Report
on
"Secure Audit Logs to Support Computer Forensics"

[Bruce Schneier and John Kelsey]
Counterpane Systems

by

Atit S Gaonkar
asgaonka@asu.edu
(1217031322)

Spring
2020

Course
"Computer and Network Forensics"

Instructor
Dr. Jaejong Baek

Contents

1 Introduction

At this pace of digital advancement and rapid data generation, it is important to practice logging. Current scenario doesn't always satisfy logging-in on secure machine. So in the event of log compromise, it should be impossible for the attacker to read and modify without being detectable. Hence, there is a need to build and maintain a file of secure audit log through computationally cheap methods.

1.1 Problem Statement

Given an untrusted[1] machine μ which is neither physically secure nor tamper resistant. *[Bruce Schneier and John Kelsey]* develop a system to log on μ with minimal interaction with a trusted machine τ while guaranteeing strongest security measures. In case an attacker gains control of machine μ at time t, the attacker shouldn't be able to read, alter or delete logs made before time t without being detected. Hence maintaining Confidentiality, Integrity and Availability of this system until time t.

In rare situations where owner of the device is not the same as the owner of secrets within the device, there is an urgent need to employ audit mechanism able to determine if there has been some attempted fraud. The purpose of this audit mechanism is to be able to detect any signs of manipulation, not to prevent all possible manipulation.

Numerous applications are in need of such protocols. To give an analogy, consider an electronic wallet as the untrusted machine (μ) containing its own set of instruction, code and data to work independently. Apart from its independent work, it also does communicate occasionally with its corresponding servers, which are assumed to be trusted (τ). In an event of e-wallet being compromised, the servers should be able to detect the compromise, so that no more traffic flows through the compromised device. Moreover the audit mechanism should be able to withstand tampering in-order to notify the servers. More applications are discussed in Section 2.3: Application

2 Novel Approach

[Bruce Schneier and John Kelsey] develop a computationally cheap method for making all log entries generated prior to the logging machine's compromise impossible for the attacker to read, modify or destroy undetectably.

2.1 Terminologies

In order to understand the methodology proposed by *[Bruce Schneier and John Kelsey]*, it is necessary to understand the terminologies used.

(i) ID_x represents a unique identifier string for an entity, x.

[1] Although *untrusted*, it isn't generally expected to be compromised

(ii) $PKE_{PK_x}(K)$ is the public-key encryption[2], using x's public key.

(iii) $SIGN_{SK_x}(Z)$ is the digital signature[3], under x's private key.

(iv) $E_{K_0}(X)$ is the symmetric encryption[4] of X using key K_0.

(v) $MAC_{K_0}(X)$ is the symmetric message authentication code using key K_0.

(vi) $hash(X)$ is the one-way hash function[5].

(vii) X, Y represents the concatenation of X with Y. Also represented as $X||Y$

(viii) p represents nonce.

Understanding of different processes is incomplete without the knowledge of various parties involved in the system. Below mentioned are three different parties within the system:

- τ is the trusted machine. It may typically be thought of as a server in a secure location. Example: Bank Servers, Government Servers
- μ is the untrusted machine, on which the log is to be kept.
- ν is a moderately trusted verifier, which will be trusted to review certain kinds of records on a log, but not trusted with the ability to change records. ν cannot access logs without interaction with τ.

2.2 Methodology

Apart from the construction rules, the working of this system can be categorised into four major modules, namely Creating the logfile, Closing the logfile, Validating the logfile and Verification Entries. The foundation is governed by certain assumption, definitions and rules. It is imperative to comprehend the importance of these underlying principle.

The untrusted machine μ initially shares a secret key with trusted verification machine τ. Using this secret key, a logfile is created. Four major aspects of security protecting the logfile is mentioned below.

- **Short-term Hashing**: As soon as the log is written, the log authentication key is hashed. The old version of authentication key is overwritten by the new hashed value of authentication key. Hence irretrievably deletes the previous value.
- **Encryption**: As the encryption keys are derived from hashed value, it is possible to give encryption for individual logs to partially trusted users. Any altercation by those users can be easily detectable.

[2]RSA or ElGamal
[3]RSA or DSA
[4]AES, DES, IDEA, or Blowfish
[5]SHA-256 or SHAKE-256

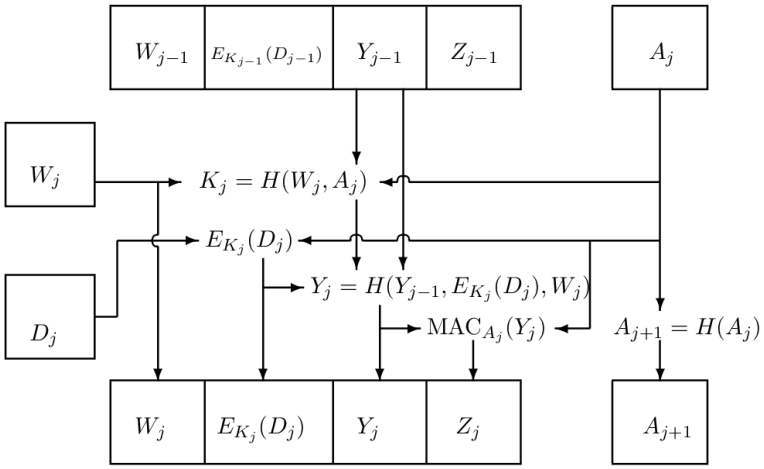

Figure 1: Process Architecture

- **Single Valued Verification**: Every log entry contains an element in a hash chain that serves to authenticate the values of all previous log entries. Similar to Block-chain. Using this single hash values, it is possible to validate the whole log file.
- **Permission Mask**: Permission masks acts as access right given to partially trusted users. Every log entry has its own permission mask. Different partially trusted users can be given access to different kinds of entries. As the encryption keys are partly derived from log entry type, lying about the access rights to gain access to restricted log makes it almost impossible for the partially trusted user to get the right access key.

2.2.1 Assumption

Assumption have be made in order to reflect reality and cope with various unknown factors. Below mentioned are few assumptions made that are important for functioning and relevance of this system.

- Untrusted machine μ occasionally communicates with trusted machine τ.
- Unavailability of constant reliable high-bandwidth channel between μ and τ.
- The attacker cannot compromise machine τ.
- After μ is compromised, it eventually manages to communicate with τ.
- μ has both short-term and long-term storage available.
- Existence of cryptogrphic tools to communicate securely between μ and τ.[6]

2.2.2 Construction Rules

The process of log entry follows certain architecture. Every entries in logfile makes use of fixed format and are constructed using following procedure:

(i) D_j represents the data to be logged in j^{th} entry of ID_{log}. Data can be of any format, understandable and relevant to the user entering it.

[6]Usage of SSL or TLS protocol.

(ii) W_j is log entry type of the j^{th} entry. It serves as a permissions mask for ν. τ controls which log entry types, a particular ν will be allowed to access.

(iii) A_j is the authentication key for the j^{th} entry in the log. μ must generate a new A_0 before starting a logfile. Either this authentication key A_0 can be shared during startup or can be shared later between μ and τ.

(iv) $K_j = hash(W_j, A_j)$ This is the key used to encrypt the j^{th} entry in the log. As discussed W_j is used in the key derivation to prevent the partially trusted verifier ν, from getting decryption keys for log entry types to which the verifier is not permitted access.

(v) $Y_j = hash(Y_{j-1}, E_{K_j}(D_j), W_j)$ Hash chain maintained to allow partially trusted users, ν, to verify parts of the log over a low-bandwidth connection with the trusted machine, τ. Y_j is based on $E_{K_j}(D_j)$ instead of D_j so that the chain can be verified without knowing the log entry. Y_1 is initialized to $NULL$.

(vi) $Z_j = MAC_{A_j}(Y_j)$

(vii) $L_j = (W_j||E_{K_j}(D_j)||Y_j||Z_j)$ where L_j is the j^{th} log entry.

(viii) $A_{j+1} = hash(A_j)$

The values A_j and K_{j-1} are irretrievably destroyed as they are overwritten by the new values A_{j+1} and K_j as soon as they are computed. Figure 1 illustrates the process of writing the j^{th} entry into the log, given A_{j-1}, Y_{j-1}, and D_j.

Given that an attacker gains control over μ, at time t, $L_1, L_2, L_3, ...L_t$ be the the log entries logged until time t and current value of A is A_{j+1} what the attacker will have in hand. As $A_{j+1} = hash(A_j)$, uses hashing mechanism, it is computationally impossible[7] to retrieve A_j from A_{j+1}. given these factors, the attacker cannot compute A_{t-n} for any $0 \leq n \leq t$. Hence it isn't possible for the attacker to read, alter any previous entries without being detected. The logging machine would have also logged attacker's intrusion, but the attacker has no way of knowing if the intrusion has been detected or not. Only possibility for the attacker to harm μ is through deletion. Even then, the next communication between μ and τ will flag the damage done to the logfile.

In the event where the attacker compromises μ before computing A_j, the t^{th} log entry is excluded from being secure as the attacker has knowledge of A_{j-1}. If A_{j-1} is a normal log entry, it is not harmful to the system, but in case of A_{j-1} being the log entry representing attacker's intrusion, τ will not be able to figure out if there has been a compromise.[8]

[7]Breaking SHA-256, SHAKE-256 is computationally impossible

[8]Drawback of infrequent log entries.

2.2.3 Startup: Creating the Logfile

As discussed in Construction Rule (iii), μ should commit to A_0 in order to authenticate itself and to start a logfile. Either A_0 is communicated betwwen μ and τ initially before dispatch or as soon as they establish a communication channel.

This transfer of A_0 between μ and τ should be done as securely as possible. In-order to protect the whole chain of logs, it is important to make sure, A_0 is not leaked.[9] μ should adhere to all the log construction rule, else τ might process it as signs of tampering. Even though μ cannot communicate A_0 to τ in real time, it is safe on $\tau's$ part to unhesitatingly assume that received A_0 is not tampered with, unless any sign of tampering is found.

μ has knowledge of $\tau's$ public key and certificate of $\mu's$ public key from τ. Using these two form of knowledge, A_0 can be passed through securely using the following procedure.

(i) μ forms the following components

- K_0, a random session key.
- d, a current timestamp.
- d^+, timestamp at which μ will time out.
- ID_{log}, a unique identifier for this logfile.
- C_μ, $\mu's$ certificate from τ.
- A_0, a random starting point.
- $X_0 = (p||d||C_\mu||A_0)$.

μ forms M_0 from the above knowledge and sends it to τ. μ also stores $hash(X_0)$ locally while waiting for the response.

$M_0 = p||ID_\mu||PKE_{PK_\tau}(K_0)||E_{K_0}(X_0||SIGN_{SK_\mu}(X_0))$, where p is the nonce / protocol identifier

(ii) μ forms the first log entry L_0, with $W_0 =$ '$LogfileInitializationType'$ and $D = (d||d^+||ID_{log}||M_0)$

(iii) τ receives and verifies the initialization message. If it decrypts correctly into valid $\mu's$ signature, then τ forms $X_1 = (p||ID_{log}||hash(X_0))$. τ then generates a random session key, K_1, forms M_1 and sends it back to μ. $M_1 = (p||ID_\tau||PKE_{PK_\mu}(K_1)||E_{K_1}(X_1||SIGN_{SK_\tau}(X_1)))$.

(iv) μ receives and verifies M_1. If verified, then μ forms a new log entry, L_j, with $W_j =$ '$ResponseMessageType'$ and $D_j = M_1$. μ also calculates $A_1 = hash(A_0)$.

If μ doesn't receive M_1 by the time-out time d_1, or if M_1 is invalid, then μ forms a new log entry with $W_1 =$ '$AbnormalCloseType'$ and $D_1 = (Curr.Timestamp||ReasonforClosure)$. The log file is then closed.

[9] Attack on μ before sharing A_0 renders the whole process useless. It is important to develop μ with pre-shared A_0.

It depends from one application to another if they can take risks of logging between the time frame of sending M_0 and receiving of M_1.Secure application may not take the risk of communicating within this window. It is in the best interest to log M_1 in either case. The purpose of entering unsuccessful abort log message is to indicate τ, that there is no logfile currently in existence with the given ID_{log}.

It might be a case where an attacker could delete μ's whole logfile after compromise, and claim to simply have failed to receive M_1 during the startup.

2.2.4 Closing the Logfile

In order to close the logfile, it is important to follow certain procedure. Write the final-record message, D_f (the entry code as '$NormalCloseMessage$' and data as timestamp).

Irretrievably delete A_f and K_f. Once the file is properly closed, an attacker who has taken control of μ cannot make any alteration to the logfile without being detected. Neither can an attacker delete entries nor add others earlier in the log. Finally, the attacker cannot delete the whole log file, because of the earlier interaction between μ and τ. Any of these changes will be detected when τ sees the final logfile.

2.2.5 Validating the Logfile

When τ receives the complete and closed log, τ can validate it using the hash chain and Z_f (since it already knows D_0). Can also derive all the encryption keys used, and thus read the whole audit log.

2.2.6 Verification Entries

At times, a moderately trusted person or machine, called ν, may need to verify or read some of the logfile's records while they are still on μ. This is made possible if τ has sent M_1 to μ and if ν has a suitable channel available to and from μ. Note that this can occur before τ has received a copy of the log from μ, and before μ has closed the logfile.

(i) ν receives a copy of the audit log, $L_0, L_1, L_2, ..., L_f$, where f is the index value of the last record, from μ. Note that μ does not have to send ν a complete copy of the audit log, but it must send ν all entries from L_0 through some L_f, including the entry with M_1.

(ii) ν goes through the hash chain in the log entries (the Y_i values), verifying that each entry in the hash chain is correct.

(iii) ν establishes a secure connection with τ[10].

(iv) ν generates a list of all the log entries she wishes to read, from 0 to n. This list contains a representation of the log entry type and index of each entry to which the verifier is

[10]Insecure connection renders the whole process useless

requesting access. (Typically, only some log entry types will be allowed, in accordance with ν's permission mask.) This list is called $Q[0..n]$, where $Q_i = (j||W_j)$.

(v) ν forms and sends to τ: $M_2 = (p||ID_{log}||f||Y_f||Z_f||Q[0..n]$.

(vi) τ verifies that the log has been properly created on μ, and that ν is authorized to work with the log. τ knows A_0 so he can calculate A_f. This allows him to verify that $Z_f = MAC_{A_f}(Y_f)$. If there is a problem, τ sends the proper error code to ν and records that there is a problem with ID_{log} on μ or a problem with ν.

(vii) If there are no problems, τ forms a list, $R[1..n]$, of responses to the requests in Q. Each entry in Q gets a corresponding entry in R: either giving the verifier the decryption key for that record, or else giving it an error code describing the reason for the refusal of access. (Typically, this will be because the log entry type isn't allowed to be read by this verifier.) Note that τ computes these keys based on the log entry type codes given. If ν provides incorrect codes to τ, the keys will be incorrect and ν will be unable to decrypt the log entries. Additionally, ν will not be able to derive the right key from the key he has been given.

(viii) τ forms and sends to ν: $M_3 = (p||R[0..n])$.

(ix) ν is now able to decrypt and read, but not to change, the log entries whose keys were sent in M_4. ν is also convinced that the log entries are authentic, since a correct MAC on any hash-chain value is essentially a MAC on all previous entries as well.

(x) ν deletes the key it established with τ in Step (3). This guarantees that an attacker cannot read μ's logfile if ν is compromised later.

2.3 Application and Extensions

There is an immense possibility for further extending on this protocol. It also provides huge array of application.

2.3.1 Usage

Secure audit log can be used a an utility tool for various scenarios. Extending this tool to solve various problems encountered makes it a good utility tool.

- **Forensic Tool** - Any signs of tampering and intrusion is detected by this audit log and hence can act as forensic tool. Assuming that there is a software that reads the log and does log analysis, there is a possibility that usage of secure audit logs would prove to be invaluable to forensic analysis.

2.3.2 Practical Examples

Other examples of systems benefiting from this protocol are discussed below:

- A computer that logs various kinds of network activity needs to have log entries of an attack undeletable and unalterable, even in the event that an attacker takes over the logging machine over the network.[11]

- An intrusion-detection system that logs the entry and exit of people into a secured area needs to resist attempts to delete or alter logs, even after the machine on which the logging takes place has been taken over by an attacker.[12]

- A computer under the control of a marginally trusted person or entity needs to keep logs that can't be changed after the fact, despite the intention of the person in control of the machine to "rewrite history" in some way. This also comes up when a secure coprocessor, or "dongle," is attached to an untrusted computer.[13]

- Mobile computing agents could benefit from the ability to resist alteration of their logs even when they're running under the control of a malicious adversary.[14]

3 Critique

The idea of coming up with such a protocol / system to counter numerous existing needs. Cryptographic tool is utilized to its maximum capabilities. Although being original and state of art, it does have its strength and weaknesses.

Looking from the perspective of the authors *[Bruce Schneier and John Kelsey]* during 1999, the usage of various legacy algorithms were justified. For example: SHA-1 in 1999 was computationally impossible to break.

3.1 Strengths and Weaknesses

Below list describes the strengths and weaknesses of this system.

3.1.1 Strengths

- Usage of Short-term Hashing technique to overwrite old authentication keys with new authentication keys paves way to achieve Perfect forward secrecy. It states that session keys will not be compromised eve if private key of the recipient is compromised.

- Authentication and Non-Repudiation is provided by use of Encryption and Certificates. Execution time of asymmetric algorithm is higher than symmetric algorithm. Hence, the idea of communicating symmetric key through asymmetric encryption is commendable.

[11] [Stoll 1989]
[12] [Schneier and Kelsey 1999]
[13] [Kelsey and Schneier 1996; Schneier and Kelsey 1997b]
[14] [Riordan and Schneier 1998]

- Single Valued Verification makes it lightweight and hence works faster making side-channel attacks a little harder. It is always easier to verify the whole chain via just an initial value A_0.
- Usage of permission masks makes it almost impossible to read logs by disguising as someone else, hence maintaining confidentiality of the audit logs.
- Hash function keep the integrity of the audit logs intact.

3.1.2 Weaknesses

- Assuming that once μ is compromised, it eventually manages to communicate with τ in itself a weakness to this protocol. If it doesn't manages to communicate, there is no possibility for τ to know that μ is compromised.
- Even though single valued verification makes it lightweight, it also induces a weakness. Single point of contact, A_0 makes it more vulnerable to initial attacks. Getting hold of A_0 will compromise the whole chain.
- In the event where the attacker compromises μ before computing A_j, the t^{th} log entry is excluded from being secure as the attacker has knowledge of A_{j-1}. If A_{j-1} is the log entry representing attacker's intrusion, τ will not be able to figure out if there has been a compromise.
- Attack on μ before sharing A_0 renders the whole process useless. It is important to develop μ with pre-shared A_0.
- Once the attacker has gained control over μ, the attacker may include phony logs of intrusion at a later time, which will make τ neglect the original intrusion.
- Infrequent log entries can make the intrusion detection harder. It is advised to log every minute, in-order to counter the possibility of quick attack. Logging based on knob values, can be susceptible to numerous attacks.
- $\nu's$ compromise enables the attacker to read certain log files which ν could have easily read.

3.2 Further Improvements and Expectation

Due to it's broad range of application, there is numerous opportunity for further improvements. Below mentioned list exhibits possible extension on current protocol.

- **Controlling Abnormal Shutdowns** - changes can be introduced in-order to take care of log file being truncated during abnormal shutdown. Storing a message of abnormal shutdown in non-volatile method every time a new A_j is computed and irretrievably deleted once the log is entered. This might help τ and μ understand the abnormal events.

- **Voice Line Initialization** - Usage of DTMF tones to reduce the complexity of creating the logfile. Authenticating μ and format of M_0 through DTMF tones might increase initiation process

- **Peer-to-Peer** - Replacing τ with peer-to-peer model. Usage of this technique can be compared to emergence of block-chain technology.

3.3 Limitations

Every solution has its limitations either by the nature of its design or implementation. As perfectly said by the author *"no security measure can protect the audit log entries written after an attacker has gained control of μ"*, trying to control this device after compromise is futile, except the fact that the machine logged the event of compromise. Once compromised, the attacker can log on machine μ as needed. All that is possible is to stop the attacker from accessing logs before the event of compromise. Mechanisms are in place to know if there has been any attempt at tampering with logs logged before time t.

Entire mechanism is based on the assumption that the attacker cannot compromise machine τ. Was there a case of constant reliable high-bandwidth available, it would have been easy for the untrusted machine μ to write logs, encrypt and send it to τ. Moreover, there is no cryptography tool that can prevent log deletion. Only possible measure is the ability to detect that the files have been deleted.

Understanding of machine μ is important for smooth functioning of this protocol. The amount of time μ is online and frequency of μ expected to be compromised should be known, indicating availability of this protocol. For example, if μ is expected to be compromised every hour, it is necessary to make sure that μ communicates with τ atleast twice or thrice within an hour. In order to know if its possible to communicate with τ we shall be abreast of the time μ is online.

3.4 Conclusion

Although cryptographic tools prevent fraud to some extent, the main aim of using these tools are to detect fraudulent activities. It acts as an indirect piece of evidence to corroborate the detected fraud. An unalterable logging system would make the work of attackers much harder. This scheme, combined with physical tamper-resistance and periodic inspection of the insecure machines, could form the basis for highly trusted auditing capabilities.