Logging Key Assurance Indicators in Business Processes*

Fabio Massacci
Dipartimento di Ingegneria e Scienza dell'Informazione
Università di Trento
Fabio.Massacci@unitn.it

Gene Tsudik
Computer Science Department
University of California, Irvine
gts@ics.uci.edu

Artsiom Yautsiukhin†
Dipartimento di Ingegneria e Scienza dell'Informazione
Università di Trento
evtiukhi@disi.unitn.it

ABSTRACT

Management of a modern enterprise is based on the assumption that executive reports of lower-layer management are faithful to what is actually happening in the field. As some well-publicised major recent disasters (such as Barings, AllFirst-Allied Irish Bank, ENRON, Societé Generale) have shown, this assumption is not well-founded. Intermediate managers can misrepresent the actual state of their systems in order to hide negative events or to "doctor" reports which have been already produced. Existing security approaches which guarantee integrity of logs and related reports do not protect the system against these threats, if they are directly applied to a multi-layered corporate structure. In this paper, we extend existing approaches by constructing a logging scheme which ensures that, at each level, logs are both correct and consistent.

Categories and Subject Descriptors

K.6.4 [**Management of computing and information Systems**]: System Management—*Management audit*; H.2.7 [**Database Management**]: Database Administration—*Logging and recovery, Security, integrity, and protection*; J.1 [**Computer Applications**]: Administrative data processing—*business*

General Terms

Management, Security

Keywords

logs, enterprise, auditing, tamper-proofed logs

*This work was supported in part by the EU-IST-FP6-IP-SERENITY and IST-FP7-IP-MASTER projects as well by the grant from the Fulbright Foundation.

†Contact author.

ASIACCS' 09, March 10-12, 2009, Sydney, NSW, Australia

1. INTRODUCTION

Traditionally, logs have been the instruments of operating system administrators. They recorded noteworthy events, such as: user activity, program execution status, system resource usage and data changes. The objective is to provide an insight into the past and current states of the system. Logs have been used to detect intrusions and user mis-behaviour. Keeping accurate system audit trails and reviewing them in a timely fashion is one of the pillars of computer security [9].

As IT started to play a major role in commercial activities, the purpose of logs and even the nature of logged data have changed. Compliance and audit requirements for IT business processes [8, 1] have shifted attention to application logs which need to provide equally detailed information for both internal and external use. With the increased complexity of organisations, using logs is sometimes the only way for deriving critical business data.

For instance, in the frantic world of a stock exchange, transaction logs are the main source of evidence for determining the value at risk in a trader's portfolio (i.e., the presence of may-go-sour deals) in order to limit its further exposure [16]. Manipulating this data may help a trader to get a larger bonus and perhaps ruin the whole company, as in the (in)famous Rusnak [16] and Kevriel [6] cases. For this reason, a number of regulations and standards (e.g., see [10]) require the high-level view of the system to be consistent with the lower-level view. Moreover, logs must guarantee some reasonable security properties.

The simplest solution is to use certified logs [5, 14]: the logging system collects all required events, stores them as records and signs the records to preserve log integrity. These low-level records contain a vast amount of information about the past behaviour of the system. A super-log of this type is hardly practical: intermediate managers typically need only a partial view of this information (e.g., monthly income, frequency of specific orders) in order to pass on a digest to higher layers (e.g., see [15, 12]). Managers often use *only* aggregated values (indicators or metrics) provided by lower managers to calculate their own *Key Indicators (KI-s)*. Thus, a lower manager can fake the data in its favour, hiding negative events (e.g., fraudulent transactions) which will remain unnoticed as long as only low-level events are logged (and not reviewed). This is precisely what happened in AllFirst and Enron cases, where the veracity of the executives' claims was not verified vis-a-vis corresponding lower-level events.

1.1 Contributions

In this paper, we investigate how log integrity can be guaranteed in a multi-layered enterprise. There are two main contributions. First, we propose a logging scheme which ensures that a middle manager does not insert fake reports. Second, we do not assume that the log controller at each layer is fully trusted from the beginning. In our model, it can start behaving maliciously at any time. The main difference between our work and prior art [5, 14, 3] is that our threat model allows the dishonest controller to start malicious observations long before performing actual malicious actions, i.e., changing logs. Since the controller has full reign over the system producing the log, malicious observations cannot be detected by the log schemes.

2. EXAMPLES AND MOTIVATION

An enterprise log system can be seen as a layered structure where lower-level logs are used to form higher-level ones. The lowest-level logs often contain information about real-world events collected by an enterprise software itself (e.g., SAP R/3) or other programs (e.g., OS-s, IDS-s, etc.). This data is then used by corresponding managers to create reports about system productivity, its security level, financial status, and so on. For one report, several logs may be needed and the same log can be used for forming different reports. Received reports are, in turn, aggregated into higher-level reports, until the highest-level – usually intended for a top manager or a client – is reached.

In the following, we use the terms 'log manager' and 'log controller' interchangeably. We also consider 'report' and 'log' to be synonymous.

In one of the most *memorable* recent cases, fraud committed by J. Rusnak in the 1997-2002 period caused losses of $691 million for Allfirst Bank [16]. It is a good example of how log forgery can lead to substantial fraud. One of the key aspects of the fraud was that J. Rusnak inserted fake deals into the bank system in order to show good daily profit-and-loss value. This value is used to check if a trader exceeded the allowed loss limit ($200 000). An important point was that, during the calculation of this value, it was not checked if deals were actually "settled" (finished). In other words, higher managers (back office) used aggregated data (accomplished deals) faked by a lower manager (J. Rusnak) for creating their KI (profit-and-loss value). Before the periodic audit – which would have revealed unsettled deals – J. Rusnak deleted all records of fake deals from the system (log tampering).

Another prominent example is the Enron bankruptcy case. (Enron was the seventh largest public corporation in the USA[12]). One of the biggest problems was caused by unfair transactions between Enron and one of its subsidiaries (LJM). This resulted in huge losses for the corporation. Such transactions were possible because everybody (even the Audit Committee):

> "...relied entirely on management representation, with no supporting documentation or independent inquiry into facts..." [12, page 32].

In other words, the Board of Directors (higher entity) used only fake management representations (reports of executive entities) without considering supporting documents, i.e., lower-level logs.

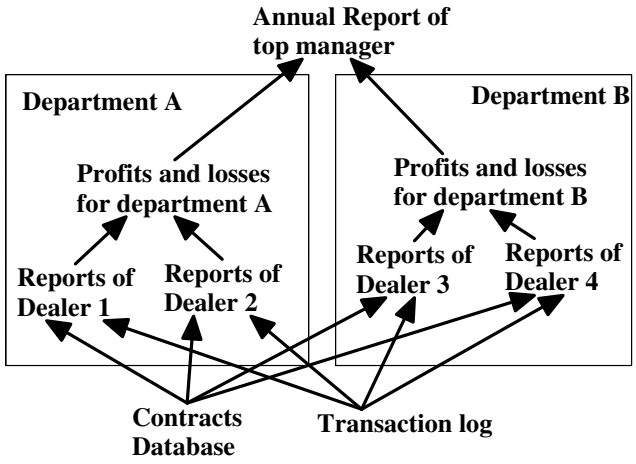

Figure 1: Sample log scheme: a bank with 2 departments and 4 dealers.

In this paper we use the following running example based on the two cases presented above in order to show how the proposed approach copes with both problems.

EXAMPLE 1. *Consider a log system in a bank. The lowest-level logs contain "hard" evidence of real-world events, such as negotiated contracts between bank's dealers and their counterparts as well as transaction records. Bank's dealers insert reports about negotiated deals into the bank computer system. The bank branch office (middle manager) uses these reports to generate daily profit-and-loss values for the department. The top manager periodically (e.g., yearly) assesses employees' and middle managers' performance using these values. A sample scenario corresponding to a bank with two departments and four dealers is shown in Figure 1.*

Our goal is to provide a multi-level log scheme such that, at each level, logs cannot be forged, i.e., neither changed nor inserted.

3. FLAT VS. LAYERED LOGS

For each level in Example 1 we can build a secure log system by using techniques from [5]. A controller at each level can be seen as untrusted computer U in [5]. Trusted computer T in our case is the log controller at the next upper level[1]. Of course, in our case, the upper-level controller is not fully trusted; it can change the value used for checking log integrity at the current level. On the other hand, it is interested in genuine low-level logs, since its reports are based on precisely this sequence of records. We also need a semi-trusted verifier V in order to check the integrity of the current log.

The scheme proposed in [5] can be applied to each pairs of managers separately. However, in order to provide a formal description of the proposed scheme, reports from different levels must be bound. Otherwise, a manager can honestly provide authentication values for lower-level logs and still generate reports that do not reflect actual events.

[1]Presence of other logs required for composition of upper-level logs is not important here.

EXAMPLE 2. *In our example, a middle manager B can report fake daily profit-and-loss values to demonstrate "excellent" management skills and to be promoted. The verifier that only checks the integrity of reports might not detect this fraud.*

Another problem with this scheme in the multi-layered structure is that the scheme of [5] has been designed to offer log integrity against an attacker who *at some point* compromises the system. Whereas, in an enterprise, a manager (potential adversary) can begin collecting keys stealthily. Then, at some point in time, a manager can change the log and, more importantly, recompute the MAC chain.

EXAMPLE 3. *A dishonest dealer 2 starts recording authentication keys at certain time, τ_r. It does not engage in overtly hostile actions but records the keys. When the dealer detects a bad transaction at time $\tau_c > \tau_r$, it changes the corresponding record and recomputes all MACs since τ_r.*

This leads us to the definitions of *correctness* and *representativeness*. However, we first define multi-layered logs more formally:

DEFINITION 1. A *Multi-layered log* (ML) is a tuple $\langle L, D\rangle$, where L is a set of simple logs and D is a set of log dependencies. Each dependency is represented as an ordered pair (i.e., directed edge) $\langle l^s, l^t\rangle$ which indicate that the *target* log l^t depends on *source* log l^s.

We also use the following two functions:

DEFINITION 2. *$Source_D(l^i) = \{l | \langle l.l^i\rangle \in D\}$ taking a current log l^i returns a set of lower-level logs required for generating the current one.*

$Log^i : 2^L \mapsto L$. The function $Log^i(Source_D(l^i))$ generates the log of the i-th controller. Note, that each controller may have its own logging function.

In general *log l^i is* correct *if it has not been tampered with*, i.e., once records are inserted into the log, they cannot be changed. However, we relax this definition to make it equivalent to the forward integrity property.

DEFINITION 3. We say that a log l^i is *correct* if, when validating it at time τ_{com}, it is computationally impossible to change it before any time $\tau_c < \tau_{com}$.

We refer to the property which binds values bottom-up as *representativeness*:

DEFINITION 4. We say that a log l^i is *representative* if $Log^i(Source_D(l^i)) = l^i$. Lowest level logs are representative by default.

We define the properties of a multi-layered log system as follows:

DEFINITION 5. We say that a multi-layered log system ML is *correct* and *representative* if *all* of its simple logs are correct and representative.

4. MULTI-LAYERED LOG SCHEME

Our strongest anticipated attacker is a malicious manager who tampers with the logs. The proposed scheme improves [5] by ensuring that a new entry inserted by a manager is not a forgery (representativeness). Moreover, the proposed scheme provides protection against more powerful attackers who know MACs for the whole chain. Nevertheless, the manager is still forced to delete the keys to provide some protection of logs between commitments of MACs. This is protection against the attacker which incidentally (without collecting the keys) wants to change the log.

Our notation for trusted/untrusted parties is different from that in other secure logging papers [5, 14, 3] since, in our context, it can be misleading. Our notation is as follows:

- C^i – an entity managing the current log.
- C^k – an entity managing the next higher-layer log[2].
- V^i – verifier checking the current log.
- r^i_j – j-th message/record in current log l^i.
- F – a strongly collision-resistant one-way function.
- H – a collision-resistant one-way hash function.
- mac_{K_0} – symmetric message authentication code (MAC) under key K_0.

Algorithms 1, 2, 3 show the procedure for C^i controller.

Algorithm 1 Initialisation

1: Generate two symmetric keys A^i_0 and B^i_0
2: Commit B^i_0 to verifier V^i and A^i_0 to C^k who securely store them.
3: Receive and store premise signed MAC S^i_0 from V^i.
4: Assign an empty value to MAC $\mu C^i_{-1} = Null$
5: Create initial entry r^i_0

Algorithm 2 For each entry r^i_j

1: Receive lower-level logs $Source_D(l^i)$ (except for $j = 0$).
2: Check correctness of logs $Source_D(l^i)$.
3: Generate new log $r^i_j = Log^i(Source_D(l^i))$ (except for $j = 0$).
4: Append new entry r^i_j to log l^i.
5: Compute two MACs: $\mu C^i_j = H(\mu C^i_{j-1} || mac_{B^i_j}(r^i_j))$ and $\mu V^i_j = H(\mu V^i_{j-1} || mac_{A^i_j}(r^i_j))$
6: Generate new keys: $A^i_{j+1} = F(A^i_j)$ and $B^i_{j+1} = F(B^i_j)$ F.
7: Securely erase: μC^i_{j-1}, μC^i_{j-1}, A^i_j and B^i_j.

When the log needs to be closed, C^i generates a special closing record r^i_f, inserts it into the log and commits the last part of the log according to the above procedures.

In the proposed scheme, the verifier does not store any MAC values. The signed value S^i_j of the latest verified log (steps 9-11 in Algorithm 3) is stored by C^i. This value is later used by the verifier to seal the integrity of logs.

The proposed procedure uses two MACs (and two evolving symmetric keys A^i and B^i) for dual verification, as in [5]. The MACs are sent to both C^k and to V^i since neither entity is fully trusted.

The signature is needed to avoid an attack whereby a dishonest C^i (at time τ_c) changes past logs and recomputes the whole MAC chain. One alternative is for the verifier to store these values and perform the check during verification

[2]Note that C^{i-1} is not always the upper controller.
[3]Verification of S^i_0 is not required

Algorithm 3 Log commitment

1: C^i commits the last part of log $\langle r_q^i, ..., r_j^i \rangle$ and μC_j to C^k.
2: C^k checks representativeness of μC_j^i by computing it with previously updated MAC of record $q-1$ (μC_{q-1}^i)
3: C^k generates its new entry $r_n^k = Log^k(Source_D(l^k))$
4: C^i commits the last part of log $\langle r_q^i, ..., r_j^i \rangle$ and previously signed MAC S_{q-1}^i to V^i.
5: V^i recovers μV_{q-1}^i decrypting ${S_{q-1}^i}^3$.
6: V^i checks the correctness using previously received source logs: $Log^i(Source_D(l^i)) = l^i$
7: C^i commits μV_j^i to V^i.
8: V^i checks that μV_j^i is genuine by computing a MAC chain for l^i since time $q-1$ using μV_{q-1}^i.
9: V^i signs the MAC $S_j^i = mac_{K_V}(\mu V_j^i)$ with its key $K^{V,m}$
10: V^i updates $K_{V,m+1} = F'(K_{V,m})$
11: C^i stores signature S_j^i and deletes the previous one S_{q-1}^i.

phase of a specific part of the log only. However, this would require more storage on the verifier.

When V^i needs to perform an integrity check of i-log he requests C^i for the entire l^i and the latest MAC μV_f^i and the lower-level controller for their logs $Source_D(l^i)$. Then V^i recomputes the MAC of the latest entry (r_f^i) $\mu V_f'^i$ and compares it with the received value μV_f^i. Note that, the verifier has to first check the signature of the received MAC value μV_f^i. If the values are the same, the log is assumed to be correct. Then, V^i checks that all entries have been correctly derived by comparing the result of $Log^i(Source_D(l^i))$ and l^i (i.e., checks representativeness). If the results are the same, the verifier concludes that i-log is correct and representative. C^k can also check correctness of l^i, although it cannot do the same for representativeness, as it has no access to $Source_D(l^i)$.

5. RELATED WORK

Bellare and Yee in [3] introduced forward integrity for logs and proposed a scheme that enforces it. They considered a single computer that generates and logs events. In [2], this scheme was adapted for use with public cryptography.

Schneier and Kelsey [14, 11, 13] proposed several schemes based on a very similar cryptographical model. However, they provided a more elaborate protocol for logging events and sending records to a trusted machine, which (by their assumption) cannot be compromised. Tsudik and Ma [5] improved this model by reducing storage required for keeping authentication values by providing a cumulative logging scheme. Subsequently, Chong, et al. [4] showed how to implement Schneier and Kelsey's scheme using tamper-resistant hardware.

In another recent result [7], Holt described secure logging software which included several aforementioned forward-secure schemes. This work also described multiple log scheme where parent (high level) nodes store initial secrets for their children (low level). Note, that logs themselves are not sent to parent nodes; thus, the scheme cannot protect the system against a more powerful attacker considered in this paper.

Waters, et al. [17] developed a secure logging scheme based on identity-based encryption (IBE). The main feature of this scheme is the ability to search encrypted logs using keywords.

6. CONCLUSIONS

We proposed a multi-layered log scheme which protects against dishonest middle managers inserting fake records or changing existing records. In our model, the attacker is more powerful than those considered in prior literature [3, 5, 14]. The proposed scheme offers two ways of verification: by an upper manager interested in genuine low-level logs (which are used to create reports) and by a verifier. Such double checking is needed because neither the upper manager nor the verifier are fully trusted. The scheme also insures that logs do not contain fake events. This new feature is achieved by checking the evidence of events on lower levels of the log scheme. For future work we are going to implement the scheme in enterprise settings. Another interesting issue would be a formal verification of the proposed scheme.

7. REFERENCES

[1] Basel Committee on Banking Supervision, International convergence of capital measurement and capital standards, 2006.
[2] M. Bellare and S. Miner, A forward-secure digital signature scheme, CRYPTO'99, 1999.
[3] M. Bellare and B. Yee, Forward integrity for secure audit logs, UCSD Technical Report, University of California at San Diego, 1997.
[4] C. Chong, Z. Peng and P. Hartel, Secure audit logging with tamper resistant hardware, Technical Report TR-CTIT-02-29, Univ. of Twente, 2002.
[5] D. Ma and G. Tsudik, A new approach to secure logging, IFIP DBSEC'08, 2008.
[6] Fox Business, Futures trader responsible for $7b fraud. `http://www.foxbusiness.com`.
[7] J. Holt, Logcrypt: forward security and public verification for secure audit logs, AusGrid'06, 2006.
[8] ISACA, CobiT, `www.isaca.org/cobit/`, 2008.
[9] ISO/IEC. ISO 17799, 2001 IT Governance Institute, *IT Control Objectives for BASEL II: The important of Goverance and Risk Management for Complience.*, 2007. `http://www.isaca.org`
[10] IT Governance Institute. *IT Control Objectives for BASEL II. The important of Goverance and Risk Management for Complience.*, 2007. `http://www.isaca.org`
[11] J. Kelsey and B. Schneier, Minimizing bandwidth for remote access to cryptographically protected audit logs, RAID'99, 1999.
[12] Permanent Subcommettee on Inverstigations of the Comittee on Governmental Affairs of the United States Senate, The Role of The Board of Directors in Enron's Collapse, `http://news.findlaw.com/hdocs/docs/enron/senpsi70802rpt.pdf`, 2002.
[13] B. Schneier and J. Kelsey, Cryptographic support for secure logs on untrusted machines, USENIX Security Symposium, 1998.
[14] B. Schneier and J. Kelsy, Secure audit logs to support computer forensics, *ACM TISSEC*, Vol. 2, No. 2, pp. 159–176, May 1999.
[15] United States Districs Court Southern Districs of New York, United States of America vs. Bernard J. Ebbers, `http://news.findlaw.com/hdocs/docs/worldcom/usebbers504ind3s.pdf`, 2004.
[16] Wachtell, Lipton, Rosen and Katz, Report to the Board of Allied Irish Banks, p.l.c., Allfirst Financial Inc. and Allfirst Bank Concerning Currency Trading Losses. Available from `http://www.aibgroup.com`, March 2002.
[17] B. Waters, D. Balfanz, G. Durfee and D. Smetters, Building an encrypted and searchable audit log, ISOC NDSS'04, 2004.