2009 Second International Workshop on Computer Science and Engineering

A Logging Scheme for Database Audit

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

***Abstract*—Database audit can strengthen the security of database. Logging database activities is usually the first step of implementing database audit. In this paper, we present a logging scheme for database audit. Unlike native database logging and auditing mechanism, our scheme is to monitor and log database activities through analyzing network traffic. The architecture of our scheme contains three principal components: packets capturing, packets parsing and data storage. First capture the packets to and from the database; then, by analyzing database communication protocols, parse the captured packets; finally, use the parsed results to support database audit.**

Keywords-logging; database audit; network traffic analyzing

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. Database auditing involves observing a database so as to be aware of the actions of database users. This is often for security purposes, for example, to ensure that information is not accessed by those without the permission to access it. This paper is to provide a logging scheme to support database audit, which is a post-event mechanism to harden a database.

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.

Our approach to implement a database logging scheme is to tap into the network with an agent and monitor traffic flowing into and out of the database system. 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 addresses and connecting user names, which are what we want from the logging process, will come out when finishing packets parsing. At last, store the results safely in an independent server for upcoming audit analysis. The following parts of this paper are organized as follows: section 2 will discuss some database logging approaches, and section 3 will present some related work. The architecture of our logging scheme will be described in detail in section 4. We will conclude the paper in section 5.

II. DATABASE LOGGING APPROACHES

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.

Although major database system venders support various functions to users such as generating new audit rules, managing the log, and recovery of the original database if any corruption has been found due to malicious users' activities, simply turning audit function on does not guarantee the completeness of database audit because more and more threats are coming from within. The need to address increasing security breaches from within requires more effort to secure the data than is provided by the native database's auditing tools. For example, it is necessary to restrict the privilege to modify sensitive data even if the user is a DBA. In addition, privileged users must not be allowed to modify audit logs in any case because the audit log is the key component for the investigation and system recovery.

Triggers are often used when implementing audit in native database. Using triggers has a few advantages. First, it is easy to implement. Second, it is flexible in the sense that one who writes a trigger can freely modify it. However, even though a trigger is easy to write, it is also hard to maintain. Triggers require modification whenever any change is made to the database schema. Triggers are easily disabled and enabled later. So database systems that rely primarily on triggers for auditing are not very effective in preventing or detecting an inside threat from privileged user who can freely manipulate the trigger. In addition, internally developed tools using triggers do not capture all activity that it is desirable to monitor. For example, native database capabilities may not be able to detect attacks which are done by embedding SQL command in applications. Furthermore, triggers might interfere with runtime performance so may lead to performance degradation. If a large volume of audit logs have to be stored in a table in real time, it causes undesirable overhead. In addition, all internal activity can not be absolutely independent from DBA.

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.

Passive and inline methods are similar in that they can both monitor network traffic. For this reason, they share common characteristics. Both methods are not able to monitor activities of users who are directly logged into the database. This means that they are not able to protect that database against insider threats. Even with the downfall, both methods are frequently used because they do not impact the database system performance. Also, network traffic does not consist solely of database network packets, so it is important that they have capability to filter packets not specific to database servers.

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 databases and reports to the central auditing system where audit 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.

The fourth method of logging and monitoring is a system integrated method. System integrated monitoring methods act as part of the database system. This method consists of all monitoring methods provided by Oracle, SQL Server, Sybase, and all other database system that provide auditing methods. System integrated monitoring provides complete logging. However, like any other programs that run in the database system, it will degrade the database performance seriously.

III. Related Work

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

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 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. 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 both inline and 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 activities.

IV. System Architecture

Figure 1 illustrates the principal components and arrangement of our architecture of a network traffic analyzing based logging scheme. We use BPF (Berkeley Packet Filter) [6] filtering mechanisms to examine packets and match them against given criteria. The reason that we use this framework for our filtering module is that it is powerful, easy to use and accessible via the libpcap library we use for packet capturing. The libpcap library 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.

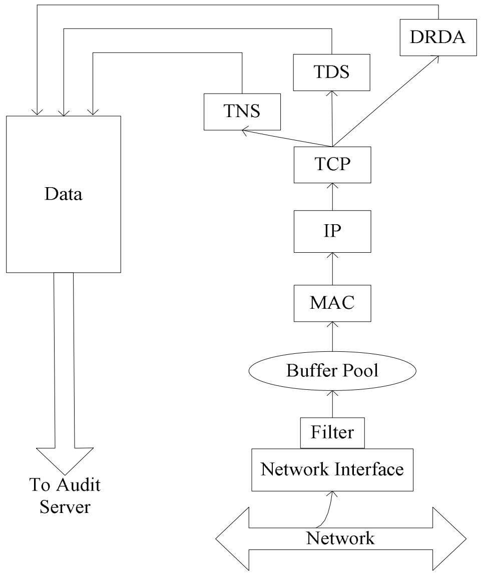

Figure 1. The architecture of the logging system

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.

A. *Packets Capturing*

The databases we are monitoring are Oracle, SQL Server and DB2, and the communication protocols they used are respectively TNS (Transparent Network Substrate), TDS (Tabular Data Stream) and DRDA (Distributed Relational Database Architecture). 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 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 NIC (Network Interface Card) connected to the network to observe is set to promiscuous mode by the operating system, so that all packets that are observed by that NIC are accessible by the packet capture application (i.e. our logging system).

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.

B. *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.

To address the limitations of manual protocol analysis, automatic protocol reverse engineering techniques have been proposed. The goal of these techniques is to automatically generate the specification of an application-level protocol, given as input one of two different sources: The first source is the application program that implements a particular protocol. So far, researchers have proposed a static analysis approach that takes as input a binary program and outputs the set of inputs that this program accepts [7].

The second source of input for automatic protocol reverse engineering systems is network traffic. This is also the method we take in experiments. More precisely, a number of systems have been proposed that analyze network traces generated by recording the communication between a client and a server. To this end, the network traces are examined for the occurrence of common structures or bytes that indicate a special meaning for the protocol. While experiments have shown that these systems are able to analyze real-world protocols, their precision is often limited [8].

C. *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, which is on another independent host from the host who hold the monitored database, usually on the same host as the monitoring system resides in.

Logging database activities is often the first step of a database auditing system. The data generated by monitor will be accessed by these systems for their own purpose.

V. Conclusion

By logging and auditing, database security can be strengthened. In this paper, we present a scheme of database logging, which monitor and log the database activities through analyzing network traffic. It contains three principal components: packets capturing, packets parsing and data storage. First capture the packets forwarded to the database; then, by analyzing database communication protocols, parse the captured packets; finally, use the parsed results to support database audit.

Compared with native logging and auditing mechanism in database, network-based database monitoring and logging 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 database administrators can disable the native auditing features and therefore avoid to be monitored, 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 feature work will focus on how to solve or evade this issue when implementing database audit.

Acknowledgment

This work has been supported by Co-Funding Project of Beijing Municipal Education Commission under Grant No. JD100060630.

References

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

[2] A. C. 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] M. Roesch, “Snort - Lightweight Intrusion Detection for Networks,” Proc. 13th Systems Administration Conference - LISA '99, pp. 229-238, 1999.

[7] J. Newsome, D. Brumley, J. Franklin, and D. Song, “Replayer: Automatic Protocol Replay by Binary Analysis”, Proc. 13th ACM Conference on Computer and Communications Security (CCS), 2006.

[8] G. Wondracek, P. M. Comparetti, C. Kruegel, and E. Kirda, “Automatic Network Protocol Analysis.” Proc. 15th Annual Network and Distributed System Security Symposium (NDSS'08), Feb. 2008.