A Framework for Database Auditing

Lianzhong Liu, Qiang Huang
School of Computer Science and Engineering, Key Laboratory of Beijing Network Technology
Beihang University
Beijing, China
lz_liu@buaa.edu.cn, hqbuaa@gmail.com

***Abstract*—Database auditing can help strengthen the security of database. In this paper, we present a framework of database auditing, which log the database activities through analyzing network traffic, execute audit analysis through event correlation and generate alarms if an anomaly or a violation of security regulations is detected. Compared with native auditing mechanism in database, our approach has an obvious advantage of providing zero-impact to the performance of the database or the applications that access it. In addition, using third-party auditing component complies with the principle of separation of duties.**

Keywords-database auditing; logging; audit analysis

I. INTRODUCTION

Databases are behind the systems that affect almost every aspect of our lives—our bank accounts, medical records, employment records, phone records—almost every piece of information of significance in our lives is maintained by a modern relational database management system. If database systems—the systems we all implicitly trust to hold our most sensitive data—are not secure, the potential impact on our lives, and even on our broader society, could be devastating[1]. As organizations increase their adoption of database systems as the key data management technology for day-to-day operations and decision making, the security of database becomes crucial.

By implementing user identification, authentication, access control and other pre-event measures, the security of database system can be elevated. But it is far from secure enough, because these measures are not capable of monitoring and logging database activities, which can help audit the system from a secure standpoint.

Traditional database security mechanisms such as data encryption and access control are rendered useless by a trusted employee who has—or can easily obtain—the right credentials [2]. In addition, more users in the enterprise are getting database access, including DBAs (Database Administrators), application developers, software engineers, and even marketing, HR, and customer support representatives. If an enterprise or an organization has not implemented such measures as database monitoring or auditing, then their invaluable assets reside in database are at high risk. At the same time, enterprises are under increasing regulatory and market pressure to protect sensitive information.

The only solution to this problem is to implement database auditing. Database auditing involves observing a database so as to be aware of the actions of database users. Security and data center teams must roll up their sleeves to implement and enforce a set of best practices to address the insider threat. IT architects must then bolster policies by using database auditing besides other security features built into all major database platforms.

A critical subset of users, including DBAs and any systems administrators with access to sensitive information, should be closely monitored and audited. It's also advisable to take auditing functions out of the hands of DBAs and put into the hands of a compliance officer or other group, such as a security or network operations team, for the sake of segregation of duties, which is a critical principle in the field of security.

Though all major database platforms have built-in access controls and auditing functions, administrators and auditors may find it cumbersome to search through raw logs for unusual behavior. Third-party database monitoring tools can augment database auditing and logging functions by aggregating and correlating log events and searching for deviations from normal activity patterns.

This paper is to present a framework for database auditing. Our approach is to tap into the network with an agent, monitor traffic flowing into and out of the database system, extract the log information useful to audit analysis and store the log in another server. Then, audit analysis will be executed and alarm will be generated if any database anomaly or violation of security regulation is detected. The following parts of this paper are organized as follows: section 2 will survey some related work. The logging component will be described in section 3. Audit analysis with event correlation will be provided in section 4. We will conclude the paper in section 5.

II. RELATED WORK

To our knowledge, there is no open source project or publication on this subject. However, there are some related commercial products.

Application Security's AppRadar [3] gets SQL commands from the network. Agents for Sybase, Oracle and IBM DB2 databases tap into network and make a copy of every packet through network switching equipment using a monitor or Switched Port Analyzer (SPAN) port. This does not cause transaction latency on the database or network traffic because

978-0-7695-3896-9/09 $26.00 © 2009 IEEE
DOI 10.1109/ICCIT.2009.207

it does not block any packet. For SQL Server, an agent is deployed in the system where SQL Server is running.

Lumigent's Audit DB [5] simply collects logs from native database products so they purely rely on the native database's audit log. Agents are installed to the system where native database resides in. Agents monitor, alert and collect all database transactions from the SQL Server transaction log, Oracle redo log or the native database audit feature (Trace or Oracle Audit). This approach is the simplest solution for logging multiple heterogeneous databases. However, this is not fundamentally different from native database's approach. So, Audit DB is still not able to detect attack from inside because if audit log in native database is modified on purpose, the result directly affects the analysis of Audit DB. Their contribution is to transform different log format to their own format.

Guardium's SQL Guard [4] can operate as either inline or passive server and can migrate from monitoring to prevention modes. If SQL Guard is deployed as inline, it sits between user terminal and database and checks every incoming packet to see if it is related to an attack or unprivileged activity. If it thinks a packet is a dangerous one, it will block the packet so the user issued command cannot reach to the database, thus operating in prevention mode in this case. Blocking is powerful function and may be useful in the sense that it has prevention capability. It also has the potential to have a false positive response. In worst case, a legal user's safe SQL command can be blocked. Some products provide pattern learning functionality to lower the false-positive. AppRadar and Audit DB do not have pattern-learning, or profiling, feature. However, Guardium's solution automatically suggests customized policies via a "learning mode" that establishes a baseline of normal activity and looks for anomalous behaviors. (Policies are generated or derived from baseline of normal activity.)

III. The Logging Component

The first step of database audit is logging. With a proper logging method, database activities and other database information like system use and performance can be recorded on administrator’s demand. Then, the logs can be utilized in audit trail analysis and database usage report generation, so as to (i).determine if security and other policies are being violated and if so, provide evidence of violation; (ii).discover attacks to the database; (iii).help recover a database if there is any damage happened to it.

Compared with native auditing and logging mechanism in database, network-based database monitoring provides zero-impact to the performance of the database or the applications that access it, providing the ultimate level of protection without penalty. In addition, as database administrators can disable the native auditing function and avoid to be monitored, using native auditing does not meet the compliance with the policy of separation of duties. What’s more, the intrusion to the host where a database resides may allow an attacker to disable or tamper with the auditing system and even to destroy the database.

A. Network-base Methods

Monitoring methods deal with approaches to collecting data or logging [2]. The first method consists of passive server or device that taps into the network and monitors traffic flowing into and out of the database system. AppRadar [3] is an example of software that uses the passive monitoring method. Passive monitoring will not slow down the network and it will cause no overhead in database system. Although passive monitoring does not cause any performance delays, it is not able to monitor encrypted traffic.

The second way to logging database activities based on network is to put devices directly inline with the network so that all packets going in and out are intercepted and passed by the device. Guardium’s SQL Guard [4] is a system that is able to work as both passive and inline modes. Inline monitoring methods can block attacks or misuse. It is also able to monitor encrypted packets. Decryption of packets and verifying of each packet may cause network latency.

The third method is agent based monitoring where the extraction of logs is done directly from the database. Example of agent based monitoring is Lumigent’s Audit DB [5]. Audit DB distributes individual agents that extracts log from each of the database(s) and reports to the central auditing system where analysis and reporting are done. Agent based monitoring is able to validate all database activity not affected by encryption. Although this method provides complete logging, it may affect the performance of database systems. Also with multiple agents, complexity of system increases.

B. Our Approach—Passive Mode

We choose passive mode among these methods for its high efficiency and flexibility. The agent run in passive mode to capture packets with the feature called port mirroring of the switch. Another method involves connecting the agent to the database server with a hub, and set the network card to work in promiscuous mode, which will also work. After packets have been captured, protocol analyzing is utilized to parse the packets. SQL statements, database commands and other information like client IP address and connecting user name, which are what we want from the logging process, will come out when finishing packets parsing.

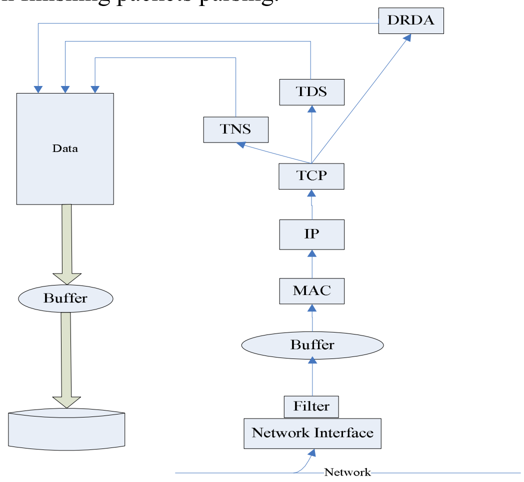

Figure 1. The architecture of the logging component

Figure 1 illustrates the principal components and arrangement of our architecture of a network traffic analyzing based logging scheme. We use the Berkeley Packet Filter (BPF) [6] filtering mechanisms to examine packets and match them against given criteria. The reason that we use this framework for our filtering module because it is powerful, easy to use and accessible via the libpcap library we use for packet capturing. libpcap provides a function that compiles strings to BPF programs, as well as a function that takes a BPF program and a packet and returns whether the packet matches the filter rule or not.

From the database logging approaches discussed above, we pick out the passive mode for its high efficiency and good extendibility. We divide the approach into three stages: packet capturing, packet parsing and data storage. Buffering between the stages accommodates burst in packet arrivals and variations in packet parsing and data storage rates.

C. *Packets Capturing*

The databases we are monitoring are Oracle, SQL Server and DB2, and the communication protocols they used are TNS (transparent network substrate), TDS (Tabular Data Stream) and DRDA (Distributed Relational Database Architecture) respectively. The three application layer protocols are all based on TCP protocol, so the packet filter will discard all the non-TCP packets when capturing. After the captured packets reach the application layer, we identify them by port numbers and classify them at the same time.

The logging system we devise has to run on a computer with a network interface to the network from which data has to be cached. Such a computer system is called a packet capture system or sniffer. The sniffer's network interface card (NIC) that is connected to the network to observe is set to promiscuous mode by the operating system, such that all packets that are observed by that NIC are accessible by the packet capture application (i.e. our logging component).

There are two approaches to providing data for the packets parsing module. The first is to capture the data on-line: the data is provided directly to the packets parsing module from the wire. This is the standard mode of operation. The second approach is to provide the processing system with a trace captured off-line. In this mode of operation, the processing system can, as an alternative to drawing packets from the receive buffer pool, read them from a trace file. This facility was provided for development purposes, but it is also useful to have the ability to apply data extraction and analysis to traces collected using libpcap.

D. Protocol Analyzing

For a number of protocols (e.g., HTTP, FTP), specifications and corresponding protocol parsers are publicly available. However, there are also a large number of proprietary, closed protocols for which such information does not exist. Unfortunately, TNS, TDS and DRDA are of this kind of protocols. For these protocols, the way of determining a specification involves protocol reverse engineering.

Protocol reverse engineering is the process of extracting application-level protocol specifications. The detailed knowledge of such protocol specifications is useful for addressing a number of security problems. In addition, protocol specifications are often required for intrusion detection systems that implement deep packet inspection. These systems typically parse the network stream into segments with application-level semantics, and apply detection rules only to certain parts of the traffic.

E. Packets Parsing and Result Storage

After protocol analyzing, we get the specifications of the three database communication TNS, TDS and DRDA, which can guide our packets parsing work. With the protocol specifications, parsing database communication packets can be like parsing other packets with well-known protocols. At this point, we can extract SQL statements and database commands from packets. As what we care is the security of the database server, it is obvious that we should pay more attention on the requests to the database than the responses to the clients. A database response is noticed only when it reply to a login request, especially a failure login response.

When extracting, other information besides the SQL statements and database commands, like source and destination IP addresses, source and destination MAC addresses, should also be extracted from the packets. In addition, other information like timestamp and identifier of each entry (where the information of a whole SQL statement or a database command stored) should be tagged by system. Once all needed information is gained, the information will be inserted by entry into another database, better on another independent host from the host who hold the monitored database, usually on the same host as the monitoring system resides.

IV. ALARM GENARATION THROUGH EVENT CORRELATION

After collecting the audit log, how to analysis and make use of log info to ensure the normal function of database becomes an important problem. The diversity and complexity of databases aggravates the probability of malfunction. So it is crucial that when the business is in fault, efficiently and correctly find the root cause of malfunction, in order to reduce losses. Audit alarm is that audit the network event and system log info from the business point and then find the audit alarm relative to security. We take the method of event correlation to analyze the audit log and detect the anomaly or violation.

We make use of event correlation method based on rule-based reasoning which uses surface knowledge reasoning, it takes the mechanism of forward-chaining inference. Rule-based reasoning does not require the deep knowledge of monitoring system structure and operation principles and provide a powerful tool to reduce the probable fault [7]. So it is suitable for the changeless network ontology. And security audit is referenced to the internal network which is small and steady. So it is reasonable to use rule-based reasoning.

Rule-based reasoning is a kind of artificial intelligence system and requires a knowledge representation and reasoning mechanism to match the rule. The rules have to form "conclusion if condition". The condition contains received events together with information about the state of the system, while the conclusion may consist of actions which lead to changes of the system and can be input to other rules [8]. Production system is the most basic structure of artificial intelligence system, is the natural knowledge representation and reasoning method. In fact, the algorithm of RETE can accelerate the correlation speed of rule-based reasoning. And next, we will illustrate rule-based reasoning from three aspects: production system, CLIPS and RETE.

A. THE PRODUCTION SYSTEM

Production system uses fact and production to express the knowledge and consists of production set, assertion library and reasoning engine. Production set stores in production storage, assertion library is stored in working area. Every production contains condition and action set, the former act as the left of production and the latter act as the right. Rule engine is used to execute the rule, includes three stages: match, select and act which is called as match-select-act cycle or recognized-act cycle. In the matching stage, rule engine judge whether the conditional element meets the current environmental of working area; in the selection phase, when the plus conditional element meets but negative does not, put the each matched instance of rule into the conflict set. And then according to the existed standard of production system named the solution of conflict , select the suitable instance of rule; in the act stage, production execute the act of right of selected rule instance.

The reasoning of production system includes forward and reverse chaining inference. Forward reasoning refers that according to the existed condition, from the bottom-up to reason and finally achieve the goal of reasoning. Reverse reasoning take the method of up-bottom until suits the current condition. Use the forward reasoning, the left of production system guides the search, but the reverse reasoning use the right to instruct the search. So from the direction of reasoning, production system is divided into two categories: forward-driven and after-driven.

We take forward-driven to implement rule-based reasoning to find the root cause of audit alarm. For example, the computer A can access other computers but B→a link fault between A and B. the after of the rule is a hypothetical fault, because it is one of many fault root cause, the forward is the description of events and its relationship. When the event instance which meets the rule occurs, rule-based reasoning will correlate the events, put the relative fault hypothesis into conflict set and finally according to the standard defined give the audit alarm to notify the administrator.

B. CLIPS

Every production system must use one or more kinds of relative language to describe production. CLIPS is a new design language of artificial language, designed by NASA in the late 1980, implemented by standard C and has the style of LISP. At the same time, CLIPS is also a develop language for expert system. It has the initial event library and knowledge management system and its reasoning use the efficient algorithm of Richard to implement matching. Compared to LISP and PROLOG, CLIPS is more efficient, requires low performance of system and has good real-time features; in contrast with OPS, ART, CLIPS is more reliable, programmed easily and maintaining simply.

CLIPS model the knowledge and experience of human, facilitate the development of artificial software. In CLIPS, there are three ways to express the knowledge. Firstly, rule, always used to the heuristic knowledge based on experience. Secondly, the self defined function and the generic function. Lastly, object-oriented programming, support the five characteristics: class, abstract, encapsulate, extend and polymorphism, mainly for the process of knowledge.

C. Description of event correlation type

After the introduction of CLIPS, we give the description of event correlation's type in CLIPS. There are four basic types of event correlation, such as filter, suppression, compression and cluster.

1) Filter correlation. It discards the event instance which does not meet the existed conditional fact. For example, some event can not act as the root cause of fault. When these events occur, we should filer them. We give the presentation of filter correlation.

$$Filter\left[e_1, e_2, e_3, \cdots, e_i\right] = \left[\overbrace{e_j, e_k, \quad e_l, \cdots}^{n}\right] \qquad (1)$$

$$i > 1,\ 1 \le j \le k \le l \le n < i$$

CLIPS uses NOT to filter event instance, describes the correlation with the key word "not".

2) Suppression correlation. It suppresses the event instance having occurred of low priority after the high priority event has happened. Described as:

$$Suppression[e_i, e_j, \Pr iority(e_i) > \Pr iority(e_j)] = \left[e_i\right] \qquad (2)$$

$$i \ge 1,\ j \ge 1,\ i \ne j$$

Priority is an attribute of event instance. Suppression is implemented through predicate function. In CLIPS, uses the key word ":" to define the function. In CLIPS, use ":" to express function binding, and "&" express that assign the value that meets the function binding to variables B, so complete the process of suppression correlation.

3) Compression correlation. It compresses some event instances into one event instance in spite of the times of event. For example, the fault of database access happens some times because of link fault, then compress these events into one events express the same meaning. Described as:

$$Compression\left[\overbrace{e_i, e_i, \cdots, e_i, e_i}^{n}\right] = e_i \qquad n > 1, i \ge 1 \qquad (3)$$

4) Cluster. It correlates the events which have been already correlated, that is, the second correlation with logical operation. For example, if the rule A and B meet the operation of "and", then we cluster these two rules.

D. The process of correlation

For each rule, judging whether the fact of work area meets the right schema of left is the key process of product system matching. For the real-time system, how to decrease the matching time is very crucial, because matching takes most time of whole reasoning [9]. There are many methods to achieve this goal, such as the algorithm of RETE, TREAT, parallel matching.

RETE is an efficient algorithm which can speed the matching effectively for production system. This algorithm is proposed by Charles L. Forgy of Carnegie Mellon University in 1974. It provides a powerful implementation for expert system. As the above given, Rule engine is used to execute the rule, includes three stages: match, select and act which is called as match-select-act cycle or recognized-act cycle. RETE can implement the matching stage efficiently.

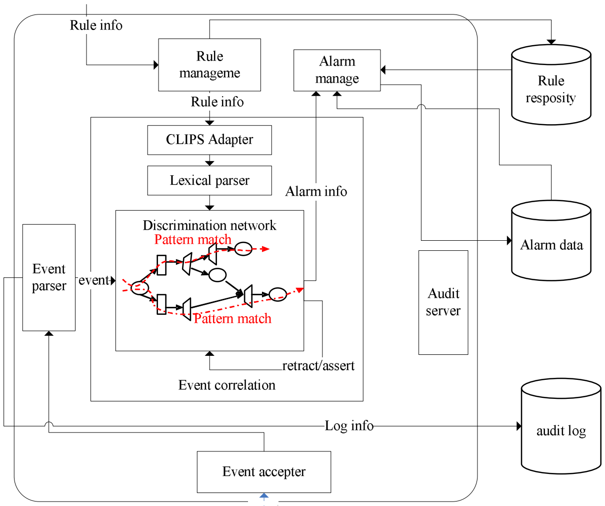

Figure 2. The process of generating audit alarm

E. The process of correlation

After introduce the requisition of rule-based reasoning, we give the whole process generating audit alarm.

As shown in Fig.2, event accepter is used to receive the audit log info. We use the event parser to parse the event. Parsed events act as input of discrimination network. We take the RETE as the algorithm of matching stage. RETE is a quick matching algorithm and makes use of discrimination network to match. Discrimination network is a single root and directed-acyclic graph. Through it, we can store some matching results and correlate with other events. In order to complete the matching stage, the parsed event and rule expressed in CLIPS is the requisite inputs. After the phase of matching, generate the audit alarm.

CONCLUSION

By implementing database auditing, database security can be strengthened. In this paper, we present a framework of database auditing, which log the database activities through analyzing network traffic, execute audit analysis through event correlation and generate alarms.

Compared with native auditing mechanism in database, our approach has an obvious advantage of providing zero-impact to the performance of the database or the applications that access it, providing the ultimate level of protection without penalty. In addition, as DBAs can disable the native auditing features and avoids to be audited; using native auditing does not meet the compliance with the policy of separation of duties. However, network-based logging has its shortcoming too, if the database communication has been encrypted, the method of passive packets capturing will be invalid. Our future work will focus on this issue.

ACKNOWLEDGMENT

We would like to acknowledge the support of Co-Funding Project of Beijing Municipal Education Commission under Grant No. JD100060630.

REFERENCES

[1] Litchfield D, Anley C. The Database Hacker's Handbook, Wiley Publishing Inc., USA, 2005.

[2] Andrew Conry-Murray. The threat from within. Network Computing, http://www.networkcomputing.com/showArticle.jhtml?articleID=166400792. 2005.

[3] Application Security, Inc. AppRadar http://www.appsecinc.com/.

[4] Guardium. Guardium SQL Guard. http://www.guardium.com/.

[5] Lumigent. Audit DB. http://www.lumigent.com/.

[6] Steven McCanne, and Van Jacobson, "The BSD packet filter: a new architecture for user-level packet capture". In Proc. USENIX Winter 1993 Conference, pp. 259-270, 1993.

[7] Nick Cercone, Aijun An, Christine Chan. "Rule-Induction and case-based reasoning", IEEE Transactions On Knowledge and Data Engineering, pp. 166-174, 1999.

[8] Andreas Hanemann, Martin Sailer, " A framework for service quality assurance using event correlation techniques", in Proc. 2005 the Advanced Industrial Conference on Telecommunications/Services Assurance with Partial and Intermittent Resources Conference/Elearning on Telecommunications Workshop, 2005.

[9] Jeong A Kang, Albert M.K. Cheng, "Recuding matching time for OPS5 production systems", Computer Software and Application Conference, pp. 429-434, 2001.