A Privacy-Preserving Interdomain Audit Framework

Adam J. Lee, Parisa Tabriz, and Nikita Borisov
University of Illinois at Urbana-Champaign
adamlee@cs.uiuc.edu, {tabriz, nikita}@uiuc.edu

Abstract

Recent trends in Internet computing have led to the popularization of many forms of virtual organizations. Examples include supply chain management, grid computing, and collaborative research environments like PlanetLab. Unfortunately, when it comes to the security analysis of these systems, the whole is certainly greater than the sum of its parts. That is, local intrusion detection and audit practices are insufficient for detecting distributed attacks such as coordinated network reconnaissance, stepping-stone attacks, and violations of application-level trust constraints between security domains. A distributed process that coordinates information from each member could detect these types of violations, but privacy concerns between member organizations or safety concerns about centralizing sensitive information often restrict this level of information flow. In this paper, we propose a privacy-preserving framework for distributed audit that allows member organizations to detect distributed attacks without requiring the release of excessive private information. We discuss both the architecture and mechanisms used in our approach and comment on the performance of a prototype implementation.

1 Introduction

Effective audit mechanisms are necessary to ensure the safe and correct operation of many different industries. In computer security, the use of verifiable audit trails is central to the maintenance of robust and secure computing systems. Audit logs are widely used as a means of verifying the secure operation of a system and often times provide the input datasets for intrusion detection systems. Until recently, the largely centralized nature of the computational resources utilized by an organization has implied that the examination of local audit trails was sufficient for detecting attacks targeting a particular organization. This notion is rapidly becoming obsolete.

In recent years, the accessibility of the Internet has caused fundamental changes in the ways that entities and organizations interact. Data and resources are often shared across organizational boundaries and businesses are increasingly outsourcing tasks such as supply-chain management and billing to third parties. As organizations grow into larger virtual organizations, more points of vulnerability emerge and attackers can carry out much more distributed forms of attack. Traditional means of distributed audit cannot be used within virtual organizations due to concerns over both log privacy and the safety of centralizing sensitive data storage [20]. The fact that two organizations are willing to cooperate for the purpose of carrying out a particular type of interaction does not, and should not, imply that they completely trust one another. In many cases, the types of data records used to conduct security audits are considered sensitive, as they could reveal information about internal network structures, the types of software running in restricted areas, or private customer information. Clearly, it would be beneficial for organizations to pool their resources to detect attacks, but current data-handling practices prevent this in the general case.

In this paper, we propose the design of a framework for privacy-preserving distributed audit. Specifically, we define mechanisms through which members of an *audit group* can cooperate to

detect attacks and patterns of abuse without disclosing sensitive information to one another. We specify a framework through which audit records generated at each site can be cryptographically altered and submitted to a central audit clearinghouse for examination. Our framework enables the outsourcing of both *historical* (i.e., long term analysis) and *online* intrusion detection and audit systems while minimizing the threats of leaking sensitive information to either the auditor or other collaborating sites within the audit group. This framework has the following specific properties:

- Our framework is general enough to be used in a variety of situations including the detection of coordinated network attacks launched by external adversaries and the detection of internal abuses of trust between group members. We wanted to develop a solution that is not limited to a specific class of problems.
- Our framework does not require that members of an audit group exchange sensitive information with one another. That is, detecting attacks does not require that members of the audit group trust each other to examine one another's sensitive data.
- The central auditor has access only to cryptographically altered audit records.
- Group members are able to verify the accuracy of any reports generated by the auditor.
- Group members can obtain probabilistic guarantees regarding the completeness of information provided by the auditor.

The rest of this paper is organized as follows. In Section 2, we further motivate the need for a privacy-preserving distributed audit system. Section 3 describes in detail the system architecture of our framework. Section 4 presents the threat model adopted in this paper; the supported data types, levels of obfuscation, and cryptographic obfuscation methods utilized by our framework are discussed in Section 5. The details of our prototype implementation are presented in Section 6 along with benchmark data for the obfuscation methods described in Section 5. In Section 7, we discuss several interesting properties of our framework. In Section 8 we survey related work in the areas of distributed audit and privacy-preserving data sharing and we conclude with a summary and discussion of future work in Section 9.

2 Motivation

There are a number of situations in which distributed audit facilities can be beneficial to groups of entities collaborating across security domains. Perhaps the most intuitive case involves the detection of coordinated attacks and network reconnaissance tactics targeting the collaborative group. A recent wide-scale study on the threat of correlated attacks revealed that 20% of the offending sources from their investigation launched correlated attacks and those attacks account for over 40% of their investigated intrusion alerts [16]. They additionally characterize correlated attacks as often being targeted towards small groups (4-6 members). As these groups remain fairly stable over time, the authors support collaboration and trust establishment for correlated alert detection. The coordinated attacks and compromises that occurred across nodes in the TeraGrid during the Spring of 2004 [26] are examples of this phenomenon.

Another interesting type of attack that can be detected through the use of distributed audit is the stepping-stone attack. During a stepping-stone attack, the attacker routes his traffic through multiple nodes in an effort to prevent detection mechanisms from pinpointing the true source of the attack. The BRO intrusion detection system [24] provides stepping stone detection across the domain the system is monitoring [36]. Correlated stepping stones in collaborating networks could be detected if local stepping stone alerts were shared and analyzed by a central auditor. Correlated

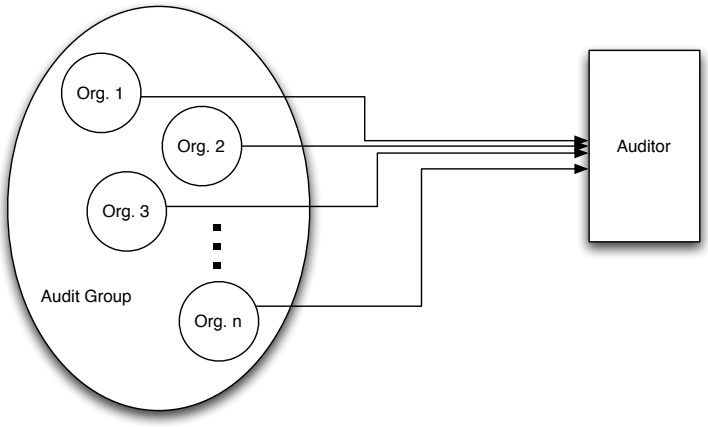

Figure 1: System architecture overview.

alerts could lead to fuller stepping stone path reconstruction, and ultimately, identification of the true origin of the attacker. Distributed audit can also help detect abuses of the applications or trust relationships used to bootstrap a virtual organization by both external attackers and malicious insiders through the detection of separation of duty constraint or other application-level policy violations. Unfortunately, members of many virtual organizations lack the level of trust in one another required to share the information that could lead to the detection of the above types of service violations.

A privacy-preserving framework for distributed audit can have positive implications not only when members of an audit group are mutually distrustful (as one might expect in an administratively diverse virtual organization), but across the entire spectrum of audit member trust levels. For instance, consider the case where members of an organization have a high degree of trust in one another and already carry out some form of distributed audit process (perhaps using a mechanism similar to that proposed in [20]). A compromise of the central point of analysis for audit data could release a large amount of sensitive information; this is undesirable in general, but particularly so if personal information, such as customer or employee activities, are part of the audit process. In this case, the offending organization may suffer from bad publicity as it could be required by laws such as California SB 1386 [7] to publicly notify the effected persons of the security breach. The use of a privacy-preserving audit framework could limit the damage caused by such a compromise by revealing only encrypted or otherwise obfuscated data. Similarly, the privacy-preserving nature of this framework could assuage the fears associated with the sharing of audit trail information in more tightly-coupled organizations. This could facilitate better audit practices between departments within the same organization or members of distributed experimentation frameworks, such as PlanetLab [25]. Although the administration of PlanetLab nodes allows host-level audit information to be collected and analyzed centrally, this cannot currently be done for IDS alerts or other network anomalies observed on the networks surrounding these nodes, as PlanetLab nodes operate in widely distributed administrative domains. Clearly, a privacy-preserving distributed audit framework benefits distributed organizations with all levels of trust between members; in the remainder of this paper, we discuss the architecture, mechanisms, and analysis of such a framework.

3 System Architecture Model

In this section, we discuss the system architecture of our audit framework. The main components of this system include organizations, audit groups, and auditors, which are presented in Figure 1 and described in detail below.

Organizations represent individual security domains, within which no data anonymization is necessary. Each organization may run intrusion detection systems, system activity loggers, or other audit mechanisms to track the security-state of its constituent systems. Organizations that have some degree of mutual trust in one another band together to form *audit groups*. The members of an audit group run a group key management protocol to agree upon a shared secret that they use for log obfuscation (the details of which are described in Section 5.2). Prior to contracting an auditor, the member organizations of an audit group develop software that can be used to detect the constraint violations that they are interested in through the examination of their obfuscated audit records (as in [20]). Each member organization then signs this code to indicate that it conforms to their standards and submits the signed code to the auditor (or auditors) who will analyze their collective audit records.

Auditors are entities in the system who are paid or otherwise contracted to analyze the obfuscated audit records generated by an audit group to detect constraint violations that cannot be detected by one organization alone. For example, auditors may attempt to detect both internal abuses of trust within the audit group and also coordinated attacks carried out by malicious outsiders. Upon receiving signed detection code from an audit group, the auditor allocates the audit group a virtual machine within which that code will be run. Since any auditor can service multiple audit groups, this prevents a malicious group from harming the execution environment of other audit groups while still permitting audit groups to author their own attack detection programs. As audit records are submitted by the audit group, the auditor passes these records into the code provided by the audit group and reports the alerts generated by this code.

4 Threat Model

In this paper, we aim to design a framework that can afford higher privacy guarantees and detect a wider range of attacks than existing Internet-scale information sharing systems by assuming a more limited system size. In particular, the interdomain audit framework described in this paper is designed for use in medium-scale collaborative environments such as business-to-business supply chains and grid computing systems. With this consideration, we describe the threat model that we have chosen to adopt for our audit framework.

Many solutions to data outsourcing problems (e.g., [2, 11, 20]) assume that the third-party who gains control of the data behaves according to the *honest-but-curious*, or *semi-honest*, model [14]. In this model, the third-party follows all specified protocols exactly, but may record any traffic that it processes and attempt to extract information from the data-store in an offline manner. Active attacks (e.g., returning incomplete or spurious data) are strictly prohibited within this model. Initial discussions of our framework assume an honest-but-curious auditor solely for clarity of discussion; in Section 7.1 we show that our system remains operational even when the auditor follows a more Byzantine threat model. Specifically, we show that audit group members can detect when incorrect alerts are raised by the auditor and can probabilistically determine whether the auditor is withholding information from the audit group.

We assume that members of the audit group trust one another enough to maintain any group secrets necessary for processing audit records sent to the auditor. This seems to be a viable

assumption, as exposing these secrets could potentially reveal the audit information of *all* group members, including the member who leaked the secrets. Note that this explicitly forbids collusion between group members and the auditor in our model. We do, however, assume that group members may attempt to send faulty "probe" audit records to the auditor in hopes of triggering constraint violations that would reveal information regarding the legitimate events detected by other members of the audit group.

5 Data Formats and Obfuscation Levels

In this section, we discuss the types of data that we have identified as being important for interdomain audit. We additionally describe the levels at which each type of data can be obfuscated and the methods through which these obfuscation levels can be attained in practice.

5.1 Supported Data Types

Audit logs are comprised of a collection of audit records, each of which is a compound structure containing multiple data values. We now discuss each of the data types supported by our system.

Identifiers An identifier is simply a label that is associated with a given record. Examples could include a log level (e.g., DEBUG, WARN, etc.) associated with a given line of output or a key field used to link a particular audit record with other records.

Numbers Numbers are totally-ordered identifiers upon which mathematical operations such as addition and subtraction can be performed. Examples include the number of packets received on a given port, the amount of time a particular process was running, or the length of a file received by an FTP daemon.

Tree structures Tree structures are hierarchically-organized identifiers in which a child node may have at most one parent node. Examples include IP addresses, domain names, and file system paths. The notion of prefix matching is important in these cases. For example, we may want to find all source IP addresses in the 128.221.8.* subnet or all files in subdirectories of `/usr/bin`.

Partially-ordered sets A partially-ordered set (poset), $\langle P, \preceq \rangle$, is a set of identifiers, P, along with a partial-ordering relation, $\preceq$. Partial orderings are often used to describe relationships between roles in RBAC systems or relationships between different types of objects in a system.

Lists Lists are variable-length collections of items of the same simple data type (e.g., identifiers, trees, posets, or numbers). Lists are particularly useful for containing variable-length information fields, such as the group of flags set in a TCP packet or other attribute lists.

The above list of data types was chosen based upon studying the types of input data accepted and emitted by popular security logging systems and discussing applications of interdomain audit frameworks with security practitioners. While this list cannot be guaranteed to be complete, it appears that the data formats currently used for attack detection and system analysis can all be represented using the above data types. We now discuss the levels at which each data type can be obfuscated prior to its disclosure to the audit process.

5.2 Obfuscation Levels

Given that data can be represented using the types described in Section 5.1, we now wish to define the various granularities at which data items can be disclosed. In Section 5.3, we provide a rough overview of the methods by which our framework enables these disclosure granularities.

Full disclosure The most basic level of disclosure is full disclosure. That is, the unmodified data item will be released to the auditor.

Local exact match In some cases, entities within an audit group might wish to allow an auditor to determine whether a data item disclosed by one member of the audit group matches a data item disclosed by other members of the audit group, but not disclose any further information about the data item (this is likely to occur with respect to the key field(s) of an audit record). For instance, if members of an audit group are monitoring mutual users of their systems, it is important that the auditor be able to determine whether two audit records refer to actions taken by the same user, even if these actions were observed by different audit group members. However, the auditor should not be able to determine the actual value of a data item disclosed for local exact match only, nor should they be able to leverage information reported by entities outside of the audit group to infer additional information about the value of the item.

Portion dropping In the event that only certain portions of a tree structure are required (e.g., the class-B network that an IP address is in), our framework supports a portion drop transformation. This allows the data producer to drop portions from the left and/or right of a tree structure to reduce the amount of data actually disclosed.

Local prefix match In some cases, it is beneficial for an auditor to determine whether two data elements match in prefix. For instance, the audit group may be interested in determining the portions of the IP address space from which the most failed login attempts originate. The auditor should not be able to determine any semantic information from the matched tuples other than the fact that they have a common prefix.

Local greater-than It should be possible to disclose partially ordered data in such a way as to allow the auditor to determine whether one data item is greater than another data item (with respect to the defined partial order) without revealing any further information regarding the actual values of the data items.

Basic numeric transformations In many cases it is desirable to carry out very simple transformations on numerical data fields. Our framework supports adding jitter to values, the use of uniform and non-uniform bucketing, and scaling.

Local blinded arithmetic In some cases, members of the audit group may wish to instruct the auditor to generate summary statistics over the data values in their audit records without wanting to disclose the actual contents of audit records. For instance, it may be beneficial to allow the auditor to maintain running totals without knowing what the current total is. These "blinded totals" could then be reported to the members of the audit group at regular intervals.

Complete obfuscation Data that is completely obfuscated should look like random bit patterns to the auditor. These values should, however, be decipherable by entities in the audit group in the event that audit records are released back to some subset of the audit group.

Table 1 summarizes the applicability of each obfuscation method presented above to each of the data types defined in Section 5.1. We now proceed to discuss how several of the above mentioned obfuscation methods can be implemented.

5.3 Obfuscation Method Implementation Details

We now provide details regarding how the above obfuscation goals are attained in our implementation. The more simplistic methods (i.e., drop, jitter, bucketing, and scaling) are not discussed.

	Identifier	Number	Tree Structure	Poset	List
Full Disclosure	√	√	√	√	√
Jitter	-	√	-	-	√
Bucketing	-	√	-	-	√
Scaling	-	√	-	-	√
Drop	-	-	√	-	√
Local Exact Match	√	√	√	√	√
Prefix Preserving Match	-	-	√	-	√
Local Greater-than	-	-	-	√	√
Blinded Summation	-	√	-	-	√
Complete Obfuscation	√	√	√	√	√

Table 1: Obfuscation methods applicable to each data type.

Throughout the remainder of this paper, we make the following assumptions regarding the cryptographic capabilities of the members of the audit group:

- The members of an audit group have a shared secret, s, perhaps computed using a group Diffie-Hellman scheme [29, 6].
- There exists a cryptographic hash function, $h : \{0,1\}^* \rightarrow \{0,1\}^l$, where l is the fixed output length of $h(\cdot)$. We assume that $h(\cdot)$ is a one-way and collision-resistant function.
- There exists a symmetric-key encryption algorithm, such as AES. We denote the encryption (resp. decryption) of a message m using key k by $e(k,m)$ (resp. $d(k,m)$). Note that $d(k,e(k,m)) = m$.
- There exists a homomorphic encryption algorithm, such as that presented in [21], with encryption operation $E(\cdot,\cdot)$ and decryption operation $D(\cdot,\cdot)$. Given a key pair $\langle k, k^{-1}\rangle$, $D(k^{-1}, E(k,m)) = m$ and $E(k,m) \times E(k,m') = E(k, m+m')$.

5.3.1 Local Exact Match

To provide local exact match functionality, each member of an audit group must be able to modify data items irreversibly in a predictable manner. This will allow the auditor to match equivalent data items without revealing the actual contents of the identifier. We first define a function $\kappa_{hash}(\cdot)$ as follows:

$$\kappa_{hash}(x) = h(\text{“HASH\_SALT”} \mid x) \tag{1}$$

The above equation generates a seemingly random salt value from its input (note that $\mid$ denotes concatenation). Let us define the value $salt = \kappa_{hash}(s)$. Given this salt value, we now define a function, $f_{lem}(\cdot)$ which alters any data value to meet the local exact match disclosure criteria:

$$f_{lem}(id) = h(salt \mid id) \tag{2}$$

The collision-resistance and one-way properties of the function $h(\cdot)$ imply that the most efficient way for an adversary to recover the value of $x \in X$ from $f_{lem}(x)$ is through a brute-force search of the space $\{0,1\}^l \times X$.

5.3.2 Local Prefix Match

The local prefix match functionality can be attained through modified use of the use of the $f_{lem}(\cdot)$ function on each “portion” a hierarchical data item, as follows:

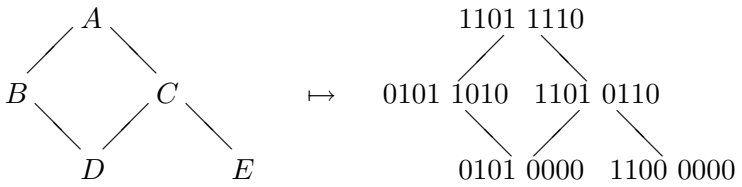

Figure 2: An example partially ordered set and one possible obfuscated encoding

$$f_{lpm}([x_1, x_2, \ldots, x_n]) = [f_{lem}(x_1), f_{lem}(x_1 \mid x_2), \ldots, f_{lem}(x_1 \mid x_2 \mid \ldots \mid x_n)] \tag{3}$$

Unlike the prefix-preserving hash method discussed in [12], the above method can be computed over data structure without requiring multiple passes over input data, as it does not preserve the data length of the field being obfuscated. This method is thus suitable for use in an environment where data is streamed out to the auditor, instead of only being useful when static files are to be anonymized and released. However, the above method works at a higher granularity than the method presented in [12].

5.3.3 Local Greater-Than Relation

To provide the local greater-than relation functionality on a partially ordered set, we first define an order-preserving homomorphism from $\langle P, \preceq \rangle$ to $\langle P', \preceq' \rangle$. We have chosen to represent the transformed poset using Bloom filters [5], which are data structures that enable efficient testing of set membership. To insert a data item $p \in P$ (where $|P| = n$) into a Bloom filter, p is processed using a function $map : P \rightarrow \{0,1\}^m$ that combines the outputs of k hash functions; any of the resulting bit positions that are set to 1 are then set in the Bloom filter. Testing for set membership involves simply computing the value that would be inserted into the Bloom filter and then testing to see if those bits are set in the filter. If they are not set, the data item is definitely not in the set; if those bits are set, then with high probability the data item is in the set. The tradeoffs in choosing k, m, and n and their effects on the false positive rate are discussed in detail in [5].

To encode a partially ordered set $\langle P, \preceq \rangle$ to allow the local greater-than relation, we represent each element of $p \in P$ as a Bloom filter containing a transformed mapping of itself and transformed mappings of all elements in P that it dominates. More specifically, for some $p \in P$, its corresponding element $p' \in P'$ is a Bloom filter containing the items $\{map(CTR \mid salt \mid p)\} \cup \{map(CTR \mid salt \mid q) \mid q \preceq p\}$ where CTR is a counter value initialized to zero. Clearly, given any $q', p' \in P'$, $p' \preceq' q'$ if the encoding of q' contains the encoding of p'. After the audit group determines its shared secret, k (and therefore the value *salt*), each member independently maps each partially ordered set $\langle P, \preceq \rangle$ into its altered representation $\langle P', \preceq' \rangle$ as described above and checks each $\langle P', \preceq' \rangle$ for false positives. If any false positives are discovered, CTR is incremented and the process is repeated. Figure 2 shows an example of this poset transformation.

This method of representing partially ordered sets has several interesting properties. First, unless the auditor has observed all 2^m data items in the set P', it cannot be certain that it has observed the entirety of the structure of $\langle P, \preceq \rangle$. Similarly, given two data items $p, q \in P'$ such that $p \preceq' q$, the auditor can obtain a lower-bound on the distance between p and q in the structure of P' but cannot tell the distance between p and q conclusively unless all 2^m items in P' have been observed.

5.3.4 Local Blinded Summation

In general, performing arbitrary arithmetic operations on encrypted data is a difficult problem. However, we can perform summations over encrypted data, given the existence of a homomorphic encryption function. Even this simple arithmetic operation enables the detection of a wide range of *threshold* events. For instance, this is useful if an audit group is interested in learning the number of failed login attempts made on each account used across the audit domain or determining which ports are seeing high volumes of connections, all without sharing private information between members of the audit group. Given a homomorphic encryption algorithm and keys k and k^{-1} derived from the group secret s, we define $f_{lbs}(\cdot)$ as follows:

$$f_{lbs}(x) = E(k, x) \tag{4}$$

That is, the homomorphic cryptosystem is used to encode the values to be obfuscated. Given values of this form, the auditor can compute summations without knowing what values are being summed by simply multiplying values together. Reports can be sent periodically to members of the audit group who can easily reveal the private summation using $D(k^{-1}, \cdot)$.

5.3.5 Complete Obfuscation

The complete obfuscation functionality can be implemented similarly to the local exact match functionality. We first define the function $\kappa_{sym}(\cdot)$ which generates a symmetric key from some initial value:

$$\kappa_{sym}(x) = h(\text{“SYMMETRIC”} \mid x) \tag{5}$$

Let us define k_{sym} by selecting the needed number of bytes from the beginning the output of $\kappa_{sym}(s)$. Should more bytes be needed, then a longer key can be generated as follows:

$$\begin{aligned} k_1 &= \kappa_{sym}(s) \\ k_2 &= \kappa_{sym}(s \mid k_1) \\ k_3 &= \kappa_{sym}(s \mid k_1 \mid k_2) \\ &\vdots \\ k_{sym} &= k_1 \mid k_2 \mid k_3 \mid \ldots \end{aligned}$$

Again, the necessary number of key bytes can be taken from the beginning of k_{sym}. Note that this process is similar to that used in the SSH protocol specification [35]. Given k_{sym}, we now define a function f_{co} that embodies the complete obfuscation functionality.

$$f_{co}(x) = e(k_{sym}, nonce \mid x) \tag{6}$$

Given a data item x, $f_{co}(x)$ hides the value of x from the auditor by encrypting the concatenation of x with a randomly chosen 128-bit nonce value with a key unknown to the auditor. The inclusion of the nonce value prevents the auditor from determining whether two ciphertext values refer to the same plaintext. However, any member of the audit group who is given $f_{co}(x)$ can easily retrieve x.

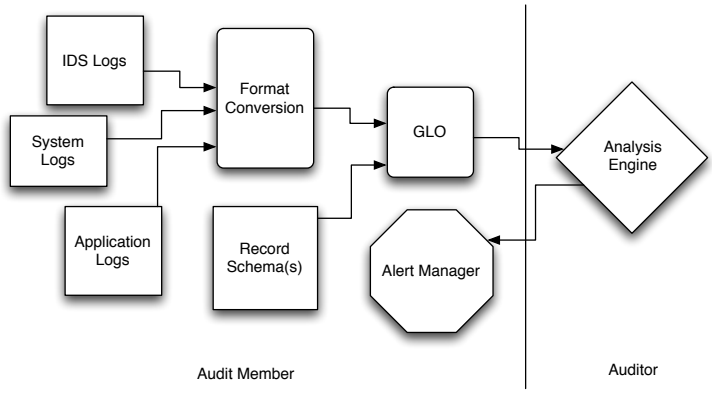

Figure 3: Software Architecture Overview

6 Implementation and Evaluation

6.1 Implementation

In this section, we introduce the software architecture necessary to support our framework for distributed audit. As pictured in Figure 3, all security logs at a member site are converted to a standard format before they are input to a general log obfuscator, which we refer to as GLO. In addition to the format converted logs, members supply a collection of policy schemas that define the record format types and associated obfuscation methods GLO should use to obfuscate the logs. The processed log records are then forwarded to the auditor and used as inputs to the audit software written by the members of the audit group. If an auditor detects an alert, the set of records acting as evidence for the alert is then forwarded back to the alert managers run by each group member that contributed to this record set.

GLO is implemented using Java 1.5 and the Bouncy Castle Cryptography API. Each of the types and obfuscation methods defined in Section 5, excluding local blinded summation, are supported in our current implementation of GLO. As there is no support for homomorphic encryption through the Java API or its supported libraries, we leave a detailed evaluation of the blinded arithmetic operation for future work. While homomorphic encryption is more computationally expensive than the other obfuscation methods supported by GLO, we expect that blinded arithmetic will be most used in less time-dependent procedures, such as daily port traffic summary reports based on batched audit logs. In this case, the extra time incurred by the overhead of using more complicated cryptographic operations would be acceptable.

To provide for flexible log processing, we have future plans to support a more standardized security log format for data processing. There is a recent and ongoing effort to define such a standardization. The Intrusion Detection Message Exchange Format (IDMEF) [8] is the most developed proposal, consisting of an XML definition for audit logs. IDMEF is still a work in progress, but some IDS conversion plug-ins for popular IDSes like Snort [27] already exist.

6.2 Evaluation

The efficiency of GLO, or how fast GLO can process audit records, is highly dependent on the obfuscation methods defined in the policy schema. With this consideration, we provide benchmarking details for several of the obfuscation methods outlined in Section 5.3 in Table 2. The shared key used in cryptographic obfuscation is 128 bits long. The GLO application was run on a Pentium 4

Data Type	Obfuscation Method	Processing Rate (Records/Second)
Number, Identifier, Poset	Local Exact Match	16341, 19937, 20729
Number, Identifier, Poset	Full Obfuscation	15657.3, 19274, 20220
Number	Scale	17778.8
Number	Uniform Bucket	17923.5
Number	Non-Uniform Bucket	19580.2
Tree	Full obfuscation	15281
Tree	Local Exact Match	15780.2
Tree	Prefix Preserving Match	9646.9
Tree	Drop Portion Match	20155.2

Table 2: Obfuscation processing rates applicable to each data type.

Source IP	Destination IP	Source Port	Destination Port	Protocol	Timestamp	Sequence Number
$< tree >$	$< tree >$	$< number >$	$< number >$	$< identifier >$	$< number >$	$< number >$
full disclosure	prefix preserving match	full disclosure	local exact match	full obfuscation	jitter	scale

Table 3: IDS log format and obfuscation policy.

2.5GHz processor with 512 MB of RAM running the Linux operating system; we consider this to be a modest computing resource compared to what a typical organization might dedicate to intrusion detection technology. Each rate reflects GLO's average processing speed over 10 repeated trials for obfuscating a large number of batched records of the given type. From these results, we can see that the most resource-intensive obfuscation method is the prefix-preserving match method, while every other method is able to handle more than 15,000 records per second.

It is useful to compare the processing rate of GLO to typical alert generation rates from IDS, system, and application logs. According to logging rates obtained from the security operations staff at our institution, typical high-volume IDS logging mechanisms average 30.69 generated records per second, and are the single largest contributor to security audit logs. We tested GLO with a general network audit log record format similar to that used by tcpdump [30] and an appropriate policy schema. The record format and schema are displayed in Table 3. Given this format and policy, GLO obfuscated 4295.1 records per second, a processing rate over 100 times faster than the alert generation rate reported by our institution. Since obfuscation can be performed in parallel, there is no reason that an organization could not run multiple instances of GLO to perform processing, but our results suggest that even a single instance of GLO can process audit records fast enough for real-time alert correlation

6.3 Distributed Password Guessing

One of the most common ways an attacker gains access to a system is through brute force password cracking schemes to gain user or root privileges on a machine. These can include manual login attempts, dictionary based attacks, and use of automated tools that randomly generate usernames and passwords to login with. System administrators prevent these types of attacks by limiting the number of times a user can attempt to login at a given host. If a user surpasses some attempt threshold within a given window of time, the system may delay future login attempts, create a security alert, or lock the user account. Users in a virtual organization often have access to resources at multiple sites and commonly use the same login username and password combination across sites. An adversary could attempt a password guessing attack by distributing login attempts across all sites within the virtual organization. In this way, he can stay beneath thresholds imposed

Timestamp	Application	Error Message	Username	Source IP	Source Port
$< number >$	$< identifier >$	$< identifier >$	$< identifier >$	$< tree >$	$< number >$
full disclosure	local exact match	full obfuscation	local exact match	local exact match	local exact match

Table 4: Authentication log format and obfuscation policy for detecting distributed password guessing.

by local security logging mechanisms without sacrificing the actual number of login attempts he is allowed to make per unit time. If, however, login failure records from all sites were collected and analyzed by a central auditor, this attack could more easily be detected.

To demonstrate a simple but important use case for privacy-preserving distributed audit, we examine the ability of our architecture to detect password guessing attacks. System authentication logs clearly store sensitive data in the form of usernames and, in some cases, obfuscated passwords. Even system authentication logs that record usernames only can hold information relating to a password if a user accidentally types their password or a portion of it at a login prompt. This data is then stored in clear text and is highly valuable to an adversary with log access. Additionally, some password guessing attacks may go unnoticed with only local detection capabilities but can be detected using distributed audit. For these reasons, we evaluate the resource requirements needed to obfuscate authentication log records and correlate a distributed password guessing attack.

We present the overhead required to obfuscate authentication logs and perform detection of distributed password guessing. We use the `auth.log` system logging file found in `/var/log/` of the Linux file system as an example record format for login logs. An example of this file logging, its respective record format, and a suitable obfuscation policy are provided in Table 4. The processing rate to obfuscate authentication logs is 5649 records/second. The central auditor only needs to maintain one state variable per user to keep track of the aggregate number of login attempts. Using a hash table to store all member authentication logs would require $O(n)$ space to store the current status of each of the n users and constant time to find or update a user's authentication state. These are identical to the space and time requirements needed to do this checking at a local site, but by correlating log records, we are now able to detect both site-specific and distributed password guessing attack while better protecting user privacy.

7 Discussion

In this section, we discuss how our system allows members of an audit group to detect when their auditor is withholding or fabricating information, or when a malicious member of the audit group probes the auditor in an attempt to learn what events other group members are reporting. We also discuss several important points related to information disclosure that users of this system should bear in mind.

7.1 Catching Liars and Cheaters

In Section 4, we introduced our threat model of a Byzantine auditor, which we assume to be more than reasonable as an auditor would likely be a paid third-party that has financial incentive to honestly provide the service that it is offering its customers. While members of the audit group develop and sign the software run by the auditor, they still require assurance that it is this exact software that the auditor is actually executing during alert correlation. If the auditor uses a trusted computing platform that supports virtual machines and flexible code execution, such as

Terra [13], members can verify that their software is being executed in its original form. This prevents tampering from the auditor or any other individual with access to the detection code.

Group members within our framework can also detect false alarms raised by the auditor and probabilistically detect if an auditor is not providing complete alert reports. To protect the authenticity and integrity of audit records, members can sign each record before submitting them to an auditor. This process can be implemented during GLO's log obfuscation using Merkle trees to sign large batches of records. When any alert is raised, the audit records contributing to the attack should be returned with the corresponding alert. At this point the authenticity of the records can be validated offline. If the overhead of signing audit records is undesirable, log records can still be validated out-of-band by administrators of each member's domain. To detect whether an auditor is withholding alerts, members can collaborate to plant fictitious log sequences that should result in alert detection if the auditor is behaving correctly. This assurance can be brought arbitrarily high by raising the frequency of these fictitious log sequences provided to the auditor.

While we assume that group members will behave due to some underlying level of established trust as well as to protect their own privacy, we accept the possibility that some audit members may try to game the auditor to reveal sensitive information about another member's audit logs. For example, a dishonest member may try to create fictitious, probing log records that will signal an alert if and only if a specific action is taking place in another member's network. In many scenarios, we envision this type of gaming the system to be difficult or at least require a large number of false audit records to successfully signal another network's response. In [4], the authors describe an instance of this type of probe response attack that is successful in identifying the sensor locations of the SANS Internet Storm Center. Since our system assumes a much more limited size than Internet-wide audit log contributors, members can help prevent gaming the system by writing rules that raise alerts if suspicious probing behavior is detected at the auditor.

7.2 Information Disclosure

In Section 5, we showed that each obfuscation method provided some level of security guarantees when used in isolation. However, in practice, the guarantees that they provide can be reduced for a number of reasons. We now discuss two such reasons: functional dependencies between fields in an audit record and the incorporation of external knowledge.

It is most often the case that the fields comprising an audit record are related to some extent, as each record describes the details of a particular event. Because of this relationship between the fields of an audit record, it could sometimes be the case that the value of one field "leaks" information about the value of another field. As a contrived example, consider a failed login audit record that reports the userid for which the login was attempted and the userid spelled backwards. Clearly, fully obfuscating only the userid field does not guarantee any level of security, as the userid spelled backwards can be used to derive the userid. More realistic examples of functional dependencies can arise in network traces, as it is sometimes the case that knowing the destination port number of a connection can reveal the transport layer protocol used for that connection. Similarly, in [17], the authors show that unobfuscated timestamp fields can be used to fingerprint the transmitting devices in a packet trace. To preserve the privacy of the data reported to the auditor, it is imperative that the functional dependencies between fields of an audit record be determined as best as possible so that these types of data leakage can be prevented.

Another problem relating to information disclosure is the incorporation of external observations or other "common knowledge" by the auditor. For example if packet-level traces are being reported to the auditor and the auditor knows that most of the traffic destined for its audit group is HTTP traffic, then it can determine that the most frequently seen destination port value deobfuscates to

the value 80. In this particular example, it is unlikely that the audit group would consider such a disclosure to be sensitive, as port numbers are not typically obfuscated even when data traces are anonymized for public release [22]. This does serve to illustrate the underlying problem, however. Functional dependencies and the incorporation of external knowledge are not mutually-exclusive threats; it could be the case that information acquired using external knowledge could be used to extract more information using a previously-obfuscated functional dependency, or vise-versa.

It is our conjecture that there does not exist any purely technological solution to the above two problems. For instance, prior to the discovery presented in [17], no one would have assumed that timestamp fields could identify the source of a packet stream. To help reduce the chance of inadvertent data disclosure through undiscovered functional dependencies or the incorporation of external knowledge, the data fields reported to the auditor should be chosen carefully. To detect most attacks, it is often times not necessary to use *every* data field reported by the IDS or packet capture device generating the raw audit logs. By selectively choosing the data sent to the auditor, the number of fields which can be used to deobfuscate data will be reduced. Similarly, given a restricted data set, it becomes easier to determine the functional dependencies relating fields within an audit record, thereby making it easier to determine an appropriate obfuscation policy.

8 Related Work

This paper builds on two distinct areas of prior work: distributed audit systems and privacy-preserving data sharing. We now review related work in these two areas.

8.1 Distributed Audit

In [20], the authors provide a framework for distributed audit occurring within one organizational domain. This framework is not suitable for use in our scenario as information is shared freely with the monitoring entity and is thus undesirable due to the privacy requirements and threats to centralized data discussed in Section 1. DShield [10] and DeepSight [9] gather information from a wide range of contributors. This information is processed to detect traffic fluctuations and other indicators of wide-scale attacks such as worms. These projects can only detect Internet-wide activity, and not specific forms of intrusion at any one site or among smaller group of collaborators.

There has been a substantial body of work in building distributed architectures for intrusion detection systems. The first proposed systems, such as DIDS [28] and the McAfee IntruShield Security Manager Appliance [19], collected data from multiple monitoring sensors and performed centralized processing of data to create alerts. More recent systems have focused on reaching higher scalability [32, 34], however, they are not designed for cross-domain collaboration and often share unaltered, raw data and alerts between monitoring nodes. Perhaps the most closely related work to ours with respect to distributed audit is [18], in which the authors develop a framework for the privacy-preserving sharing of network traffic logs. Their framework is more general than that of DShield and DeepSight, as anyone can browse the submitted records, though it is still specific to network traffic records. While there have been many system proposals for distributed intrusion detection architectures, there is no prior solution for general privacy protection necessary to make any audit system feasible in a cross-domain scenario.

8.2 Privacy-Preserving Data Sharing

Network packet trace anonymization has been one effort to promote network data information between collaborating entities. This technique deals with anonymizing large volumes of network

data so that it can be released for public scrutiny. Pang and Paxson [23] present a policy-based framework for the anonymization of large packet traces such as FTP sessions. This work differs from ours in that we wish to preserve enough of the data to allow interesting distributed audit and correlation to occur between sites while they want to make data safe for *public* disclosure.

At an opposite extreme from systems that prepare data for public disclosure lies the secure multi-party computation problem. This problem arises when a group of k entities wishes to calculate the value of some function of k private inputs. Although this problem is theoretically solvable in general [33], the general solutions tend to be rather inefficient. Efficient secure multi-party computation algorithms for specific problems (such as secure computation of a k^{th} ranked element [1] and collaborative forecasting and benchmarking [3]) exist, however distributed audit appears to be a general enough area that devising an efficient secure multi-party solution would be nearly as difficult as defining a secure general-purpose secure multi-party computation protocol. The work presented in this paper is an attempt to place a point on the privacy/efficiency spectrum somewhere between network trace anonymization and secure multi-party computation.

Privacy-preserving data mining and database obfuscations (for a survey of this area, see [31]) aim to allow trends to be discovered over distributed datasets without revealing the sensitive information contained within. Typically, datasets are randomly perturbed (as in [15]) to achieve this goal. The end result of this is that large trends can be discovered without revealing specific information. We aim to provide a framework which can be used to detect not only these high-level events, but also more specific types of anomalies that cannot be detected using the aforementioned techniques. In [2, 11], the authors provide a means for owners of distributed databases to carry out privacy-preserving versions of operations such as set intersection and join. This is useful in the context presented in these papers, though these operations cannot be used efficiently to share information as required for the real-time detection of the types of anomalies that we wish to identify (especially if one wishes to use a state-machine approach, as in [20]).

9 Conclusions and Future Work

In this paper, we have described in detail an architecture for privacy-preserving distributed audit. In this architecture, logs can be collected from any number of sources, formatted according to the data types defined in Section 5.1, obscured using the methods presented in Section 5.3, and presented to a central auditor for processing. We have shown that this architecture allows mutually distrustful entities from different security domains to collaboratively detect both distributed attacks against their combined infrastructure and abuses of the trust relationships existing between their security domains. We have implemented the portion of this architecture responsible for obfuscating audit records and deobfuscating results returned by the auditor. In this paper, we present results indicating that this subsystem can process audit records at a rate several orders of magnitude higher than required to keep up with the IDS records generated at our institution.

In the future, we plan to fully implement the architecture presented in Section 3 and to support compatibility with the IDMEF standard. We then hope to work more closely with security operations staff to deploy and test this system in a production environment. In addition to these system-level tasks, we feel that further treatment of the trade-offs discussed in Section 7 is important so that users of this system can better quantify the privacy guarantees afforded by the obfuscation methods that they choose to implement.

References

[1] G. Aggarwal, N. Mishra, and B. Pinkas. Secure computation of the k-th ranked element. In *Eurocrypt*, May 2004.

[2] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 86–97, 2003.

[3] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private collaborative forecasting and benchmarking. In *ACM Workshop on Privacy in the Electronic Society (WPES'04)*, Oct. 2004.

[4] J. Bethencourt, J. Franklin, and M. Vernon. Mapping internet sensors with probe response attacks. In *Proceedings of the USENIX Security Symposium*, August 2005.

[5] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, Jul. 1970.

[6] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *Proceedings of the 8th ACM conference on Computer and Communications Security (CCS'01)*, pages 255–264, 2001.

[7] California Senate Bill SB 1386, Sept. 2002. ⟨`http://info.sen.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html`⟩.

[8] H. Debar, D. Curry, and B. Feinstein. Intrusion detection message exchange format. IETF Internet-Draft, Jan. 2005. ⟨`http://www3.ietf.org/proceedings/05mar/IDs/draft-ietf-idwg-idmef-xml-14.txt`⟩.

[9] DeepSight analyzer. Web site, 2006. ⟨`http://analyzer.symantec.com/`⟩.

[10] DShield–distributed intrustion detection system. Web Page, 2006. ⟨`http://www.dshield.org`⟩.

[11] F. Emekci, D. Agrawal, and A. E. Abbadi. ABACUS: A distributed middleware for privacy preserving data sharing across private data warehouses. In *Proceedings of Middleware 2005*, volume 3790 of *Lecture Notes in Computer Science*, pages 21–41. Springer-Verlag, 2005.

[12] J. Fan, J. Xu, M. Ammar, and S. Moon. Prefix-preserving IP address anonymization. *Computer Networks*, 46(2):253–272, Oct. 2004.

[13] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *Symposium on Operating Systems Principles*, 2003.

[14] O. Goldreich. Secure multi-party computation. Working draft, Version 1.4, Oct. 2002. ⟨`http://www.wisdom.weizmann.ac.il/~odedg/pp.html`⟩.

[15] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. Random data perturbation techniques and privacy preserving data mining. *Knowledge and Information Systems Journal*, 7(4):387–414, 2005.

[16] S. Katti, B. Krishnamurthy, and D. Katabi. Collaborating against common enemies. In *Internet Measurement Conference*, 2005.

[17] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2), Apr.–Jun. 2005.

[18] P. Lincoln, P. Porras, and V. Shmatikov. Privacy-preserving sharing and correlation of security alerts. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[19] McAfee IntruShield Security Manager. Web site, May 2006. ⟨`http://www.mcafee.com/us/enterprise/products/network_intrusion_prevention/intrushield_security_management_system.html`⟩.

[20] A. Mounji, B. L. Charlier, D. Zampunieris, and N. Habra. Distributed audit trail analysis. In *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, Feb. 1995.

[21] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology—EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag, 1999.

[22] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. *Computer Communication Review*, Jan. 2006.

[23] R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM'03)*, 2003.

[24] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[25] Planetlab. Web site, May 2006. ⟨`http://www.planet-lab.org/php/pr/`⟩.

[26] P. Roberts. Update: Hackers breach supercomputer centers. *COMPUTERWORLD Security*, Apr. 2004. ⟨`http://www.teragrid.org/news/apps/0404/computerworld2.html`⟩.

[27] M. Roesch. Snort, intrusion detection system. Web site, May 2006. ⟨`http://www.snort.org`⟩.

[28] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur. DIDS (distributed intrusion detection system) – motivation, architecture, and an early prototype. In *Proc. 14th NIST-NCSC National Computer Security Conference*, 1991.

[29] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM conference on Computer and communications security (CCS'96)*, pages 31–37, 1996.

[30] Tcpdump public repository. Web site, May 2006. ⟨`http://www.tcpdump.org`⟩.

[31] V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Record*, 33(1):50–57, 2004.

[32] Y.-S. Wu, B. Foo, Y. Mei, and S. Bagchi. Collaborative intrusion detection system (CIDS): A framework for accurate and efficient IDS. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC'03)*, Dec. 2003.

[33] A. C. Yao. Protocols for secure computation. In *Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science*, 1982.

[34] V. Yegneswaran, P. Barford, and S. Jha. Global intrusion detection in the DOMINO overlay system. In *Proceedings of the The 11th Annual Network and Distributed System Security Symposium (NDSS'04)*, 2004.

[35] T. Ylonen and C. Lonvick. The secure shell (SSH) transport layer protocol. IETF RFC 4253, Jan. 2006. ⟨`http://www.ietf.org/rfc/rfc4253.txt`⟩.

[36] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the 9th Annual USENIX Security Symposium*, Aug. 2000.