An event based framework for facilitating database activity tracking

Josip Saban
Hypo Alpe Adria International
Hypo Alpe Adria Platz 1
Klagenfurt, Austria
josip.saban@hypo-alpe-adria.com

Toni Grzinic
Croatian Research and Academic Network
J. Marohnica 5
Zagreb, Croatia
Toni.Grzinic@CARNet.hr

Leo Mrsic
Lantea Grupa d.d.
Ul. grada Gospića 1/A,
Zagreb, Croatia
leo.mrsic@lanteagrupa.hr

Abstract. *During a database analytic investigation investigator tracks intruder's actions on the system until incident occures - the investigator identifies that the intruder has, indeed, accessed the database in an unauthorized way. Combined with the data about the actions following the incident, it is also crucial to collect data about user activity on the server before the incident so that a log of actions can be created.*

The goal of this paper is to propose which data should be collected before the security incident occurs, focusing on two parts:
Users on the level of the operating system which have access to either the shared file system or the direct access to the operating system by using remote connection
SQL database users (either native or domain users), key tables with sensitive data and the activity of users in relation to those tables

Keywords. Database system, misuse detection, fraud transactions

1 Introduction

Discovering historical facts about user activity is a part of usual duties of investigators searching for a way to determine which actions an intruder performed within a database server. Databases are often used by applications to store, sort, and manipulate data. These applications can range from web-based, online banking applications designed to transfer funds that use databases to store client account information, to stand-alone applications that use a database solely to store application configuration settings.

A properly configured server environment should be spread across different network zones; network zones are logical boundaries that typically restrict inbound and outbound traffic depending on the application layers that reside within the zone. There are three main network zones:

- Untrusted zone - often contains data that is not verified and cannot be trusted
- Semi-trusted zone - contains data that at one point was verified, but due to exposure to untrusted zone hosts, now cannot be fully verified and trusted
- Fully trusted zone - data within this zone is normally under full control of the organization and, therefore, is fully trusted - the trusted zone is rarely directly connected to untrusted hosts
-

From the server security perspective databases typically consist of two parts - data files and backup files. Both of these files belong (in an ideal case) in a fully trusted zone (this scenario is usual for intranet applications, where databases hold sensitive data).
In the cases where a fully trusted scenario is not possible (when public access is required, either for display in applications or in a process of data validation inside of a business process which has public interfaces) databases are hosted in semi-trusted zone.

In this paper we shall try to define a required data collection process that will take into account the database and the environment in which the database is installed, including an analytic model that could be used for active data analysis. Although the database itself has to be secured, a great deal of security depends on the network settings, database user types (database native or domain users) and general environment of the server on which it is installed.

The main motive of the paper is to create a framework that will bridge the gap in the features available in the database and the underlying operating system. It will be used to store correlated events and information which will provide information about access to sensitive or otherwise protected data in the database. For example, in the banking environments, there is a special class of customers that are marked as "protected" - usually public officials - whose accounts should not be accessed by curious employees - this access needs to be specifically supervised and reported, specially for internal audit and compliance

processes. In addition, commercial DBMS systems like Microsoft SQL Server offer specialized audit solutions, in this case SQL Audit, which offer logging of almost all action on the database server. When those events occur, they are reported either in the form of alerts to audit administrators or saved to the audit database which is usually stored on the separate server.

Also, we cannot only look to database tables for analysis, but we also have to take into account other database objects that could be used to access or manipulate sensitive data. This process should, ideally, be in place at the time when database system goes into production, and at the latest when the system starts to be accessed from public networks.

2 Related work

Khanuja and Adane [1] created a list of all available database artifacts that can be tracked, and, although connected firmly to MySQL database, presents some interesting pointers on which database objects to track. Unfortunately, due to connection to just one database vendor and lack of proposed model, does not present a valid starting point for tracking database activity.

In the work of Gawali and Gupta [3] a valid model framework is designed, separating the tracked database from data warehouse that collects and stores the data, at the same mentioning no specific vendor. The work is significant because it calculates algorithm complexities of storing tracked data using different algorithms, proposing in the end a variation of hash algorithm that tracks all changes to rows, and creates hashes of content at certain point in time, though mentioning that this approach suffers from performance penalties which could be problematic in high - traffic databases.

Stahlberg, Miklau and Levine [4] present a detailed analysis of low-level file systems on which work different MySQL and PostGRE data storage solutions. They assess performance of these data storage solutions and present costs and benefits of each one. Although more concentrated on MySQL, PostGRE is also analyzed, specially with the relation to its clustering abilities, which is the key output of this work. As data is divided to multiple servers it becomes critical to collect data from all cluster instances.

Technically not so sophisticated, but still significant from retrospective aspect is the work from Khanuja and Adane [5], which creates a template of steps that need to be taken in order to create a successful database analytics solution. One key aspect that authors especially state is that algorithm used for authorization are equally important as those for authentication, which is usually forgotten.

Fowler [6] in his paper describes a case study for a real-life forensic research which presents a starting point for any forensic process and, although tightly related to one vendor, Microsoft SQL Server, gives an insight in the complexity of the topics. He covers all phases, from verification of the incident existence, evidence collection to data analysis. This presents a true read for anyone interested in applied database forensic analysis.

In [13] authors Low and Teoh present a fingerprinting method for detection of changed records (instead of the more standard hashing method). It replaces selected text columns with predefined patterns and then, using regular expressions, checks whether the content has changed. Although the concept is sound, the performance issues of string manipulations of this kind in relational databases prevent the usage of this method in high transaction environments - the performance decreases linearly with the increase of transaction sizes or users.

The seminal work in misuse detection systems is the work of D.Denning [7]. She presented a rule-based expert system that collects host data (login and session activity, command and program execution, file access) and from this data creates usage profiles. Misuses are detected from anomalous profiles that deviate from normal usage profiles.

Jin et al [9] realized that traditional database security mechanisms do not protect databases from misuse. The authors similarly to Denning proposed architecture for a database intrusion detection system. Their system can detect misuse by collecting requests to the database server on the network and host level. They used a multi agent approach with agents on supervising host monitoring changes and network traffic. Their system is supervised only for Oracle 10.2.0.1.

For SQL Server databases Zhang and Chen [2] proposed and implemented a rule based system based on rough set theory. Database server events are collected using the standard SQL Server profiler (for versions 2000 and 2005). They used 47 attributes as input to their classification model; every rule consisted of rule head and rule options. Rule head contains network specific data - source and target IP addresses, ports and net masks. Rule options contain alarm and pattern information needed for monitoring. Zhang and Chen rules look similar to the network intrusion detection system Snort[1].

[1] http://www.snort.org/

3 Methodology

In order to gather data we establish a pre-event analytic environment where we track two major data-sources:

- Operating system
- Database system

Model is created with a presumption of a Microsoft Windows Server 2008 R2 and Microsoft SQL Server 2008 R2 database. Operating system, in the context of data analysis, will be used for collecting following data:

- Local and domain users who have access to the server, and their logon history
- Access to the folders of those users where the database files are stored

That data will be collected on the regular basis, gathered and parsed from system activity log in order to acquire logon information for each user, combined with data from the local server store of the active domain and local server users. We also track user right changes, in the separate process, on the folder that contain database files.

Database data collection focuses on three key areas:

- Key database tables – this will be a static table, manually maintained by the analytic database system administrator, that will list all database tables that have analysis value
- Database users – a list of all database users (local and domain users) that have connection right to the database; also we collect a list of their privileges in relation the selected key database tables
- User activities – we collect data on the activity of users in relation to the selected key tables – these include, but are not limited to, length of connection, executed SQL command, time of those activates, plus special flags which are raised if such activities are performed

When the data collection process is active in both areas, it is combined in the dedicated analytic database which is, ideally, stored on the separate database server and secured only to be accessible to the data analytics team (both physical server and database server). When the data from both sources is gathered it is combined into one data model which represents the main data source for further analysis in the case of the security incident.

3.1 Server data collection

Data collected on the server side is comprised of users that have remote or direct access to the server combined with privileges on the folders where database files are located.

Users are divided into two groups - local server users (which exist only on the server and are not proxies of any domain-level process) and domain level users which are given rights to the server. Those users are divided into two subgroups:

- Those that have direct remote access to the server, using VPN or any kind of remote access solution (full or limited access to the server)
- Users that only have shared folder privileges and that can access the database through direct data access

User activity data is collected by parsing through the security section of Windows Server logs, while the list of users and their privileges on the system are acquired through the user catalog (for local users) or domain catalog (for domain users). Login and logoff activities are marked with category number 12544 and 12525 respectively, network access events (not Active directory access) are recognized by the logon type 3 and their source IP address can be retrieved from the attribute “IpAddress” contained in the event log. Network access stands for access to Windows share folder and similar activities.

Windows systems use different login protocols, the options in the Windows security subsystem architecture are: Negotiate, Kerberos, NTLM, SChannel (secure channel) and Digest. Interactive logins, where user enters user credentials in the initial Windows logon form (provided by the Graphical Identification and Authentication DLL - MSGINA.dll), is handled by the User32 process and has LogonType 2 which stands for the interactive login. LogonProcess and LogonType are fields stored in the additional section of logon events. Active Directory uses Kerberos for authentication, where domain controller and client exchange tickets in order to authenticate user logins; the logon process is called Kerberos and domain logins are marked with type 3. Active Directory events differ from interactive logins, and store information about host and controller communication. More about Windows login types can be found in Gupta’s paper Windows logon forensics [13] and Russinovich’s Windows Internals [14].
Folder location of the database files, in general, should not be accessible for users that do not have interactive remote logon rights (folders with database files should never be available through shared folders). Privileges on the folder are retrieved on the defined period and combined with the previously collected data. User and folder privilege data are then combined in one dataset, using the user ID and data collection time as the primary key for identifying a state in time.

3.2 Database data collection

Database data collection is divided into three distinct parts:

- Key tables
- Database users
- Database activity

First step in the process of defining a database monitoring process is to define key tables which hold relevant data as tracking entire set of tables would give us no additional value. These tables are listed in the analytics database with the following properties:

- Table name
- LogUPDATE tracked (are update commands tracked)
- LogDELETE tracked (are delete commands tracked)
- LogINSERT tracked (are insert commands tracked)
- LogDataDEFINITION tracked (are data definition changes tracked)
- TableNOTIFICATION needed (should we immediately notify responsible person on any table event)

Next step is to retrieve, on periodic basis, all database users and their privileges on the selected database tables. This is done by retrieving data from database system catalogs and saving it regularly to the dedicated table in the analytics database. Again we retrieve both local database users, domain users and groups which have access to the database based on domain settings.

The last element we store is the user activity on the selected tables, in accordance with the data retrieval or changes that we have deemed important. This is done in real-time (when the event occurs), unlike the previous activities which are collected in a periodically executed process.

Table "KeyTables" is manually maintained by analytics database administrators, and holds table names for which we are collecting data, plus additional information which information are we tracking for the selected table - the granularity is one the level of DDL statements, data definition changes and notification of defined responsible persons about selected tracked operations. Table "ActivityLog" holds actual logged data in the form of executed command and the number of affected rows.

To make data retrieval database vendor independent triggers are used for acquiring information we need, both for the data modification (DML triggers) and data structure changes (DDL triggers). Additional requirement, that cannot be generically recommended because it depends on database vendor, is that selected tables cannot be dropped during operational database work, but this has to be implemented based on a specific case.

Once all this data is stored, it is combined with the data collected on the server side, using the same user ID and data collection time as reference, while the real-time data is also stored in the analytics database but their create time is not used as a referential key.

It is important to note that local database user names usually do not correspond to the local server users (even though they may have the same name they are not the same user), and are treated as separate entities. On the other hand, domain users on the database can, but do not have to, exist on the server, as domain users do not need server access to gain database access.

3.2 Framework's model

The created model (Figure 1), in detail explained in Table 1, is used to store both the data from the server and from the database collection processes. Each entity is described in detail, along with each entities attributes.

Table 1. Entities descriptions of the proposed framework

Entity Name	Entity Description
WindowsUsers	Master data collected about server local and domain users
WinLoginActivity	History of login activity of users defined in table "WindowsUsers"
KeyTables	List of tables we are tracking for structure changes and data modification
SQLUsers	Database users, both local database users and domain users
DDLActivityType	List of tracked activities, which are listed previously in the chapter about database data collection
ActivityLog	A combined log of database users, key tables and DDL activity types, with details about each event

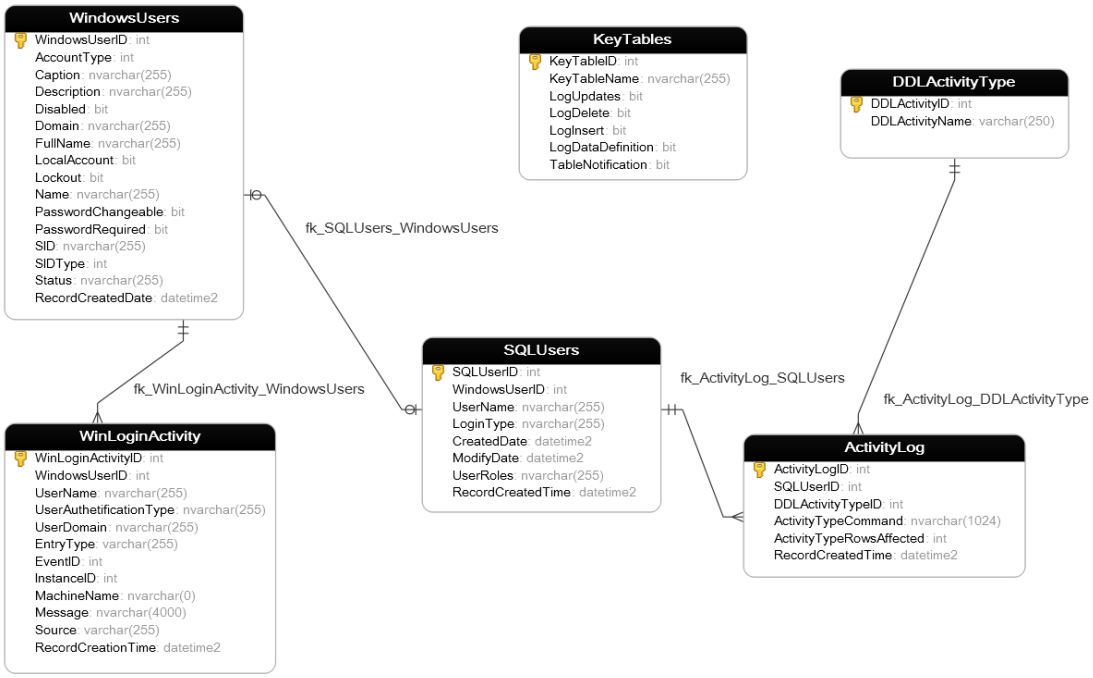

Figure 1 – Entity relationship diagram of the proposed framework

4 Conclusion and further work

Recent research in database analytics have shown that the need for correct, secure and reportable history of key database data is evident, both in commercial and academic environments, especially in environments that are subject to possible forensic research. Current findings suggest that, although much of the research is already done, standardized data collection in time before the incident is the key to later analysis, and this paper proposes such data model, which, although implemented on dedicated vendor technology, can be replicated in any environment with minimal changes. Also, this model provides sufficient data quality and event granularity that it can be useful for further analysis.

Some database systems offer integrated auditing solutions, like the before - mentioned Microsoft SQL Server Audit, but in most cases analysis of audit data is not supported. Also, we are collecting data about the underlying operating system's user activity which is also a key parameter for further analysis on database actions. Our framework would be used as basis for analyzing collected information and isolating potentially malicious or otherwise important events.

As a next step we would propose to define key attributes for the classifications of the events, which are able to differentiate normal and abnormal behavior of users. Decision attributes would be used for creation of a rule-based decision support system, ideally based on fuzzy logic concepts, that would be used to predict improper cases of key tables usage as an early warning system. Prerequisite for building a classification model is a training set with previously abnormal events. Data from this model can also be used as a basis for modeling reports that are used for historical or forensic data analysis.

References

[1] Khanuja, H. K., & Adane, D. S. (2012). A Framework for database forensic analysis,. *Computer Science & Engineering*, *2*(3).

[2] Zhang, J., & Chen, X. (2012). Research on Intrusion Detection of Database based on Rough Set. *Physics Procedia*, *25*, 1637-1641.

[3] Gawali, P. P., & Gupta, S. R. Database Tampering and Detection of Data Fraud by Using the Forensic Scrutiny Technique.

[4] Stahlberg, P., Miklau, G., & Levine, B. N. (2007, June). Threats to privacy in the forensic analysis of database systems. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (pp. 91-102). ACM.

[5] Khanuja, H. K., & Adane, D. D. (2011). Database Security Threats and challenges in Database Forensic: A survey. In *Proceedings of 2011 International Conference on Advancements in Information Technology (AIT 2011), available at http://www.ipcsit. com/vol20/33-ICAIT2011-A4072.pdf*.

[6] Kevvie Fowler (2007) Forensic Analysis of a SQL Server 2005 Database Server, *SANS Institute InfoSec Reading Room*

[7] Denning, D. E. (1987). An intrusion-detection model. *Software Engineering, IEEE Transactions on*, (2), 222-232.

[8] Lee, S. Y., Low, W. L., & Wong, P. Y. (2002). Learning fingerprints for a database intrusion detection system. In *Computer Security—ESORICS 2002* (pp. 264-279). Springer Berlin Heidelberg.

[9] Jin, X., & Osborn, S. L. (2007). Architecture for data collection in database intrusion detection systems. In *Secure data management* (pp. 96-107). Springer Berlin Heidelberg.

[10] Sunil Gupta, Windows Logon Forensics https://www.sans.org/reading-room/whitepapers/forensics/windows-logon-forensics-34132, Sans Institute, January 2013

[11] Russinovich, Mark E., David A. Solomon, and Alex. Ionescu.Windows Internals, Part 1 (6th Edition) ,page 555, Microsoft Press, 2009.

[12] Spalka, A., & Lehnhardt, J. (2005). A comprehensive approach to anomaly detection in relational databases. In *Data and Applications Security XIX* (pp. 207-221). Springer Berlin Heidelberg.

[13] Low, W. L., Lee, J., & Teoh, P. (2002, April). DIDAFIT: Detecting Intrusions in Databases Through Fingerprinting Transactions. In *ICEIS* (pp. 121-128).