大数据平台 Hadoop 的安全机制

本章学习目标

- 了解 Hadoop 的安全机制
- 理解 Hadoop 组件的安全机制
- 分析 Hadoop 的安全性
- 掌握 Hadoop 的安全架构

Hadoop 是常用的大数据处理平台，其安全性对于业务数据处理尤为重要。本章主要向读者介绍 Hadoop 的安全机制、Hadoop 组件的安全机制、安全技术工具、Hadoop 的安全性分析，并对其安全架构进行阐述。

3.1 安全威胁概述

Hadoop 是 Apache 旗下基于 Java 语言实现的开源大数据计算处理框架，允许使用简单的编程模型在计算机集群上对大型数据集进行分布式处理。正因为 Hadoop 在处理大数据方面具有高效性、高容错性、高可扩展性和低成本等优势，当前许多企业或机关事业单位等都采用 Hadoop 来存储和处理海量数据。Hadoop 由于自身的业务特点，一般都部署在用户内网中，所以在早期设计的时候不是太注重安全方面的设计，而更多的是专注于实现业务的功能。下面是网上报道过的关于其安全漏洞的案例。

Apache 的 Ambari 引用给 Hadoop 带来了很多便利，可以直接通过外部的管理对 Hadoop 的生态组件进行管控，但在这个过程中由于外部技术的引用，导致一些外部应用层的漏洞，主要是 SSRF 伪造请求漏洞。恶意攻击者通过 SSRF 攻击，远程对 Hadoop 服务以及进程进行操纵和读取数据。

MapReduce 信息漏洞主要是由于数据文件、用户产生的数据以及加密密钥都存储在同一个文件和磁盘中，导致恶意用户获取到加密密钥，并读取了数据块中的数据。

Ambari 重定向漏洞是由于 target 的 URI 参数被修改成了黑客指定的任意网址，由此造成钓鱼攻击，甚至结合 Linux 底层的操作系统漏洞以及 Hadoop 的其他漏洞还能实现恶意代码的加载。

2017 年，Hadoop 的一个新的漏洞被曝光，漏洞的主要原因在于 Hadoop 引入了 Docker 组件，但是在 Linux 这层没有输入过程的认证，并且 Docker 命令又是通过 root 身

份来执行的。因此，黑客就通过 Docker 命令伪造了 root 身份，然后对 Hadoop 进行了全线账户的提权，实现了整个操作系统的权限控制。

通过上述案例可以发现，这些安全漏洞的产生大部分都是由于 Hadoop 的身份认证授权没有做好。早期 Hadoop 在默认情况下，是没有身份认证和访问控制机制的，基本上是继承了 Linux 的权限控制体系。另外，它在数据传输过程中和静态数据保存过程中都没有有效的加密措施。

3.2 Hadoop 安全机制

3.2.1 基本安全机制

为了增强 Hadoop 的安全机制，从 2009 年起，Apache 专门组织了一个团队，为 Hadoop 增加基于 Kerberos 和 Delegation Token 的安全认证和授权机制。Apache Hadoop 1.0.0 和 Cloudera CDH3 之后的版本添加了安全机制。Hadoop 提供了两种安全机制：Simple 机制和 Kerberos 机制。

1. Simple 机制

Simple 机制是 JAAS（Java Authentication and Authorization Service）协议与 Delegation Token 结合的一种机制，JAAS 提供 Java 认证与授权服务。

（1）用户提交作业时，JobTracker 端要进行身份核实，先是认证到底是不是这个人，即通过检查执行当前代码的人与 JobConf 中的 user.name 中的用户是否一致。

（2）然后检查 ACL（Access Control List）配置文件（由管理员配置）确认是否有提交作业的权限。

一旦通过认证，会获取 HDFS 或者 MapReduce 授予的 Delegation Token（访问不同模块有不同的 Delegation Token），之后的任何操作，比如访问文件，均要检查该 Token 是否存在，以及使用者跟之前注册使用该 Token 的用户是否一致。

2. Kerberos 机制

Kerberos 机制是基于认证服务器的一种方式。简单来说，大数据平台 Hadoop 中有一个专门的认证服务器 KDC，可以把它看作户籍派出所，它可事先给所有的平台用户发放户籍证明，即 keytab（密钥表）。之后某个用户要使用大数据平台，就要拿着这个证明先去 KDC 认证，认证无误之后，才能够使用大数据平台服务。Kerberos 机制工作原理示意如图 3.1 所示。

在身份认证方面，在 Hadoop 集群内部，一般使用基于 Kerberos 的身份认证机制。主要原因在于，该机制具有如下优势：①可靠，Hadoop 本身并没有认证功能和创建用户组功能，只是使用依靠外围的认证系统；②高效，Kerberos 使用对称密码操作，比使用 SSL 的公钥密码快；③操作简单，用户可以方便地进行操作，不需要很复杂的指令，比如废除一个用户，只从 Kerberos 的 KDC 数据库中删除即可。

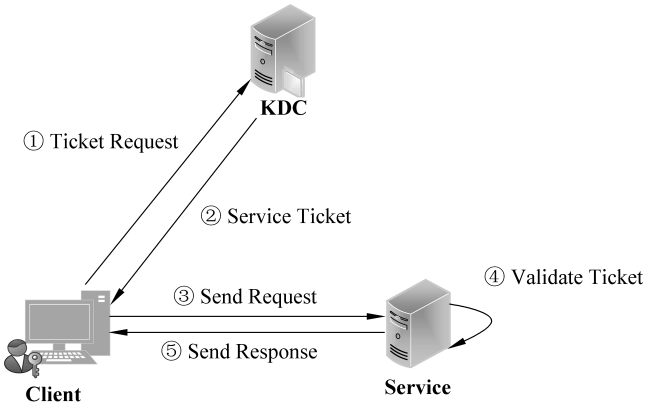

图 3.1　Kerberos 机制工作原理示意

3.2.2　总体安全机制

Hadoop 的安全机制如下所述。

(1) Hadoop 的客户端通过 Hadoop 的 RPC 库访问相应的服务，Hadoop 在 RPC 层中添加了权限认证机制，所有 RPC 都会使用 SASL(Simple Authentication and Security Layer)进行连接。其中，SASL 协商使用 Kerberos 或者 DIGEST MD5 协议。

(2) HDFS 使用的认证可以分成两部分：第一部分是客户端与 NameNode 连接时的认证；第二部分是客户端从 DataNode 获取 Block 时所需要的认证。第一部分使用 Kerberos 协议认证和授权令牌(Delegation Token)认证，这个授权令牌可以作为接下来访问 HDFS 的凭证。第二部分则是客户端从 NameNode 获取一个认证令牌，只有使用这个令牌，才能从相应的 DataNode 获取 Block。

(3) 在 MapReduce 中用户的每个 Task 均使用用户的身份运行，这样就防止了恶意用户使用 Task 干扰 TaskTracker 或者其他用户的 Task。

(4) HDFS 在启动时，NameNode 首先进入一个安全模式，此时系统不会写入任何数据。NameNode 在安全模式下会检测数据块的最小副本数，当一定比例的数据块达到最小副本数时(一般为 3)，系统就会退出安全模式，否则补全副本，以达到一定的数据块比例。

(5) 当从 HDFS 获得数据时，客户端会检测从 DataNode 收到的数据块，通过检测每个数据块的校验和(Checksum)来认证这个数据块是否损坏。如果损坏，则从其他 DataNode 获得这个数据块的副本，以保证数据的完整性和可用性。

(6) MapReduce 和 HDFS 都设计了心跳机制，Task 和 DataNode 都定期向 JobTracker 和 NameNode 发送信条数据。当 JobTracker 不能接收到某个 Task 的心跳数据时，则认为该 Task 已经失败，会在另一个节点上重启该任务，以保证整个 MapReduce 程序的运行。同理，如果 NameNode 收不到某个 DataNode 的心跳消息，也认为该节点已经死掉，不会向该节点发送新的 I/O 任务，并复制那些丢失的数据块。

3.3 Hadoop组件的安全机制

Hadoop安全性与其组件安全机制息息相关。本节进一步介绍Hadoop中RPC、HDFS、MapReduce的安全机制。

3.3.1 RPC安全机制

RPC是指远程过程调用，也就是说，两台不同的服务器（不受操作系统限制），一个应用部署在A上，一个应用部署在B上，若A想要调用B上的某个方法，由于不在一个内存空间，不能直接调用，因此需要通过网络来表达调用的语意和传达调用的参数。

Hadoop集群是Master/Slave（主/从）结构，Master包括NameNode和JobTracker，Slave包括DataNode和TaskTracker。NameNode可看作分布式文件系统中的管理者，主要负责管理文件系统的命名空间、集群配置信息和存储块的复制等。NameNode会将文件系统的MetaData存储在内存中，这些信息主要包括文件信息、每一个文件对应的文件块的信息和每一个文件块在DataNode中的信息等。DataNode是文件存储的基本单元，它将Block存储在本地文件系统中，保存了Block的MetaData，同时周期性地将所有存在的Block信息发送给NameNode。Client就是需要获取分布式文件系统文件的应用程序。就通信方式而言，Client与NameNode、NameNode与DataNode都是在不同进程、不同系统间的通信，因此Hadoop要用到RPC。

RPC安全机制是在Hadoop RP中添加权限认证授权机制。当用户调用RPC时，用户的Login Name会通过RPC头部传递给RPC，之后RPC使用SASL确定一个权限协议（支持Kerberos和DIGEST-MD5两种），完成RPC授权。

3.3.2 HDFS安全机制

客户端获取NameNode初始访问认证（使用Kerberos）后，获取一个Delegation Token，该Token可以作为接下来访问HDFS或者提交作业的凭证。为了读取某个文件，客户端首先要与NameNode交互，获取对应数据块的Block Access Token，然后到相应的DataNode上读取各个数据块，而DataNode在初始启动后向NameNode注册时，已经提前获取了这些Token，当客户端要从TaskTracker上读取数据块时，首先认证Token，通过后才允许读取。

1. Delegation Token

当用户使用Kerberos证书向NameNode提交认证后，从NameNode获得一个Delegation Token，之后该用户提交作业时可使用该Delegation Token进行身份认证。Delegation Token是用户和NameNode之间的共享密钥，获取Delegation Token的任何人都可以假冒该用户。只有当用户再次使用Kerberos认证时，才会再次得到一个新的Delegation Token。

当从NameNode获得Delegation Token时，用户应该告诉NameNode这个Token

的 renewer(更新者)。在对该用户的 Token 进行更新之前,更新者先向 NameNode 进行认证。Token 的更新将延长该 Token 在 NameNode 上的有效时间,而非产生一个新的 Token。为了让一个 MapReduce 作业使用一个 Delegation Token,用户通常需要将 JobTracker 作为 Dolegation Toben 的更新者。同一个作业下的所有任务使用同一个 Token。在作业完成之前,JobTracker 确保这些 Token 是有效的;在作业完成之后,JobTracker 就可以废除这个 Token。

NameNode 随机选取 masterKey,并用它生成和认证 Delegation Token,保存在 NameNode 的内存中,每个 Delegation Token 都有一个 Token,存在 expiryDate(过期时间)。如果 currentTime>expiryDate,该 Token 将被认为是过期的,任何使用该 Token 的认证请求都将被拒绝。NameNode 将过期的 Delegation Token 从内存中删除,另外,如果 Token 的 owner(拥有者)和 renewer 废除了该 Token,则 NameNode 将这个 Delegation Token 从内容中删除。Sequence Nunber(序列号)随着新的 Delegation Token 的产生不断增大,唯一标识每个 Token。

当客户端(如一个 Task)使用 Delegation Token 认证时,首先向 NameNode 发送 Token。Token ID 代表客户端将要使用的 Delegation Token。NameNode 利用 Token ID 和 masterKey 重新计算出 Delegation Token,然后检查其是否有效。当且仅当该 Token 存在于 NameNode 内存中,并且当前时间小于过期时间时,这个 Token 才算是有效的。如果 Token 是有效的,则客户端和 NameNode 就会使用它们自己的 Token Authenticator 作为密钥、DIGEST MD5 作为协议相互认证。以上双方认证过程中,都未泄露自己的 Toke Authenticator 给另一方。如果双方认证失败,意味着客户端和 NameNode 没有共享同一个 Token Authenticator,那么它们也不会知道对方的 Token Authenticator。

为了保证有效,Delegation Token 需要定时更新。假设 JobTracker 是一个 Token 的更新者,在 JobTacker 向 NameNode 成功认证后,JobTracker 向 NameNode 发送要被更新的 Token。

NameNode 将进行如下认证。

(1) JobTracker 是 Token ID 中指定的更新者。

(2) Token Authenticator 是正确的。

(3) currentTime < maxDate。

认证成功之后,如果该 Token 在 NameNode 内存中,即该 Token 是有效的,则 NameNode 将其新 expiryDate 设置为 min(currentTime+renewPeriod, maxDate),如果这个 Token 不在内存中,说明 NameNode 重启丢失了之前内存中保存的 Token,则 NameNode 将这个 Token 添加到内存中,并且用相同的方法设置其 expiryDate,使得 NameNode 重启后作业依然可以运行。JobTracker 需要在重新运行 Tasks 失败之前,向 NameNode 更新所有的 Delegation Token。

注意:只要 currentTime < maxDate,那么即使这个 Token 已经过期,更新者依然可以更新它。因为 NameNode 无法判断一个 Token 过期与否(或是否被废除),或是由于 NameNode 重启导致其不在内存中。只有被指定的更新者可以使一个过期的 Token 复活,即便攻击者窃取到了这个 Token,也不能更新使其复活。

masterKey需要定时更新，NameNode只需要将masterKey而不是Tokens保存在磁盘上。

2. Block Access Token

早期的Hadoop并没有对Block(数据块)添加访问控制，对于一个未认证的客户端，只要它能获得数据块的Block ID，就可以读取数据块。除此之外，任何人都可以向DataNode写任意数据。

当用户向NameNode请求访问文件时，NameNode进行文件权限检查。NameNode根据对用户所请求的文件(即相关的数据块)是否具有相应权限来做出授权。然而，对于DataNode中的数据块，以上权限授权是无用的，因为DataNode没有文件的概念，更不用提文件权限了。

为了在HDFS上实施一致的数据访问控制策略，需要一个机制来将NameNode上的访问授权实施到DataNode上，并且任何未授权的访问将被拒绝。

NameNode通过使用Block Access Token向DataNode传递数据访问权限授权信息。Block Access Token由NameNode生成，在DataNode上使用，其拥有者能够访问DataNode中的特定数据块，而DataNode能够认证其授权。

Block Access Token通过对称密钥机制生成，NameNode和所有的DataNode共享一个密钥。对于每一个Token，NameNode使用这个共享密钥计算出一个加密的哈希值(MAC)，这个哈希值就是Token Authenticator。Token Authenticator是构成Block Access Token的必要部分。当DataNode收到一个Token时，它使用自己的密钥重新计算出Token Authenticator，并将其与接收到的Token中的Token Authenticator进行比较，如果匹配，则认为这个Token是可信的。因为只有NameNode和DataNode知道密钥，所以第三方无法伪造Token。

若使用公钥机制生成Token，则计算成本较为昂贵。其主要优点是：即使一个DataNode被攻陷，攻击者也不会获得能够伪造出有效Token的密钥。然而，通常在HDFS部署中，所有DataNode的保护措施都是相同的(相同的数据中心、相同的防火墙策略)。如果攻击者有能力攻陷一个DataNode，就能够利用相同的手段攻陷所有的DataNode，而不必使用密钥。因此，使用公钥机制不会带来根本性的差异。

理想情况下，Block Access Token是不可转移的，仅其拥有者可以使用它。Token中包含了其拥有者的ID，无论谁使用这个Token，都要认证其是否为拥有者，所以没有必要担心Token的丢失。在当前的安全机制中，Block Access Token中包含其拥有者的ID，但DataNode并不认证其拥有者的ID，预计以后会添加相关认证。

无须更新或者废除一个Block Access Token。当一个Block Access Token过期时，只需获取一个新的Token。Block Access Token保存在内存中，无须写入磁盘中。Block Access Token的使用场景如下：HDFS客户端向NameNode请求一个文件的Block ID和所在位置；NameNode认证该客户端是否被授权访问这个文件，然后将所需的Block ID和对应的Block Access Token发送给客户端；当客户端需要访问一个数据块时，将向DataNode发送Block ID和对应的Block Access Token；DataNode认证收到的Block

Access Token,判断是否客户端允许访问数据块。HDFS 客户端把从 NameNode 获取的 Block Access Token 保存在内存中,当 Token 过期或者访问到未缓存的数据块时,客户端会向 NameNode 请求新的 Token。

无论数据块实际存储在哪里,Block Access Token 在所有的 DataNode 上都是有效的。NameNode 随机选取计算 Token Authenticator 的密钥,当 DataNode 首次向 NameNode 注册时,NameNode 将密钥发送给该 DataNode。NameNode 上有一个密钥滚动生成机制以更新密钥,并定期将新的密钥发送给 DataNode。

3.3.3 MapReduce 安全机制

MapReduce 也是 Hadoop 中的核心组件之一,它为海量的数据提供计算。其安全机制如下。

1. 作业提交

用户提交作业(Job Submission)后,JobClient 需与 NameNode 和 JobTracker 等服务进行通信,以进行身份认证和获取相应令牌。授权用户提交作业时,JobTracker 会为之生成一个 Delegation Token,该 Token 将被作为 Job 的一部分存储到 HDFS 上,并通过 RPC 分发给各个 TaskTracker,一旦作业运行结束,该 Token 失效。

2. 作业控制

用户提交作业时,可通过参数 mapreduce.job.acl-view-job 指定哪些用户或者用户组可以查看作业状态,也可以通过 mapreduce.job.acl-modify-job 指定哪些用户或者用户组可以修改或者关闭作业。

3. 任务启动

TaskTracker 收到 JobTracker 分配的任务后,如果该任务来自某个作业的第一个任务,则会进行作业本地化:将任务运行相关的文件下载到本地目录中,其中,作业令牌文件会被写到 ${mapred.local.dir}/ttprivate/taskTracker/${user}/jobcache/${jobid}/jobToken 目录下。由于只有该作业的拥有者可以访问该目录,因此令牌文件是安全的。此外,任务要使用作业令牌向 TaskTracker 进行安全认证,以请求新的任务或者汇报任务状态。

4. 任务(Task)运行

用户提交作业的每个 Task 均是以用户身份启动的,这样,一个用户的 Task 便不可以向 TaskTracker 或者其他用户的 Task 发送操作系统信号,对其他用户造成干扰。这要求为每个用户在所有 TaskTracker 上建一个账号。

5. Shuffle

当一个 MapTask 运行结束时,它要将计算结果告诉管理它的 TaskTracker,之后每个 ReduceTask 会通过 HTTP 向该 TaskTracker 请求自己要处理的那块数据,Hadoop

应该确保其他用户不可以获取 MapTask 的中间结果，其做法是：ReduceTask 对“请求 URL”和“当前时间”计算 HMAC-SHA1 值，并将该值作为请求的一部分发动给 TaskTracker，TaskTracker 收到后会认证该值的正确性。

3.4 安全技术工具

为保障 Hadoop 生态环境安全，除 Hadoop 自身的安全配置之外，也可以借助外部安全工具。根据侧重点不同，安全技术工具可分为系统安全工具和认证授权工具两类。其中，系统安全工具包括 Ganglia、Nagios、Ambari 等，认证授权工具主要指 Apache Sentry 与 Ranger。本节将主要对以上几类安全工具进行详细介绍。

3.4.1 系统安全工具

1. Ganglia

Ganglia 是 UC Berkeley 发起的一个开源集群监视项目，用于测量数以千计的节点。Ganglia 的核心包含 gmond、gmetad 及一个 Web 前端，主要用来监控系统性能，如 CPU、内存、硬盘利用率、I/O 负载、网络流量情况等。通过曲线可以看到每个节点的工作状态，对合理调整、分配系统资源、提高系统整体性能起到重要作用。

每台计算机都运行一个收集和发送度量数据的 gmond 守护进程。接收所有度量数据的主机可以显示这些数据，并且可以将这些数据的精简表单传递到层次结构中。这种层次结构模式使得 Ganglia 可以实现良好的扩展，同时 gmond 带来的系统负载非常少，使得它成为在集群中各台计算机上运行的一段代码，不会影响用户性能，但所有这些数据多次收集就会影响节点性能。网络中的“抖动”发生在大量小消息同时出现时，可以通过将节点时钟保持一致的方式来解决问题。

gmetad 可以部署在集群内任一节点或者通过网络连接到集群的独立主机，它通过单播路由的方式与 gmond 通信，收集区域内节点的状态信息，并以 XML 数据的形式保存在数据库中。

由 RRDtool 处理数据并生成相应的图形显示，以 Web 方式直观地提供给客户端。

2. Nagios

Nagios 是一个监视系统运行状态和网络信息的监视系统，能够监视所指定的本地或远程主机及服务，同时提供异常通知功能。

它可以运行在 Linux/UNIX 平台上，同时提供一个可选的基于浏览器的 Web 界面，以方便系统管理人员查看网络状态、各种系统问题及日志等。

Nagios 可以监控的功能包括：

(1) 监控网络服务(如 SMTP、POP3、HTTP、NNTP、PING 等)。

(2) 监控主机资源(如处理器负荷、磁盘利用率等)。

(3) 简单的插件设计使得用户可以方便地扩展自己服务的检测方法。

(4) 并行服务检查机制。

(5) 具备定义网络分层结构的能力,采用 parent 主机定义来表达网络主机之间的关系,这种关系可被用来发现和明晰主机死机或不可达状态。

(6) 当服务或主机问题产生与解决时,将告警发送给联系人(通过 E-mail、短信、用户定义方式)。

(7) 定义一些处理程序,能够在服务或者主机发生故障时起到预防作用。

(8) 自动的日志滚动功能。

(9) 支持并实现对主机的冗余监控。

(10) 可选的 Web 界面用于查看当前的网络状态、通知、故障历史、日志文件等。

(11) 通过手机查看系统监控信息。

(12) 指定自定义的事件处理控制器。

3. Ambari

Apache Ambari 是一个基于 Web 的工具,用于配置、管理和监视 Hadoop 集群,支持 HDFS、MapReduce、Hive、HCatalog、HBase、ZooKeeper、Oozie、Pig 和 Sqoop,同时提供了集群状况仪表盘,如 Heatmaps 和查看 MapReduce、Pig、Hive 应用程序的能力,以友好的用户界面对性能特性进行诊断。

Ambari 充分利用了已有的优秀开源软件,巧妙地将它们结合起来,在分布式环境中具有集群式服务管理能力、监控能力、展示能力。相关的开源软件包括:

(1) 在 Agent 端,采用 puppet 管理节点。

(2) 在 Web 端,采用 ember.js 作为前端 MVC 框架和 NodeJS 相关工具,handlebars.js 作为页面渲染引擎,在 CSS/HTML 方面使用 Bootstrap 框架。

(3) 在 Server 端,采用 Jetty、Spring、JAX-RS 等。

(4) 同时利用 Ganglia、Nagios 的分布式监控能力。

Ambari 采用 Server/Client 的框架模式,主要由 ambari-agent 和 ambari-server 两部分组成。Ambari 依赖其他已经成熟的工具,如 ambari-server 依赖 python,而 ambari-agent 依赖 ruby、puppet、facter 等工具,也依赖一些监控工具,如 Nagios 和 Ganglia 用于监控集群状况。其中,puppet 是分布式集群配置管理工具,也是典型的 Server/Client 模式,能够集中管理分布式集群的安装配置部署,主要语言是 ruby;facter 是使用 Python 编写的一个节点资源采集库,用于采集节点的系统信息,如操作系统信息。由于 ambari-agent 主要使用 Python 编写,因此使用 facter 可以很好地采集节点信息。

Ambari 项目目录介绍见表 3.1。

表 3.1 Ambari 项目目录介绍

目 录	描 述
ambari-server	Ambari 的 Server 程序,主要管理部署在每个节点上的管理监控程序
ambari-agent	部署在监控节点上运行的管理监控程序

续表

目　录	描　述
Contrib	自定义第三方库
ambari-web	Ambari页面UI的代码，作为用户与ambari-server的交互
ambari-views	用于扩展ambari-web UI中的框架
Docs	文档
ambari-common	ambari-server和ambari-agent共用的代码

3.4.2 Apache Sentry

1. Sentry 介绍

Apache Sentry是Cloudera公司发布的一个Hadoop开源组件，它提供了细粒度级、基于角色的授权以及多租户的管理模式。它在Hadoop生态中扮演着“守门人”的角色，看守着大数据平台的数据安全的访问。它以插件的形式运行于组件中，通过关系型数据库(或本地文件)来存取访问策略，对数据使用者提供细粒度的访问控制。

Sentry仅支持基于角色的访问控制。无法直接向用户或组授予权限，需要在角色下组合权限。只能将角色授予组，而不能将角色直接授予用户。

Sentry授权包括以下4种角色。

(1) 资源。资源是要管理访问权限的对象，可能是Server、Database、Table或者URL(如HDFS或本地路径)。Sentry支持对列进行授权。

(2) 权限。权限的本质是授权访问某一个资源的规则。

(3) 角色。角色是一系列权限的集合。

(4) 用户和组。一个组是一系列用户的集合。Sentry的组映射是可以扩展的。默认情况下，Sentry使用Hadoop的组映射(可以是操作系统组或LDAP中的组)。Sentry允许将用户和组进行关联，可以将一系列用户放入一个组中。Sentry不能直接给一个用户或组授权，需要将权限授权给角色，角色可以授权给一个组，而不是一个用户。

2. Sentry 特性

Apache Sentry为Hadoop使用者提供了以下便利。

(1) 能够在Hadoop中存储更敏感的数据。

(2) 使更多的终端用户拥有Hadoop数据访问权。

(3) 创建更多的Hadoop使用案例。

(4) 构建多用户应用程序。

(5) 符合规范(如SOX、PCI、HIPAA、EAL3)。

在Sentry诞生之前，对于授权，有两种备选解决方案：粗粒度级的HDFS授权和咨询授权，但它们并不符合典型的规范和数据安全需求，原因如下。

(1) 粗粒度级的 HDFS 授权：安全访问和授权的基本机制被 HDFS 文件模型的粒度所限制。五级授权是粗粒度的，因为没有对文件内数据的访问控制：用户要么可以访问整个文件，要么什么都看不到。另外，HDFS 权限模式不允许多个组对同一数据集有不同级别的访问权限。

(2) 咨询授权：咨询授权在 Hive 中是一个很少使用的机制，旨在使用户能够自我监管，以防止意外删除或重写数据。这是一种“自服务”模式，因为用户可以为自己授予任何权限。因此，一旦恶意用户通过认证，它不能阻止其对敏感数据的访问。

通过引进 Sentry，Hadoop 目前可在以下方面满足企业和政府用户的 RBAC(Role-Based Access Control，基于角色的访问控制，将在 5.5 节具体介绍)需求。

(1) 在安全授权方面，Sentry 可以控制数据访问，并对已通过认证的用户提供数据访问特权。

(2) 在访问控制的粒度方面，Sentry 支持细粒度的 Hadoop 数据和元数据访问控制。Sentry 在服务器、数据库、表和视图范围内提供了不同特权级别的访问控制，包括查找、插入等，允许管理员使用视图限制对行或列的访问。管理员也可以通过 Sentry 和带选择语句的视图，根据需要在文件内屏蔽数据。

(3) Sentry 通过基于角色的授权简化了管理，能够方便地将访问同一数据集的不同特权级别授予多个组。

(4) 多租户管理：Sentry 允许为委派给不同管理员的不同数据集设置权限。

(5) 统一平台：Sentry 为确保数据安全，提供了一个统一平台，使用现有的 Hadoop Kerberos 实现安全认证。

3. Sentry 体系结构的组件及工作流程

1) 融合层(Binding)

Binding 实现了对不同的查询引擎授权，Sentry 将自己的 hook()函数插入各 SQL 引擎的编译、执行的不同阶段。这些 Hook()函数起两大作用：一是起过滤器的作用，只放行具有相应数据对象访问权限的 SQL 查询；二是起授权接管的作用，使用 Sentry 之后，Grant/Revoke 管理的权限完全被 Sentry 接管，Grant/Revoke 的执行也完全在 Sentry 中实现；所有引擎的授权信息也存储在由 Sentry 设定的统一的数据库中，这样就实现了对引擎的授权的集中管理。

2) 策略引擎(Policy Engine)

这是 Sentry 授权的核心组件。Policy Engine 判定从 Binding 层获取的输入的权限要求与服务提供层已保存的权限描述是否匹配。

3) 策略读取(Policy Provider)

Policy Provider 负责从文件或数据库中读取原先设定的访问权限。Policy Engine 以及 Policy Provider 其实对于任何授权体系来说都是必需的，因此是公共模块，后续还可服务于别的查询引擎。

Sentry 权限管理流程如图 3.2 所示。

Apache Sentry 的目标是实现授权管理，它是一个策略引擎，被数据处理工具用来认

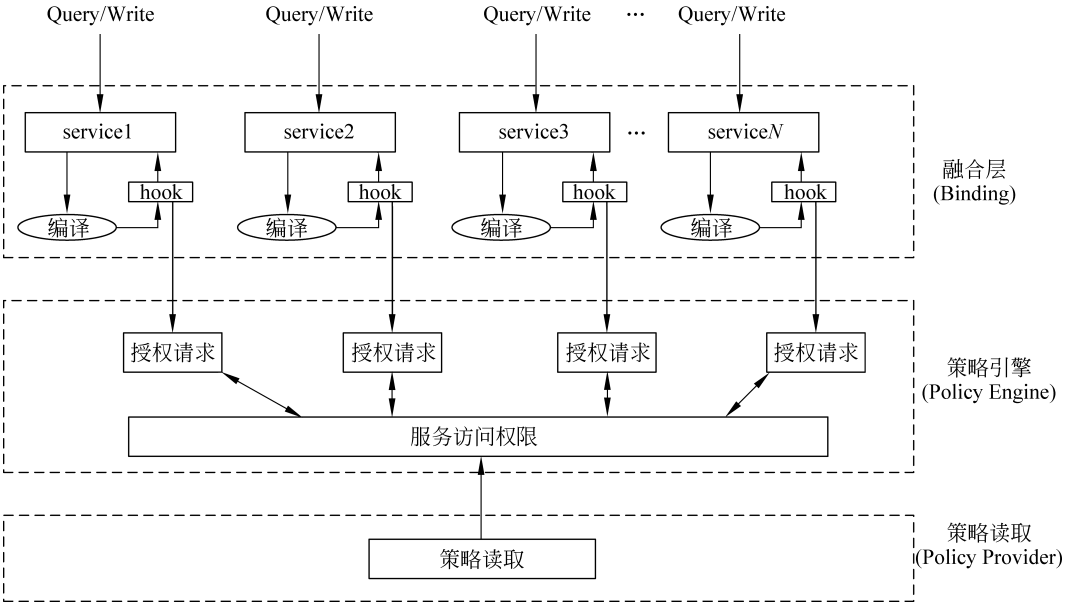

图 3.2 Sentry 权限管理流程图

证访问权限。它具有高度扩展性,可以支持任意的数据模型。例如,它支持 Apache Hive 和 Cloudera Impala 的关系数据模型,以及 Apache 中有继承关系的数据模型。

Sentry 提供定义和持久化访问资源策略的方法。目前,这些策略可以存储在文件里,也可以存储在能使用远程过程调用(Remote Procedure Call,RPC)服务访问的数据库的后端。数据访问工具,如 Hive,以一定的模式辨认用户访问数据的请求,例如从一个表读一行数据或者删除一个表。这个工具请求 Sentry 认证此次访问是否合理。Sentry 构建请求用户被允许的权限映射并判断给定的请求是否合理。请求工具根据 Sentry 的判断结果来允许或禁止用户的访问请求,这就是 Sentry 的工作流程。

3.4.3 Apache Ranger

1. Ranger 概念

Apache Ranger 提供一个集中式安全管理框架,提供统一授权和统一审计的能力。它可以对整个 Hadoop 生态中的组件(如 HDFS、YARN、Hive、HBase、Kafka、Storm 等)进行细粒度的数据访问控制。通过操作 Ranger 控制台,管理员可以轻松地通过配置策略来控制用户访问 HDFS 文件夹、HDFS 文件、数据库、表、字段权限。这些策略可以为不同的用户和组来设置,同时权限可与 Hadoop 无缝对接。

Ranger 由 Ranger Admin、Ranger UserSync、Ranger TagSync 与 Ranger Plugin 组成,架构如图 3.3 所示。

1) Ranger Admin

用户管理策略包含 3 部分:Web 页面、Rest 消息处理服务以及数据库。可以把它看

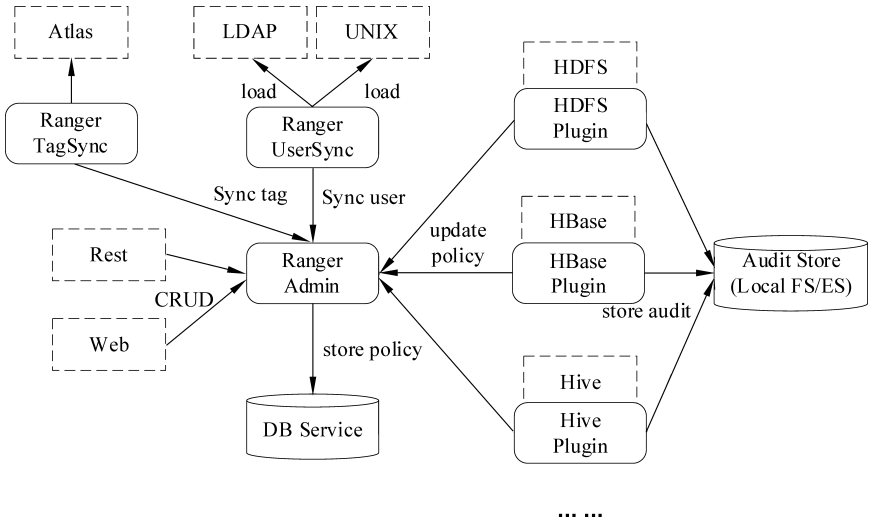

图 3.3　Ranger 的架构

成数据的集中存储中心(所有的数据都集中存在这里,但是其他组件也可单独运行存在),具体有以下作用。

(1) 接收 UserSync 进程传过来的用户或用户组信息,并将它们保存到 MySQL 数据库中,在配置权限策略时需要使用这些用户信息。

(2) 提供创建 policy 策略的接口。

(3) 提供外部 Rest 消息的处理接口。

2) Ranger UserSync

Ranger UserSync 用于将 UNIX 系统或 LDAP 用户或用户组同步到 Ranger Admin。

UserSync 是 Ranger 提供的一个用户信息同步接口,可以用来同步 Linux 用户信息与 LDAP 用户信息。通过配置项 SYNC_SOURCE=LDAP/UNIX 来确认是 LDAP 还是 UNIX,默认情况是同步 UNIX 信息。

3) Ranger TagSync

同步 Atlas 中的 Tag 信息,基于标签的权限管理,当一个用户的请求涉及多个应用系统中的多个资源的权限时,可以通过只配置这些资源的 Tag 方便快速地授权。

4) Ranger Plugin

Ranger Plugin 是如 HDFS、Hive、HBase 等一个个大数据组件所提供的插件接口,其工作主要是从 Ranger Admin 处拉取该组件配置的所有策略,然后缓存到本地,当有用户请求时提供鉴权服务。

2. Ranger 权限模型

Ranger 权限模型由一条条权限策略组成。Ranger 权限策略主要由 3 方面组成,即用户、资源、权限,如图 3.4 所示。

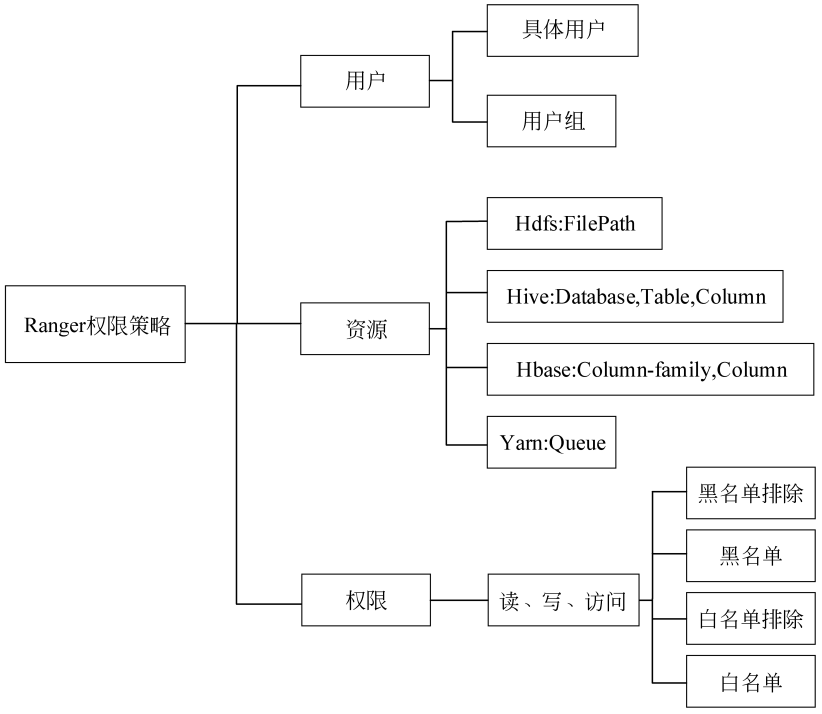

图 3.4 Ranger 权限策略的组成

用户：Ranger 可以对用户进行分组，一个用户可以属于多个分组。Ranger 支持对用户或者用户组配置某资源的相关权限。

资源：对于各个不同的组件，资源的表述都不相同。比如，在 HDFS 中，文件路径就是资源，而在 Hive 中，资源可以指 Database、Table 甚至 Column。

权限：Ranger 可以对各个资源的读、写、访问进行限制，具体可以配置白名单、白名单排除、黑名单、黑名单排除等条目。

Ranger 管理员配置好组件的相关策略后，并且将 Ranger 插件安装到具体的大数据组件中，Ranger 就开始生效了。这时用户访问 Ranger，就会进行权限的校验过程。用户访问资源权限的校验流程图如图 3.5 所示。

当用户要请求某个资源时，会先获取和这个资源有关联的所有配置的策略，之后遍历这些策略，然后根据黑、白名单判断该用户是否有权限访问该资源。从图 3.5 中可以看出，黑名单、黑名单排除、白名单、白名单排除匹配的优先级如下。

(1) 黑名单的优先级高于白名单。

(2) 黑名单排除的优先级高于黑名单。

(3) 白名单排除的优先级高于白名单。

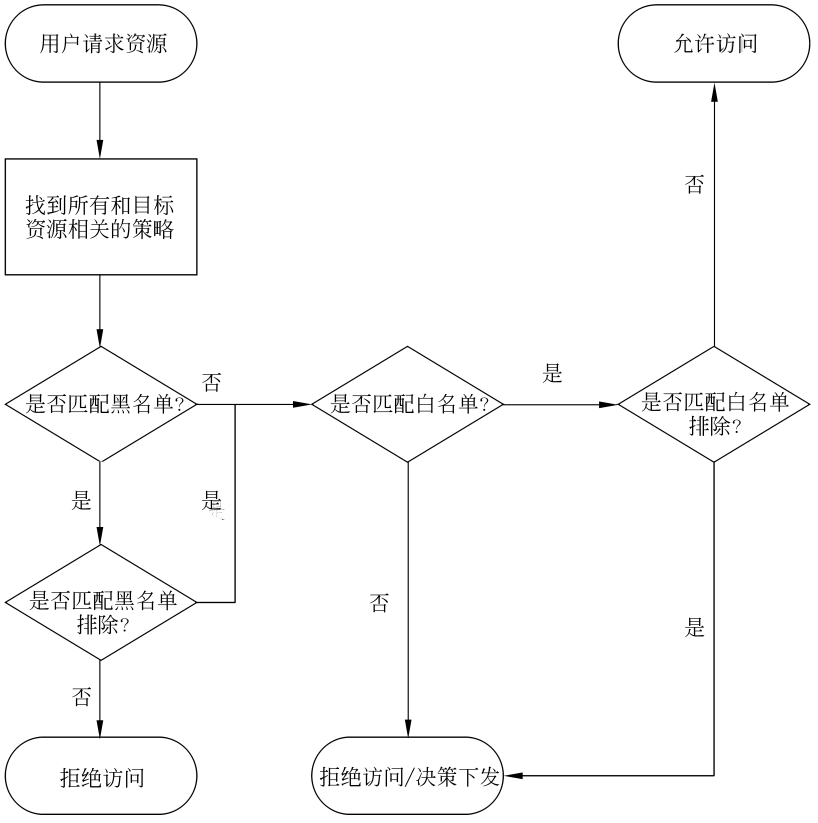

图 3.5　用户访问资源权限的校验流程图

3.5　Hadoop 的安全性分析

Hadoop 的安全问题，其中一方面是 Hadoop 本身的安全能力，另一方面是对 Hadoop 的安全性进行补充的策略。

3.5.1　Hadoop 面临的安全问题

在 1.0 版本之前，Hadoop 基本没有任何安全机制，因此面临着各方面的安全威胁，主要包括以下几方面。

(1) 如何强制所有类型的客户端(比如 Web 控制台和进程)上的用户及应用进行认证?

(2) 如何确保服务不是服务冒充的(比如 TaskTracker 和 Task，未经授权的进程向 DataNode 出示 ID 以访问数据块等)?

(3) 如何根据已有的访问控制策略和用户凭据强制数据的访问控制?

(4) 如何实现基于属性的访问控制(ABAC)或基于角色的访问控制(RBAC)?

(5) 怎么才能将 Hadoop 跟已有的企业安全服务集成到一起?

(6) 如何控制谁被授权可以访问、修改和停止 MapReduce 作业?

(7) 怎么才能加密传输中的数据?

(8) 如何加密静态数据?

(9) 如何对事件进行跟踪和审计,如何跟踪数据的出处?

(10) 对于架设在网络上的 Hadoop 集群,通过网络途径保护它的最好办法是什么?

上述部分问题可通过 Hadoop 自身的能力解决,但也有很多是 Hadoop 所无能为力的。究其原因,主要是 Hadoop 最初开发时并没有考虑安全因素。

3.5.2 Hadoop 生态圈安全风险

按照 Hadoop 最初的设想,它假定集群总是处于可信的环境中,由可信用户使用的相互协作的可信计算机组成。这就导致整个 Hadoop 生态圈一度被暴露在安全的风险之下。Hadoop 生态圈的安全风险主要有 5 类。

1. 安全认证

任何用户都可以伪装成为其他合法用户,访问其在 HDFS 上的数据,获取 MapReduce 产生的结果,从而存在恶意攻击者假冒身份,篡改 HDFS 上他人的数据,提交恶意作业破坏系统、修改节点服务器的状态等隐患。由于集群缺乏对 Hadoop 服务器的认证,攻击者假冒成为 DataNode 或 TaskTracker 节点,加入集群,接受 NameNode 和 JobTracker。一旦借助代码,任何用户都可以获取 root 权限,并非法访问 HDFS 或者 MapReduce 集群,恶意提交作业、修改 JobTracker 状态、篡改 HDFS 上的数据等。

2. 权限控制

用户得知数据块的 Block ID 后,可以不经过 NameNode 的身份认证和服务授权,直接访问相应的 DataNode,读取 DataNode 节点上的数据或者将文件写入 DataNode 节点,并可以随意启动假的 DataNode 和 TaskTracker。对于 JobTracker,用户可以任意修改或者关闭其他用户的作业,提高自身作业的优先级,JabTracker 对此不作任何控制。其中,无论是粗粒度的文件访问控制,还是细粒度地使用 ACL 进行访问控制,都会或多或少抢占 Hadoop 集群内部的资源。可从外部在行为上进行额外的权限控制,尤其支持有 Hive 的 Hadoop 环境。只需要判别 Hive SQL 语句的对象和当前用户的 Key 是否匹配,就可以通过通信阻断等方式达到权限控制的目的。

3. 关键行为审计

默认情况下,Hadoop 缺乏审计机制,但可以通过 Hadoop 系产品添加日志监控来完成一部分审计功能。通常通过日志的记录来判断整个流程中是否存在问题。这种日志的记录缺乏特征的判断和自动提示功能。完全可以利用进行改进后的审计产品来进行审计,只审计客户端的行为即可追查到恶意操作或误操作行为。

4. 静态加密

默认情况下，Hadoop 对集群 HDFS 系统上的文件没有存储保护，所有数据均是明文存储在 HDFS 中，超级管理员可以不经过用户允许直接查看和修改用户在云端保存的文件，这就很容易造成数据泄露。采用静态加密的方式，对核心敏感数据进行加密处理，使得数据密文存储，可防止泄露风险。具体细节可见本书 6.1 节及 6.3 节相关内容。

5. 动态加密

默认情况下，Hadoop 集群各节点之间，客户端与服务器之间数据明文传输，使得用户隐私数据、系统敏感信息极易在传输的过程被窃取。解决动态加密一般会提供一个附加的安全层。对于动态数据而言，即传输到或从 Hadoop 生态系统传送出来的数据，利用简单认证与安全层(SASL)认证框架进行加密，通过添加一个安全层的方式，保证客户端和服务器传输数据的安全性，确保在中途不会被读。具体细节可见本书 6.2 节及 6.4 节相关内容。

3.5.3 Hadoop 安全应对

当 Hadoop 不具备用户所要求的安全能力，那么它只能转而集成第三方工具，或使用某个厂商提供的安全加强版 Hadoop，或采用其他有创造性的办法。随着 Hadoop 在云上的广泛运用，很多公司都对 Hadoop 提出了安全应对方案，行业内也涌现出很多 Hadoop 安全补充工具，如 Yahoo 提出的 Kerberos 体系解决安全认证问题、ACL 解决访问控制问题。厂商发布安全产品来弥补 Hadoop 的不足，主要基于以下考虑。

(1) 没有“静态数据”加密。目前 HDFS 上的静态数据没有加密。那些对 Hadoop 集群中的数据加密有严格安全要求的组织，被迫使用第三方工具实现 HDFS 硬盘层面的加密，或安全性经过加强的 Hadoop 版本。

(2) 以 Kerberos 为中心的方式——Hadoop 依靠 Kerberos 做认证。对于采用了其他方式的组织而言，这意味着要单独搭建一套认证系统。

(3) 有限的授权能力——尽管 Hadoop 能基于用户及群组许可和访问控制列表(ACL)进行授权，但对于有些组织来说，这样是不够的。很多组织基于 XACML 和基于属性的访问控制使用灵活动态的访问控制策略。尽管肯定可以用 Accumulo 执行这些层面的授权过滤器，但 Hadoop 的授权凭证作用是有限的。

(4) 安全模型和配置的复杂性。Hadoop 的认证有几个相关的数据流，用于应用程序和 Hadoop 服务的 Kerberos RPC 认证，用于 Web 控制台的 Http Spnego 认证，以及使用代理令牌、块令牌、作业令牌。对于网络加密，也必须配置 3 种加密机制，用于 SASL 机制的保护质量，用于 Web 控制台的 SSL，HDFS 数据传输加密。所有这些设置都要分别进行配置，并且很容易出错。

总的来说，针对 Hadoop 存在的潜在风险，其自身也在逐步完善中，主要表现在：

(1) HDFS 的命令行不变，但在 Web UI 中添加权限管理。

(2) MapReduce 添加 ACL,包括:管理员可在配置文件中配置允许访问的 user 和 group 列表;用户提交作业时,可知道哪些用户或者用户组可以查看作业状态,使用参数 mapreduce.job.acl-view-job;用户提交作业时,可知道哪些用户或者用户组可以修改或者关闭 Job,使用参数 mapreduce.job.acl-modify-job。

(3) MapReduce 系统目录(即 mapred.system.dir,用户在客户端提交作业时,JobClient 会将作业的 job.jar、job.xml 和 job.split 等信息复制到该目录下)访问权限改为 700。

(4) 所有 Task 以作业拥有者身份运行,而不是启动 TaskTracker 的那个角色。

(5) Task 对应的临时目录访问权限改为 700。

(6) DistributedCache 是安全的。DistributedCache 分为两种:shared 可以被所有作业共享;private 只能被该用户的作业共享。

3.6 Hadoop 安全技术架构

基于前面对 Hadoop 安全性的分析,要想实现安全的 Hadoop,可以从以下 8 方面予以考虑。

(1) 认证。提供单点的登录,通过这种方式实现用户身份的认证。

(2) 授权。要能够确定用户访问权限,比如对数据集的访问、服务请求等。

(3) 访问控制。要有一个基于细粒度和角色的访问控制机制。

(4) 数据加密。包括数据处理过程中对中间数据的处理,以及在数据传输和数据存储过程中的处理,都需要引入数据加密的技术。

(5) 网络安全。是否有一个有效的隔离措施,以及对业务行为的安全控制是否有保障。

(6) 系统安全。Hadoop 大部分基于 Linux 操作系统,要有对底层操作系统漏洞的防护。

(7) 基础架构安全。物理层面的安全以及基于安全架构逐步完善的安全措施。

(8) 审计监控。是否能够对所有的用户行为和授权行为等进行有效监控,并基于这些来识别潜在的危险行为。

3.6.1 Hadoop 认证授权

Hadoop 使用的是 HDFS 分布式存储文件系统。当用户登录到 DataNode 访问数据时,用户的权限可以访问到 DataNode 目录下的所有数据块。这实际上是一个安全风险,因为 Hadoop 没有对用户、用户组和服务进行有效的认证,当执行 Hadoop 命令时只是单单通过 whoami 确定用户身份,这个过程中黑客可以编写 whoami 脚本模拟超级用户。

提高 Hadoop 的安全要从两个层面着手:一是用户层次访问控制;二是服务层次访问控制。其中,用户层次访问控制要有对用户和用户组的认证机制,具体包括:

(1) Hadoop 用户只能访问授权的数据。

(2) 只有认证的用户可以向 Hadoop 集群提交作业。

(3) 用户可以查看、修改和终止他们的作业。

(4) 只有认证的服务可以注册为 DataNode 或 TaskTracker。

(5) DataNode 中数据块的访问需要保证安全,只有认证用户才能访问 Hadoop 集群中存储的数据。

服务层次访问控制,即服务之间的互相认证,具体包括:

(1) 可扩展的认证,Hadoop 集群包括大量的节点,认证模型需要能够支持大规模的网络认证。

(2) 伪装,Hadoop 可以识别伪装用户,保证正确的用户作业隔离。

(3) 自我服务,Hadoop 作业可能执行很长时间,要确保这些作业可以自我进行委托用户认证,保证作业完整执行。

(4) 安全的 IPC,Hadoop 服务要可以相互认证,保证它们之间安全通信。

Kerberos 是 Hadoop 的安全基础,未来 Hadoop 的所有安全措施都要基于 Kerberos 认证原理才能实现。Kerberos 是网络的认证协议,可以实现在网络中不传输密码就能完成身份认证,其原理在于通过对称加密算法生成时间敏感的授权票据。Kerberos 具体的认证过程见本书 4.3.1 节相关内容。

3.6.2 Hadoop 网络访问安全

关注安全的企业往往都有一套统一身份认证管理机制,用来管理企业内部的终端、网络设备、上网行为等。Kerberos 则实现了 Hadoop 内部的身份认证管理。Kerberos 如何和企业统一身份认证管理进行有效结合是实现企业网络生态安全的第一步。

向 Hadoop 请求的客户端用户需要在 EIM(统一身份认证管理)系统中注册身份,再由 EIM 向终端用户发放 Kerberos 授权票据,这时客户端会使用该票据向 Hadoop 请求。整个过程中 EIM 系统需要和 Hadoop 的本地 KDC 进行数据同步,建立跨域信任。

如图 3.6 所示,目前主流的 Hadoop 网络安全措施是通过防火墙将客户端和 Hadoop 集群进行逻辑隔离的。防火墙作为网络层面的控制,对恶意端口、无用协议进行有效过滤,之后部署有很多 Hadoop 生态组件工具的网关服务器,客户端用户通过统一登录网关服务器对 Hadoop 集群进行维护以及提交作业。这样就初步实现了网络层的访问控制、用户的认证授权,以及行为的访问控制过滤。

随着 Hadoop 的发展,如图 3.7 所示,它又在网关层上引入了开源项目 HUE,基本的架构和前面类似,只不过多了 HUE 的相关功能。HUE 可以与 EIM 系统的身份认证结合,支持 LDAP 同步用户和用户组信息,采用 HttpFS 代理,通过 SPANWFO-Base 认证协议访问 SSL 加密,实现了细粒度的用户访问控制。

随后,Hadoop 又有了 Knox 网关集群,如图 3.8 所示,它结合了 HUE 和传统的优势,内部还是以网关的形式做代理,实现了防火墙的功能对端口和协议进行过滤,同时对用户进行细粒度的访问控制,不仅如此,它还实现了单点登录。

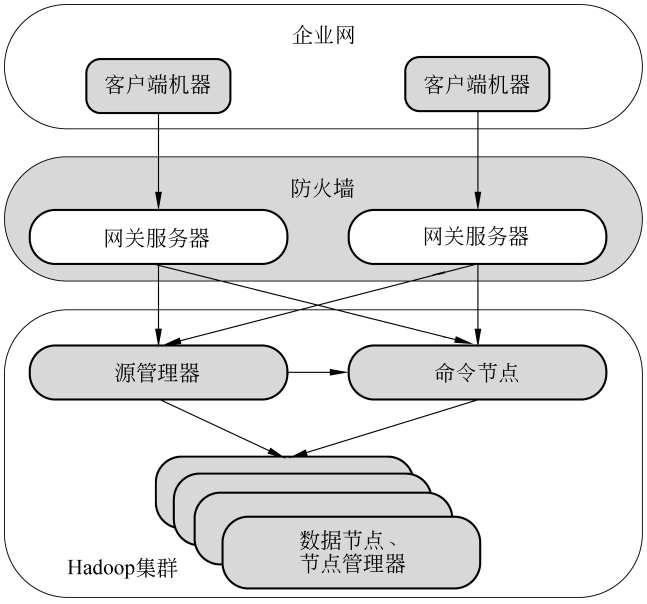

图 3.6 Hadoop 网络访问安全措施

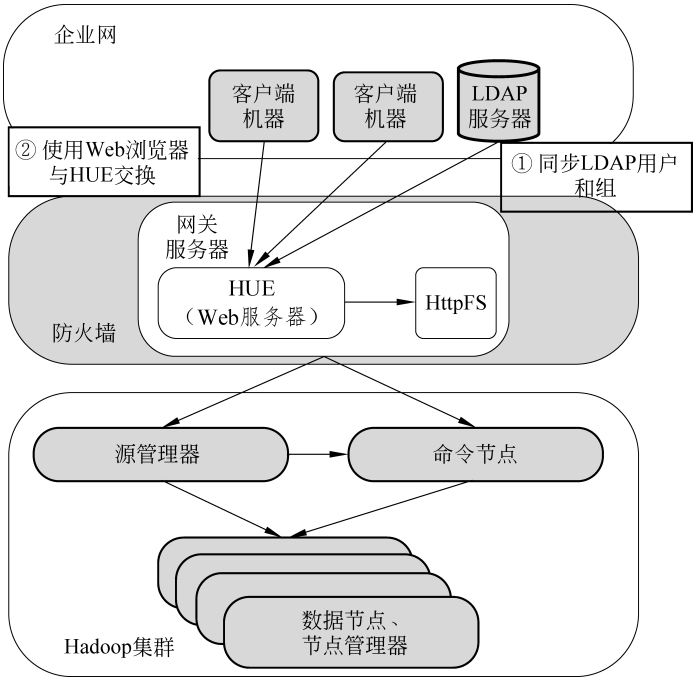

图 3.7 结合 HUE 的 Hadoop 网络访问