TRANSACTION LOG ON WORM ARCHITECTURE WITH BUILT-IN AUDIT HELPER

BY

YUN HEE LEE

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Advisor:

Professor Marianne Winslett

Abstract

Regulatory Compliance in current industries has become a mandatory requirement of every day business. Many of regulations and policies set standard of conduct by insiders, who, by definition, have been entrusted with authorized access to the system. Therefore, existing security measures that aim at protecting the system from unauthorized access from outside are ineffective in providing sufficient protection and compliance solutions.

In considering the compliance solution for current industries, it is vital that the cost of compliant is justifiable for companies. That is, if the costs such as system migration and operational cost outweigh the business value and societal benefits brought on by compliance, there is no rationale for the companies to actively install compliance solutions.

This thesis reviews and extends transaction log on WORM (TLOW) architecture [4], a practical compliance solution that supports long-term immutability for relational tuples. Given that majority of enterprise information systems are supported by RDBMS, TLOW allows for smooth transition to the compliant system minimizing the cost and user resistance in its adoption. This thesis aims to solidify the TLOW architecture by internalizing the Audit Helper (AH) module, allowing for securer and more efficient operations by the module.

To my mother.

Acknowledgments

This thesis would not have been possible without the guidance, support and encouragement from my advisor, Professor Marianne Winslett. She was always there with her insight to discuss and analyze my research interests, guiding me to discover research topics that are relevant in real world. Under her supervision, I learned to keep broader perspective of the research topic at hand, and question its practicality and real value. This has made my graduate studies so much more rewarding than I ever expected.

My deepest gratitude is also due to Ragib Hasan, for his unreserved help and advice. Ragib always gladly offered his deep technical and research knowledge whenever I faced a roadblock, guiding me to navigate the problem and resolve it. His answers to my questions and consultations have been invaluable throughout my research.

I am grateful for many wonderful friends I met at UIllinois, without whom I cannot have endured the life of a graduate student, which can be daunting at times. These friends have constantly reminded me to look beyond the next homework or deadline, and enjoy the great privilege of being at UIllinois studying and sharing laughter with them.

I am thankful for my family whose support, love and encouragement enabled me throughout my academic career.

Table of Contents

List of Figures

List of Abbreviations

AH	Audit Helper
HIPAA	Health Insurance Portability and Accountability Act.
LDA	Log-consistent DBMS Architecture.
SOX	Sarbanes-Oxley Act.
TLOW	Transaction Log On WORM.
WORM	Write-Once-Read-Many.

List of Symbols

H	Additive (incremental) Hash Function.
D_c	Current Database Snapshot.
D_o	Old (previous) Database Snapshot.
$\mathcal{H}$	Hashed Transaction Log (output of Audit Helper).
r	Regret Interval.
$\mathcal{L}$	Transaction Log.

Chapter 1

Introduction

1.1 Background

Regulatory Compliance in current industries is no longer just a "recent phenomenon" but a mandatory requirement in every day business. Advances in information technology in the past decade has had a profound impact on industrial enterprises' internal infrastructure and their business, by making it easier than ever to process, share, and store increasing amount of information. Such expansion of scale and capability of information systems and its management have naturally driven the need for security measures to ensure that its impact is contained. Security and risk managements of systems from unauthorized accesses and malicious attacks from outside entities, including hackers, corporate spies or even natural disasters, have been an integral part of its evolution. However, concerns for insider attacks have been relatively recently brought into headlights following a number of major corporate and accounting scandals, for example, those involving Enron and WorldCom. A lot of existing security measures such as access control and intrusion detection are often ineffective against insider attacks because by definition, insider is someone who has been entrusted with authorized access to the network, and also may have knowledge of the network architecture [2]. This implies that defense against insider attacks is partly of administrative issues, requiring policies and regulations to set the guidelines of what is socially acceptable use of the systems privileges the insiders have. As a result, US Alone has seen more than 10,000 state and federal regulations mandating secure management of business records [1, p. 358]. These regulations cover wide varieties of industries, and some of the well known ones include:

- Sarbanes-Oxley Act (SOX)

  Enacted in 2002, SOX requires financial and accounting transparency from all publicly listed companies, subject to independent audits.

- Health Insurance Portability and Accountability Act (HIPAA)

  Enacted in 1996, and governs fair health insurance coverage and a national standard for electronic

health and/or medical data storage and transactions.

- SEC (Securities and Exchange Commission) Rule 17a-4

  Requires brokerage and trading companies dealing with financial securities (e.g. bonds, stocks) to provide data retention, indexing and accessibility for their business records and transactions.

- Gramm-Leach Bliley Act

  Enacted in 1999 and requires financial institutions to have protection policy for data integrity and security.

These regulations govern different industries and enforce different rules, but commonly impose heavy financial penalties or criminal charges on companies that fail to comply. For example, in 2005, Deloitte & Touche was fined $50 million when the firm failed to detect accounting fraud at Adelphia Communications Corp., which is the largest penalty the SEC had ever levied against an accounting firm [11].

In their pure form, these industrial regulations are almost always technology-free, meaning that there are no specific technological implications in implementing them. However, the fundamental requirements of a compliant information system features can be generalized as follows:

1. Tamper-proof (long term) data retention

   Secure and trust-worthy storage of data in such a way that it is not subject to both outsider and insider attacks. Retention period differs between regulations and data, and willful destruction or tampering can result in criminal charges.

2. Auditable trail of data

   Recording any access and changes (legitimate or otherwise) to the stored information.

   Tamper proofing may be partially achieved by stringent auditing of accesses/changes, which will act as a deterrent for any tampering attempts. Sanitization of the audit record/logs is also an important requirement for protecting privacy of users, as these records are likely to be viewed by third party auditors.

3. Querying of data across versions

   Ability to search and query the retained data is of vital operation for auditors. In enabling this functionality, it is important to ensure that it cannot be abused in a way that allows adversaries to hide or alter the historic view of the data (e.g. index manipulation).

4. Secure deletion with support for litigation holds

   Guaranteed deletion beyond recovery is required for some regulations, after the predefined retention period is expired. However, if a piece of data is subpoenaed in court, it must be retained regardless of the expiration.

These requirements are derived from the common data management and provability obligations underlying many of the aforementioned regulations. For more detailed list, please refer to [3].

1.2 Write-Once-Read-Many Storage Servers

Perhaps the most prevalent technology currently available for business organizations' compliance solution is the Write-Once-Read-Many (WORM) storage devices. However, WORM is not a new technology that was developed solely for compliance solutions. WORM devices were developed in the late 1970s mainly for the archival purposes. WORM technology allows data to be written to the device once and are intentionally not rewritable to prevent accidental deletion of data, allowing data retention in general, which in turn provides a suitable foundation for compliance solutions.

The common CD-R and DVD-R are technically WORM storage media, but in the context of regulatory compliance, the WORM server refers to a specialized storage system that provides data retention that is immune to deletion or modification even by a superuser of the system. Most of currently available WORM storage servers, as marketed by many vendors including IBM and HP [5, 6], support immutable data retention at file level, making it not directly applicable for the majority of enterprises that have their systems backed up by databases. It is possible that such WORM servers be used as-is to provide compliance solutions, by utilizing them as a backup storage to which the snapshot of the organizations database are dumped at regular basis. However, this is inefficient use of storage space and the time to make a copy of the database snapshot, because both increase exponentially as the size of the database increases. Because of these costs, it is only feasible to create and store database snapshots at relatively regular time interval, say daily or weekly, and this opens up the window of time when a transaction commits data to the database but not yet copied to the WORM server, during which the data could be tampered with. A naive solution to this, as discussed in [8], is to either store each tuple and subsequent versions of it in a separate file on WORM, or to store new version of database snapshot after each transaction commits. However, the transaction overhead of these solutions will be too expensive to warrant normal database operations, let alone tamper proofing the data. Also, unless the database is locked while the database snapshot is copied, thus suspending the

normal usage, an adversary could still tamper with the data.

It is clear that WORM server cannot provide compliance solution by naively adopting it as it is or as a replacement for the database storage server. It is vital that the solution is practically applicable in the real industry settings, taking into account that from the organization's point of view, spending funds to become compliant should be justified by added business values. Naive solutions as above where operational costs more than offset the benefit of achieving compliance would only increase organization's resistance and deferment to adopting the compliance solution. Researchers have proposed compliant DBMS with tuple-immutability leveraging on the capabilities of WORM servers [8, 4], introducing more viable compliance solutions.

1.3 Contributions

This work is an extension of the Transaction Log on WORM (TLOW), the compliant relational database architecture introduced in [4]. The main contribution of this thesis is in improving the security and efficiency of term-immutable data retention and audit process of transactional history in TLOW. More specifically, this work focuses on the integration of the Audit Helper (AH) module into the DBMS kernel to take advantage of the reliability and security provided by the DBMS, and hence minimizing the vulnerability of isolated AH module and its interaction with the DBMS. An underlying assumption is that commercially available databases are secure; this assumption is based on the fact that DBMS is a tried-and-true technology where security of data stored is one of the fundamental requirement. Another supporting argument is that the commercial DBMS' source code is rarely available to public, and this provides some level of protection from hackers.

As part of integrating AH module into DBMS kernel, the following contributions are made:

- Examine threats of compromised external AH.
- Justify the changes to the DBMS kernel by analyzing the benefits of kernel-AH module.
- Discuss design goals of Kernel-AH, including improving security, reliability, performance and ease of use.
- Provide experimental results of Kernel-AH against TPC-C benchmark, and compare the performance and efficiency of the new system.

The rest of this thesis is organized as follows: Chapter 2 discusses the Transaction Log on WORM (TLOW) architecture which this thesis aims to improve, and in Chapter 3 the motivations for this thesis including threats of current TLOW architecture are outlined. Chapter 4 evaluates the implementation and experimentation of the new improvements proposed, followed by the conclusion in Chapter 5.

Chapter 2

Transaction Log on WORM Architecture

This work extends on a previous work by Hasan et al. called TLOW, which is a compliant relational DBMS architecture. TLOW leverages on the WORM storage server's tamperproof guarantee at file level to support term-immutable tuples, drawing out a more feasible compliance solution for most of the enterprises in the industry.

Cost of compliance is one of the biggest obstacle for companies becoming compliant with policies and regulations in the respective industries, and is often more severely encountered by smaller companies. In an EIU survey, more than 40 percent of IT managers in the US and Asia Pacific and 35 percent of EMEA respondents cited budgetary constraints as one of the biggest obstacles in achieving compliance objectives [7]. Therefore it is important that the compliance solution is easily adoptable and inexpensive, such that it ultimately adds business value to justify the cost incurred in the process, such as the cost of system migration and training. The benefit of making compliance easily achievable by small or large corporates translates to social benefit. Given that the relational database is still the mainstream technology of choice for supporting information systems, TLOW introduces a compliant relational DBMS architecture that is a step further in the direction of practical compliance solution for the real industry.

2.1 Threat Model

2.1.1 Threat Parameters

TLOW describes two parameters for its main threat model, as shown in figure 2.1. These parameters represent the time windows of attack by adversaries, in that minimizing them will decrease the chances of attack.

The *regret interval* refers to the minimum time that can be assumed between a successful commit of a tuple and when an adversary tries to tamper with it. For example, daily backup of database snapshots in WORM, which is a common practice in the industry, has a regret interval of a day. Email compliance under

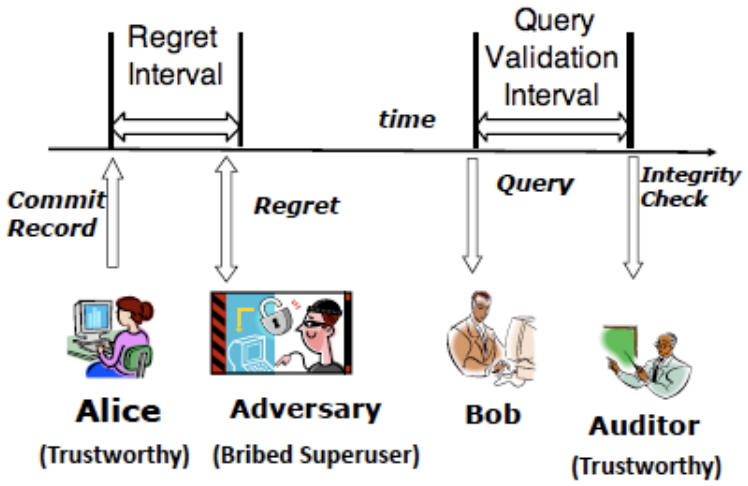

Figure 2.1: Threat Parameters

SEC enforces zero regret interval, meaning that the email must be archived on WORM server before it is forwarded to the recipient. The *query validation interval* is the time between when a user issues a query and when the auditor validates the query result. This interval represents the period during which an adversary may tamper with the data and then revert the modifications. The result of this is that the affect of the tampering can persist and possibly propagate in the database without the trace of the original tampering being detected by an audit process. For example, an adversary could double the net income just in time for an annual report generation, and revert the changes after the report is filed. This type of tamper-untamper attack is called "State Reversion Attack". Because the auditing process for a company is often disruptive to its normal day-to-day operations including system and data usage, in reality many industries mandate a relatively infrequent, say yearly, audits. This opens up the vulnerability towards the state reversion attack. TLOW counters this problem by adopting probabilistic detection of state reversion attack by more frequent and faster internal audits. The Audit Helper (AH) module of TLOW is responsible for this task, which is discussed further later in this thesis.

2.1.2 Regret-Based Threat Model

TLOW focuses on regret-based threat model for transaction-time databases [4]. This threat model reflects the malicious activities that regulations such as SOX and SEC Rule 17a-4 are intended to prevent, and it assumes that the procedures and systems external to database are not compromised, including DBMS. That is, when the data is initially written to the database it is correct and authoritative, as it is assumed that the data is generated and captured according to strict internal policies and processed by secure software.

This rules out the possibility of the adversary abusing the policy or attacking the software; while these are all valid attack scenarios, database normally has no control outside of itself, and therefore it makes sense to limit the threat model and doing so does not affect the correctness of the TLOW architecture.
Regret-based threats are concerned with insider attacks, where authorized users possibly with super user access tries to tamper with the history of stored data, by update, insert or delete record with past datetime stamp. For example, a mal-intentioned financial executive might bribe the DBA to delete records for liabilities and change or insert new backdated records inflating the company's income and cash flow so that the balance sheets only show favorable performance, deceiving shareholders and investors. This example is similar to the Enron scandal in 2001 [10].

Given that the external system is assumed to be "healthy", i.e. not hacked, an adversary can forge history by changing the database file or log files, using text editor or a separate, non-compliant DBMS. Or with a super user access, the adversary can also read or write any internal files through DBMS within the limits allowed by the DBMS and the server hosting it. TLOW architecture makes the following premises for possible attack scenarios:

- DBMS and storage server prevent tampering attempts on the transaction log while DBMS is up. That is, only the DBMS is able to write to the transaction log while DBMS is up and running (transaction integrity).
- Adversary cannot tamper with data while it resides in the DBMS page cache, such that when a new transaction arrives at the DBMS, the DBMS correctly executes it.
- DBMS has nonzero regret interval.
- There are only negligible delays in communication between the DBMS and storage servers and in performing simple writes to storage, such as the appending of a new blog to a log.
- Auditor has a trustworthy source of information regarding the time of each crash or shutdown and system reboot at the WORM server time, occurred since the last audit.
- WORM storage server operates properly, including the anti-tampering provisions of WORM server clock.
- DBMS and WROM storage server clocks are roughly synchronized, with limit of within $r/2$ time unit difference at all times, where r is the length of the regret interval.

History forgery is certainly not the only possible form of attack that a compliant system is concerned with. Regulatory compliance is a complex issue with administrational requirements such as policies and code of conduct, as well as system requirements that extends the conventional definition of a secure system. For TLOW architecture to be fully compliant, above assumptions must be supported by the means of other security controls and techniques, for example access control, database backup and crash recovery, which are normally expected features of any commercial DBMS.

2.2 Architecture

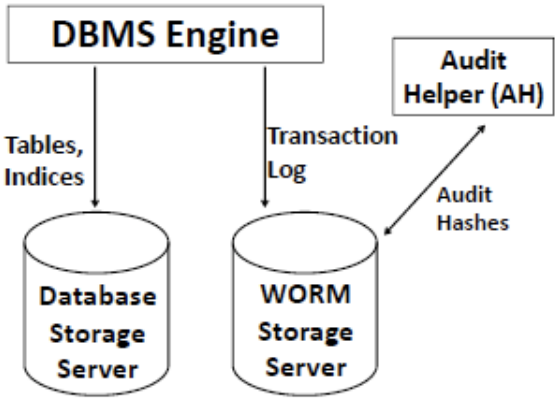

Figure 2.2: TLOW Architecture

TLOW architecture employs transaction-time database, where every insertion, modification or deletion of records inserts a new version of tuple into the database with updated *start time* (or equivalently, *end time*) timestamp that marks the transaction commit time of the action involving the tuple. For deletion, a special end-of-life tuple is inserted as a "flag", marking it as deleted. TLOW utilizes transaction-time databases in order to retain all versions of tuples, using historic versions of tuples to verify if a modification was made by legitimate system-database interactions or by data tampering attempts. The transaction-time DBMS can be provided by a software layer encapsulating an ordinary DBMS, and is compatible with legacy applications that only interacts with the most recent tuple versions through transparent query modifications, performed by transaction-time software layer. This adaptable solution for transaction-time database is consistent with low cost of compliance imposed by TLOW architecture.

The name TLOW, transaction log on WORM, is quite self-explanatory of its overall architecture; TLOW stores the current database instance on ordinary storage, and the transaction log $\mathcal{L}$ and the snapshot of the

database contents as of the last audit on WORM storage, as shown in figure 2.2. The DBMS engine must perform transaction log flush to WORM and new log file creation every $r/2$ time unit in order to detect clock tampering attacks on DBMS.

Auditing process then uses the data that are stored in WORM, which provides the necessary assurance regarding the integrity of the data and thus trustworthy auditing given that WORM is used as trusted computing base in the overall TLOW architecture.

2.2.1 Audit Process

TLOW uses cryptographically strong incremental hash function H to support more efficient audit process. The hash function H has following characteristics:

- H operates on a set $\{a_1, ..., a_n\}$, that is, duplicate elements are ignored and the order of elements does not affect the result of hash.
- As an additive hash function, it is simple to compute $H(\{a_1, ..., a_n\})$ given $H(\{a_1, ..., a_{n-1}\})$ and a_n. Intuitively, this is equivalent to $H(\{a_1, ..., a_n\}) = H(\{a_1, ..., a_{n-1}\}) + H(\{a_n\})$.
- H is preimage resistant such that given h, it is hard to find $(\{a_1, ..., a_n\})$ such that $h = H(\{a_1, ..., a_n\})$ (i.e. H is one-way function), and given one set $\{a_1, ..., a_n\}$, it is hard to find a different set $\{a'_1, ..., a'_n\}$ $(\neq \{a_1, ..., a_n\})$ that computes to the same h, i.e. $H(\{a_1, ..., a_n\}) = H(\{a'_1, ..., a'_n\})$.

The audit process by TLOW is shown in figure 2.3. An auditor stores a snapshot of database state and its cryptographic hash to the WORM Storage server at the end of each audit. These are signed by the auditor to guarantee the authenticity and integrity of data. Then during the next audit process, the auditor checks for the signature and then generates hashes from the transaction log $\mathcal{L}$ and the current database snapshot. Because the hash function H is additive (incremental), if this audit is to succeed, the sum of the hash of previous database snapshot stored in WORM and the generated hashes of $\mathcal{L}$'s new tuple entries should be identical to the hash of the current database snapshot. TLOW identifies this property as the *tuple completeness condition*, represented with the following notation:

$$H(D_c) = H(D_o) \cup H(\mathcal{L})$$

where D_c refers to the current database snapshot, D_o the old (previous) database snapshot after the last audit.

Intuitively, this means that the auditing cost is dependent on the size of current database snapshot D_c and the transaction log $\mathcal{L}$ which is only limited by the size of the DBMS and disk on which the $\mathcal{L}$ is stored,

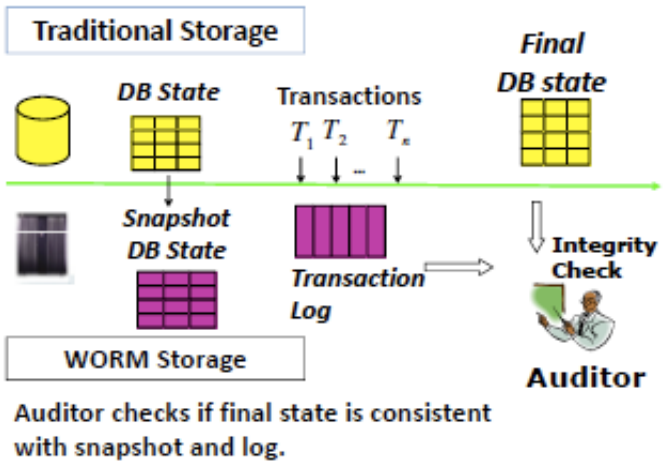

Figure 2.3: TLOW Audit Process

potentially making the audit process too inefficient. Given that this auditing process needs to take place while the ordinary business and operations are put on hold, slow auditing process is prohibitively costly for the organizations. TLOW employs several techniques to speed up the auditing process; The Audit Helper is an automated real-time auditor introduced by TLOW, which will discussed in the following section. Another audit performance optimization TLOW uses is to effectively reduce the size of database. This cost is due to the need to compute $H(D_c)$ at every audit. Because TLOW uses transaction-time database, the size of the database, and thus the time cost of scanning the database, steadily grows over time. By migrating old versions of tuples to WORM and using time-split B^+ tree, both audit processing and normal transaction processing can be optimized without losing the ability to retain historic data.

It should be noted that the tuple completeness checking is not the only part of the auditing process. For example, the auditor must make sure that the index files are not tampered with, which may be modified or deleted to enable other data tampering attempts to be hidden in the database. Detailed description of other audit tasks are explained in [4].

2.3 Audit Helper

Audit Helper (AH) in fact serves two purposes within TLOW architecture as described above. Its primary function is to perform real-time incremental auditing, allowing the time cost of audit process to be spread out during the normal database operations between audits rather than incurring the total cost at once, during every audit. This is achieved by the AH module reading new entries in $\mathcal{L}$ as soon as it is written, and

computing the hash of new tuples found in the log and outputting the result to a file $\mathcal{H}$ on WORM. This is the same tasks performed by auditors during the audit process, only difference is that AH is automating the process in real-time. There is a time restriction that the hash needs to reach WORM within $r/2$ time units, in order to thwart the adversary's attempts to tamper with data. Also, in order to assure the authenticity of AH and prevent the adversary from tampering AH to breach the security of TLOW, AH signs its $\mathcal{H}$ with a key that the auditor knows, but adversary cannot obtain.

During audit time, the auditor cross checks $\mathcal{L}$ and $\mathcal{H}$, both sanity checking and to make sure that each entry in $\mathcal{H}$ has corresponding legitimate COMMIT record (i.e. not backdated entry by the adversary) in $\mathcal{L}$, and it is within $r/2$ time units of each other. If these checks pass, auditor needs to compute the hash of current database snapshot and test if that equals the sum of the $\mathcal{L}$ hash and the hash of previous database snapshot. On the other hand, if either the sanity check fails or a non-trustworthy entry is found in $\mathcal{L}$, for example, untimely COMMIT timestamp, then the auditor has to manually carry out the audit process disregarding AH and $\mathcal{H}$.

AH also provides probabilistic detection of the state reversion attack. While perfect detection is possible with a technique called *hash-page-on-read* [Mitra?], authors of TLOW reason that the cost of perfect detection is likely to offset the benefit gained; this is acceptable given that the state reversion attack is by no means the most prevalent form of data tampering. Another point is that state reversion attack usually requires at least some duration of time for the tampered data to persist in the database, in order to be effective. A quick tamper-untamper attack could generate evident anomaly that is easier to detect than a longer tamper-untamper attack, which could generate a more trend-like changes in the record. Obviously, if a state reversal attack succeeds within the timeframe of subsequent audits, it will not be detected by the tuple completeness check and its effect will go undetected. By having the AH perform real-time incremental auditing, the timeframe for the attack is dramatically reduced to subsequent transactions, with upper limit of $r/2$ time units.

There is clearly a tradeoff between perfect detection and the cost of doing so, which should be left for policy makers to define the priority of, as the policy and regulation define compliance.

Current implementation of AH in TLOW architecture is as a stand alone module that is not programatically integrated into the overall architecture. AH can be hosted on the DBMS server or on its own server, with the restriction that the host server clock must be synchronized with WORM server clock within $r/2$ time units. While this separation may provide system portability and modularity between the normal DBMS

operations and audit operations, compliance and audit support are the main focus of TLOW architecture; this suggests that prioritizing security and trustworthy of audit support is more important than preserving the side effect of implementation design. These observations are the basis of the design decision to integrate the AH module fully into the DBMS kernel (kernel-AH), and the next section discusses in further detail the motivations and expected benefits of the kernel-AH.

Chapter 3

Motivation for Kernel-AH

This section lays the groundwork for Kernel-AH, discussing motivations for porting AH module into DBMS kernel, design goals and the tradeoff between expected benefits and costs.

The underlying assumption of this thesis is that the commercial DBMS has built-in security measures, for both the system and its data, that are assurable - at least to a certain level that enables the "clients" of the DBMS to entrust data storage and management. Porting AH module into the DBMS kernel allows AH to leverage on this accepted security guarantees, and to therefore improve the overall security of TLOW architecture.

The isolation of AH module from DBMS in the current implementation of TLOW is the main area of focus through which to achieve architectural enhancement. Given that the commercial DBMS is securer than an external plug-in module, communication within the DBMS should be more protected compared to that between the DBMS and the plug-in where the communication link is outside the control either parts. The goal is to manage all modules within a single system in its entirety.

Another assumption is that it is very unlikely that the source code of the commercial system to be available to public, adding to the security of the system from external hackers trying to exploit certain features or possible bugs to attack it or the stored data.

3.1 Threats of Tampered AH (TAH)

[4] outlines how TAH may allow backdated tuple insertion. An adversary first tampers with the current instance of database to include a pre-configured backdated tuple t', and then replaces the AH with a tampered AH (TAH). TAH is modified so that when the DBMS commits a transaction T while the TAH is running, TAH hashes the new tuples in T, say t_T, plus t' and outputs it to $\mathcal{H}$. Then the original AH may be restored to carry out auditing as usual. Because the sum of hashes in $\mathcal{H}$ equals $H(D_c)$, the auditor will not detect

the false data t. This shows that if the database and TAH reflect the same data tampering in a coordinated manner, masking the falsely inserted tuple is possible. Backdated modification or deletion is considerably more difficult, which invloves removing the hash of original tuple t and adding the hash of backdated tuple t' and write to $\mathcal{H}$ after the transaction T commits:

$$H(t_T - t + t') \Rightarrow \mathcal{H}$$

Note that because the hash function is strictly one-way, above computation is assumed to be impossible.
Adversary can also attack TLOW by disrupting the AH functionalities, increasing the query validation interval and effectively allowing more time to plant a state reversion attack. This is because without AH's frequent audits, tuple completeness check cannot be done to detect state reversion attack that is underway. A possible attack scenario is that the adversary may replace AH with TAH which adds arbitrary values to $H(D_c)$ so that $\mathcal{H}$ no longer mirrors the state of database and obscure what the "correct" state of the database is. This particular attack will be deteced in the next internal audit through the tuple completeness check, but if the adversary's goal is to simply disrupt the system, in the similar way was the DOS attack, then the attack succeeds, as forensic analysis must take place to determine the health of the system

Researchers of TLOW suggest that these attacks can be thwarted by requiring AH to digitally sign its $\mathcal{H}$ using a key that the adversary cannot easily obtain, with inexpensive TPM to safely maintain a key seed that is known to the auditor to generate the hash key for auditing. Then AH can use a cryptographic hash function h_k with key k, to compute the hash $h_k(i)$ where i is the new tuple found in the log, and appends this to $\mathcal{H}$. Also, apart from keyed tuple hashes, AH needs to be able to prove itself as a legitimate copy to the auditors, for example using a certified code.
These cryptographic techniques and access controls are industry standards and therefore provide viable solutions for protecting AH, but such supplemental security measures add to the complexity of the overall architecture and the cost of system management. We believe the same benefits of TLOW architecture can be achieved with kernel-AH, where AH functionalities are hosted and performed as a part of DBMS kernel operations rather than a stand-alone module. The following section describes design goals guiding the implementation of kernel-AH, focusing on preserving both DBMS and AH functionalities while minimizing disruption to the kernel.

3.2 Design Goals

3.2.1 Security

Major threats to AH and consequently to TLOW architecture are explained in the previous section. Using the existing security software and hardware components can provide AH the same level of security guarantee that is asserted by TLOW, albeit not bulletproof; for example, TPM attack is possible [**?**]. Based on this consideration, the biggest damage the adversary can cause is nullifying AH module, by attacking the correctness of audit and disabling frequent internal audits.

Kernel-AH has benefit of escalating threats and attacks to AH to be handled by the DBMS, which is trusted to have securer protection. For example, any attacks to replace or kill the kernel-AH must go through the security walls of DBMS. This is considerably more difficult given that any form of attack is likely to require rebooting the DBMS which is very detectable either by an audit or by system users.

An important consideration in implementing and supporting AH functionality in the kernel is that it must not introduce new security holes in the DBMS so that the trustworthiness of DBMS is not compromised. Additionally, the DBMS users should be able to configure and use kernel-AH through a narrow API that allows access to full functionality of AH while limiting possible abuse of the system. Given that AH is originally designed to be an automatic internal audit processor that does not require much user control, such implementation is logically possible.

A side-effect of kernel-AH is that there is no need to separately authenticate AH to auditor with cryptographic hash function to write to $\mathcal{H}$ or TPM to manage keys, and having less sensitive information to manage generally means less threats that can be exploited by adversaries.

3.2.2 Reliability

The first and foremost reliability concern follows from the security of modifying the database kernel. The kernel modification must ensure that security of DBMS is not affected, as well as that the existing DBMS functionalities are unchanged, such that the DBMS remains reliable. Note, however, that the purpose of Audit Helper is to additionally store audit information based on the data stored in the database, without changing the new or existing data. With AH only carrying out "read-only" operations within the database kernel, there is no conflict between kernel and AH operations and therefore kernel-AH implementation with no database function change is possible.

The original implementation of AH allows transparency between AH and DBMS in regards to crashes in either modules, that both AH and DBMS can withstand and perform as normally when the other module

crashes and restarts. This is consistent with having minimal effect to the normal DBMS functionalities and the kernel-AH implementation should maintain this property.

3.2.3 Performance

One of the biggest motivation behind kernel-AH is in improving the efficiency of AH operations. In the current TLOW implementation, AH scans transaction log $\mathcal{L}$ to locate and re-parse a new tuple record before it hashes the tuple to write $\mathcal{H}$. In effect, AH continuously polls the database log file(s) in a predefined time interval to detect new tuples in log entries. Better efficiency can be expected from kernel-AH by removing this costly scanning and re-parsing of log entries, as the kernel-AH module can have direct access to the tuple objects after the tuple is written to both the data store and to the transaction log by the DBMS kernel. Another aspect of performance is latency. In the current implementation, by having the AH as a separate operation from the normal database transaction processing, AH does not interfere with the database reads and writes and thus there is no impact on the transaction latency of DBMS. Ideally, the kernel-AH should maintain this property in order to lessen user resistance because system users are likely to relate the visible behaviors of the system, and their direct experiences, to system usability. This is heavily dependent on the implementation of kernel-AH, but an observation is that because AH operations are never in conflict with database operations, similar performance may be achievable with kernel-AH by multithreaded DBMS processes. With a single-threaded implementation, kernel-AH is likely to have higher run time overhead compared to the original implementation.

3.2.4 Easier Compliance

The ultimate goal of incorporating AH module into DBMS kernel is to fully support TLOW architecture in a single system. This simplifies configuration and management of the system, compared to separately installing and looking after multiple modules to provide full functionality of the system.

AH plays relatively auxiliary role in providing regulatory compliance in the overall TLOW architecture. The two main functionalities that AH perform, which are improving the speed of the audits and providing probabilistic detection of state reversion attacks, are either already provided by basic TLOW design, or can be replaced with an alternative technique. For the audits, the auditor can perform tuple completeness check on the entire $\mathcal{L}$ since the last audit without the help of AH which results in a considerably longer audits, and for the detection of state reversion attacks, hash-page-on-read technique can be used for complete detection at the cost of transaction performance overhead. However, these incur prohibitively expensive costs to the organizations both in terms of time and resources, for example having to suspend the normal business usage

of the database for audits. Arguably, this type of business cost determines how practical it is to adopt the compliance system such as TLOW. This shows that promoting better protection and integration of AH in the TLOW architecture has significant business value as well as the systematic benefits describe above. This is very important as regulatory compliance is more relevant to administrative decision than technical one.

Chapter 4

Empirical Evaluation

Kernel-AH has been implemented as a proof of concept system on Berkeley DB version 4.7.25, following the design goals described in the previous section. The main implementation task was to reorganize, but not to extend the TLOW architecture. That is, the implementation of the TLOW with kernel-AH is based on the original TLOW system, without any changes to the functionalities supported.
This section describes some of the the implementation decisions, analyses the performance of kernel-AH and possible future improvements.

4.1 Implementation

There are some notable implementation decisions in porting AH into the DBMS kernel. Following the security considerations above, users of the DBMS are provided with narrow API that allows them to configure kernel-AH in a restricted way. Specifically, the extent of user interaction required for kernel-AH is limited to setting or removing the compliance log ($\mathcal{H}$) directory through `DB_ENV.set_compliance_dir` function to activate or deactivate the AH module within the DBMS kernel, respectively. These additional configuration functions are provided in a consistent fashion as the existing DMBS-provided ones, such as setting the database file directory and log directory, with the goal of supporting seamless API. Without the compliance log directory set (i.e. set to `NULL`), DBMS operates as per normal.

The actual AH functionalities, which are hashing of new tuples and writing to $\mathcal{H}$, are performed following the normal transaction log ($\mathcal{L}$) writes after each transaction commits. The current implementation executes these operations serially. Further performance gain may be possible through multithreaded implementation, but proving that AH can be ported into the DBMS kernel with the least disruption was deemed to be of higher priority.
The difference between the external AH and kernel-AH is the time of their operation. External AH hashes

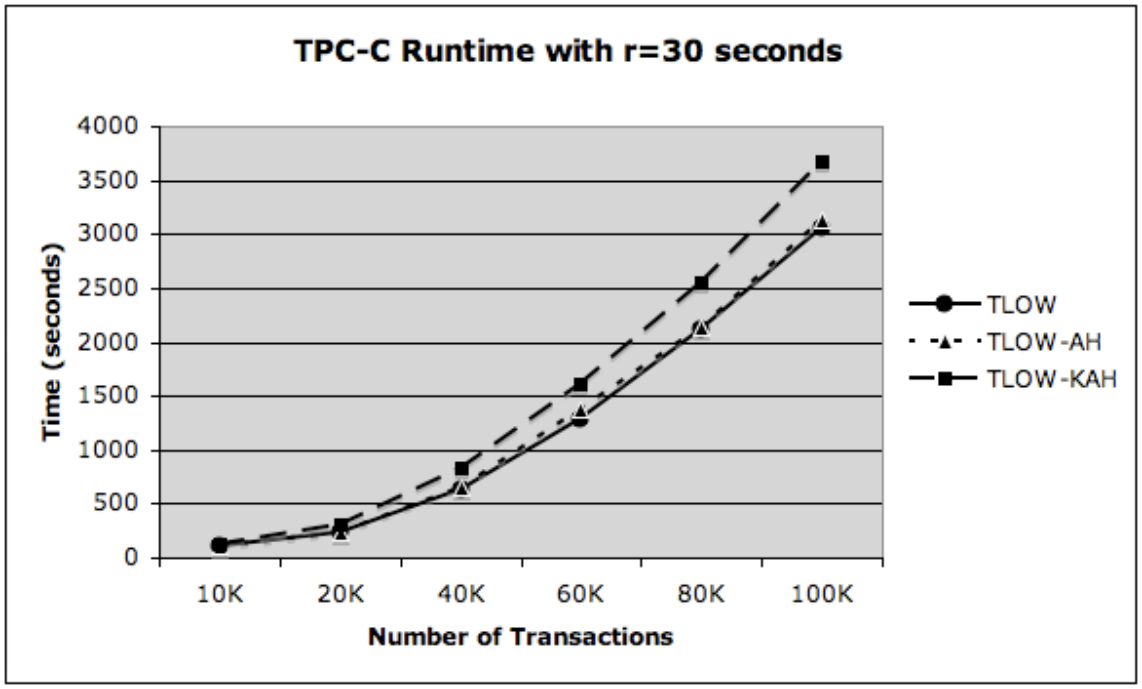

Figure 4.1: TPC-C Benchmark Results

newly inserted tuples in bulk when it detects a new log file and flushes the results to $\mathcal{H}$ file on WORM every $r/2$ seconds. On the other hand, kernel-AH processes individual tuple as soon as it is written to the transaction log file $\mathcal{L}$.

4.2 Experimentation

Test environment for kernel-AH is consistent with that of TLOW evaluation [4]. The Shore implementation of TPC-C benchmark was used to evaluate the performance of TLOW with kernel-AH. The DBMS was hosted in a machine with a Pentium dual core 2.8 GHz processor, 512KB L2 cache, 4GB RAM, and a 1TB hard disk. WORM server was simulated by a Pentium 2.8 GHz single core processor, 512KB L2 cache, with a portion of its local disks exported as an NFS volume. The DBMS mounted the WORM volume over NFS, and stored the logs there. Audit Helper's output, $\mathcal{H}$, is also stored in the simulated WORM server.

TPC-C performance was measured with regret intervals r of 30 seconds for TLOW with and without external AH (TLOW-AH), and TLOW with kernel-AH (TLOW-KAH). The resulting TPC-C run times are shown in figure 4.1.

Overall, TLOW-KAH adds 21% overhead on average, with lower overhead for larger transaction test sets.

Given the simpleness of current kernel-AH implementation where new tuples in the transaction log $\mathcal{L}$ entries are hashed and written to $\mathcal{H}$ sequentially, this performance result is arguably the upper bound of kernel-AH performance. In contrast, TLOW and TLOW-AH share latency within 3% of each other because the external AH of TLOW-AH operates concurrently with the normal database operations. Possible improvements for TLOW-KAH performance through more sophisticated implementation is discussed in the next section.

It should be noted that audit processing was not a part of TLOW-KAH experimentation, as there are no changes to the format or information written to the $\mathcal{H}$ file or the process of auditing itself. TLOW-KAH in fact generates larger $\mathcal{H}$ file due to its higher frequency of hash calculation after every transaction commit, compared to bulk hashing of new tuples in regular interval by TLOW-AH. Supposing that the incremental hashing function H used in audit process has no logical limit of the number of elements in the set on which it operates, larger $\mathcal{H}$ file should not affect audit time too grievously.

4.3 Future Work

As the TPC-C benchmark result shows, the performance of TLOW with kernel-AH is considerably slower than the original implementation of external AH module in TLOW-AH architecture. This is an expected behavior given that TLOW-KAH serially writes to $\mathcal{H}$ compared to the external AH module that writes to it in parallel to other DBMS operations. This suggests that current implementation of kernel-AH can benefit from parallelizing its process through multi-threaded implementation, significantly improving its performance. Because the incremental hash function H for auditing process safely ignores the order of elements in calculating the hash, synchronization of $\mathcal{H}$ writes in transaction order is not vital in its multithreaded implementation. However, keeping the information of correct transaction order may provide additional audit function for DBMS activities.

Another possible enhancement for TLOW-KAH is to allow new tuple hashes to be internally buffered before it is written to the $\mathcal{H}$ file, so that less number of I/O is required. This requires more intricate design to handle rollback and recovery scenarios to ensure the buffered hash sums are not lost in the event of possible DBMS crashes. Assuming than the hash sums will be calculated for the log entries that have been committed and flushed to the transaction log $\mathcal{L}$, if DBMS crashes after the $\mathcal{L}$ is written but with intermediate hash sum still buffered in memory, $\mathcal{H}$ and $\mathcal{L}$ will no longer mirror the same state unless the buffered hash set is

recovered. Note that because information in $\mathcal{H}$ depends on $\mathcal{L}$, such recovery actions will have no interference with that of $\mathcal{L}$ or DBMS itself.

Chapter 5

Conclusion

This thesis reviewed the TLOW architecture, that supports long-term immutability for relational tuples, providing a realistically feasible compliance solution for enterprise systems that are backed by RDBMS. The AH module of TLOW is a vital component that allows for the practicality of the solution. AH speeds up the periodic audit process that many of the current industrial regulations mandate, thus minimizing the system downtime and the business cost incurred by the downtime.

This thesis presented an extension to the current TLOW with AH architecture, that provides a better protection of the AH functionalities. The original AH is implemented as a stand-alone module that is external to the database kernel, requiring a separate security measures to protect the module and its interaction with the DBMS. For example, an adversary may destroy the AH module completely and remove its functionality from TLOW architecture, or may replace the module with a tampered AH that will hide a backdated tuple insertion in the database by reflecting the same modification in the hash generation. In order to prevent these attack scenarios, extra authentication steps to verify the version of AH module to the auditor using a key that only the auditor knows. What results is a disjoint system that requires more effort in its maintenance and operation, as well as their associated costs. The TLOW-KAH introduced in this thesis aims to improve these drawbacks by porting the AH module into the database kernel, whose functionality can be turned on or off by database users through a simple API. By supporting complete features of TLOW in a single DBMS module, it eliminates the need for any additional security measures or system maintenance, as the AH module can benefit from the protection provided by the DBMS itself.

The current implementation of TLOW-KAH serially handles AH operations following every transaction commit log writes, which inevitably adds to the operational latency compared to TLOW-AH, on average 21% across 10k to 100k transactions measurements, as shown in the experimental results. Given that AH functionality is never in conflict with normal database operations, it is expected that it can be implemented in a multithreaded processes which will lessen the latency considerably. Given the simpleness of current implementation, the 21% overhead is arguably the upper bound of kernel-AH performance.

Overall, TLOW-KAH implementation has proven that AH module can be incorporated into the database

kernel without affecting the normal DBMS operations, albeit notable transaction overhead due to naive implementation. While DBMS kernel change is often disputed in researches and/or system enhancements, a wholly complete single system has operational advantages which TLOW-KAH seeks to benefit from. It is believed that a more sophisticated implementation of TLOW-KAH should allow such advantages fully realized without performance depreciation.

References

[1] Gertz, M and Jajodia, S. 2007. *Handbook of Database Security: Applications and Trends.* Springer.

[2] *Webopedia*. http://www.webopedia.com/TERM/I/insider_attack.html

[3] Hasan, R. 2009. *Trustworthy History and Provenance for Files and Databases.* University of Illinois at Urbana-Champaign PhD Dissertation

[4] Hasan, R., Winslett, M. and Mitra, S. 2009. *Efficient Audit-based Compliance for Relational Data Retention.* UIUC Dept. of CS Tech Report UIUCDCS-R-2009-3044.

[5] *IBM System Storage N series with SnapLock Compliance and SnapLock Enterprise.* http://www-03.ibm.com/systems/storage/network/software/snaplock/

[6] *WORM Data Protection Solutions.* http://h18000.www1.hp.com/products/storageworks/wormdps/index.html

[7] *Whitepaper: Compliance : creating business value* www.atosorigin.com/NR/rdonlyres/...A9CF.../wp_risk_management.pdf

[8] Mitra, S., Winslett, M., Snodgras R., Yaduvanshi, S., Ambokar, S. 2009. *An Architecture for Regulatory Compliant Database Management.* ICDE.

[9] *TPM Reset Attack* http://www.cs.dartmouth.edu/ pkilab/sparks/

[10] *Enron Scandal* http://en.wikipedia.org/wiki/Enron_scandal

[11] *SEC Charges Deloitte & Touche for Adelphia Audit.* http://www.sec.gov/news/press/2005-65.htm