二进制翻译技术将符号模拟与本地执行紧密集成, 使细粒度和高效的符号模拟成为可能; QSYM 还降低了传统混合执行引擎严格的健壮性要求, 以获得较高的性能优化。MOPT[46]发现现存工具的变异策略大多使用均一分布作为变异操作的调度策略, 但不同变异操作触发新覆盖的效率差距很大。基于粒子群算法, MOPT 设计了一种调度策略, 迭代求解变异操作分布的全局最优解, 提高单次变异的平均效率。UnTracer[47]在覆盖反馈的模糊测试流程中加入了轻量级的过滤分析, 不能触发新覆盖的测试用例将不会被追踪, 从而提升整体的性能。

模糊测试的定制化是针对特定目标的有效漏洞挖掘手段之一。Razzer[14]针对操作系统内核中广泛发生的数据竞争漏洞, 提出用静态分析来定位潜在存在数据竞争的代码, 再结合确定性的线程交错技术控制线程调度, 以提供精确的并行执行信息, 降低内核模糊测试工具的不确定性, 缩小了内核数据竞争漏洞的触发窗口。SRFuzzer[15]针对 SOHO(小型公用/家用)路由设备的模糊测试进行了定制: 输入生成方面, 设计了 KEY-VALUE 和 CONF-READ 两个输入语义模型提高语义正确性; 漏洞覆盖方面, 设计了多种异常监控机制和多样化的变异规则, 涵盖内存破坏、Web 等多种类型的漏洞, 最终发现了 10 款路由设备的 97 个 0day 漏洞。JANUS[48]面向文件系统的模糊测试对输入生成阶段进行定制, 通过从文件系统镜像中精确定位并提取元数据, 然后基于文件系统的运行时状态来生成新的文件操作, 在镜像和文件操作两个输入维度上探索状态空间。SQUIRREl[30]是针对传统关系型数据库的模糊测试框架, 设计了一种中间表示(IR)维护结构化的 SQL 查询, 在 IR 上进行测试用例生成和变异, 保证语法和语义的正确性, 同时借鉴灰盒模糊测试的覆盖反馈进行迭代优化, 在 SQLite、MySQL、PostgreSQL 和 MariaDB 上发现了 12 个 CVE 漏洞, 但均为缓冲区溢出、内存泄漏等传统漏洞。Kelinci[49]和 Javafuzz[50]是面向 Java 程序的模糊测试工具, 二者均基于 AFL 的覆盖反馈算法实现, 但前者采用代理的方式与 AFL 直接交互, 后者用 Java 语言重写了模糊测试逻辑。Kelinci 和 Javafuzz 属于基于变异的模糊测试, 不适合处理结构化输入。Zest[51]面向具有结构化输入的 Java 程序, 基于 JQF[52]提供的 Java 程序模糊测试框架, 对覆盖反馈算法进行修改, 结合基于生成的模糊测试技术, 最终产生语法正确且结构多样化的输入, 能够较好地探索 Java 程序语义分析阶段的缺陷, 但 Zest 及 JQF 不支持多线程程序, 且高度依赖烦琐的需人工编写的生成器, 更重要的是, 它们只能监控 JVM 抛出的异常和超时异常, 很难发现真正影响程序安全性的漏洞。上述模糊测试技术及工具, 并未考虑对数据仓库及查询引擎中特有攻击面的支持, 无法直接应用于 Hive 及相关组件的漏洞检测与安全性分析。

当前与分布式处理平台及相关组件的缺陷检测、异常检测相关的研究工作大多侧重于对可能影响平台或组件的功能可用性以及性能的缺陷进行检测或定位[12-13]; 而围绕 Hive 的研究工作重点关注对 Hive

查询性能的优化[51-52], 对 Hive 及相关组件的安全性缺少足够的重视。本文借鉴软工领域和模糊测试技术领域的方法论, 针对数据仓库及查询引擎 Hive 的安全性测试提出定制化模糊测试与组件间安全检测相结合的两阶段测试与检测方法, 弥补相关研究工作对安全性重视不足的问题。

8 总结

本文以 Hadoop 平台上的 Hive 数据仓库及查询引擎为切入点, 归纳了 Hive 在查询解析过程中, 以及在与 Hadoop 平台或其他第三方组件交互过程中面临的两个主要攻击面, 并针对性地设计了一个两阶段安全性检测方案。方案的第一阶段针对 Hive 因接收、解析用户查询所引入的攻击面, 对传统模糊测试技术进行定制化扩展, 重点挖掘 Hive 自身代码中可能导致权限提升、授权绕过等利用效果的漏洞; 第二阶段针对 Hive 因与其他组件交互引入的攻击面, 重点检测可能通过组件间交互触发的漏洞, 并进行预警。基于上述方案实现的原型工具 HiveAttacker, 在 Hive 两个历史版本及最新版本中共挖掘出 8 个漏洞, 其中包含 2 个最新版本中仍未修复的漏洞; 在搭建的真实 Hive 运行环境中检测并确认出因组件交互引入的安全威胁 7 处, 验证了方案的有效性。

下一步拟开展的工作包括将所设计的安全性检测方案推广到 Hive 之外的大数据查询系统以及 Hadoop 之外的大数据平台上, 以及针对组件间安全威胁的自动化确认。

参考文献

[1] iCloud leaks of celebrity photo[Z]. https://en.wikipedia.org/wiki/ICloud_leaks_of_celebrity_photos. Nov. 2020.

[2] Apahce Hadoop[Z]. https://hadoop.apache.org/. Nov. 2020.

[3] Apache Hive[Z]. https://hive.apache.org/. Nov. 2020.

[4] Miao L Y. Research on hadoop security mechanism[D]. Nanjing: Nanjing University of Posts and Telecommunications, 2015.
(缪璐瑶. Hadoop 安全机制研究[D]. 南京: 南京邮电大学, 2015.)

[5] Apache Hadoop Security Vulnerabilities[Z]. https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-22215/Apache-Hadoop.html. Nov. 2020.

[6] Apache Hive Security Vulnerabilities[Z]. https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-30309/Apache-Hive.html. Nov. 2020

[7] Kerberos: The Network Authentication Protocol[Z]. https://web.mit.edu/kerberos/. Nov. 2020.

[8] Apache Ranger[Z] http://ranger.apache.org/. Nov. 2020.

[9] Apache Sentry[Z] https://sentry.apache.org/. Nov. 2020.

[10] JIRA of Apache Hadoop Common[Z]. https://issues.apache.org/ jira/projects/HADOOP/issues. Nov. 2020

[11] JIRA of Apache Hive[Z]. https://issues.apache.org/jira/projects/HIVE/issues. Nov. 2020.

[12] Lu J, Li F, Li L, et al. CloudRaid: Hunting Concurrency Bugs in the Cloud via Log-Mining[C]. *The 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018: 3-14.

[13] Lu J, Liu C, Li L, et al. CrashTuner: Detecting Crash-Recovery Bugs in Cloud Systems *via* Meta-Info Analysis[C]. *The 27th ACM Symposium on Operating Systems Principles*, 2019: 114-130.

[14] Jeong D R, Kim K, Shivakumar B, et al. Razzer: Finding Kernel Race Bugs through Fuzzing[C]. *2019 IEEE Symposium on Security and Privacy*, 2019: 754-768.

[15] Zhang Y, Huo W, Jian K P, et al. SRFuzzer: An Automatic Fuzzing Framework for Physical SOHO Router Devices to Discover Multi-Type Vulnerabilities[C]. *The 35th Annual Computer Security Applications Conference*, 2019: 544-556.

[16] Han H, Oh D, Cha S K. CodeAlchemist: Semantics-Aware Code Generation to Find Vulnerabilities in JavaScript Engines[C]. *Proceedings 2019 Network and Distributed System Security Symposium*, 2019.

[17] Yun I, Lee S, Xu M, et al. QSYM: A Practical Concolic Execution Engine Tailored for Hybrid Fuzzing[C]. *The 27th USENIX Conference on Security Symposium*, 2018: 745-761.

[18] Chen P, Chen H. Angora: Efficient Fuzzing by Principled Search[C]. *2018 IEEE Symposium on Security and Privacy*, 2018: 711-725.

[19] Wang J J, Chen B H, Wei L, et al. Superion: Grammar-Aware Greybox Fuzzing[C]. *2019 IEEE/ACM 41st International Conference on Software Engineering*, 2019: 724-735.

[20] AddressSanitizer[Z]. https://github.com/google/sanitizers/wiki/AddressSanitizer. Nov. 2020.

[21] MemorySanitizer[Z]. https://github.com/google/sanitizers/wiki/MemorySanitizer. Nov. 2020.

[22] Apache Tez[Z]. https://tez.apache.org/. Nov. 2020.

[23] Apache Spark[Z]. https://spark.apache.org/. Nov. 2020.

[24] Apache Hadoop Yarn[Z]. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html. Nov. 2020.

[25] HDFS Architecture[Z]. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. Nov. 2020.

[26] Attack Surface Analysis[Z]. https://cheatsheetseries.owasp.org/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet.html. Nov. 2020

[27] CVE-2018-1282[Z]. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1282. Nov. 2020.

[28] CVE-2015-7521[Z]. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-7521. Nov. 2020.

[29] “Wikipedia of Fuzzing”[Z]. https://zh.wikipedia.org/wiki/%E6%A8%A1%E7%B3%8A%E6%B5%8B%E8%AF%95. Nov. 2020.

[30] Zhong R, Chen Y H, Hu H, et al. SQUIRREL: Testing Database Management Systems with Language Validity and Coverage Feedback[EB/OL]. 2020: arXiv: 2006.02398. https://arxiv.org/abs/2006.02398.

[31] Zalewski M. American Fuzzy Lop (2.52b)[Z]. http://lcamtuf.coredump.cx/afl, Nov. 2020.

[32] Peach Fuzzer[Z]. https://www.peach.tech/products/peach-fuzzer/. Nov. 2020.

[33] A fork and successor of the Sulley Fuzzing Framework[Z]. https://github.com/jtpereyda/boofuzz. Nov. 2020.

[34] LLVM. LibFuzzer - A Library For Coverage-guided Fuzz Testing, 2017[Z]. http: //llvm.org/docs/LibFuzzer.html. Nov. 2020.

[35] Vulnerabilities found in Ranger[Z]. https://cwiki.apache.org/ confluence/display/RANGER/Vulnerabilities+found+in+Ranger. Nov. 2020.

[36] T.J. Watson Libraries for Analysis[Z]. https://github.com/wala/WALA. Nov. 2020.

[37] Fonseca R, Porter G, Katz R H, et al. X-trace: A pervasive network tracing framework[C]. 4th USENIX Symposium on Networked Systems Design & Implementation, 2007.

[38] Javassist by jboss-javassist[Z]. https://www.javassist.org/. Nov. 2020.

[39] Hodován R, Kiss Á, Gyimóthy T. Grammarinator: A Grammar-Based Open Source Fuzzer[C]. *The 9th ACM SIGSOFT International Workshop on Automating TEST Case Design*, *Selection*, *and Evaluation*, 2018: 45-48.

[40] ANTLR[Z]. https://www.antlr.org/. Nov. 2020.

[41] HIVE-23461[Z]. https://issues.apache.org/jira/browse/HIVE-23461. Nov. 2020.

[42] HIVE-12495[Z]. https://issues.apache.org/jira/browse/HIVE-12495. Nov. 2020.

[43] CVE-2018-8009[Z]. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8009. Nov. 2020.

[44] CVE-2017-7677[Z]. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7677. Nov. 2020.

[45] CVE-2017-15713[Z]. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-15713. Nov. 2020.

[46] Lyu C, Ji S, Zhang C, et al. MOPT: Optimized mutation scheduling for fuzzers[C]. *28th USENIX Security Symposium*, 2019: 1949-1966.

[47] Nagy S, Hicks M. Full-Speed Fuzzing: Reducing Fuzzing Overhead through Coverage-Guided Tracing[C]. 2019 *IEEE Symposium on Security and Privacy*, 2019: 787-802.

[48] Xu W, Moon H, Kashyap S, et al. Fuzzing File Systems *via* Two-Dimensional Input Space Exploration[C]. *2019 IEEE Symposium on Security and Privacy*, 2019: 818-834.

[49] AFL-based fuzzing for Java[Z]. https://github.com/isstac/kelinci. Nov. 2020.

[50] coverage guided fuzz testing for Java[Z]. https://github.com/ fuzzitdev/javafuzz. Nov. 2020.

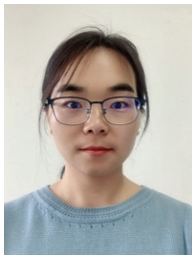
[51] Padhye R, Lemieux C, Sen K, et al. Semantic Fuzzing with Zest[C]. *The 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019: 329-340.

[52] Padhye R, Lemieux C, Sen K. JQF: Coverage-Guided Property-Based Testing in Java[C]. *The 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019: 398-401.

李文超 于 2018 年在吉林大学软件工程专业获得学士学位。现在中国科学院大学网络空间安全专业攻读硕士学位。研究领域为系统安全。研究兴趣包括漏洞挖掘、模糊测试、大数据安全。Email: liwenchao@iie.ac.cn

李丰 于 2013 年在中国科学院大学获博士学位。现任中国科学院信息工程研究所副研究员。研究方向为程序分析与软件漏洞挖掘。Email: lifeng@iie.ac.cn

薄德芳 于 2019 年在北京邮电大学软件工程专业获得学士学位。现在中国科学院大学网络空间安全专业攻读硕士学位。研究领域为系统安全。研究兴趣包括静态缺陷检测。Email: bodefang@iie.ac.cn



周建华 于2013年在北京科技大学通信工程专业获硕士学位, 现任中国科学院信息工程研究所工程师。研究领域为软件安全分析。研究兴趣包括代码审计、漏洞挖掘。Email: zhoujianhua@iie.ac.cn

霍玮 于 2010 年在中国科学院大学计算技术研究所获博士学位。现任中国科学院信息工程研究所博士生导师, 中国科学院青年创新促进会成员。主要研究领域包括软件漏洞挖掘、利用和安全评测、基于大数据及知识图谱的软件安全分析、信息系统安全分析等。Email: huowei@iie.ac.cn