

Improving the Protection of Logging Systems

Dmitriy Ayrapetov, Archana Ganapathi, Larry Leung
UC Berkeley Computer Science
Berkeley, CA USA

ABSTRACT

Filled with valuable data such as transactions, records and confidential information, system logs are lucrative targets for attack. Most computer systems use simple logging schemes that offer no protection after compromise. Various schemes have been proposed to strengthen logging systems by allowing verification of log integrity, protecting the confidentiality of the contents, and securely relocating logs on remote systems. Such logging systems increase overall system security by preventing certain attacks on the logs and providing accuracy guarantees for post-attack forensics. This paper summarizes the existing proposals in Secure Audit Logging, presents unaddressed problems in these proposals, and offers suggestions to enhance Secure Audit Log technology.

Keywords

Secure Audit Logs, computer forensics, intrusion detection, tamper evidence, forward integrity.

INTRODUCTION

After a security breach, logs are prime targets for attacks because they store sensitive information with weak security. Valuable information stored in logs range from confidential data such as financial transactions, to incriminating evidence such as proofs of intrusions into the system. Most often, logs are stored as sequential entries to a plain text file protected merely by the permissions checking built into the operating system. There are no guarantees of correctness or authenticity of the log entries protected by such mechanisms. Once the operating system has been compromised, the logs are at mercy of the intruder.

There can be several motivations to access log data. An attacker might want to steal confidential information that might be present in various application logs, such as those of an e-commerce system. Such information includes credit card or social security numbers, misuse of which can constitute identity theft. This type of an attack relies on the fact that log entries are stored in clear text with no encryption. An attacker might also want to insert a fake transaction into the logs for financial or personal gain. This attack would exploit the easily editable nature of log files. Sequence numbers attached to each event could also be defeated with the assistance of a simple script. Another scenario might be the case where the log contains the attacker's steps and incriminating evidence. In this case,

the attacker might want to completely erase those entries to cover up evidence.

Periodic backing up of the logs only solves the relatively minor problem of having redundancy. The granularity of such an approach, however, is inadequate to protect the log stream against attacks between backups. Such an approach might also let an attack go undetected since a backup might pick up the latest entries in tainted logs as valid data, with nothing to compare against for verification and reconstruction purposes. Barring this issue, backup storage can prove to be yet another source of difficulties. For example, printing logged events to a paper feed brings in the issues of preservation of the logs as well as the ability of the system to cope with the possible load. Periodically printing hashes of the log data in widely distributed publications such as newspapers introduces an upper bound on the quantity of such hashes. Thus, rather than focusing on data back-up mechanisms, we need to focus on improving the robustness of the logging system itself.

Relying on the security features of the underlying system is insufficient for logging mechanisms. Security features internal to the logging systems must be present to protect the log entries from attacks such as the ones above. Such internal mechanisms include cryptographic authentication codes that provide guarantees of a log's integrity. Encryption can also be used on the message payload to protect the confidentiality of the contents. Messages can also be interleaved in order to prevent unauthorized modification of the contents. Such methods not only present technical barriers to the intruders, but also allow for fast verification of the log integrity. These techniques can also enhance intrusion detection by triggering automated lockdown of the system upon detection of inconsistencies in the logs. The tainted logs may also identify which entries in particular were modified or deleted, consequently better exposing the attack while aiding computer forensics and/or audit trail analysis.

A TAXONOMY

In order to keep our discussion of secure logging system consistent, we first need to establish some terminology in order to differentiate the different types of log protection and attacks on logging systems.

Log Protection

The first, and perhaps the most important property that a logging system should possess is *tamper evidence*. A

tamper evident system provides a method of verification that reliably reveals the occurrence of tampering. It does not directly prevent tampering from occurring. Secure audit log systems fall into this category.

The next property is *tamper resistance*. A system that is tamper resistant has mechanisms in place make it deter various attacks on the logged information. While tamper evidence is a binary feature (either tamper evident or not), tamper resistance is best described as characteristics ranging from resistance from unauthorized reading to deletion prevention. Also, tamper resistance may come in various degrees of strength. The tamper resistance and tamper evidence features of a system are often synergistic.

A *tamper proof* system is the ideal logging system that leaves absolutely no means for compromising the logs that all systems strive for. All tamper proof systems are by definition tamper resistant against any attack.¹

Attacks

Attacks on logging systems can be classified in the following categories along with their motives:

- *Read* – Unauthorized access to sensitive data contained in the log messages allow for scenarios such as the stealing confidential information.
- *Delete* – Removal of a single entry or blocks of entries, often for the purpose of evidence removal.
- *Write* – Insertion or modification of a log entry, allowing for the falsification of an event.
- *Denial of Service* – Overwhelming of a log system's resources in order to disrupt normal operation, either delaying or preventing logging.
- *Flooding* — Overwhelm the logs with valid, but meaningless events in to mask illicit events in a flood of otherwise normal events.
- *Abuse of Trust* – A trusted user abuses existing privileges to bypass protection mechanisms and gain unauthorized access to the logging system and its contents.
- *Extraordinary events* – Destruction of a crucial part of the system to disable services and hinder forensics

Proposed secure audit log systems address some of these concerns, typically focusing only on the read, write and delete attacks.

THREAT MODEL

Secure audit logs use a threat model that encompasses the entire lifecycle of the attack, from compromise to recovery and forensics. Initially, we assume that a system is fully trusted and has properly configured logging mechanisms. After a certain length of time, an attacker with virtually unlimited resources and knowledge about the software configuration attacks the machine and successfully gains administrative privileges. At this point, the attacker swiftly compromises all software on the system including the kernel and may read and modify any data left on the system, including the log records. The attacker will conceal the intrusion by hiding signs of the compromise, often by letting services run as usual. Typically, the longer the attack goes unnoticed, the more damage an attacker can cause. At this point, the system administrator must somehow find out about the attack and limit the damage done by taking the system down. The system is then audited to reveal clues about the attacker's method of attack, purpose of attack, and the damage done.

EXISTING TECHNOLOGIES

The earliest ideas in this field stem from research in secure time stamping. A chain of time stamps are sequentially linked whereby a time stamp's certificate is derived based on that of the previous one [1]. This concept was further refined into a property called "Forward Integrity" that provides useful post-attack guarantees about data integrity when applied to system logs, as shown by Bellare & Yee [2]. They also outline various possible implementations and address the performance and feasibility of such implementations. Schneier & Kelsey take a similar concept and describe a scheme that provides log chaining, log-level permissions and verification by semi-trusted parties [3]. They also reduce the risk of the trusted verification key holder being compromised by suggesting a scheme for distributing trust among various nodes. There are also several RFCs and IETF drafts in progress that look to introduce security to existing logging systems. One such technology is the syslog-sign draft, which describes a remote logging system built on top of existing syslog infrastructure, that securely transmits and stores log entries [4].

These technologies are designed for different aspects of the problem of securing the logging systems. The research technologies, as well as syslog-sign, approach the problem of secure logging by treating the log stream as an entity to be compromised. They therefore aim to make it harder to tamper with the log and keep the changes undetected. Syslog-sign also includes message source authentication.

Both research technologies [2,3] fall under the category of tamper evident, since they allow tamper detection and do not make any guarantees about data integrity after the logs have been compromised. Both systems use existing cryptographic primitives such as symmetric encryption,

¹ The idea of a tamper proof system is useful for theoretical discussions about what can possibly be done with any logging system and probably don't exist since an attacker with physical access could destroy any known system.

hashing and one way functions to append message authentication codes to a log entry so that a verifier can confirm the authenticity of a single entry, as well as detect discontinuities in the log stream. Before launching into a discussion of the details of each method, note some of the terms for cryptographic primitives that are used to build these systems:

Cryptographic primitives

- MAC – Message Authentication Code. An authentication code generated with and verifiable by a key.
- Encryption – Standard symmetric encryption using a key K . This is denoted as $E_k(x)$.
- One way function or Hash - a non reversible function that is used to evolve the encryption/authentication keys between epochs. It is also used for verification of field contents. It is important for the function to be one-way so that the attacker, given an arbitrary epoch's key, can not compute keys for previous epochs and thus gain the power to falsify valid entries. (e.g. SHA-1) This is denoted as the function $H(x)$.

A detailed discussion of these primitives can be found in [5].

The proposals that work towards strengthening the message stream all share a similar technique of dividing time into epochs in which different keys are used. Although their use of this temporal division is different, their variations share common characteristics of being non-overlapping, sequential and having all log entries entered within the epoch boundaries, and never in between. The common assumption made by all these technologies is that there is at least one trusted entity and the underlying cryptographic primitives are secure.

Forward Integrity

Forward Integrity (FI) is a property proposed by Bellare and Yee to be associated with logging systems. A logging system can be considered having the FI property if it contains information that is sufficient to confirm or rebuke allegations of log stream modification before the moment of system compromise.

An example of such a system would work by generating a MACs of the log payloads in order to provide a mechanism of checking the authenticity of the entry's payload. Besides the normal message payload, a log entry in such a system contains an authentication code that is derived by applying the MAC function to the payload using the current epoch's key.

Epochs are evolved based on time or after a number of messages. Epoch changes are denoted by special messages signaling the start and end boundaries. All log entries are written only between these boundaries, which are defined to

be non-overlapping. An epoch transition not only sees epoch boundary messages written to the log stream, but also an evolution of the encryption key (A) used by the MAC algorithm and any other cryptographic primitives that might be used in the system. The encryption key for the next epoch is computed by an application of a one-way function to the current key. ($A_{n+1}=H(A_n)$) Immediately upon the completion of this computation, the previous key is erased from the system².

Bellare and Yee do not limit the implementations of their proposal to only having a message and the MAC of the message in a log entry. They suggest that any data, linear in size on the total message, could be added to provide further protection. This suggestion leads to discussion of further techniques.

The forward integrity property in this design comes from the fact that the compromise of a key in any epoch does not reveal any previously used keys without reversing the one-way function. Therefore the attacker is not able to modify the log entries written in epochs prior to the compromise. Only with the knowledge of the initial key and the one-way function used to evolve keys, can any undetected modification be possible to the log stream. It is for this reason that the original key should be very well guarded. The verifier, given the initial key and the function used to evolve the key, can go through all log entries and check whether the hash of the payload corresponds to the one stored in the log entry.

Proposed Implementations

Bellare and Yee examine variations among the implementations of various systems that possess the FI property. One could, for example, use random instead of sequentially computed keys for the MAC algorithm. These randomly generated keys would be stored on a secure remote host for the verifier to use. This particular approach is not desirable however, since there are too many keys to be stored and creates a single point of failure at the trusted host storing the keys.

An alternative method would allow the logging system to use seemingly random keys without the necessity of storing every single one on the logging host. This method turns to using pseudo random functions (prf). Each epoch would have a secret value s , which would evolve between epochs via $s_{j+1}=\text{prf}_{s_j}(0)$ and the hashing key for each epoch would be computed as $k_j=\text{prf}_{s_j}(1)$. Therefore only the initial s_0 would need to be kept on the logging host.

In order to add deletion detection to the FI schemes discussed, sequence numbers and end of epoch markers would be introduced. The position of the log entry within

² Special care must be taken in deleting the previous epoch's key from the memory. An attacker might otherwise be able to read the memory for the key and modify that epoch's logs. [2]

the epoch would be hashed along with the log entry's payload, thus further preventing insertion of false entries and allowing detection of deleted entries in an epoch.

Secure Logs on Untrusted Systems

Another scheme proposed by Schneier and Kelsey uses similar Forward Integrity authentication codes for verifying log entries, but also improves the design by building in payload encryption, verification hash chaining and the ability to allow semi-trusted parties to verify the logs [3]. Semi-trusted party verification allows verification of the logs without exposing all log data and the original authentication key to the verifier.³

Following the structure of the log entries proposed for forward integrity, Schneier and Kelsey add two additional fields to each log entry (something suggested in the Bellare and Yee's approach). One of the fields identifies the log type and allows for permissions masks. The second field is a chain of hash values of previous logs that tie the log stream together, preventing modification of previous log entries and enabling semi-trusted third party verification.

The scheme generally works in a similar fashion to the FI systems. Time is divided into epochs (E) with each epoch having a unique authentication key (A) associated with it. This key evolves between epochs via a one way function ($A_{j+1}=H(A_j)$).

In this system however, the key is not used directly, but in conjunction with other fields. For example, the encryption key (K) used for encrypting the payload (D) of each log message is calculated by using a hash function on a tuple consisting of the log entry type (W) and the authentication key. ($K_j=H(A_j, W_j)$) The message is then encrypted using this key K_j . Therefore the payload field of the log entry gets $E_{K_j}(D)$.

The next field in the log entry is the verification hash chain field (Y), which enables semi-trusted party verification of previous log messages. The hash chain field contains a hash of the current message's encrypted payload and message type field, as well as the previous log entry's hash chain field. What is written to the log entry is $Y_j=H(Y_{j-1}, E_{K_j}(D), W_j)$. This field provides stronger insertion and deletion detection than the sequence number scheme proposed previously in [2]. Also, given any known correct log entry with Y_j , a third party can verify all previous log entries without knowing the log data or any epoch keys.⁴

³ The previously described forward integrity designs all assumed that a verifier would be fully trusted and was trusted with the master key at the initial epoch. The master key could also be used to modify any log entry of any epoch. The trusted verifier is a single point of failure in these designs.

⁴ Note that if a log entry was periodically published in broadcast medium such as a newspaper, even a

A verifier with access to the encrypted log entries can verify the hashes to detect modifications to the logs. Removing a single log entry j in this chain invalidates the verification hash value in $j+1$, since the value expected by the verifier in Y_{j+1} now comes from Y_{j-2} . Likewise, inserting an event j into the log stream will result in an invalid hash chain field of the following log entry, $j+1$. The verifier would compare the hash chain value of the $j+1$ entry against the computed value that would use the corresponding field of the j^{th} entry, rather than that from $j-1$. In this case there would not be a match between what is expected in the field and what is written, and thus the insertion is would be detected.

The last field in a log entry in this proposal is a MAC of all other fields combined, using the current epoch's authentication key. Like previously proposed Forward Integrity MACs, this protects against any modification to the log entry.

One of the benefits of this proposal is that the attacker cannot read any of the log entries written in the past, since they are encrypted using K_j , the hash of the FI key and the log entry type, W . The W field is also used as a sort of permissions mask in the system to determine who can read what entries. Schneier describes a scheme that allows semi-trusted parties (including the logging system itself) to read selective log entries based on the permission mask. Because the log data is encrypted with the hash of the FI key A_j and W_j , the epoch key A_j will not be revealed while reading log entries.

Similar to the Forward Integrity schemes described by in Bellare and Yee, this scheme requires an external trusted system to hold the initial FI key to prevent modification of the log data and to help with the verification. Since any design with a trusted party is clearly a single point of failure, Schneier also presents an extension to this scheme that distributes trust among a network of insecure peers to reduce risks.

Syslog-Sign

These Forward Integrity addresses the secure logging problem on a single machine, but often logs need to be aggregated onto a single machine to reduce maintenance and ease analysis. The Syslog-Sign IETF draft [4] aims to add origin authentication, message integrity and features that would guarantee uniqueness of messages and continuity of the log stream. Currently a work in progress authored by John Kelsey and Jon Callas, syslog-sign aims to be the least intrusive by extending syslog, a popular remote logging protocol. The proposal adds two different types of messages, signatures and certificates, that secure the normal log messages. Therefore syslog-sign can be adopted with little disruption since the syslog APIs are the same and the

compromise of the master key at the initial epoch would not be enough to modify written log entries before that entry!

logging host can optionally chose to ignore syslog-sign messages.

The idea behind the implementation of syslog-sign is in utilizing parts of the MSG field to implement some of the suggestions put forth in the papers above such as deletion detection, message integrity and message sequencing. Additionally, this draft also proposes including origin authentication, which would be useful in a scenario where multiple systems distributed on the internet are reporting to the same logging host. This would guard against network attacks that would spoof messages to seem like they are coming from the compromised device.

Syslog-sign allows for out of order message delivery and can recover from lost messages. Upon receiving a certain block of messages, syslog-sign expects a message of type 'signature block' which contains hashes of the previously sent messages. These hashes are compared against what has been received thus verifying the existence of an expected message. If the messages come within different signature blocks, verification routines allow for a message to arrive to the logging host after its signature block has been received. This can be achieved either in an offline or an online mode. (The latter using the replay window approach [4].)

Messages that are missing or altered can be detected by checking if either their hash does not match what is expected in the signature block, or there are hashes in the signature block for which there are no corresponding messages.

UNADDRESSED PROBLEMS

Secure audit logging systems attempt to provide tamper evidence in read/write/delete attacks and some log truncation attacks, but are susceptible to denial of service, flooding, log truncation, and abuse of trust attacks. A clever attacker could employ a combination of these attacks to bypass the log security mechanisms.

Denial of Service

A Denial of Service attack (DOS) refers to a common technique of overwhelming a system's resources in order to cause a system outage. Such attacks are common today on network systems such as web servers and could be use on a logging system to disrupt logging services.⁵

Secure logging systems can make direct DOS attacks harder by using efficient algorithms and fast verification routines. An attacker that wants to slow verification to a halt could create a flood of log entries for the system to verify. Since cryptographic primitives are computationally intensive by nature, their excessive usage can make the system more

vulnerable to a DOS attack. Forward Integrity systems proposed by Bellare and Yee are designed with efficiency in mind, and are not very computationally complex, requiring one cryptographic calculation per log entry. The design proposed by Schneier and Kelsey however, uses four cryptographic calculations per log entry. These operations are needed to support semi-trusted third party verification.

A DOS attack on the network would prevent verification of the log entries on forward integrity schemes. An intruder might overload or disrupt the network connectivity of either the machine generating the logs or the trusted party, thus disrupting verification in forward integrity schemes. Since the forward integrity schemes all rely on a remote host for storing the initial epoch key, their verification protocols require network communication. By disrupting normal verification with a DOS attack, the attacker can have more time on a compromised system and hence do more damage.

A DOS attack on the network of a remote logging system disrupts logging and may shutdown a system. The current implementation of syslog uses the UDP protocol for remote logging. A disruption of the network between a compromised system and a remote logging host creates a window within which log messages will be queued. A DOS attack of this sort on the remote logging system could make a large backlog of log entries on the clients that could shutdown the system or disable logging. Since syslog write are synchronous, this may shutdown all services that require syslog as well. Recovery of these logs once the network is restored is another problem.

Flooding

Flooding logs with valid, but meaningless log messages create another sort of a denial of service attack. There are several implications of such an attack:

- The attacker floods the logs until the storage capacity is full, effectively disabling the logging system.
- The attacker floods the system with valid but meaningless events such as login attempts to obscure a real attack. Hidden amongst these log entries may be the actual attack but masked by the plethora of valid log entries. Well-hidden attacks will force the auditor to manually sort through the logs, making it difficult to detect.

A flooding attack is a type of DOS attack, except it employs a deeper understanding of the log structure to prevent verification.

Log Truncation

The Log Truncation attack defeats the deletion detection properties of a secure audit logs by removing all log entries past a certain point. Using a combination of a DOS attack and a system failure, this following recipe can truncate the logs of the proposed systems:

⁵ A disruption of this sort can be caused by numerous network attacks such as DNS, BGP compromises or brute force attacks such as TCP SYN flooding.

1. Disable verification using a DOS attack on the trusted host
2. Break into the system and gain administrative access.
3. Commit malicious actions
4. Truncate the logs in a forward integrity system to an epoch boundary before the attack. Delete queued logs in a remote logging system.
5. Force a system crash

Afterwards, a system will find that its current logging key is lost. How to recover from such a situation is unclear since it could either be a normal system failure or an attack.

Normally, a truncation attack would be detected using the same tamper evident techniques in forward integrity. It would notice that the current key isn't the current epoch key and thus set off alarms. If the key were stored on non-volatile memory, this attack could be avoided. But this creates more problems as described in [2] unless properly secured with specialized hardware to protect the key.

Neither forward integrity proposal adequately addresses this problem and its unclear whether it can be solved without specialized hardware.

Abuse of Trust

Every system has trusted administrators that install and configure software on a system, including the logging system. A malicious administrator could abuse the privileges entrusted to weaken the security of the audit logging system.⁶ One option is to copy an earlier epoch's key from memory and then later using it to modify log entries to cover up fraud. Alternatively, the key could be taken from the trusted system if the administrator also controlled that system. After taking control of the secure audit logs, the system is at the complete mercy of the system administrator.

Assuming the epoch keys aren't compromised, contents of the log are also susceptible to abuse of trust attacks. Log contents are encrypted in Schneier's scheme to protect the log contents from administrators or users with read access from abusing their privileges and reading confidential information contained in the logs for unintended purposes. For example, a prying system administrator may want to track a user's website accesses through the log data. Restricting an access to logs may hinder auditing, but makes sense when the data contained in the logs is confidential information and the administrator only needs access to specific types of logs relevant to maintaining the system. Schneier's scheme supports permissions, a basic building block for this protection, but confidential

information and system status data are often mixed together in similar log messages. For example, a web server's error messages that allow bad links to be detected also allow for administrators to determine the users who wanted a specific resource.

System Failures

An attacker can always cover up any attack by unrecoverably destroying the system. For example, a secure format utility could destroy all log data and make recovery impossible. A malicious system administrator may "accidentally" reformat a hard drive to cover up information theft. All software-only logging systems are susceptible to this kind of attack. Tamper evidence may not be possible as distinguishing accidental and purposeful destruction of a system is not easy.

Some solutions

Many of these attacks presented above have either no known solution or partial, costly solutions. Further research into these problems is necessary to improve log security. In the following section, we describe any known solutions to these problems and any ideas for further research.

Denials of Service attacks are easily detectable, but are very hard to counter. Smarter management of resources makes these attacks more difficult but not impossible. Many systems, especially network services, already have known DOS vulnerabilities and DOS attacks can be built by overwhelming any service. Implementing DOS detection would help alert administrators of an attack and help trace an attack back to the source. Attacks using a DOS on a forward integrity logging system typically attempt to disrupt communication between the system and its trusted party with the initial epoch key. A secure communication channel between the two systems is a costly but effective way to prevent many network DOS attacks. This forces the attacker to launch the DOS from one of the two machines, a significantly riskier attack, or use another type of DOS. Without a secure link, any software response to a DOS by shutting down system functions or blocking remote hosts will either reduce security or allow a DOS to escalate into the system's vital services. There is no clear software only solution for a DOS attack.

The log truncation attack could be tamper evident by providing resistance to deletion or non-volatile storage of the key. If the attacker cannot easily find where the incriminating logs are, he or she cannot truncate the logs and make the attack non-evident by covering up the attack with a system failure that erases the current epoch key. A proposal to provide stronger deletion resistance is described below in future work. Another option is to constantly store the key in nonvolatile memory so upon reboot, the last epoch will be apparent. Storing the key on disk is an option but may create more security risks. Another option is to use

⁶ The malicious insider, an employee with malicious intentions or a spy, is a common phenomenon and is noted in an analysis of tamperproof equipment [7].

special tamper resistant hardware such as a smart card to store the epoch key and also prevent copying of the key.

Any abuses of trust attacks are notoriously difficult to defend against since the attacker has full knowledge of a system's security, access to the system, and the trust of his or her peers. If a system starts out uncompromised and the administrators have no direct physical access, a system that logs all changes to binaries and all significant actions on a system could prevent some abuses of trust at the system administrator level such as subverting security measures and deleting important records. Though logging all actions is poor protection since most system administrators can upgrade software and install backdoors in the software without being detected. The logging software could be compromised to never log certain actions. Attacks based on key compromise have some solutions. Using tamper resistant hardware to store the epoch key would prevent the administrator from simply copying the key and using it to modifying log entries. For trusted host attacks, the trust could be distributed across many insecure systems as described in [3] to reduce the risk of compromise. Using distributed trust systems like the one described, peer administrators could also review each other's accesses to the logs to make inappropriate reads of confidential log data riskier to an insider.

Attacks employing full system failures are likely to be impossible to stop with only software solutions. We could detect any DOS attacks used to shut off the system and also report any attempts to reformat the hard disk, but this activity could easily be normal maintenance. Physical attacks on the hardware are impossible to detect without special hardware. No good solutions are known in this area since even special hardware can only prevent a small percentage of attacks.

A summary of the attacks and solutions is shown in Figure 1.

Attack	Solutions
Denial of Service	Detection, secure channel
Log Flooding	Automation of log analysis
Log Truncation	Tamper resistance, key protection
Abuse of Trust	Distribute trust, key protection
System failure	none

Figure 1. Attacks and Solutions

FUTURE WORK

In addition to solving some of the problems identified above, we believe that further work combining the ideas from various security fields will make secure audit log technology a more compelling addition to modern systems. The technology can support more than post-attack computer forensics; secure audit logs could help support real-time defense mechanisms such as intrusion detection systems. A

scheme for providing deletion resistance is presented. Also, a combination of remote logging and forward integrity schemes is explored.

IDS Integration

Secure Audit Logs provide evidence of tampering with logs while Intrusion Detection Systems (IDS) currently notify administrators of tampering of key system files, a warning of an attack on a computer system. An IDS like Tripwire monitors key system files such as the configurations in /etc and binaries such as login that an attacker may compromise after a successful break-in to further weaken security and install back doors to control the system. Previously, an IDS could not monitor attacks on the logs because log files that defy the simple digest checking techniques employed by most systems. The Secure Audit Log technology in this paper provides the verification primitives that would allow an IDS to detect of modification on past log records, cutting off another opportunity for an attacker to cover up an attack.

Integration with an IDS requires that a system be able to verify its logs periodically. Current designs support this feature through communication with the remote trusted host that holds the initial epoch key. Clearly, there is a tradeoff between time between self audits and bandwidth costs. Work needs to be done to minimize traffic and increase the speed of this operation.

Schneier's scheme could support IDS verification by revealing one dummy log entry to the IDS after every epoch. Given a log entry, the IDS could use the backward hash chain to verify the correctness of the rest of the log. Unfortunately, this require significant amount of network traffic over time as every revelation implies a conversation with the remote trusted party. Also, this scheme is susceptible to a network denial of service attack since cutting off the network would disable the IDS's log verification. At this point, a system is faced with the dilemma of whether to shut down services or to continue without verification. Shutting down would allow a network denial of service to turn into a system denial of service, allowing attackers to easily disrupt vital services. Continuing without verification would reduce the security of the system and its ability to respond to an attack.

Deletion Resistance

Secure audit logging techniques can greatly benefit from research into making logs deletion resistant to make the truncation attack described above fully evident. One idea to provide better tamper resistance is by entangling all the log record so an attacker would not be able to selectively delete log records. An attacker would have to corrupt the log or delete the whole log to cover the attack, a bold action likely to alert administrators.

One proposed idea is to use an encrypted database to store the data. Encrypted databases are based on related ideas to Shamir secret sharing and are applicable to privacy-

protecting data storage schemes where users want to store sensitive data on a server and have full control over what information is revealed to the server after a query [6]. The encrypted database resides on the logging system and the trusted machine or machines can issue queries on the encrypted database to reveal certain log entries upon demand. (Similar to Schneier's scheme, the trusted party controls queries on the encrypted data.)

Given an encrypted database, secure logging system inserts the same log entries and MACs as presented above in forward integrity schemes. Since the database is encrypted and the internal representation of data in a database is opaque to the logging system, single records can't easily be located in the database. An attacker must delete the entire database or corrupt it to remove a log entry. This makes the log truncation/system failure attack presented above impossible to execute without evidence.

Support for Distributed Systems

Most system administrators will want to consolidate the log data into a central repository and audit the data centrally. This approach eases auditing for large groups of systems and may provide additional insight into attacks spanning multiple systems, but may create other log security problems. Generally, centralized schemes are more vulnerable to the denial of service attacks mentioned above and create a single point of failure for the whole system.

The Syslog-sign IETF draft addresses how to securely transfer log messages to a central repository and could be used in conjunction with log entries created via Schneier's FI scheme, but no concrete research in this area has addressed potential security problems this combination creates. While this scheme addresses deletion detection well in both logging system and central repository, it suffers from all the weaknesses of FI and more. If an attacker gains entry to the system, the syslog-sign key could be compromised and used to send normal messages to the repository. Auditors could not detect the attack directly from the central logs and instead are fooled by the fake log entries. Further work into adapting secure logging technologies to distributed systems is needed.

CONCLUSION

Secure Audit Logs technology promises to provide guarantees on a log's authenticity, supporting post-attack

computer forensics and intrusion detection. Techniques such as forward integrity provide the cryptographic basis for log integrity schemes. Past research into this area has addressed a number of problems such as semi-trusted authentication, distributed trust, and secure remote logging, but has not addressed many vulnerabilities caused by denial of service, flooding, log truncation, abuse of trust, and system failures as described above. Only a small fraction of these problems have solutions and some problems seem impossible to solve through software only methods. To improve the security of audit logs, we suggest further research into these problems as well as integration with intrusion detection mechanisms, deletion resistant through encrypted databases, and support for distributed systems.

ACKNOWLEDGMENTS

We wish to thank several people who have provided us with feedback and valuable input in writing this paper. First, we thank Doug Tygar for his guidance and ideas for attacks on logging systems. Next, we thank David Wagner for providing us additional references in this field. Finally we thank everyone who has contributed to refining and editing this paper.

REFERENCES

1. S. Haber ,W.S. Stornetta, "How to Time Stamp a Digital Document" *Advances in Cryptology – Crypto '90*, Springer-Verlag, 1991, pp.437-455
2. M. Bellare ,B.S.Yee, "Forward Integrity For Secure Audit Logs" 1997 University of California, San Diego.
3. B. Schneier ,J.Kellsey "Cryptographic Support for Secure Logs on Untrusted Machines"
4. J. Kelsey, J.Callas "Syslog-Sign Protocol DRAFT". Network Working Group. June 2002.
5. A.J.Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997
6. D. X. Song, D. Wagner, A. Perrig. *Practical Techniques for Searches on Encrypted Data*.
7. R. Anderson and M. Kuhn, "Tamper Resistance: A Cautionary Note," *Proceedings on the Second USENIX Workshop on Electronic Commerce*, Nov 1996, pp. 1-11.