A PKI-BASED SECURE AUDIT WEB SERVICE

Wensheng Xu, David Chadwick, Sassa Otenko
Computing Laboratory, University of Kent,
Canterbury, CT2 7NZ, England
{w.xu, d.w.chadwick, o.otenko}@kent.ac.uk

ABSTRACT
For many applications, access control and other business related information of all user transactions should be kept in secure log files for intrusion and misuse detection or system audit purposes. Because the log files may be stored on or moved to an untrusted machine and may attract attackers because of the large amounts of potentially sensitive information contained in them, we would like to guarantee that in the event an attacker gains access to this machine, we can limit his ability to corrupt the log files and we are able to detect any compromises afterwards. We also may want to ensure that he can gain little or no information from the log files. In this paper we propose a secure audit web service (SAWS) which can provide a secure audit trail service for multiple clients. The secure audit trail generated by SAWS can be stored on any untrusted machine and it is impossible to be modified or destroyed without detection, and its integrity can be validated by any client. Optionally, the audit file can be encrypted, making it impossible for unauthorised parties to read its contents.

KEY WORDS
Secure audit trail, Public Key Cryptography, Web Service, secure hash, trusted computing base

1. Introduction

With the widespread use of the Internet, more and more resources and services are available on the web. To better help resource web sites improve their system security and reliability, a history log or audit trail is usually necessary to record all the accesses to the system, so that these can be later inspected either daily or periodically during system audits. An audit trail is especially important and necessary for access control services as it forms a significant part of the front-line defence for detecting system misuse or intrusion attempts. Unfortunately, since a large amount of sensitive information e.g. those related to access control decisions, may be contained in the audit trail, and the audit trail may be stored on or moved to an untrusted machine, this may attract attackers to try to read or alter the log records, e.g. remove traces of their actions from it. We would like to guarantee, in the event that an attacker does gain access to this machine, that although we are unable to stop him altering or removing the log records or the whole audit trail, nevertheless we are able to limit his ability to undetectably corrupt the log files and are able to detect the compromise afterwards. Optionally and in addition, we can ensure that he will gain little or no information from the log files by encrypting them prior to storage. This secure audit trail capability is crucial for many applications in order to prevent audit trails from being tampered with undetectably. Whilst there are already some secure auditing schemes for applications Schneier and Kelsey [1] rely on a central trusted machine, whilst Chong et al [3] rely on a Java iButton and their system is only for Digital Rights Management. For a virtual organization or distributed application which spans multiple servers, multiple application components will produce log records in different digital formats that contribute to the whole system security analysis, in which case a centralised secure audit trail system which provides general auditing services may be required. In this paper, a PKI-based secure audit web service (SAWS) is described which records audit information for distributed applications in virtual organizations.

The primary functionality of a secure audit service is to provide permanent secure storage for log records, so that it can reliably detect when tampering has occurred. Schneier and Kelsey have developed a cryptographic mechanism for securing the contents of an audit log against unauthorised reading, which provides tamper detection [1]. The mechanism relies on symmetric encryption, with a central trusted server holding the decryption keys. Reading and verification of the log records is accomplished with the help of the central trusted server. But because of this, when the central trusted machine is not available, then it’s impossible for users to read and verify the audit trails – this could cause an inconvenience for users, a central point of failure, a central point of attack, and potentially a bottleneck to performance.

In this paper, we modify Schneier’s scheme by using public key cryptography to enable independent reading and verification of the audit logs, without the need for the central trusted machine. We also use the recent developments in Trusted Computing Bases (TCBs) [2] to store the private and secret keys of the audit service, so that the external central trusted server is no longer needed for this either. The remainder of this paper is structured as follows: In section 2, the requirements for our secure audit web service (SAWS) are described. In Section 3, the architecture of SAWS is presented. Section 4 describes the security mechanisms adopted in SAWS. Section 5 we present a discussion and summary of this paper.

2. Security Requirements for the Secure Audit Web Service

User access requests and activities should be collected and sent to SAWS, so that administrators can at a later point in time know who did what, including who was given access to which resources, and who was denied access, for example when wanting to track down an attacker. Since several applications may share the same SAWS, this leads to the following basic security requirements:

- Append mode of access: Only append mode of access should be allowed, so that users or applications cannot rewind the audit file and delete or modify information that has already been stored there
- Authorised writing: Only authorised parties should be able to append log records to the audit trail. Though unauthorised applications or attackers may gain access to the audit trail and try to append fake log records to the audit trail, or modify or remove the audit trail, this should be detected by the tamper detection mechanism.
- Timestamps: Every record in the audit trail should be timestamped by SAWS to provide a trusted record of when the audit data was received. We note that if SAWS is trusted to record the audit data without tampering with it, then it should also be trusted to append the correct time to the data. Therefore we do not propose to use a secure time stamping service. If this is insufficient for some applications, then because the format of the recorded data in each log record is application-specific and is determined by the client applications themselves, they may contain another timestamp provided by the client for cross checking purposes. However the use of the latter is application dependent.
- Secure communication: The communications between a SAWS client and the SAWS server should ensure tamper resistance, data integrity and authorised connection.
- Secure storage on untrusted media: Since an audit trail may be stored on untrusted machines, the SAWS security mechanism should ensure persistent and resilient storage of the audit trail, and ensure detection of tampering of the audit trail – modification, deletion, insertion, truncation, or replacement. If tampering is detected, SAWS should be able to notify the security auditor.
- Support multiple simultaneous clients. SAWS should be easily and conveniently accessible via a web service interface, and it should be able to serve multiple client applications simultaneously.
- Performance efficiency: The performance of SAWS should be as efficient as possible. Since our initial target client is the PERMIS authorisation system, and this can make 500 access control decisions per second on a PC [7], we made 500 records per second the minimum performance requirement for SAWS on the same platform.
- Contents transparency: SAWS should be able to record any digital content coming from any SAWS client.
- Confidentiality and authorised reading: Since the audit trail may contain sensitive information, then the secure audit mechanism should optionally be able to ensure that only authorised applications or people have the privilege to read the audit trail.

Based on the above requirements, we propose the following architecture and security mechanisms for SAWS.

3. Architecture of the Secure Audit Web Service

There are three types of application in the SAWS system – the SAWS server, the audit trail viewing tool (VT), and the SAWS client. The SAWS server is issued and configured with two public/private key pairs – an encryption/decryption public/private key pair and a digital signing/verifying private/public key pair. Optionally, the SAWS clients and the VT may all be issued with their own encryption/decryption public/private key pairs. The encryption/decryption public/private key pairs are used for confidential transmission of information between the different components, whilst the signing/verifying private/public key pair is used by the SAWS server to digitally sign the log file so that any application can verify the integrity and authenticity of the log file by using the SAWS server's public key certificate. The structure of SAWS is shown in Figure 1 (The VT is not shown in this figure).

The core component in the SAWS server is the Java Secure Audit Trail Service (J-SATS). J-SATS is responsible for receiving log messages from SAWS clients, for securing them (as described in Section 4) and then writing the secured audit records into one or more permanent audit log files on untrusted machines.

Three different client interfaces are provided for SAWS to facilitate different application scenarios:

- a Java programmable interface, the J-SATS API,
- a direct SSL-TLS socket communication interface for TCP based clients and
- a SOAP server over SSL interface for web service based clients.

When the client is remote from the SAWS server and is accessing it via the Internet, to ensure secure communications between SAWS clients and the SAWS server over the Internet, we adopt SSL to ensure mutual authentication, message integrity and message confidentiality. We did initially intend to use WS-Security [6], but the performance of this only allowed us to process 2 records per second which was an order of magnitude worse than SOAP over SSL (which was still an order of magnitude worse than our performance target). To meet our performance target of 500 records per second, we had to use the Java API.

the log record is stored in the TCB. This allows detection of a truncation attack.

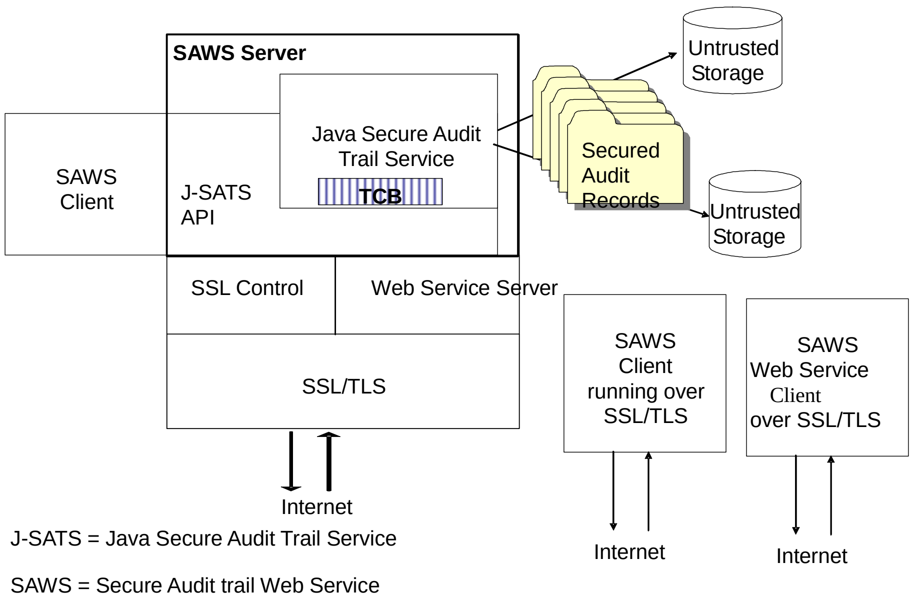
SAWS Server
Java Secure Audit Trail Service
TCB
SAWS Client
J-SATS API
Secured Audit Records
Untrusted Storage
Untrusted Storage
SSL Control
Web Service Server
SSL/TLS
SAWS Client running over SSL/TLS
SAWS Web Service Client over SSL/TLS
Internet
Internet
Internet
J-SATS = Java Secure Audit Trail Service
SAWS = Secure Audit trail Web Service

Figure 1. Architecture of the Secure Audit Web Service

4. Security Mechanisms of SAWS

4.1 Secure Storage of the Audit Trail

To ensure secure storage of the audit trail on an untrusted machine, the following measures are adopted.

1) When starting a new audit record file, the SAWS writer generates a new random secret key *RN*, which is then stored securely by SAWS so that only it can recover it after a crash. *RN* is stored in the TCB and is also encrypted with the public encryption key of the SAWS server and stored as the first record of the new audit file. *RN* will be used in the calculation of the secure hash which is appended to the end of each log record, in order to detect modification of a record's contents before the audit file is finally digitally signed and closed i.e. in the case when SAWS prematurely crashes.

2) The SAWS server places the file name and digital signature of the previous audit file as the second record in the new audit file. This chains the log files together and stops an attacker from completely deleting one or more audit files without detection. The digital signature stops an alternative authentic file being substituted for the correct one by renaming it.

3) Each record is given a sequence number which prevents records from being inserted or deleted in the middle of the audit trail. To detect truncation of the file from the end after a crash, the current sequence number of

4) SAWS adds the authenticated name/ID of each client to each record they submit before writing it to the audit trail. This is to stop one client masquerading as another in the data that it submits. SAWS uses its own ID for each system record that it writes.

5) SAWS keeps a plain accumulated hash of the entire audit file as the log file is built up. This is stored in the TCB (along with the current sequence number) in order to detect a replacement attack after a crash (i.e. an attacker uses an old version of the audit trail to replace the latest version of the audit trail after his intrusion into the storage system).

6) When the audit file is complete, the plain accumulated hash is written to the end of the file and the plain accumulated hash is digitally signed with the signing private key of the SAWS server. The signature and public key certificates of SAWS are also written to the file. Now anyone can verify the completeness of the file with the verifying public key of the SAWS server.

7) Optionally the audit file can be stored in several different computers in different locations to defend against truncation and replacement attacks. This will also allow recovery in cases where one or more (but not all) copies are tampered with. Of course, the more copies there are at different locations, the more chances there are of compromise. But since our purpose is to detect tampering rather than to prevent it, and to ensure that at least one genuine copy is preserved for audit purposes, then as long as not all the copies are attacked our system is secure.

8) If confidentiality is required, optionally SAWS

Sn_j	UID_j	ST_j	T_j	L_{j-1}	L_j	$Encrypt_j$	M_j	H_j

Figure2. Format of a Log Record in the Secure Audit Trail

can generate a random symmetric encryption key which is subsequently used to encrypt the audit records. The key is encrypted using the SAWS encryption public key and the viewing tool public key and both copies are stored in the audit file. Optionally the key can also be encrypted with the public keys of all the SAWS clients, so that they can each independently verify and view the audit log.

9) After every 1 second in the idle state, the SAWS server writes a heartbeat record to the audit file, so if a system crash happens at any time we can know the time of the crash to the nearest second.

4.2 Log Record Format

The format of a SAWS log record is shown in Figure 2. All log records in the log file use the same format. The *j*th log record LR_j is defined as:

$$LR_j = Sn_j \,||UID_j \,||\, ST_j \,||\, T_j \,||\, L_{j-1} \,||\, L_j \,||Encrypt_j \,||\, M_j \,||\, H_j$$

(1) Sn_j is the sequence number (4 bytes).

(2) UID_j is the User ID of this record; it indicates the identity of the client that provided this log record. Every SAWS client is assigned a unique user ID after authentication, and this mapping is held in the **SysClientID** record (see later). The UID_j = 0x00 is reserved, and indicates that the log record is written by the SAWS server itself (1 byte).

(3) ST_j is the log record type (1 byte).

(4) T_j is the timestamp of the *j*th log record in format of a long integer (8 bytes).

(5) L_{j-1} is the length of the last log record LR_{j-1} (4 bytes).

(6) L_j is the length of the current log record LR_j (4 bytes).

(7) $Encrypt_j$ is the encryption indicator for the encryption method used for this log message. It could be **SymmetricEncryption** (0x01), **AsymmetricEncryptionForSAWS** (0x02), **AsymmetricEncryptionForVT** (0x03), **AsymmetricEncryptionForClient** (0x04), or **NoEncryption** (0x00) (1 byte).

(8) M_j is the *j*th log message received from the SAWS client or from the SAWS server itself (indefinite bytes).

(9) H_j is the secure hash value for this record (20 bytes).

(10) || represents the concatenation operation.

Generally there are the following types of log records in the audit trail: **ClientLogData**, **SAWSRandomNumber**, **SymmetricEncryptionKey**, **SAWSLastFile**, **SAWSAccHash**, **SignatureRecord**, **SAWSCert**, **SysSAWSStartup**, **SysSAWSShutdown**, **SysHeartbeat**, **SysUnauthorisedConnectionAttempt, SysAuditorNotification**, **SysClientID**. For log records coming from SAWS clients, their log record type should be **ClientLogData**. The other log record types are for holding SAWS support data for the audit trail.

4.3 Initialisation, Validation and Recovery of the Secure Audit Trail

When the SAWS server first starts up it generates two asymmetric key pairs – the SAWS encryption/decryption key pair and the signing/verifying key pair. Both the private keys will be written to and protected by the TCB if this is available. In addition, the encryption/decryption key pair is exported in PKCS#12 format for backup by the administrator. The private signing key is never exported, but if a TCB is not available, then it is stored in (and recovered from) an encrypted file, using an administrator supplied seed password and an internal key generation algorithm. If an external certificate authority is available, SAWS will attempt to get both public keys certified using either CMC-PKCS#10 [4] or CMP [5]; otherwise self-signed certificates are created. SAWS also generates a secret random number and encrypts it using the SAWS encryption public key, and saves it to the **SAWSRandomNumber** record, so in the future, on recovery, SAWS will be able to retrieve it. The secret random number will be used to calculate the secure hash that is appended to each log record for authenticity checking purposes. If a configuration parameter requires it, SAWS will optionally generate a symmetric encryption key and encrypt it with the audit trail viewing tool (VT) encryption public key and the SAWS encryption public key, and save it to the audit trail as two **SymmetricEncryptionKey** records, so that both the VT and SAWS are able to retrieve this symmetric key at a later time. Optionally, the encryption public keys of SAWS clients may be used as well to encrypt the symmetric key. This symmetric key will be used to encrypt/decrypt the client log data to be stored in the log file if confidentiality of the audit trail is required. (Note that clients can independently send encrypted records to SAWS if they want record level confidentiality.) The administrator is prompted for the name of the previous log file, SAWS opens this, validates it by checking its digital signature, then stores the file name, the plain accumulated hash and signature of the previous log file in the **SAWSLastFile** record.

After initialisation, the SAWS server can then receive SAWS client log messages, calculate the secure hashes and the accumulated hash, optionally encrypt the log records, and save them as **ClientLogData** records in the log file.

Every time the SAWS server restarts, it needs to first perform validation of the current log file. This entails the following operations:

- Recompute the plain accumulated hash of the whole log file and check if the signature in the log file is present and correct. If it is, this validates the entire log file.
- SAWS then confirms that the last sequence number of the log file is equal to that stored in the TCB. This can prevent a replacement attack of the entire log file.

If the signature validates, then SAWS will start a new log file as described above. If any error is found, or the signature is absent, this means that SAWS either terminated prematurely or the system crashed. In this case SAWS needs to validate the incomplete file. SAWS retrieves the secure random number from the log file, checks the secure hash on each record and checks the sequence numbers. SAWS displays an error message for each record whose secure hash does not validate correctly, and for each pair of records whose sequence numbers are not sequential. SAWS also displays the details of the last audit file that is recorded in the current log. The administrator can determine whether to subsequently validate this or not, depending upon the outcome of the current process. SAWS recomputes the accumulated hash of the log file, and compares this and the last sequence number in the log file with those stored in the TCB – if they are the same, SAWS can be sure that the log records in the current log file are authentic and complete, in which case, SAWS adds the certificate record to the audit file, writes the plain accumulated hash to the file, digitally signs it, writes the signature record at the end of the file and closes it. It then starts a new log file as above.

The validation of a complete audit file by any application is straightforward. It can open the audit file, recompute the accumulated hash of the whole log file, and check the signature using the certificate inside the log file.

5. Summary and Conclusions

In this paper we have presented a secure audit web service (SAWS) that can provide secure audit trail services to multiple distributed applications. SAWS is able to receive and save client log messages in a secure audit trail file, which can be stored on an untrusted machine. Any party can subsequently verify its authenticity and optionally only authorised parties can read it. Any type of tampering with the secure audit file, such as modification, deletion, insertion, truncation, replacement or unauthorised appending, can be detected. In addition, all the audit trail files are chained together in order to detect the loss of one or more complete files. Whilst SAWS does not directly prevent intrusion or misuse of web resources, nevertheless it can aid the detection of intrusions or misuses by providing tamper resistant evidence of them after the fact. SAWS combined with a trusted computing base (TCB), or other physical tamper-resistant hardware can form the basis for highly trusted auditing capabilities. SAWS is designed to be general purpose, and any application can make use of its services for logging and audit purposes. Compared with Schneier's method [1], the basis of trust is shifted from the central trusted machine in Schneier's method to SAWS's own trusted store and optionally, an external Certification Authority. This simplifies and reduces the security requirements for the trust basis, thus it can bring more convenience and flexibility to SAWS applications and SAWS administrators.

6. Acknowledgements

The authors would like to thank both UK JISC for partially funding this work under the DyCom project and the EC for partially funding this work under the TrustCoM project.

References

[1] B. Schneier, J. Kelsey. "Secure audit logs to support computer forensics". *ACM Transactions on Information and System Security*, 2(2), 1999, 159-176.
[2] http://www.trustedcomputinggroup.org/
[3] C. N. Chong, Z. Peng, P. H. Hartel. "Secure audit logging with tamper-resistant hardware". *Proceedings of 18th IFIP TC11 Int. Conf. on Information Security, Security and Privacy in the Age of Uncertainty*. D. Gritzalis, S. De Capitani di Vimercati, P. Samarati and S. K. Katsikas (eds.) , published by Kluwer Academic Publishers, Boston, Massachusetts, held in Athens, Greece, May, 2003, pp 73-84.
[4] M. Myers, X. Liu, J. Schaad, J. Weinstein. "Certificate Management Messages over CMS". RFC 2797, April 2000.
[5] C. Adams, S. Farrell. "Internet X.509 Public Key Infrastructure Certificate Management Protocols", RFC 2510, March 1999.
[6] OASIS. "Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)". OASIS Standard 200401, March 2004.
[7] D. Chadwick, O. Otenko. "A Comparison of the Akenti and PERMIS Authorization Infrastructures", in Ensuring Security in IT Infrastructures, proceedings of the ITI First International Conference on Information and Communications Technology (ICICT 2003) Cairo University, Editor Mahmoud T El-Hadidi, pp5-26, 2003