available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Secure log management for privacy assurance in electronic communications

Vassilios Stathopoulos[a], Panayiotis Kotzanikolaou[a,b,*], Emmanouil Magkos[c]

[a]Authority for the Assurance of the Communications Security and Privacy (ADAE), 3 Ierou Lochou 15124 Maroussi, Greece
[b]University of Piraeus, Department of Informatics, 80 Karaoli and Dimitriou, 18534 Piraeus, Greece
[c]Ionian University, Department of Informatics, 7 Tsirigoti square, 49100 Corfu, Greece

ARTICLE INFO

Article history:
Received 15 June 2007
Accepted 9 July 2008

Keywords:
System logging
Network providers
Internal attacks
Integrity
Digital signatures

ABSTRACT

In this paper we examine logging security in the environment of electronic communication providers. We review existing security threat models for system logging and we extend these to a new security model especially suited for communication network providers, which also considers internal modification attacks. We also propose a framework for secure log management in public communication networks as well as an implementation design, in order to provide traceability under the extended security model. A key role to the proposed framework is given to an independent Regulatory Authority, which is responsible to maintain log integrity proofs in a remote environment and verify the integrity of the provider's log files during security audits.

1. Introduction

Communication privacy is considered as a valuable asset by the providers of electronic communication networks, such as Internet providers, fixed and mobile telephony providers. Indeed, incidents of privacy violations against their subscribers may cause severe impact with commercial and legal consequences. Moreover, security incidents in communication networks may also lead to abuse of services and economical loss for the providers. For these reasons, the network providers usually perform internal security audits with technical and procedural measures, in order to verify and maintain an acceptable security level. In addition to internal audits, in many countries Regulatory Authorities are responsible to regulate and externally audit the security level of public network providers, in order to preserve communications' security and privacy for the citizens.

Electronic communication networks are usually protected with state-of-the art technology which protects network perimeter and defines access controls, authentication and encryption mechanisms. Although security measures are in place in communication networks and well-defined standards exist there are still security holes. Threats such as external intrusion, communication interception, unauthorized access to private data such as the Call Data Records (CDRs) and abuse of privileges by insiders must be considered. Existing vulnerabilities such as overestimation of security measures, non-conformance with security measures and lack of dependable and secure logging and auditing mechanisms increase the security risks. Since it is not always possible to prevent security

* *Corresponding author*. Authority for the Assurance of the Communications Security and Privacy (ADAE), 3 Ierou Lochou 15124 Maroussi, Greece.
E-mail addresses: v.stathopoulos@adae.gr (V. Stathopoulos), p.kotzanikolaou@adae.gr (P. Kotzanikolaou), emagos@ionio.gr (E. Magkos).
0167-4048/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2008.07.010

breaches, it is required to have in place adequate detective security measures.

System logging is the most important detective security measure. Log files are maintained in almost every system and they are usually examined during security audits, either external or internal. Indeed, during regular security audits, log files may be examined and correlated, in order to assure that the intended technical measures are in place and that the security policies and procedures are implemented. During non-scheduled security audits, *e.g.* as a response to a security incident, log files are analyzed in order to discover the cause of the incident, such as lack of security measures, non-conformance with security procedures or system misconfigurations.

1.1. Our contribution

In this paper we are focusing on secure logging for electronic communication network providers. We propose an extended security model for logging in network providers, which also considers internal modification attacks. We also propose a framework for secure log management in public communication networks as well as realistic implementations, which are more resilient to the identified security threats. A key role to the proposed framework is given to an independent Regulatory Authority. Each provider is responsible to send integrity proofs of its log files to the Regulatory Authority, which in turn is responsible to securely store the integrity proofs and verify the integrity of the log files. Note that although the proposed framework focuses on telecommunication environments, it could be appropriately modified to apply to other environments facing similar threats and security requirements.

Our paper is motivated from the recently announced interception case in a mobile telecommunications provider in Greece (see for example Schneier, 2006). As the Greek authorities and the provider itself revealed, part of the core network of the provider was compromised by some unknown trojan-like program. According to published information, the malicious software infected the core network. Then, it activated the lawful interception (LI) component in the infected elements, which is by default installed in inactive mode, and made possible the call interception of several subscribers.[1] The malicious program turned off several logging procedures in order not to alarm about its presence or the fact that the LI component had been activated. The underestimation of several security threats and vulnerabilities regarding logging management and mechanisms did not allow the immediate detection of the incident.

The rest of this paper is organized as follows. In Section 2 we review the related work in secure logging. In Section 3 we describe an extended security model for system logging in telecommunication networks. In Section 4 we set requirements for logging in the extended security model. In Section 5 we propose a generic framework for secure logging which incorporates technical and procedural security controls under the guidance of a trusted authority, while in Section 6 we describe an implementation design. Section 7 analyzes the security and finally, Section 8 concludes this paper.

[1] The announced list of the victims included among others the Prime Minister, Ministers and Ex-Ministers.

2. Related work

First, we will examine existing security models for logging. Then, we will review secure logging systems proposed in the literature.

2.1. Security models for logging

In the general case, the logs are generated by one or more log *Generators* (devices, systems, software, etc.) and are sent to a *Log Server* through a relay mechanism. Existing threat models include

- *Trusted generators and marginally trusted log server*. In this model (*e.g.* Schneier and Kelsey, 1998) the logs are generated and relayed to the Log Server within a trusted environment. However, although the Log Server is protected, it cannot be guaranteed that it will not be compromised. Consequently, in this threat model the security attacks which are mainly considered are disclosure and modification attacks against the stored logs.
- *Distributed log generators and marginally trusted log server*. In this case (*e.g.* Accorsi, 2006) the log generators and the Log Servers are in a fully distributed environment. The Log Servers are considered, as in the previous case, marginally trusted. In addition to the previous model, attacks during the transmission of logs are considered. Since the logs are generated in a distributed environment, the log messages are not assumed by default to originate from the claimed device. Hence, attacks against the transmission of log entries are also examined such as impersonation attacks against log generators and/or the Log Server, and modification and disclosure attacks against log messages during transmission.

2.2. Secure logging systems

Several secure logging systems have been proposed in the literature. We group these into several categories.

2.2.1. Real systems

In real logging systems, the security of logging and auditing procedures is usually relied on the assumption that the host's operating system is not corrupted, which is a flawed assumption. Secure systems aim at improving the robustness of the logging system itself without relying on the security features of the underlying system. The *Syslog-sign* IETF draft (Kelsey and Callas, 2002) describes a logging system with message source authentication and message integrity, built above *Syslog*, a cross-platform standard for remote logging on a central repository. In the ReVirt system (Dunlap et al., 2002) the OS is moved to a virtual machine and the integrity of the logging system is protected against external attacks with OS-level privileges.

2.2.2. Tamper evidence

Cryptographic research in secure logging systems aims at building logs that are irrevocably tamper-evident. In a scheme

presented by Bellare and Yee (1997), the key that authenticates the logs entries is sequentially computed using an one-way cryptographic function, in order to achieve the *forward integrity* property, *i.e.* if an adversary compromises the current authentication key, she cannot modify old entries without being detected.

Schneier and Kelsey (1998) propose a secure logging scheme that detects attempts to delete or modify log entries on a host that has been compromised. The log entries are linked using a technique called *hash chaining* (Haber and Stornetta, 1990), where each entry contains a digest of the previous entry. Moreover, each entry is encrypted and authenticated using an ''epoch'' secret that is evolved using an one-way cryptographic function. The initial secret is shared with an external trusted party *T* who is able to independently verify the integrity of the logged data. Because the secrets used to encrypt and authenticate the log records are permanently deleted at the end of each epoch, an external attacker is not able to go back in time and disclose, modify or delete logged data, as this will be detected by *T*. This means that the scheme of Schneier and Kelsey (1998) satisfies the forward integrity property. Moreover, it allows the selective disclosure of the encrypted log data, using permission masks as an authorization list.

In recent works (Chong et al., 2002; Waters et al., 2004; Accorsi, 2006; Holt, 2006), the scheme of Schneier and Kelsey (1998) has been extended to provide for keyword searching on the encrypted data using public-key cryptography (Waters et al., 2004) to enable tamper-evident remote logging in resource-poor devices (Accorsi, 2006) or to detect software tampering in DRM systems (Chong et al., 2002). In another work, the *LogCrypt* system (Holt, 2006) extends and implements the scheme of Schneier and Kelsey (1998) to support public-key signatures for accountability and public verifiability of the submitted logs, while it also discusses secure aggregation of multiple logs in a distributed logging system. Secure aggregation of multiple logs for forensic computing was also addressed in a recent scheme that uses distributed *Merkle Trees* (Kawaguchi et al., 2005) for the collection of the log files.

2.2.3. *Tamper-resistance*

Deletion or altering of the log entries in untrusted systems cannot be prevented simply by cryptographic means; instead, some physical assumptions must be made, such as tamper-resistant hardware and/or physically secure communication channels. Tamper-resistance has also been employed in secure storage solutions and in securing database systems maintained in untrusted environments (Maheshwari et al., 2000). In storage-based IDS (Pennington et al., 2003), the IDS embedded in the storage device's firmware continues to work (in a tamper-resistant manner) even if an external source runs software with OS-level privileges. Log files may also be dispersed onto multiple hosts (*e.g.* Shen et al., 2004; Arona and Rosti, 1999), expecting that a non-negligible fraction of them will remain honest. Log files can also be replicated to increase availability, which may be expensive in terms of storage and network bandwidth. Other ways to protect logs from tampering also include networked file systems or cryptographic file systems for secure distributed storage (Kher and Kim, 2005), as well as Write Once Read Many (WORM) optical drives and continuous-feed printers. Such solutions are vulnerable to hardware faults or system attacks.

3. An extended security model: semi-trusted log generators and semi-trusted Log Servers

Existing threat models do not consider *insider attacks* and *collusion attacks* between log generators and Log Servers. Our threat model integrates and extends existing threat models in order to protect log files from attacks which have been identified within the environment of public network providers. For example, a possible threat may be that the log generators deliberately send modified log messages or that the stored logs are deliberately modified after their storage to the Log Server with the active participation of the Log Server administrator. A motive for such an attack would be to avoid the consequences of security problems during external security audits.

Our threat model assumes that all entities involved in the logging process are *semi-trusted*, including the generator(s) of log entries and the Log Server(s) that store the log files. The security of the proposed framework will be examined against a polynomial-time adversary, which may eavesdrop or manipulate the communication. Since the adversary is polynomially bounded, the probability that the adversary can decrypt messages or forge signatures is negligible, provided that the adversary has no access to the secret key used. We consider both passive and active adversaries. The passive adversary is able to eavesdrop on any message exchanged between log generators and the Log Server. The active adversary is able to inject erroneous or arbitrary messages. More specifically, we consider (i) modification attacks on the stored logs from compromised Log Servers; (ii) modification attacks on log events from compromised log generators; and (iii) modification attacks from colluding log generators and Log Servers.

Although during these attacks, collusions of compromised systems are allowed and the adversary may control the majority of the systems, we assume that the adversary cannot control all the Log Servers and all the log generators simultaneously. We believe that this assumption is justified since, if the attacker is allowed to actually manipulate all internal systems with full access privileges, then security cannot be established, unless a number of physical assumptions (such as tamper-resistance) can be tolerated. The adversary can also disrupt message delivery for a limited amount of time but is not allowed to ultimately prevent the delivery of messages.

4. Requirements for secure and verifiable log management

Logging information may provide accountability to the provider's network since it may be used to trace a possible misuse, mainly regarding system and network administration functions (such as commands to control operations or configuration). Bellow, we describe the basic functional and security requirements regarding the operation of a secure logging service.

4.1. Functional requirements

The basic functional requirements for secure log management in the examined security model are

(1) *Real-time and continuous logging*. The logging service should guarantee a continuous operation without interruptions or excessive delays. Logging interruption may cause loss of log events that may be critical for the audits.
(2) *Completeness*. The log files should contain all the necessary log events, in order to achieve a valuable audit. If logging information is missing, then this may cause invalid audit results.
(3) *Transparency*. Any operation related to logging (such as collection of logs, processing, storing and distribution of relevant information to third parties) should operate as transparently as possible.
(4) *Performance*. High scalability and stability are also two important requirements that should be taken into account.
(5) *Interoperability*. The logging mechanisms should be able to operate with most of the network and system commercial technologies hence generic interfaces should be the main concern.
(6) *Uniformity*. The content structure of the log files should be organized into discrete log files avoiding syntactical complexities and providing generic log information categories and offering a common formalization for all providers' networks.

4.2. Security requirements

The log management system should fulfil the following security requirements:

(1) *Confidentiality*. The log files should not be disclosed to unauthorized entities, since they contain information related with the communication.
(2) *Forward secrecy*. If all secrets concerning a period are revealed to an external attacker, the secrecy of past log entries should not be affected.
(3) *Integrity*. The log file should be protected from unauthorized modification for internal or external entities.
(4) *Forward integrity*. If all secrets concerning a period are revealed to an external or internal attacker, the integrity of past log entries should not be affected.
(5) *Access control*. Access to the log files should be controlled and only authorized entities should have access rights to particular entries.
(6) *Traceability*. Any read or write access to the log file should be traceable in order to be able to trace the source of an illegal action.

5. A generic framework for secure and verifiable logging

Provided that log files are important evidence, a security framework is required that will guarantee the availability, confidentiality and integrity of logging operations and log files. Based on the requirements described above, we propose a generic framework for secure log management, under the semi-trusted environment described in Section 3. We assume the existence of a trusted Regulatory Authority RA which is responsible to assure that the providers preserve communications' security and privacy. In regular audits or after a security incident, the RA may examine the log files of the provider in order to determine the cause of the incident. The framework consists of the following phases.

5.1. Defining what should be logged

Before any security measures are taken, it is important to explicitly define what is important to be logged. This decision involves both the provider and the RA. From the provider's side, an effective logging supports system maintenance, troubleshooting and internal security audits. From the RA's side, logging information helps in investigating the cause of a security incident and as evidence in a court of law.

In order to determine the events that must be logged within a provider, a *Log Reference Model* is defined, as shown in Fig. 1. This model is an abstract representation linking functions, *i.e.* general categories of network and operational events to the corresponding log *files* that monitor these functions, through the *services* which implement the functions. This model analyzes the logging needs from three different views, called planes. These planes are

(1) *Functional plane*. It models the network and operational events within a network, without taking into consideration implementation details, architectural or topology constraints and design requirements. Suggestively and not limitedly in a provider's environment the following categories of functions should be logged.
   (a) *Security functions* (*e.g.* system access control, password management, user management, lawful interception, data retention).
   (b) *Service management functions* (*e.g.* monitoring, troubleshooting, management services) and
   (c) *Network management functions* (*e.g.* network configuration, network connectivity, routing).
(2) *Service plane*. It describes all specific services which are executed within the network or IT nodes. It discriminates system from application services, while it takes into consideration the OS platform, communication protocols, interconnections and hardware. Examples of services of this plane are the SNMP service, the DSL service, the password management service or the AAA service.
(3) *Logging plane*. It describes specific commands and events of each used service, which can be grouped into separated log files. For example, the command ''show user'' (captured for displaying user names) will be logged in a log file named ''password management''. This log file will correspond to the password management service, which implements part of the security management functions.

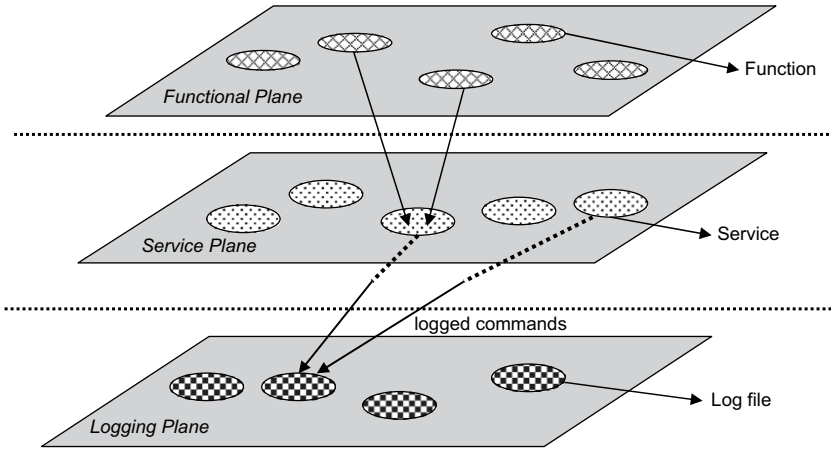

Fig. 1 – An abstract representation of a Log Reference Model.

5.2. *Defining the operational requirements of each log file*

After the list of functions to be logged has been mapped to services and consequently to log files, the operational requirements of each log file must be determined. In particular, requirements in this category define for each log file, the structure of the log file (*i.e.* the fields contained in a log file), the generation frequency, the storage requirements (*i.e.* the form, the duration and the location of storage) or any other requirement regarding the usage of log files for auditing purposes.

Organizing and managing the logging requirements depend on the agreement among the providers and the RA. This constitutes an administrative procedure without requiring any extra equipment at the providers' premises.

5.3. *Securing log files against external attacks*

In order to secure the log files from external and common internal attacks all the functions which have been identified in the previous phases must be securely logged. This can be based on standardized secure log systems such as the *LogCrypt* (Holt, 2006) or the Schneier–Kesley secure logging system (Schneier and Kelsey, 1998). We briefly describe this approach (see Fig. 2). The Log Server is supplied with an initial symmetric key A_0. The log file consists of consequent log entries $L_1,\ldots,L_n$. The key A_0 is updated for each new log entry, through a cryptographic one-way hash function hash, *i.e.* $A_i = \text{hash}(A_{i-1})$. Each log data entry L_i contains the log data D_i, which is encrypted and integrity protected. In order to encrypt the log data, a key K_i is derived from A_i, by hashing the concatenation of the key A_i with the permission mask W_i of the data entry D_i. Thus, $K_i = \text{hash}(W_i, A_i)$ and the encryption is $E_{K_i}(D_i)$. For integrity protection of the log entries, a hash chain is used. Each log entry L_i contains the hash value (for Y_0 a padding value is used) as well as the Message Authentication Code $Z_i = \text{MAC}_{A_i}(Y_i)$. Thus, each time only one MAC and one hash value are stored, which contain all the previously hashed results. This preserves forward integrity against outsiders, since if the key A_i is compromised to an adversary, the adversary cannot modify past log entries without being detected.

5.4. *Securing log files against internal attacks*

Although after each log entry L_i is stored in the log file and A_i has been updated to A_{i+1}, the previous key A_i is deleted, it is possible for a compromised Log Server to modify the log file. Suppose that the system is compromised at the time t_i. Thus the current key, say A_i, is revealed to the adversary and also that the adversary has access to the log files from the time t_i and after. The adversary does not change the log entries at that time. Then, at time t_j, $j > i$, the adversary modifies the log entries i, $i+1,\ldots,j$. By using the key A_i, the keys $A_{i+1},\ldots,A_j$ are reconstructed and the original log entries are replaced by the manipulated log entries. Such attacks are practically difficult to prevent if they originate from malicious insiders (*e.g.* the log file administrator) and the best we can hope for is traceability.

Note that a simple replacement of the MACs with digital signatures does not provide traceability against a compromised Log Server administrator. Indeed, although digital signatures would provide non-repudiation for the Log Server administrator, the storage of the signatures within the Log Server makes the system vulnerable to insider attacks. A compromised Log Server administrator would simply replace the log entries with the ones of her choice and then would compute new valid signatures for the modified log files. In addition to the above, if the MACs for all the log entries were replaced by digital signatures as in Accorsi (2006), the system would become totally inefficient and impractical for a large-scale application, such as within a network provider, where log entries are generated with extremely high frequency.

In order to enable traceability of such attacks, we extend the secure logging system of Schneier and Kelsey (1998) by considering a set of technical and organizational protective and detective measures. We make use of digital signatures under the supervisioning of a trusted Regulatory Authority RA. In addition with the secure Log Server described in the previous section, each provider will be assigned with two independent public/secret key pairs, PK_1/SK_1, PK_2/SK_2. These keys will be used to sign log files and log events, respectively,

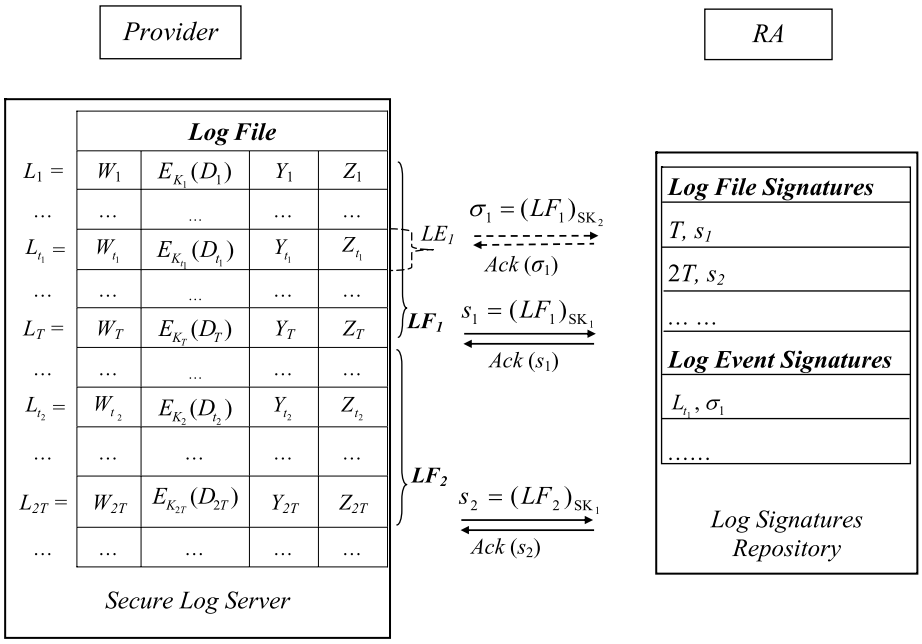

Fig. 2 – Securing log files from external and internal attacks.

as explained below. The signature keys must be certified through the corresponding digital certificates $Cert_1$, $Cert_2$. The digital certificates are issued by a mutually trusted external certification authority so that all the parties can verify the validity of the signatures generated with the keys SK_1, SK_2. The certificates are issued to the legal entity (the provider), although it is possible to additionally link them to the physical entities (the administrators), which will be authorized to access the corresponding secret keys. The key management functions such as generation, certification, revocation and updating of the signature keys may be supported by one or more independent certification authorities, which are trusted by the RA and the providers. For additional security, the distribution of the keys to the privileged users can be performed with off-line procedures, under the supervision of the trusted certification authorities.

The proposed procedure involves

(1) Asynchronous signing of the log files in predefined time periods.
(2) Real-time signing of the critical log events.
(3) Real-time signing of random log events.
(4) Remote storage of the signatures to the RA.

5.4.1. *Asynchronous signing of the log files in predefined time periods*

This procedure will ensure the integrity of the log files in discrete time periods. In order to provide integrity and accountability proofs for each log file to the RA, the provider periodically signs each log file and sends the signatures to the RA. Initially, the RA is responsible to initiate the procedure and define all necessary parameters such as the signature period *T*. The Log File Signatures are generated as follows:

At the end of the i_{th} logging period T_i, the provider instructs the Log Server to generate the hash value of the log file instance, say LF_i and then sign it by using the signature key SK_1 to produce the signature $s_i = (LF_i)_{SK_1}$ (see Fig. 2). Following, the signature s_i is send to the RA. Note that it is preferable that the log file signing operation is performed in an environment separated from the Log Server. In that case, a successful attack in the Log Server will not affect the security of the key SK_1. On receiving a signature s_i, the RA acknowledges it and stores it in a secure repository for future audits. The entire procedure is periodically repeated at the end of each period T_i.

5.4.2. *Real-time signing of the critical log events*

The asynchronous signing of the log files will permit detection of any log file modifications, which was realized *after* the signature has been send to the RA. However, it cannot detect modification attacks against the log file, which is realized *before* the signature has been generated. Moreover, the periodical signing of the log file cannot be useful if the modification attack is launched against the log events before these are stored into the log file. Such attacks may be executed during the generation of log events within a log generator or during their transmission towards the Log Server. In order to protect from such modification attacks, it is required that the log events are also protected *within* a log file signing period T_i.

Due to the fact that log event generation is a continuous process and vast amount of such events are produced in a provider's network, it is not practical to assume that all the log events can be real-time protected. For this reason, the RA has defined a list of *critical events*, *i.e.* events concerning actions which might be part of a malicious attack, in order to filter the information. Examples of critical events may include, system restart, service mode modification (*i.e.* starting a service or halting a running service), modification of users

and user privileges, modification of the log file and modification of the criticality level of a command.

The real-time signing procedure of the critical log events operates as follows: When a critical log event LE_j is generated by a log generator, this event is directly captured, signed and send to the RA. The Log Server does not intervene in the entire process. The log event LE_j will be signed with the key SK_2, producing the signature $\sigma_j = (LE_j)_{SK_2}$, which is send to the RA. Note that the critical log events are signed with SK_2, which is independent of the key SK_1 used for the signatures of the log files. In this way an adversary (internal or external) who compromised one of the keys will not be able to modify log files without being detected. On receiving a signature σ_j, the RA acknowledges it and stores the Log Event Signature to a repository for future audits.

5.4.3. *Real-time signing of random log events*

The protection of the log events cannot be based only on deterministic, predefined critical events, since this could be vulnerable to compromised insiders. For additional security, real-time signing can also be requested by the RA in random time intervals. In case where the RA issues a log event signing request, the provider will automatically retrieve the current log events from the log generator, sign them with the key SK_2 and send the signatures to the RA through the same interface as in the case of the critical log events. A random Log Event Signature request may be performed several times within a period T_i for one or more log generators, so that the RA has integrity evidence within a logging period.

5.4.4. *Remote storage of log signatures to the RA*

Although the log files are stored in the provider's side, the signatures of the logs (both log file and Log Event Signatures) are sent to the RA's side, by using a secure and always available channel. By committing to the log data and sending the commitments to the RA, even if the adversary gains access to all signing keys or any other long-term secrets, she will not be able to modify the log history and replace the actual signatures with the modified ones, without being detected, for any date before the time of the last commitment.

5.5. *Verifying log files*

In case of a security audit or after a security incident, the RA will retrieve the log files stored in the provider's side. Then, it will use the log signatures it retained along with the certificates $Cert_1$, $Cert_2$ of the provider's signature keys SK_1, SK_2 in order to verify the validity of the log file and Log Event Signatures, respectively.

During an audit, a Log File Signature s_i can be verified by the RA by using the stored signature s_i, the certificate $Cert_1$ and by accessing the actual log file LF_i retained in the provider's Log Server. A Log Event Signature σ_j can be verified by the RA by using the stored signature σ_j, the certificate $Cert_2$ and the actual log event LE_j stored at the provider's Log Server. In case where the signatures are not verified, the RA has evidence that the logs have been altered.

6. Implementation design

The implementation of the log signing procedures requires a distributed architecture with dedicated services. Fig. 3 proposes a generic implementation design, involving both the provider and the RA, which incorporates the following entities:

(1) The *Mediation Device*. It aims to have a central management and mediation role among Network Nodes and external authorities for the execution of each service. It communicates with the RA through a well-defined secure interface. It hosts a service execution software platform, the *Service Execution Environment* (SEE). The SEE serves the required business logic. This logic is implemented by the *Service Logic Program* (SLP). Each different service maintains its own SLP. Many requests, that involve the same service (*i.e.* the same SLP), may be served in parallel by generating different instances of the same logic. Moreover, an *Access Manager* is also hosted within the SEE that is responsible for implementing the communication interfaces with the required nodes. Finally, this device holds the secret key SK_2 that will be used to sign random log events and send them to the RA.

(2) The *secure Log Server*. This is a cryptographically enhanced Log Server, which also contains a Service Execution Environment (SEE). The SEE hosts the corresponding business logic, the *Logging Logic Program* (LLP), which implements the corresponding service. Moreover, an Access Manager is also hosted that is responsible for implementing the communication interface with the Mediation Device. Both the secure Log Server and the Mediation Device hold the main logic for the service management and for the execution of the services. The secure Log Server hosts specialized functions for executing the orders that are invoked by the Mediation Device. Hence, they should both communicate through a well-defined interface.

(3) The *Signature Server*. This is an isolated server that hosts the secret key SK_1. It implements one interface to receive signature requests from the Mediation Device. The secure Log Server and the Signature Server will have separate administrators. In this way, the Log Server administrator will not be privileged to access the key SK_1.

(4) The *Network Nodes*. This entity is used to model any network or IT element (routers, mail servers, etc.) within the provider's perimeter that generates log events, which are then stored to a Log Server. In other words Network Nodes are the log generators. Since our model considers the log generators as partially trusted, each Network Node hosts a dedicated *Agent*, which receives commands from the Mediation Device. The code of the Agents is signed by the RA in order to detect modifications of their program logic.

(5) The *Terminal Equipment*. This entity is hosted within the RA side. It initiates the services by invoking requests towards the Mediation Device and receives signatures from each Mediation Device of each provider. It hosts a management module, the *Signature Collection Manager*. It also stores these signatures within a Signature Repository in a secure place either within or outside this entity.

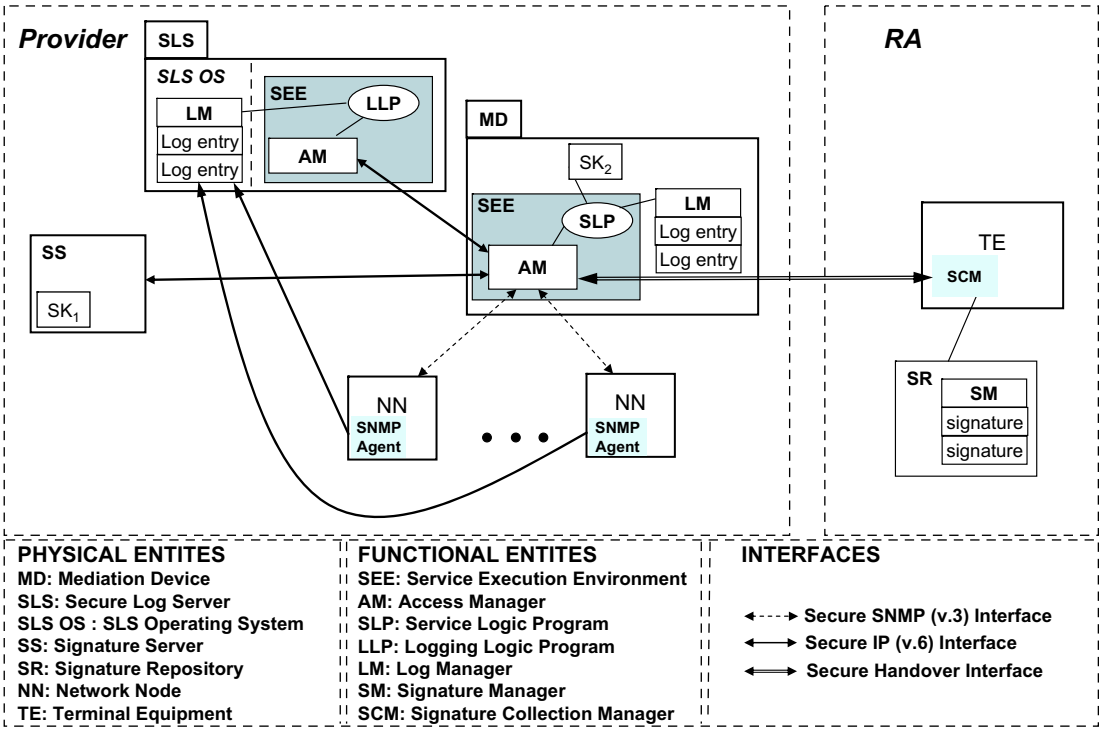

Fig. 3 – An generic implementation design of logging management.

In order to protect the communication between the various entities, three secure interfaces are implemented. A secure *Handover Interface* between the RA and the provider's Mediation Device allows encrypted and authenticated external communication based on certificates. The communication between the Network Nodes and the Mediation Device is based on the Simple Network Management Protocol (SNMP). In order to protect the transmission of the log events towards the Mediation Device, a secure version of the protocol (v3) can be applied. Finally, the communication between the Network Nodes and the Log Server can be protected by applying a secure version of IP (v6).

6.1. *Log signing services*

The proposed platform supports two log signature services: (i) the *Log File Signature* (LFS) service and (ii) the *Log Event Signature* (LES) service, which implement the corresponding procedures described above. Both services are composed by a complex set of variables and state machines. Hence modelling of this aspect requires the introduction of *sessions*. During the lifetime of a logging service various sessions of this service may be activated or deactivated.

6.1.1. *Log File Signature (LFS) service*

The LFS service (see Fig. 4) implements the asynchronous log file signing procedure described in Section 5.4.1. The LFS service operates as follows. Initially the Terminal Equipment (RA's side) invokes a *1:SetUp( )* message towards the Mediation Device for activating the first session of the LFS service. This message invocation handles authentication and authorization criteria, along with log file names and time related parameters such as the time period *T*. More than one sessions may be included in a *1:SetUp( )* message. The Mediation Device after receiving the *1:SetUp( )* message creates a new SLP instance for managing this session and then it invokes an *2:InitiateService( )* message towards the provider's Log Server. The Log Server also creates an LLP instance to handle the session.

The Log Server's ordinary job is to collect incoming log events by the Network Nodes. The Log Server classifies the collected log information into appropriate log files according to the rules that have been set by the Log Reference Model. Then the Log Server applies a hash function over the requested log file LF_i and returns the corresponding hashed value hash(LF_i) through a *3:Notify( )* message towards the Mediation Device. Following, the Mediation Device invokes a *4:Sign( )* method towards the Secure Signature Server and passes the hashed value. This interface is asynchronous. The Signature Server asynchronously signs the hashed values and invokes with a *3:Notify( )* message towards the Mediation Device. The Mediation Device forwards the corresponding signatures towards the RA within a *5:Response()* message, through the secure Handover Interface.

In case that the RA wants to negotiate for additional logging information it requests the activation of a new session by invoking a new *1':SetUp( )* message. The mediation device keeps track of all active sessions and refreshes its holdings with the new contents. It filters the information and invokes a *2':RequestReportChange( )* message towards the Log Server with the new requirements. The Log Server may reply back to confirm or deny the changes through a *3':ReportChange( )* message. Log File Signatures are sent back to RA through the same procedure. The RA may finally release some or all of the

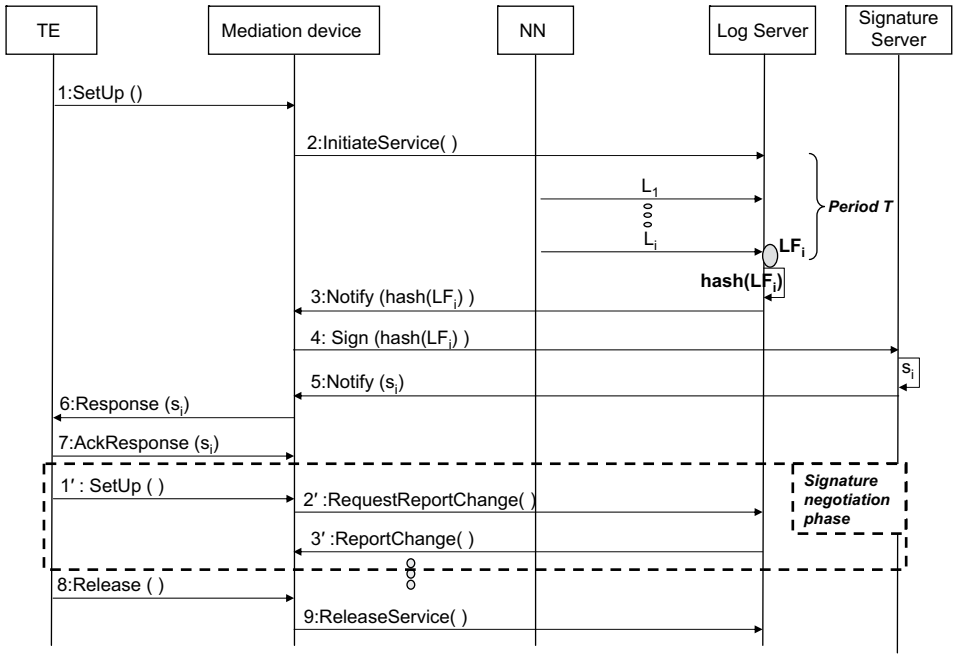

Fig. 4 – Log File Signature (LFS) Service.

active sessions that has requested by invoking a *8:Release( )* message.

6.1.2. *Log Event Signature (LES) service*

The LES service (Fig. 5) implements both the critical and the random event signing procedures, described in Sections 5.4.3 and 5.4.4, respectively. The LES service is activated by the RA through a *1:SetUp( )* message. Similarly to the LFS service, the authentication, authorization and other important information are passed through appropriate arguments of the *1:SetUp( )* message.

The critical event signing procedure is implemented as follows: The Network Nodes (log generators) send ordinary log events towards the Log Server. The Agents within the Network

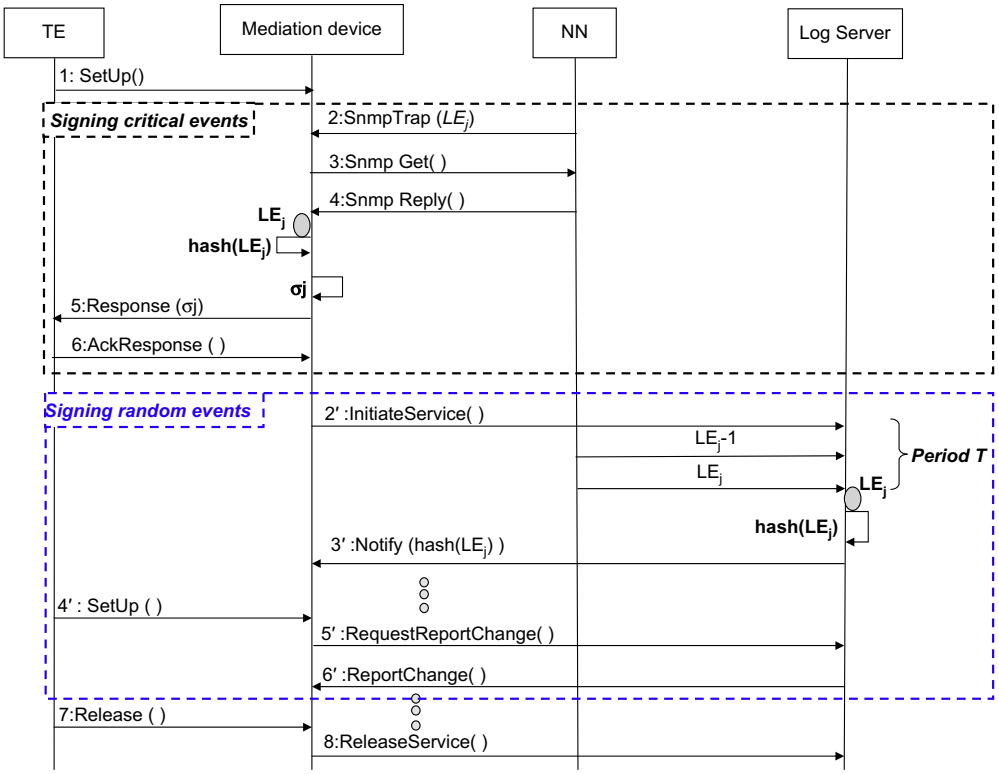

Fig. 5 – Log Event Signature (LES) Service.

Nodes are programmed to send alarms by using the format of the SNMP traps as an 2:*SnmpTrap( )* message, concerning only the critical log events. The Mediation Device automatically creates a new SLP instance for managing the session request. The Mediation Device may also request additional logging information from the Network Nodes through 3:*SnmpGet( )* and 4:*SnmpReply( )* messages. The Mediation Device groups these events, generates hash values and then signs the log events. The signatures are sent to the RA within a 5:*Response( )* message, through the secure Handover Interface.

The random event signing procedure includes the following steps. The Mediation Device collects the current log events (critical or non-critical) from the Network Nodes and from the Log Server for further processing. The log events collected from the Network Nodes are processed as in the case of the critical event signing, whereas the processing of the log events collected from the Log Server involves the following steps.

A *1:SetUp()* message initiates a new SLP instance which instructs the Mediation Device to send a *2':InitiateService()* message towards the Log Server, requesting the hash values for the selected log events. Similarly to LFS service, the Log Server generates hashes of the requested events and sends the hashed values towards the Mediation Device through a *"3':Notify()* message. The Mediation Device signs these values and forwards the signatures towards the RA. Finally, the messages *5':RequestReportChange()* and *6":ReportChange()* are used for requesting additional logging information from the Log Server.

7. Security analysis

We examine how the proposed system satisfies the security requirements set in Section 4.

7.1. Confidentiality

Since the system applies a secure Log Server system (such as Schneier and Kelsey, 1998; Holt, 2006) it inherits its security services. The Log Server encrypts each log entry and only the Log Server administrator can have access to the stored entries. By applying the principle of *dual control*, the initial key A_0 can be securely maintained by a different administrator. In this way, it will not be possible for a compromised Log Server administrator to violate the confidentiality of the log entries. Moreover, the confidentiality of the log events during their transmission from the Network Nodes (the log generators) towards the Log Server is protected since encrypted channels are used, as described in Section 6.

7.2. Forward secrecy

This property is also provided by the secure Log Server. Since the encryption key is updated for each log entry with an one-way function, an intruder who manages to access the encryption key at a given time will not be able to violate the confidentiality of past log entries.

7.3. Integrity

The asynchronous signing of the log files provides integrity proofs, for modifications which have taken place *after* the file has been signed and the signatures have been sent to the RA. In that case, the modified log file will not match the corresponding signature, which is remotely stored within the RA. The use of a signing period T sets a time limit frame to the adversary. Note that while a very long time period T would reduce the integrity protection of the protected log files, a selection of short time periods would not be efficient. According to the security needs, it is recommended that T is between 1 day to 1 week.

In addition, since the real-time integrity protection of each log entry is not practical, the real-time signing of at least the critical log events provides a balanced protection. Moreover, the use of real-time signing of random log events increases the uncertainty for an attacker, since even if the attacker is an insider with advanced access privileges, he cannot be assured whether an event will be signed and sent to the RA.

7.4. Forward integrity

Forward integrity against an external attacker is achieved due to the use of the secure Log Server system with the one-way updating of the MAC keys. Forward integrity against internal attackers is achieved through the log file and log event signing and the remote storage of the signatures. Since the RA stores the log file and Log Event Signatures, even if an insider has access to the signature keys, he will not be able to modify signatures generated in previous periods and modify the corresponding log files, without being detected.

7.5. Access control

Apart from the technical access control measures which apply to the proposed system, we make use of procedural access control measures such as the *separation of duties*. The keys SK_1, SK_2 used for log file and log event signing, respectively, are independent keys which are installed and administered in separated environments. Additionally, the log files are stored in a separate environment from the signature environment of the log files. Thus access to the Log Server does not imply access to the signature key used to protect the log file. Consequently, even if the attacker has compromised security sensitive systems such as the Log Server, he cannot be assured that his attack will not be traced. The attacker will not able to modify the signatures of the existing log files also containing traces of his actions.

7.6. Traceability

During security audits, the RA examines the log signatures stored in its repository against the log files stored in the provider's side.

If the Log File Signatures and the Log Event Signatures for the examined period are verified against the log files of the provider, then the RA has strong indications that the logs are valid. If the Log File Signatures for the examined period are verified, while the Log Event Signatures are found invalid, then the RA has evidence that the log events have been manipulated within the Log Server, since the Log Event Signatures are based on events captured at real-time during their generation. If both log file and Log Event Signatures cannot be verified, the RA has evidence of manipulation of the log files.

Finally, another measure to trace illegal behavior within the log generators is the use of signed code for the software

entities that support the logging functions, such as the SLPs installed in the Mediation Device and the alarm Agents installed in the Network Nodes. In this way, if there has been an attempt to manipulate the code of these Agents, the RA will be able to trace such an attempt, since such actions would generate critical events towards the RA. Then the RA will verify whether the running Agents have been modified by checking their signed code.

8. Conclusions

Existing secure logging systems cannot protect or detect attacks against the integrity of the log files from internal attackers. In public communication networks, however, the security requirements of log files must also consider internal attacks such as compromised log generators, compromised Log Servers or combinations of both. In this paper we consider an extended security model for logging systems and we define a generic framework for secure logging for public network providers. Through the proposed framework the logging requirements of each provider are defined, as well as the required security measures and procedures for the protection of the log files. A trusted Regulator Authority RA has a central role in this framework, in the definition of the logging requirements as well as in the storage of log integrity proofs and in the verification of the maintained log files. In addition with known security measures for secure logging, we propose the use of digital signatures in two different ways, as well as the remote storage of the signatures in the environment of the RA. Although modification attacks against log files cannot always be prevented, it is feasible to build tracing mechanisms which will impose dual controls and separation of duties and will obviate internal attacks.

REFERENCES

Accorsi R. On the relationship of privacy and secure remote logging in dynamic systems. In: Security and privacy in dynamic environments, vol. 201. Springer-Verlag; 2006. p. 329–38.

Arona A, Rosti DBE. Adding availability to log services of untrusted machines. In: Fifth annual computer security applications conference (ACSAC'99), Phoenix, AZ, USA; 1999.

Bellare M, Yee B. Forward integrity for secure audit logs. Tech. rep.. Computer Science and Engineering Department, University of California at San Diego; November 1997

Chong CN, Peng Z, Hartel PH. Secure audit logging with tamperresistant hardware. Tech. rep.. Enschede, The Netherlands: Universiteit Twente; August 2002

Dunlap GW, King ST, Cinar S, Basrai M, Chen PM. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. In: Proc. 2002 Symp. Operating Sys. Design and Implementation; 2002.

Holt J. LogCrypt: forward security and public verification for secure audit logs. In: Proceedings of Australasian information security workshop; 2006.

Haber S, Stornetta W. How to time-stamp a digital document. In: Menezes A, Vanstone SA, editors. Proc. of CRYPTO'90. Lecture Notes in Computer Science, vol. 537. Springer-Verlag; 1990. p. 437–55.

Kelsey J, Callas J. Ssyslog-sign protocol. DRAFT. Network Working Group; June 2002.

Kawaguchi N, Obata N, Ueda S, Azuma Y, Shigeno H, Okada K. Efficient log authentication for forensic computing. In: Proceedings of IEEE sixth information assurance workshop. IEEE; 2005. p. 215–23.

Kher V, Kim Y. Securing distributed storage: challenges, techniques, and systems. In: Proceedings of the first international workshop on storage security and survivability (StorageSS'05). ACM; 2005.

Maheshwari U, Vingralek R, Shapiro W. How to build a trusted database system on untrusted storage. In: Proceedings of the USENIX symposium on operating systems design and implementation; 2000. p. 135–50.

Pennington A, Strunk J, Griffin J, Soules C, Goodson G, Ganger G. Storage-based intrusion detection: watching storage activity for suspicious behavior. In: Proceedings of 12th USENIX Security Symposium, Washington, DC; 2003.

Shen Y, Lam T, Liu JC, Zhao W. On the confidential auditing of distributed computing systems. In: Proceedings of 24th international conference on distributed computing systems; 2004.

Schneier B. Schneier on security: phone tapping in Greece. Available from: http://www.schneier.com/blog/archives/2006/02/phone_tapping_i.html; 2006.

Schneier B, Kelsey J. Cryptographic support for secure logs on untrusted machines. In: Proceedings of the 7th USENIX security symposium. USENIX Press; 1998. p. 53–62.

Waters B, Balfanz D, Durfee G, Smetters D. Building an encrypted and searchable audit log. In: The 11th annual network and distributed system security symposium; 2004.

Vassilios Stathopoulos is a security auditor at the Hellenic Authority for the Assurance of the Communications Security and Privacy of Greece (ADAE). He received his BSc degree in Physics (1996) from University of Athens, his MSc degree in Communication, Control and Digital Signal Processing (1997) from University of Strathclyde, Glasgow, UK and his PhD degree (2001) from National Technical University of Athens (NTUA). He has participated, as research associate, in several European Projects. His research interests are in the field of computer network security, service creation and control and distributed processing.

Panayiotis Kotzanikolaou is a security auditor at the Hellenic Authority for the Assurance of the Communications Security and Privacy of Greece (ADAE) and a visiting lecturer at the Department of Informatics, University of Piraeus, Greece. He received his BSc in Computer Science (1998) from the University of Piraeus, Greece and his PhD (2003) in information systems security, from the same university. His research interests include network and information security, communication privacy, security protocols, and applied cryptography.

Emmanouil Magkos is a lecturer at the Department of Informatics, Ionian University, Corfu, Greece. He received his Degree in Computer Science (1997) and his PhD (2003) entitled 'Secure electronic transactions over the Internet' in the Department of Informatics at the University of Piraeus, Greece. His research interests include information security and cryptography, wireless networks and distributed systems.