

Towards a Secure Logging Mechanism for Dynamic Systems

Rafael Accorsi

Department of Telematics

Albert-Ludwigs-Universität Freiburg, Germany

accorsi@iig.uni-freiburg.de

Abstract

Logging is a central service in computing systems. It collects information about the events happening in (remote) devices and thereby provides a basis for other services within the system. However, in order to be useful, log data must be correct, for information derived from flawed sources is unreliable. In this paper, we report on ongoing work in developing a logging mechanism to securely store log data in marginally trusted remote collectors. To this end, we first characterize the security properties of log data and that of mechanisms to guarantee these properties. Further, we employ standard logging techniques to design protocols to achieve the security properties, and discuss the protocols' adequacy and limitations in guaranteeing these properties.

1 Introduction

Logging services are of primary importance in computing systems. Generally speaking, their function is merely that of collecting and archiving events communicated by devices (and/or sensors thereof) within the system. The information collected through logging builds the basis upon which several services are deployed. For example, log data is used to determine whether a system is properly configured, to provide a foundation for recovery services based on checkpoint/rollback techniques, and to detect and examine security incidents by means of audit tools. Recently, log data has also been used as evidence to decide on forensic disputes.

Irrespective of the envisaged application of log data, to be useful it must be sound, for *ex falsum quod libet*, i.e., from a falsity everything follows. Protecting log data is thus of foremost importance. However, the increasing distribution, complexity, and openness of computing systems pose a number of challenges to logging, in particular if the device notifying the events is not the same as the service storing this data. In this context, while the device's log data is stored in partially trusted collectors, accurate security guarantees regarding the integrity and confidentiality of log data should hold and, if necessary, be proved to the device.

In this paper, we report on ongoing work in developing a logging mechanism to securely store log

data in marginally trusted collectors. Our approach leverages the techniques proposed by [18] and, additionally, provides for non-repudiation of the collector's receipt of log data. To this end, we propose the corresponding protocols for initializing and updating log files, and proving collector's possession of the (correct) set of log data sent by the device.

Overall, our work contributes to clarifying the still obscure area of audit and accountability in dynamic systems. We consider *faithfulness* and *uniqueness* as the main properties of log data: while the former states that log data is a correct representation of the events the device observes, the latter states that this data cannot be arbitrarily reproduced. By ensuring these two properties, we guarantee that log data is an exact and inimitable picture of the reality. In addition to that, tamper evidence [12] mechanisms ensure that attempts to violate these properties will not go undetected, and forward integrity states that an attacker cannot read or alter log data archived before the attack [7]. We remark that, in general, logging mechanisms that fulfill these properties can also be misused to violate individual's privacy. Although we see privacy as a central issue, we deliberately do not address it in this paper.

We proceed as follows: in §2, we introduce the scenario underlying our research into logging. In §3, we characterize the meaning of security and define a threat model for logging services. We describe our method in §4, and discuss its adequacy and limitations regarding the expected security properties. In §5, we report on related work in secure logging and compare it with ours. We draw conclusions and state the upcoming research issues in §6.

2 Log in Dynamic Systems

Dynamic systems allow for the omnipresent availability of computing power, communication and (personal) data. This is a consequence of advances in distinct aspects of computing systems, namely autonomy [1], pervasiveness [16], and reachability [10], defining ubiquitous computing (UbiComp).

Apart from technical aspects, dynamic systems have distinguishing characteristics. Resource-rich and resource-poor devices share the same envi-



Figure 1: Log service setting.

ronment in a collaborative manner in that, e.g., resource-poor devices delegate the execution of computationally demanding tasks to resource-rich devices, or temporarily hire storage in remote devices. This amounts to a decentralization regarding the way in which tasks are executed, which eventually leads to a spread of data across the network.

In particular with regard to logging, resource-poor devices store their log data in remote services offering (secure) logging. While these services are trusted to carry out the logging protocol they agree on, they are not considered to be tamper resistant. Nor are they generally supposed to access log entries.

To obtain a more general scenario for the forthcoming discussion, we classify the components involved in logging according to their role [14]:

- *device* is a service that generates a log message;
- *relay* is a service that receives a message and forwards it to another service; and
- *collector* is a service that receives a message and, in one way or another, archives it.

This setting is illustrated in Fig. 1, where the arrows denote the flow of log data between the services. In this paper, we concentrate on the interplay between device and collector.

Traditional approaches to logging assume, however, that the device and the collector, albeit distributed, are within equally trusted domains. As a result, devices assume that the collector does not maliciously modify log messages. Similarly, collectors assume, e.g., that the log messages really originate at the claimed device. While these assumptions may hold in closed systems, they do not generally hold in dynamic systems. Thus, wider attack vectors arise and bring along new challenges for secure logging services.

3 Secure Logging

3.1 Requirements on Secure Logging

As we suggest above, log data can only be used if it is “correct.” We define correctness as the simultaneous fulfillment of the following properties:

- *faithfulness*: log data truly reflects the reality, that is, the information logged is a faithful representation of the messages sent by the device.
- *uniqueness*: log data shall not allow for parallel realities, i.e., it shall be impossible to intercept log data sent from device d_1 to collector

c_1 and to resend it (possibly in modified form and claiming a different device identity) to a collector c_2 .

These requirements characterize different integrity properties of log data and are implemented using different cryptographic techniques. Log services based on these techniques need to fulfill two other requirements, namely:

- *tamper evidence*: attempts to illicitly modify log data must be detectable to a verifier [12].
- *forward integrity*: a log service possesses the forward integrity property if it contains sufficient information to confirm or rebuke allegations of log data modification before the moment of the compromise [7].

Tamper evidence and forward integrity are necessary since absolute tamper resistance is infeasible [5]. This especially holds in dynamic systems [19]. We remark though that tamper evidence and forward integrity are not new and, to some extent, realized by existing log services, as we report in §5.

3.2 Threat Model and Attacks

The goal of the attacker is to access (confidential) log data and, thus, violate faithfulness and uniqueness. For this, the attacker uses different strategies, which we model using the Dolev-Yao attacker model [8] with the following characteristics:

- he can obtain any message sent over the net.
- he can compose, replicate, and send messages in his possession.
- he is a legitimate participant of the network. That is, he can act as or impersonate legitimate devices and collectors.

In order to model attacks upon stored log data, we extend the capabilities of the attacker. Namely, he can read, delete, and modify stored data. However, we assume that the attacker can only encrypt and decrypt messages and log data using the appropriate cryptographic keys.

Attacks upon log services have different facets and, as such, target different participants. Given the threat model above, faithfulness and uniqueness can be attacked as follows:

- *replay of log messages*: the attacker records a set of messages M sent by a device d to collector c . Later, he attacks d and, in order to hide collected evidence about the attack, sends a (possibly refreshed) set M to c .
- *integrity of sent and stored log data*: the attacker has access to the communication

medium and can modify data during the transmission. Moreover, if the attacker gains access to log files at the collector c , he can read and modify them at will.

- *authenticity of device*: the attacker takes over a service, modifies its identity, and starts to send log messages to a collector.
- *confidentiality of log messages*: during the transmission, the attacker intercepts and reads messages sent in clear-text over the network. Similarly, confidential log data stored in a collector may be accessible to an attacker.
- *availability of log service*: the attacker floods the collector c with irrelevant log messages, so that legitimate messages sent by the devices are not logged. The attacker can also use this setting to attack the devices sending messages to c , since his attempts are not protocolized.

These attacks may also be combined, as in [6].

4 Describing our Approach

Our approach to securing log data in dynamic systems builds on and extends the techniques to tamper evidence and forward integrity proposed in [18]. Our protocol consists of the following main steps:

1. *mutual authentication of device and collector*: apart from authentication, the two services also agree on a secret number that will be used to ascertain authenticity of log messages.
2. *initialization and construction of the logfile*: the device is in charge of applying cryptographic techniques to safeguarding the integrity of its logfile. With regard to how this log data is sent to the collector, we assume that chunks of log data are sent from the device to the collector. (As future work, we plan to investigate performance issues involved in sending each entry instead of chunks thereof.)
3. *acknowledgement of receipt from collector*: the collector computes a hash value based on the device’s messages and sends it signed together with a timestamp and protocol sequence number back to the device. The device then stores this unambiguous piece of information, as it demonstrates that the collector received the chunk of log data correctly and can be held accountable for attacks upon this data.

We focus on the last two steps. Regarding authentication, it can be tackled in several ways and our approach does not directly depend on the way it is achieved. In our experiments, we employ the Secure Sockets Layer (SSL) protocol, as it supports

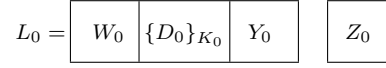


Figure 2: Initial log entry L_0 and authenticator Z_0 .

hierarchical authentication on the basis of certificates. (We remark, however, that this is not mandatory.) This is particularly useful when a collector is to archive the log data of several devices.

In describing our approach, we employ the following notation:

- c denotes the collector and d the device; D denotes log data.
- $\{X\}_K$ denotes the symmetric encryption of message X under the key K . $Sign(X)_K$ stands for the signature of X with K .
- X, X' stands for the concatenation of the messages X and X' .
- K_s stands for the public-key of the service s and K_s^{-1} the corresponding private-key of s .
- $MAC_K(X)$ denotes the message authentication code of X under K .
- $Hash(X)$ is the one way hash of X .

While we do not prescribe a particular algorithm to implement the cryptographic techniques above, we assume that these functions fulfill the expected properties in that, e.g., it is infeasible for an attacker to provoke collisions of hash values. These algorithms are described in, e.g., [17].

4.1 Initializing the Logfile

Assuming that the device and the collector successfully prove their identities to each other and agree on a secret value A_0 , d creates the log file by inserting the first entry into it.

All the entries in the logfile have the same format. We illustrate the initialization entry L_0 in Fig. 2, where the fields stand for the following information:

- W_0 is a permission mask to regulate the access to the log entry L_0 . According to [18], at the initialization phase this entry type may be set to **LogfileInitializationType**.
- $\{D_0\}_{K_0}$ is the symmetrically encrypted log data for the entry L_0 and K_0 is a random session key. To provide the necessary security guarantees, D contains not only the event to be logged, but also a timestamp d and a protocol identifier p . (The former states the actuality of the entry, the latter avoids harmful protocol interactions between different protocol steps.)
- Y_0 stands for the first link of a hash-chain.¹ The actual initialization value of Y_0 is left open.

¹In the simplest form, a hash-chain Y can be inductively defined as $Y_1 = Hash(Y_0)$ and $Y_n = Hash(Y_{n-1})$.

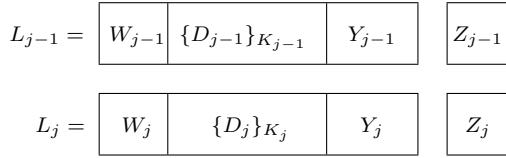


Figure 3: Adding an entry to the logfile.

- $L_0 = W_0, \{D_0\}_{K_0}, Y_0$ is the initial log entry.
- Z_0 is the message authentication code of L_0 defined as $MAC_{A_0}(L_0)$. This piece of information is used to compute the proof value associated to the whole chunk of log entries and, thence, is *not* sent along with L_0 to the collector.

The process of initializing the logfile at the collector side is interactive and subsumes the response of the collector in order to detect possible tampering.

4.2 Adding Log Entries

After creating the logfile and being sure that the collector is not tampered, the device starts adding entries to the logfile. This process is illustrated in Fig. 3, where the fields are computed as follows (observe that order is also relevant):

1. $A_j = Hash(A_{j-1})$ denotes the authentication key of the j th log entry. The confidentiality of this information is essential for the security of the log data, as it is used, either directly or indirectly, in every step of the protocol. Thus, we assume that the computation of the new value irretrievably overwrites the previous value.
2. $K_j = Hash(W_j, A_j)$ is the cryptographic key with which the j th log entry is encrypted. This key is based on the permission mask W_j . Thus, only permitted services may access the entry.
3. $Y_j = Hash(Y_{j-1}, \{D_j\}_{K_j}, W_j)$ is the j th value of the hash-chain. Each link of the hash-chain is based on the corresponding encrypted value of the log data. This ensures that the chain can be verified without the knowledge of the actual log entry.
4. $L_j = W_j, \{D_j\}_{K_j}, Y_j$, i.e., the log entry consists of the permission mask, the encrypted log data and the hash-chain value.
5. $Z_j = MAC_{A_j}(Hash(L_j, Z_{j-1}))$ is the authenticator of the j th log entry. Note that we compute the message authentication code for the whole entry instead of a field of it (in the case of Schneier and Kelsey, the hash-chain value Y). While this increases the computational cost involved in calculating this field, it improves the security of the acknowledgment phase.

This process describes how the j th log entry is computed. We address the overall security guarantees provided by this protocol phase in §4.4.

4.3 Acknowledgement Phase

The last phase of the protocol focuses on the collector and its goal is to provide irrefutable evidence regarding collector's possession of the chunk of log data sent by the device, as well as the chunk's integrity.

To this end, the following steps are carried out:

1. by receiving the chunk starting at L_j and ending at L_k (with $j < k$), the collector computes for each entry L_i the corresponding A_i and Z_i values.
2. after $k - j$ iterations, the collector obtains the authenticator $MAC_{A_k}(Hash(L_k, Z_{k-1}))$.
3. the signed proof value is sent to the device $Sign(MAC_{A_k}(Hash(L_k, Z_{k-1})))_{K_c^{-1}}$. This message includes a timestamp and protocol step identifier. Moreover, the collector irretrievably overwrites the memory space that stores A_k .
4. the device then checks whether the authenticator matches with the authenticator computed during the second phase. If it does, the collector frees the storage by deleting the chunk.

This concludes the logging protocol. The device can now ascertain the integrity of the logfile, as well as (selectively) delegate access to entries on this file. At present, we are still investigating how to perform these operations and the risks involved therein.

4.4 On the Required Security Guarantees

In §3.1, we suggest faithfulness and uniqueness as the most relevant security properties for log data. We now explore the relationship between these properties and the approach proposed above.

Assuming that the device accurately archives the events its sensors communicate, guarantees regarding faithfulness are safeguarded by the proof value sent by the collector in the last protocol phase. The message authentication code employed by the collector proves that it possesses the corresponding secret A and demonstrates that every log entry is a faithful representation of the original log data. It should also be noted that although the collector possesses the secret A , it cannot read the log data D sent by the device, as it is sent in encrypted form.

This fact is closely related with the concept of forward integrity. If an attacker takes over a collector c at time t , he is able to act as if c were not compromised. For this, he merely keeps sending the proof values as prescribed by the protocol. However, since he cannot obtain the previous values of A , it is impossible for him to read log entries archived before t . Thus, log entries up to t can be considered confidential and fulfill integrity properties.

Assuming that the cryptographic functions employed in the algorithms are sound, uniqueness is guaranteed by the use of timestamps and the hash-chains. Timestamps bring along a synchronization problem between the device and the collector. Although the probability of an attack due to timing confusion might be negligible in practice, the hash-chain works in a complementary manner and prevents old messages from being seamlessly and undetectably added to the chain.

This leads to the need for tamper evidence. The hash-chain employed in the protocol lets us verify the integrity of the whole chain by inspecting the last link in the chain. The proof value sent by the collector demonstrates that the log data has not been tampered with during the transmission or by a malicious process acting between the device and the collector. Hash chains also make it possible to detect whether (series of) log entries were eliminated.

4.5 Limits of the Logging Protocol

The logging protocol we present above is based on a number of assumptions, which outline and complement the underlying technical restrictions. We now briefly discuss the most significant limitations.

- Timepoint of a successful attack: the integrity of the logfile can only be ensured up to the point in time at which the attacker takes over the collector (or device). Once an attacker succeeds in compromising a machine, he can misuse log information at will.
- Collector’s behavior: unless it is compromised, the collector is assumed to follow the protocol. However, he is not trusted with respect to the confidentiality of log messages. As future work, we plan to investigate ways of ensuring—possibly employing approaches based on trusted computing—that the collector (as well as the relay) works as expected.
- Deletion of log entries: it is always possible for an attacker to completely (and irretrievably) delete the log file or entries thereof, in particular when we assume that no write-only hardware or media (e.g. WORM disk, paper print-out, optical jukebox) is at hand. Tamper evidence allows us to detect the deletion.

5 Related Work

A number of approaches have been proposed for securely logging information in computing systems. The majority of these approaches are based on `syslog`, a de facto standard log service whose functionalities are described in the RFC 3164 [14]. Security was not considered in developing `syslog`. In fact, it:

- provides no device authentication.

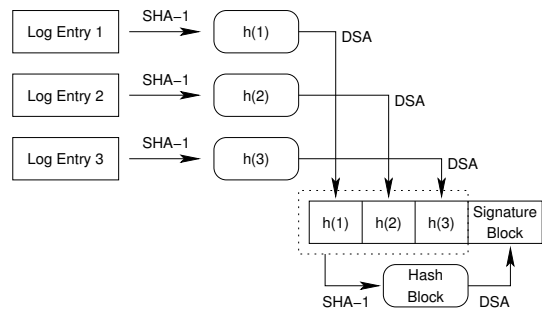


Figure 4: Creation of signature blocks in `syslog-sign`.

- ignores the original source of log data.
- transmits and stores log data in clear-text.
- uses UDP as underlying transport protocol.

Security was an afterthought. Indeed, various extensions to `syslog` have been developed.

syslog-ng. `syslog`’s new generation aims at providing a secure, reliable log service [4]. It is backward compatible with RFC 3164 and, currently, its only advantage over `syslog` is the use of TCP as transport protocol. Encrypted and signed transmission and storage are not provided.

syslog-sign. This service adds origin authentication, message integrity, replay resistance, and detection of missing messages to `syslog` [13]. This is accomplished with a cryptographically signed message that contains the signatures of previously sent `syslog` messages. The contents of this special message is called “signature block.” To our knowledge, no implementation of `syslog-sign` is available to-date.

Fig. 4 depicts how signature blocks are created. The cryptographic keys necessary to execute this protocol are calculated during the initialization phase of the protocol. `syslog-sign` does not provide confidentiality during the transmission of log data, nor does it account for encrypted storage of log entries. Moreover, since the signature blocks may be deleted after the authentication, tamper evidence and forward integrity are only partially fulfilled.

syslog-pseudo. Logfiles often store personal data and, thus, are a popular attack point. `syslog-pseudo` proposes an architecture to pseudonymize logfiles in unix-like systems [9]. This approach focuses exclusively on the privacy of users and does not take into account the security properties suggested in §3.1.

Reliable syslog. While the approaches above are based on the RFC 3164, the reliable `syslog` is based on the RFC 3195 [15] and aims to implement reliable delivery for `syslog` messages. For this, it is built on top of BEEP, the Block Extensible Exchange Protocol [2]. BEEP is a framework for building application protocols that uses TCP and allows device

authentication, mechanisms to protect the integrity of log messages, and protection against replay attacks. An implementation of reliable syslog by the San Diego Supercomputer Center is available [3].

Schneier and Kelsey. This method is the starting point of our investigation. In essence, it aims at realizing tamper evidence and forward integrity. In addition to that, mechanisms to protect the integrity of log entries, as well as their confidentiality and access control are presented [18].

However, the assumptions underlying our approach, as well as the procedures involved in achieving the security guarantees, differ in various ways. The authors assume that the device and the collector are the same machine. They also assume that the owner of a device is not the same person as the owner of the secrets within the device. In this context, log services must be in place to determine whether there has been some fraud. To our knowledge, this method has not been implemented.

6 Ongoing and Future Work

We have reported on work in progress towards a secure logging mechanism for dynamic systems based on UbiComp environments. The techniques we employ provide faithfulness and uniqueness of log data, as well as tamper resistance and forward integrity. The overall goal of our investigation is to provide secure transmission and storage of log data in marginally trusted machines.

This is merely the first step in our investigation and some issues remain to be examined. First, although our method provides new insights into the security properties and techniques for remote logging, at the moment it strongly relates to [18] and it is unclear whether simplifications are possible. A realization of the method shall shed a light on its performance and practical adequacy.

Second, the permission mask can be used by authorized services—e.g. audit or configuration services—to obtain log data directly from a collector. For this, protocols to regulate delegation should be in place and analyzed against the protection goals presented in §3.1 in order to avoid harmful protocol interactions. We plan to examine these issues and develop protocols for these purposes.

Third, in our approach we cannot make strong statements about the actual behavior of the collector. Indeed, the collector may be infected with malicious software that copies or sends log data to other hosts, or attempts to attack log entries with brute force techniques. To address this problem, we currently investigate the use of trusted computing. The idea is to remotely attestate collector's platform and, thus, its behavior [11]. Furthermore, we deliberately disregard the role of the relay. Using similar techniques we could also make statements about the relay's behavior.

Finally, if misused, such a powerful log mechanism is a privacy *endangering* technology, especially in an environment where virtually *every* piece of information can be collected and processed in order to derive further characteristics of an individual. We plan to address the privacy issues hidden in our approach to logging and whether one can draw benefits from it.

References

- [1] Autonomic computing initiative. <http://www.research.ibm.com/autonomic/>.
- [2] BEEP web site. <http://www.beepcore.org>.
- [3] Reliable syslog web site. <http://security.sdsc.edu/software/sdsc-syslog/>.
- [4] Syslog-ng web site. http://www.balabit.com/products/syslog_ng.
- [5] R. Anderson and M. Kuhn. Tamper resistance: A cautionary note. In *Proc. of the 2nd Workshop on Electronic Commerce*, pp. 1–11, 1996.
- [6] M. Bauer. Stealthful sniffing, intrusion detection and logging. *Linux Journal*, 2002.
- [7] M. Bellare and B. Yee. Forward integrity for secure audit logs. Tech. report, University of California at San Diego, 1997.
- [8] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208, 1983.
- [9] U. Flegel. Pseudonymizing unix log files. In *Proc. of the Conf. on Infrastructure Security*, vol. 2437 of *LNCS*, pp. 162–179, 2002.
- [10] G. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, 27(4):38–47, 1994.
- [11] A. Hohl, L. Lowis, and A. Zugenmaier. Look who's talking – authenticating service access points. In *Proc. of the 2nd Int. Conf. on Ubiquitous Computing*, vol. 3450 of *LNCS*, pp. 151–162, 2005.
- [12] G. Itkis. Cryptographic tamper evidence. In *Proc. of the Conf. on Computer and Communication Security*, pp. 355–364. ACM Press, 2003.
- [13] J. Kelsey and J. Callas. Signed syslog messages. IETF Internet Draft, 2005.
- [14] C. Lonvick. RFC 3164: The BSD syslog protocol. Request for Comments, 2001.
- [15] D. New and M. Rose. RFC 3195: Reliable delivery for syslog. Request for Comments, 2001.

- [16] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, pp. 10–17, August 2001.
- [17] B. Schneier. *Applied Cryptography*. John Wiley and Sons, Inc, 1996.
- [18] B. Schneier and J. Kelsey. Security audit logs to support computer forensics. *ACM Transactions on Information and System Security*, 2(2):159–176, May 1999.
- [19] F. Stajano. *Security for Ubiquitous Computing*. John Wiley and Sons, 2002.