第33卷 第17期 计 算 机 工 程 2007年9月
Vol.33 No.17 Computer Engineering September 2007
·安全技术· 文章编号：1000—3428(2007)17—0178—03 文献标识码：A 中图分类号：TP393

面向服务的分布式数据库系统的容忍入侵方法

廖 凯[1]，张来顺[1]，郭渊博[1,2]
（1. 解放军信息工程大学电子技术学院，郑州 450004；2. 武汉大学计算机学院，武汉 430000）

摘 要：以分布式数据库系统提供的服务作为入侵检测的对象，采用了基于对可观测现象检测的原理，跟踪和定位被入侵的服务，借鉴了基于日志的破坏隔离围堵策略，准确定位了被破坏的数据，并对这些数据进行处理和恢复，不需要将整个系统进行“回退”，最大限度地保证数据库中其他服务的正常运行，保证了系统的可生存性。

关键词：分布式数据库系统；容忍入侵；破坏隔离

Intrusion-tolerant Method Based on Service-oriented Distributed Database System

LIAO Kai[1], ZHANG Lai-shun[1], GUO Yuan-bo[1,2]
(1. Institute of Electronic Technology, PLA Information Engineering University, Zhengzhou 450004;
2. School of Computer, Wuhan University, Wuhan 430000)

【Abstract】 This paper presents the design of service-oriented intrusion-tolerant in distributed database systems, which detects intruded services by watching observable effects. After that it carries on damage containment policy, locating and cleaning corrupted data, which ensures that other services can work normally in spite of the systems being intruded, guaranteeing their survivability.

【Key words】 distributed database system; intrusion-tolerant; damage containment

1 概述

为了保证分布式数据库的安全，不仅需要保证授权的合法用户对数据的有效存储和操作，同时还要严格拒绝非法用户的入侵攻击。入侵检测方法的关键是怎样去检测、定位“入侵”，并给出响应和恢复。然而现有的入侵检测技术普遍存在误报率和漏报率高、效率低的问题，无法完全满足那些提供关键服务的数据库的要求。因此，怎样在系统被入侵、数据被破坏的情况下，保证系统能够控制破坏性的传播，继续向合法用户提供不间断的服务，这是分布式数据库安全防护的一个难点。

现有的提供关键服务的分布式数据库系统在检测到系统被入侵、数据被破坏之后，往往通过把数据库系统回退到上一个安全校验点，来消除数据破坏和污染。虽然这样做能把数据破坏和污染彻底清除，但所有合法用户在这段时间的合法操作也会丢失，造成了极大的资源浪费[1]。可以采取面向服务的入侵检测机制，以系统中的服务作为检测对象，通过检测服务被攻陷后表现出的可观测的现象来检测入侵的发生。从入侵者的角度看，当某个服务被攻陷后，攻击者很容易继续入侵与该服务有关联、结构类型相似的其他服务，针对这个特点，可以建立体现这些服务之间联系的关系图，在图上跟踪每一个入侵攻击实例的攻击路径。这样可以简化检测过程，提高并发程度，有利于系统同时处理多个并发的入侵攻击行为。在处理检测到的入侵攻击时，可以对传统基于校验点的方法进行了改进，提出一种基于破坏隔离围堵的安全恢复策略，在破坏隔离阶段迅速实施隔离，防止破坏的传播；在释放阶段，以日志中记录的更新为依据，在关系图上跟踪和定位破坏的传播，准确定位被破坏的数据，及时释放被错误隔离的对象，保证系统向用户提供连续的服务，从而达到容忍入侵的目的。

2 面向服务的入侵检测

破坏检测阶段采取面向服务的入侵检测机制，以分布式数据库系统中提供的服务作为检测对象，通过检测服务被攻陷后表现出来的可观测现象，检测入侵的发生，并完成对恶意事务的检测和定位。主要包括以下几个步骤：

(1)建立能够体现这些服务之间联系的关系图，简称为系统服务导图。根据实际应用中的系统特征而建立，针对不同的系统，会建立不同的导图。

以图1为例，结点 *A,B,C,…*代表该系统可以提供哪些服务，有向箭头 $A \rightarrow B$ 表示当服务 *A* 被特定类型的攻击攻陷以后，该入侵将会传到另一个结构类型相似的服务 *B* 中，称这种联系为入侵渠道。笔者主要考虑以下5类入侵渠道：

1)DoS攻击渠道：如果 *A* 被DoS攻击攻破，那么 *B* 也会被攻破；
2)网络攻击渠道：通过网络进行攻击；
3)共享文件攻击渠道：通过共享文件进行攻击；
4)共享内在区攻击渠道：通过共享内存区进行攻击；
5)超级攻击渠道：综合利用上述方法进行攻击，传播破坏。

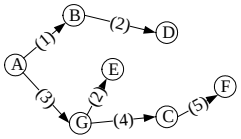

图1 系统服务导图实例

基金项目：国家自然科学基金资助项目(60503012)；国家部委预研基金资助项目(9140A16040206JB5204)
作者简介：廖 凯(1982－)，男，硕士研究生，主研方向：数据安全与恢复；张来顺，教授；郭渊博，博士后
收稿日期：2006-10-22 **E-mail：**liaokai_1982@sina.com

(2)根据系统服务导图和脆弱点描述得出攻击发生时所有可能的传播路径，由它们所组成的图称为入侵攻击图。得到脆弱点信息之后，在导图中找到相应的结点，然后根据入侵渠道，由所有能够通过入侵渠道连接起来的结点和这些入侵渠道组成的图称为入侵攻击图。步骤(1)~步骤(2)与发生的攻击实例没有关系，这两步都是在入侵发生之前就可以完成，为后期对入侵行为的检测节约时间。

(3)当入侵发生时，把每一个具体攻击实例的攻击路径在图上反映出来，为每个攻击实例建立相应的攻击子图，如图 2 所示。当检测到系统中某个服务受到攻击时，在入侵攻击图中找到与该服务对应的结点，把入侵攻击图中所有能够通过入侵渠道连接起来的结点和这些入侵渠道组成与该攻击实例相应的攻击子图。

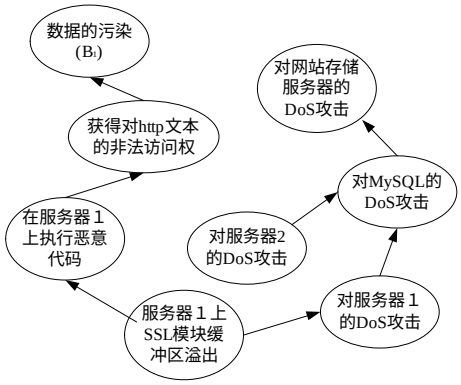

图 2　攻击子图实例

建立攻击子图后，就可以从所有结点中找出与数据操作和数据破坏相关的恶意事务来，这些恶意事务组成最初的恶意事务集合并以其为源头，在破坏隔离与恢复阶段跟踪破坏的传播中，找出被它们传染的其他数据对象。图 2 中，与事务操作相关的结点为数据污染，把得到的恶意事务记为 B_1。在数据库系统中，事务是对数据进行操作的最小单位。所有已经提交的事务都会被记录在数据库系统的日志中。为了保证系统具有较高的安全性与可信性，可以使用基于非易失性存储器进行日志的存放，日志被篡改时，可以检测出来并加以恢复。

3　基于日志的分阶段破坏隔离与释放

在检测到恶意事务之后，对破坏进行隔离围堵包括破坏隔离阶段和释放阶段 2 个阶段。在初始隔离阶段，可疑受损对象(例如被图 2 中恶意事务B_1破坏的数据对象)被迅速隔离，不允许和其他事务、对象进行通信，防止破坏传播。在释放阶段，以破坏检测阶段检测到的恶意事务集合为源头，对被破坏的数据对象进行准确定位，把初始阶段错误隔离的数据对象及时释放出来，提高数据的利用率，保障系统向用户提供连续的服务。同时，这个阶段的执行与其他没有关联的正常事务的运行是并行的，互不干扰，可以提高执行的效率。

3.1　处理单个恶意事务方案的提出及改进

为了清晰、简练地描述整个过程，可以先阐述对单个恶意事务的处理过程，然后再给出对多个恶意事务的处理。

(1)初始方案(方案 1)

初始方案主要是利用时间戳设计的。每个数据对象都与一个时间戳t_a相关联，代表在何时被更新。假定任何非法修改时间戳的行为都会被记录，并且可以恢复正确的时间戳。称日志中对一段事务的记录为一个记录段，记为H。事务B_1读取数据x，记为 $B_1R[x]$；事务B_2对数据y进行写操作，记为 $B_2W[y]$。

隔离步骤为：

1)暂停所有正在运行的事务，等隔离阶段完成后继续。这样可以防止破坏泄漏的发生：在进行破坏隔离与恢复时，如果不停止正在运行的事务，可能导致本来没有被破坏的正常事务读取被破坏的数据，使正常事务被破坏而不被处理[1]。

2)设恶意事务为M_i，如果 $t_a \in [t_L, t_H]$，把数据对象a隔离。其中，t_L是事务M_i执行的开始时间；t_H是第一阶段隔离的结束时间。只要数据对象a在隔离区间$[t_L, t_H]$上被更新过，都要被隔离起来。这里为了不造成破坏泄漏，可以采用过量评估。过量评估的定义为：假设实施的隔离集设为S_c，而实际上真正被破坏的数据构成的集合为S_d，当 $S_c > S_d$ 时，评估为过量评估。

在描述释放阶段之前，可以先把在隔离区间$[t_L, t_H]$内所有事务更新的信任关系在图上反映出来，称该图为信任图，它是对数据破坏进行精确定位的关键部分。

信任图中以横轴为访问服务器的客户端(client)，以纵轴为时间轴，在图上绘出在区间$[t_L, t_H]$内各个时刻发生在所有客户端(client)的更新(update)情况。更新是指数据对象a被事务T_i执行写操作，或者事务T_i读取数据对象a。update的结构包括：更新者的身份，更新时间，被更新数据的位置(位于分布式数据库系统中的哪台服务器上)，被更新的数据组成的集合。构造信任图时，首先在图中找到恶意事务M_i提交中涉及到的$update_1$结点，接着判断破坏的传播。如果在恶意事务M_i的$update_1$更新过数据对象a以后，$update_2$再去读a，很显然$update_2$会被破坏；$update_2$在被破坏以后再去更新别的数据对象b则b也可能被破坏。依据该原理把图上对应的各个结点连接起来，则可保证所有可能被破坏的结点都在其中，其他未被连接的结点没有被破坏。把在区间$[t_L, t_H]$内的所有更新(update)分为 2 类：

①没有被连接的 *update* 组成正确更新集合 *Rup*；

②被恶意 *update* 连接在一起的更新组成错误更新集合 *Wup*。

3)释放阶段，检查$update_i$是否属于Wup，如果$update_i$不属于Wup，说明$update_i$没有被污染，$update_i$的写集就没有被破坏，把$update_i$的写集放入S_U中，并释放。如果$update_i$属于Wup，则由专用清理破坏、恢复数据的事务来处理$update_i$的写集，等恢复完成后，$update_i$的写集也被放入S_U中，并释放。S_U是存放被释放数据对象的集合，当某个数据对象a被确认没被破坏或已经恢复被放入S_U后，如果后来又检测到a，则不需要再进行重复的工作，只需要对不在S_U中的数据进行恢复。

过去的数据破坏隔离技术在释放阶段，通常都是笼统地用清理事务把所有被隔离的事务扫描一遍，既造成工作量增加，又导致那些并没有被破坏的数据对象得不到及时释放，增加了释放延迟。

但是，方案 1 中可能会发生破坏泄漏。例如，数据对象a被$update_1$,$update_2$更新过，$t_{U_1} < t_{U_2}$，$update_1$没有被破坏，$update_2$被破坏。根据方案 1，在扫描检测$update_1$得知，它没被破坏后a被释放，并放入S_U中。检测到$update_2$被破坏时，由于a已经在S_U中，因此不会再次修复a。但事实上，a被$update_2$破坏了，它被释放后可能会被其他正确的$update$读取，造成破坏泄漏。

(2)改进方案(方案 2)

改进方案解决了初始方案中的存在破坏泄漏的问题。为了避免重复，只对区别于方案 1 的部分进行描述。在发现$update_1$没有被破坏后，不再像方案 1，直接把$update_1$的写集全部释放，而是进一步检查，如果$t_a < t_{u_1}$，证明数据对象a在$update_1$以后没有再被更新过，因此，不可能存在破坏泄漏的可能，可以把数据对象a放入S_U中并释放，称这个过程为“2 次确认”。

实例 1 如下：

H_1: $B_2W[w], B_1W[x], B_2W[u], B_3R[u], B_1W[y] commit_{B_1}, B_2R[x]$
$B_3W[l], B_4W[z], commit_{B_4}, B_2W[u], commit_{B_2}, B_3R[u], B_4W[v], B_3W[v], commit_{B_3}, B_5R[y]$
$B_6W[m], commit_{B_6}, B_5R[u], B_5W[u], B_5W[z], commit_{B_5}$

其中，在入侵检测发现阶段中，B_1为恶意事务。

$Update_1: t_1 \cdots\cdots B_2W[w],\ Client_1,\ Server_6,\ w$
$Update_2: t_2 \cdots\cdots B_1W[x],\ Client_2,\ Server_1,\ x$
$Update_3: t_3 \cdots\cdots B_2W[u],\ Client_5,\ Server_3,\ u$
$Update_4: t_4 \cdots\cdots B_3R[u],\ Client_4,\ Server_3,\ u$
$Update_5: t_5 \cdots\cdots B_1W[y],\ Client_2,\ Server_2,\ y$
$Update_5: t_5 \cdots\cdots B_1W[y],\ Client_2,\ Server_2,\ y$
$Update_6: t_6 \cdots\cdots B_2R[x],\ Client_3,\ Server_1,\ x$
$Update_7: t_7 \cdots\cdots B_3W[l],\ Client_1,\ Server_3,\ l$
$Update_8: t_8 \cdots\cdots B_4W[z],\ Client_1,\ Server_5,\ z$
$Update_9: t_9 \cdots\cdots B_2W[u],\ Client_4,\ Server_3,\ u$
$Update_{10}: t_{10} \cdots\cdots B_3R[u],\ Client_2, Server_1,\ x$
$Update_{11}: t_{11} \cdots\cdots B_4W[v],\ Client_1,\ Server_4,\ v$
$Update_{12}: t_{12} \cdots\cdots B_3W[v],\ Client_4,\ Server_4,\ v$
$Update_{13}: t_{13} \cdots\cdots B_5R[y],\ Client_5,\ Server_2,\ y$
$Update_{14}: t_{14} \cdots\cdots B_6W[m],\ Client_3, Server_4,\ m$
$Update_{15}: t_{15} \cdots\cdots B_5R[u],\ Client_5,\ Server_3,\ u$
$Update_{16}: t_{16} \cdots\cdots B_5W[u],\ Client_5,\ Server_3,\ u$
$Update_{17}: t_{17} \cdots\cdots B_5W[z],\ Client_5,\ Server_5,\ z$

图 3 为实例 1 的信任图。

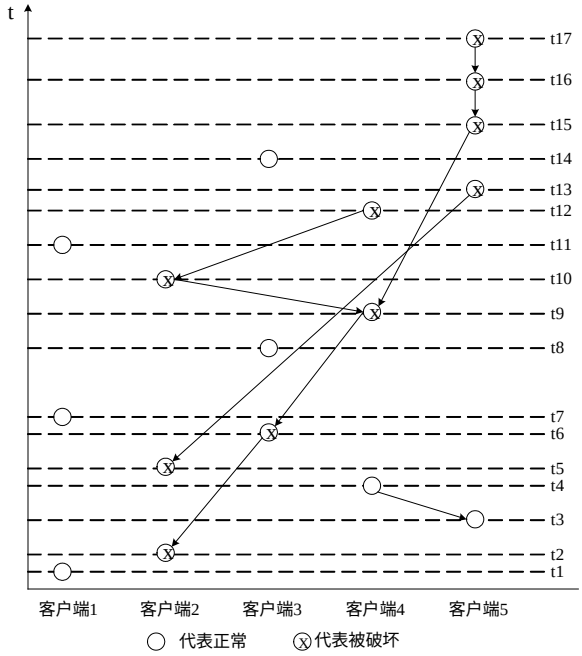

图 3　实例 1 的信任图

得到信任图后，可以迅速得到结论：$update_1$, $update_3$, $update_4$, $update_7$, $update_8$, $update_{11}$, $update_{14}$属于Rup，没有被破坏，在经过“2 次确认”以后，对应的写集中相应的数据对象放入S_U，得到释放。

3.2　同时对多个恶意事务处理方案

由于分布式系统存在的脆弱点较多、攻击手段的日趋隐蔽以及入侵检测的滞后，因此在实际应用中，往往会同时检测到多个入侵行为；另外，在处理某个恶意入侵时，也可能会有其他入侵行为发生，目前大多数解决方案在面对这 2 种情况时，通常会把这几个恶意事务合并处理，而之前的计算全部作废，导致了资源的浪费，严重影响了对恶意事务处理的效率。其实这 2 种情况很相似，本质上都是对多个恶意事务的处理。

在本方案中：

(1)采取面向服务的入侵检测机制，将数据库系统看成一个服务的集合，以服务作为检测对象，为每一个攻击实例产生相应的攻击子图，提高其并发程度，以快速、准确地完成对入侵行为的检测；

(2)对基于校验点的方法进行了改进，提出了一种基于破坏隔离围堵的安全恢复策略，以日志中记录的更新为依据，在关系图上跟踪和定位破坏的传播，避免了复杂推理过程，既可以表现攻击之间存在的交叉联系，又可以对破坏及其传播精确定位，及时把初始阶段错误隔离的数据对象及时释放出来，提高算法的效率。

比如，在处理恶意事务B_0的同时检测到另一个攻击行为，并根据攻击子图得知B_1也是恶意事务。

实例 2 如下：

$H_1: B_0W[w], commit_{B_0}, B_1W[x], B_2W[u], B_3R[u],$
$B_1W[y], commit_{B_1} B_2R[x], B_2W[y], B_4R[u], B_4W[z],$
$commit_{B_4}, B_2R[u], commit_{B_2}, B_3R[u], B_4W[v], B_1R[u]B_3W[v],$
$commit_{B_3}, B_5R[y], B_6R[v], commit_{B_6}, B_5R[u], commit_{B_5},$
$B_7W[l], commit_{B_7}, B_5W[u], commit_{B_5}, B_8R[l], commit_{B_8}$
$Update_1: t_1 \cdots\cdots B_0W[w], Client_1, Server_6, w$
$Update_2: t_2 \cdots\cdots B_1W[x], Client_2, Server_1, x$
$Update_3: t_3 \cdots\cdots B_2W[u], Client_5, Server_3, u$
$Update_4: t_4 \cdots\cdots B_3R[u], Client_4, Server_3, u$
$Update_5: t_5 \cdots\cdots B_1W[y], Client_2, Server_2, y$
$Update_6: t_6 \cdots\cdots B_2R[x], Client_3, Server_1, x$
$Update_7: t_7 \cdots\cdots B_2W[y], Client_1, Server_2, y$
$Update_8: t_8 \cdots\cdots B_4R[u], Client_6, Server_3, u$
$Update_9: t_9 \cdots\cdots B_4W[z], Client_3, Server_5, z$
$Update_{10}: t_{10} \cdots\cdots B_2W[u], Client_4, Server_3, u$
$Update_{11}: t_{11} \cdots\cdots B_3R[u], Client_2, Server_1, x$
$Update_{12}: t_{12} \cdots\cdots B_4W[v], Client_1, Server_4, v$
$Update_{13}: t_{13} \cdots\cdots B_1R[u], Client_6, Server_3, u$
$Update_{14}: t_{14} \cdots\cdots B_3W[v], Client_4, Server_4, v$
$Update_{15}: t_{15} \cdots\cdots B_5R[y], Client_5, Server_2, y$
$Update_{16}: t_{16} \cdots\cdots B_6R[v], Client_3, Server_4, v$
$Update_{17}: t_{17} \cdots\cdots B_5R[u], Client_5, Server_3, u$
$Update_{18}: t_{18} \cdots\cdots B_7W[l], Client_6, Server_3, l$
$Update_{19}: t_{19} \cdots\cdots B_5W[u], Client_5, Server_3, u$
$Update_{17}: t_{17} \cdots\cdots B_8R[l], Client_3, Server_3, l$

图 4 为实例 2 的信任图。

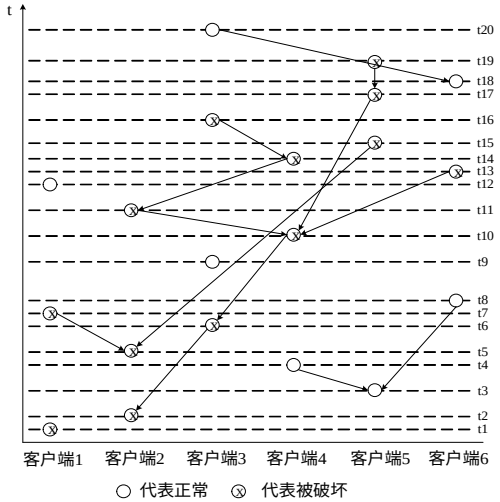

图 4　实例 2 的信任图

（下转第 186 页）