

Pseudonymizing Unix Log Files*

Ulrich Flegel

University of Dortmund, D-44221 Dortmund, Germany

Ulrich.Flegel@udo.edu

Abstract. Unix systems in many cases record personal data in log files. We present tools that help in practice to retrofit privacy protection into existing Unix audit systems. Our tools are based on an approach to pseudonymizing Unix log files while balancing user requirements for anonymity and the service provider's requirements for accountability. By pseudonymizing identifying data in log files the association between the data and the real persons is hidden. Only upon good cause shown, such as a proceeding attack scenario, the identifying data behind the pseudonyms can be revealed. We develop a trust model as well as an architecture that integrates seamlessly with existing Unix systems. Finally, we provide performance measurements demonstrating that the tools are sufficiently fast for use at large sites.

1 Introduction

In times when companies appreciate the value of personal information, customers will increasingly appreciate the protection of their valuable personal data. Survey after survey shows that privacy concerns with respect to the use of the internet are on the rise [5,6,7]. A gap has been shown between what kind of identifying data users think ought to be collected versus what kind they think actually is collected when accessing a web site. The biggest disagreement is shown for some of the data commonly logged by web servers such as machine name or IP address, browser and OS [7]. Most users would rather not access a site than reveal personal information asked by the site for registration [7]. The vast majority of users strongly values being able to anonymously use internet services [7]. This situation calls for privacy enhancing technologies that can be used today and that help service providers to comply with user expectancies as well as with legal regulations concerning personal data. Service providers that can credibly assure their customers the protection of their personal data may gain a competitive advantage over those that can't. Consider as a simple example a web site where the service provision requires no user registration and no billing. There is a reasonable user expectancy to be able to access the site anonymously.

Furthermore, prevalent statutory regulations confine the processing¹ of personal data. Considering our example web site the users would not only have

* This work is currently partially funded by the German Research Council (DFG) under grant number Bi 311/10-2.

¹ Processing, in relation to personal data, covers virtually the entire data life cycle from collection, through to erasure of the data when no longer required.

reason to expect, but they would also have a right to be able to use the site anonymously, based on the pertinent legal situation in many western countries. Refer to [3] and [8] for a more detailed discussion of the related legal issues. Unfortunately it is insufficient to merely legally interdict the misuse of personal data. Once personal data has been collected its processing is hard to control in a non-technical way, since any processing can occur on arbitrary copies of the data. Considering the example web site it is practically futile for a user to determine or even to prove that the organizational controls are ineffective wrt. the protection of his personal data.

Therefore we observe an increasing demand for strong technical enforcement of privacy principles. Multilaterally secure systems take into account and balance conflicting security requirements of all involved parties. Accordingly a multilaterally secure internet service allows customers to act pseudonymously as long as violations of the service security policy are still detectable. In return the service provider is allowed to recover the identities of customers upon good cause shown, which supports the hypothesis of violations of the security policy. Translated to the example web site this means that the IP addresses and potentially other identifying data are pseudonymized when the web server logs are generated. Only if a client exploits a vulnerability of the web server its IP address and other identifying data is recoverable.

This paper demonstrates that the approach to threshold-based identity recovery we designed in [4,3] is actually implementable and applicable in practice. Specifically the contributions of this paper are the following:

- the development of a practical architecture to integrate our pseudonymizer with existing *syslog* auditing infrastructures;
- the exploitation of a trust model supported by actually deployed systems;
- the provision of tools that implement our concepts and architecture;
- a detailed evaluation of the performance of our pseudonymizer, indicating that it lives up to the requirements posed by the audit data volume generated by a busy server system at the Center for Communication and Information Processing at the University of Dortmund.

The remainder of this paper is organized as follows: Section 2 justifies our design decision to pseudonymize *syslog* audit data and introduces the relevant concepts. In Sect. 3 we detail the design decisions for integration of the pseudonymizer, explain the trust model and outline our approach to pseudonymization. The architecture of our implementation also is illustrated in Sect. 3. In Sect. 4 we evaluate the performance of the implementation. We position our contribution wrt. related work in Sect. 5 and conclude in Sect. 6.

2 A Case for *syslog*-Pseudonymization

Most modern operating systems offer comparable audit components, e.g. AIX, BSDs, HP-UX, Linux, Solaris, Windows-NT and its descendants. All of these record the identity of the actor of an event or at least identifying features of

users. Audit components specifically designed for compliance with the Trusted Computer System Evaluation Criteria (TCSEC) (C2 or higher) [9,10] or the Common Criteria (FAU_GEN1.2 and FAU_GEN2) [11] are required to record information identifying the originator of an event. Hence, all of these audit components record personal data and need to be considered for pseudonymization. In addition to the audit components of the operating system, application processes record audit data containing personal data, such as web servers and mail exchangers.

2.1 Event Sources

When considering pseudonymization of audit data all relevant event data sources mentioned above should be covered. With our approach we aim at a non-invasive, easy to install tool that covers as many categories of audit data as possible. We therefore decided to build a pseudonymizer for *syslog* audit data. Due to its uniformity and availability in most significant Unixes many event sources in the Unix world leverage *syslog* for recording audit data. Also active network components such as routers provide for remote audit recording using *syslog*. Finally there are several third party products available for Windows driven systems to integrate with an existing Unix *syslog* infrastructure. By supporting *syslog* we can pseudonymize host-based, network-based and out-of-band audit data. Refer to [1,12,13] to learn more about *syslog*.

Event sources not using *syslog*, but directing their audit data to dedicated files are covered by our *redirector*. In practice with our tools we cover most of the audit data usually collected in Unix systems. For a more elaborate discussion on the diverse sources for audit data in Unix systems refer to [3].

2.2 Audit Data

As mentioned above, host-based, network-based and out-of-band audit data converge in *syslog*. The audit data collected via *syslog* describes events and conditions monitored by diverse components of the IT system. These components are categorized as *facilities* in *syslog* parlance. A facility that monitors an event or condition supplies a rating of its *severity* [13,12]. The *priority* of an event record is defined as the pairing of its generating facility and its severity. The severity ranges from debugging and informational over error conditions to emergency conditions. Mostly informational and diagnostics messages from the kernel and from network services are recorded via *syslog*. Refer to [1] for some annotated examples.

Also serious actions of users associated with login sessions are recorded by *syslog*. Depending on the system configuration a variety of other sorts of events may be reported to *syslog*. As an example the intrusion detection system *Snort* [14] is capable of reporting attack detections to *syslog*. Basically, *syslog* audit data is collected and used for troubleshooting and limited manual intrusion detection. Also some intrusion detection systems analyze *syslog* audit data, such as the *STAT Tool Suite* [15].

3 Embedding the Pseudonymizer

There are several opportunities for pseudonymization of *syslog* audit records along the path from the audit component to the final recipient. While it would be possible to integrate pseudonymization with all audit components, such an approach does not scale well. Also audit records should be pseudonymized before they reach the final recipient, which may be a privacy adversary.

Firstly, a straightforward approach is to pseudonymize the output of `syslogd` by reconfiguring `syslogd` to write into named pipes being read by pseudonymizers. The problem with this solution is, that a pseudonymizer is not able to determine the priority of the local output records of `syslogd`. This information is lost, but may be required for effective log analysis.

Secondly, there are two ways to pseudonymize audit records before they enter `syslogd`. We can wrap the *syslog* API function calls, but this approach does not catch audit records received from the kernel or via the network. Instead, a *wrapper* may pick up audit records from the *syslog*-sockets, while `syslogd` is configured to receive the pseudonymized audit records from the wrapper. We implemented such a pseudonymizing wrapper (see Fig. 1).

Thirdly, we can embed pseudonymization within `syslogd` by patching its source code or by replacing the existing `syslogd` by a pseudonymizing `syslogd`. The disadvantages of these solutions are that either the access to system-dependent source code of `syslogd` is required, which may not be readily available for all Unixes, or the existing implementation of `syslogd` may differ significantly from the replacement version. In addition to the wrapper we provide source code patches to optionally embed pseudonymization within `syslogd`.

Some audit components write audit records directly to files without involving `syslogd`. As long as our pseudonymizer can parse these records, it may also read from these files. Sometimes it is useful to consolidate the audit records of as many audit components as possible in one place, f.i. to use the same pseudonymizer for these records. We supply a *redirector*, which picks up the audit records from a named pipe or a growing file and deposits them by means of the *syslog* API.

3.1 Trust Model

When considering audit data generated on the local host we are always confronted with the risk of integrity loss in case an attacker achieves sufficient privilege to manipulate audit data or the process of its generation. Even in the face of these problems *syslog* audit data today is still one very important source of information that is used to resolve attack situations or to gather early indications thereof.

In the same way in which an attacker can manipulate audit data or its generation, he may also corrupt the integrity of pseudonyms in pseudonymized audit data to evade later identification. We argue that we can rely on the pseudonyms generated on a host under attack as long as we can rely on the *syslog* data generated on that host. Stronger techniques for pseudonym generation are outside the

scope of this approach. Refer to Sect. 5 for related work. Pseudonymization can be used to make inferences on the pseudonymized data impractical. However, we cannot avoid inferences that take into account additional information outside the scope of pseudonymization.

Our approach to pseudonymization balances conflicting security requirements of at least two parties. On the one hand the users of an IT system wish to use it anonymously to protect their privacy. On the other hand the IT system owner's representative responsible for the system security requires accountability in the case of policy violations. Other interests in identifying users are not considered here, such as direct marketing by means of user profiling.

To put the mutual security requirements for anonymity and accountability not at risk, neither of the above mentioned parties can be allowed to control the audit components, `syslogd` and the pseudonymizer. Hence we introduce a third party denoted as the personal data protection official (PPO), who controls above mentioned system components (see the PPO domain in Fig. 1). The PPO is trusted by the users to protect their pseudonymity and he is trusted by the site security officer(s) (SSO) to ensure accountability in the face of a security incident.

In the current implementation audit components (with the help of *redirectors*) feed audit records via a *wrapper* or via *syslog* into the pseudonymizer (see Fig. 1 and Sect. 3.2). The pseudonymizer substitutes predefined types of identifying information, henceforth denoted as *features*, by pseudonyms. We exploit Shamir's threshold scheme for secret sharing to generate the pseudonyms. Owing to the properties of the threshold scheme we use, the features behind the pseudonyms can be recovered only under certain conditions later on (see Sect. 3.2). Thus, attacks on the pseudonymity of the system users are anticipated and resisted as soon as the pseudonymized audit records leave the pseudonymizer. Note that audit records should be pseudonymized on the same physical component where they are generated by the audit components. This may not be possible in some situations, f.i. when consolidating audit data generated by routers or Windows driven systems as is proposed in Sect. 2.1. In such cases additional measures must be taken by the PPO to provide a secure channel between the audit component and the pseudonymizer.

The pseudonymized audit data is delivered to and analyzed by the SSO to acquire indications for policy violations (see the SSO domain in Fig. 1). In order to support aforementioned analysis the pseudonyms of a given identifying feature are linkable wrt. a given suspicion. Given sufficient suspicion, the pseudonyms can be used to reveal the originally substituted features in the hope to gain more information helping to identify the perpetrator. In our approach sufficient suspicion is defined by a threshold on weighted occurrences of potentially attack-related events. The recovery of a feature is possible if and only if the corresponding threshold is exceeded. Thus, the SSO's requirement for accountability is met if the definition of sufficient suspicion is satisfied, but not otherwise, conforming to the user requirement for pseudonymity. The recovery of identifying features

is done by the *reidentifier*, which is – unlike the pseudonymizer – fed with the output files of `syslogd` (see Fig. 1).

3.2 Pseudonymizing *syslog*

The pseudonymizer receives audit records from `syslogd` containing identifying features. Since each feature may contribute with a specific weight to a scenario legitimating feature recovery, the pseudonymizer is able to distinguish different event types and associates each embedded feature type with one or more scenarios and the respective weight(s). The PPO specifies apriori knowledge about the syntax of events and features to be pseudonymized. In cooperation with the SSO, the PPO models suspicions justifying reidentification by specifying associations between features and scenarios, defining the weight(s) for the contribution of each feature to its associated scenario(s), as well as the scenario thresholds. This apriori knowledge is stored in the configuration files of our tools. Consequently, feature recovery is tied to the scenarios that are configured when the features are pseudonymized.

Parsing: To locate and pseudonymize identifying features in *syslog* audit records, we have to parse them. Since the syntax of the records is mainly determined by the originating audit component, recognition of different event and feature types is based on the evaluation of syntactical contexts, e.g. by means of regular expressions for pattern matching as defined in POSIX 1003.2 Sect. 2.8. Refer to [1] for the details.

When parsing audit records we use the apriori knowledge stored in the configuration files for our tools. By sequentially matching audit component, event type context, and feature type contexts, we isolate the features of the record that are to be pseudonymized. According to the assigned weight and scenario, a number of pseudonyms is generated, replacing the respective feature. The pseudonymized audit records retain the format of the unpseudonymized version.

Generating Pseudonyms: The basic idea of our approach to generating pseudonyms is to have the pseudonymizer split an identifying feature into as many pseudonyms as are needed to pseudonymize audit records containing this feature. The pseudonyms have the property, that given a number of pseudonyms equal to the associated scenario threshold or more, but not less, the reidentifier is able to recover the corresponding feature. Secret sharing threshold schemes are suitable to fulfill these requirements.

For our purposes we exploit Shamir's threshold scheme, as described in detail in [16]. Owing to different conditions of deployment of Shamir's threshold scheme, we make some modifications with regard to its application. For a more technically inclined description of these modifications and the basic algorithms used by the pseudonymizer and the reidentifier, refer to [4].

In a nutshell, the pseudonymizer encrypts each identifying feature and the corresponding decryption key is split into shares, which are used as pseudonyms

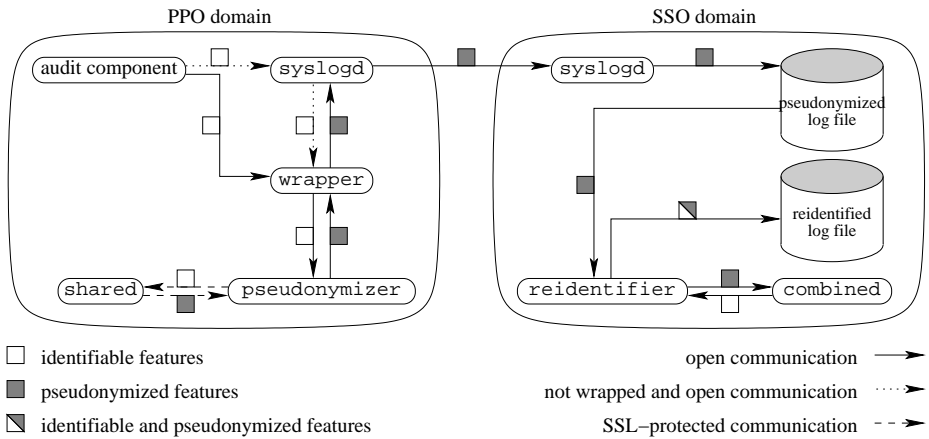


Fig. 1. Interplay with a wrapped syslogd

for that feature. The reidentifier uses Lagrange interpolation to recover the decryption key from a sufficient number of shares belonging to the same feature.

Architecture: We supply tools for the PPO and for the SSO, namely **redirector**, **wrapper**, **pseudonymizer**, **shared**, and **reidentifier**, **combined**, respectively. The tools are portable and have been used successfully on Solaris, OpenBSD and Linux.

According to Sect. 3.1 the audit components, **redirector**, **wrapper**, **syslogd**, **pseudonymizer** and **shared** are controlled by the PPO. The audit components are configured to generate only audit data required for sustained service provision, such as audit data indicating proceeding attacks against the site. The audit data is written to `/dev/log` (with the help of **redirectors**), to UDP port 514 and to other locations usually read by **syslogd**, f.i. `/dev/klog` for OpenBSD kernel audit data. In Fig. 1 the **wrapper** intercepts audit data at `/dev/log` and at UDP port 514². Audit data that cannot be intercepted³ is received by **syslogd** and sent on to the **wrapper** (see the dotted lines in the PPO domain in Fig. 1). The **pseudonymizer** is fed with audit data by the **wrapper**. This solution can be used if **syslogd** shall not be replaced or modified. Optionally we provide a patch for **syslogd** such that incoming audit data is sent directly to the **pseudonymizer** without involving the **wrapper**. For this solution **syslogd** needs to be replaced by a patched version (refer to [1] for details).

The audit data contains identifying features that need to be pseudonymized. The **pseudonymizer** parses each incoming audit record, determines the appro-

² We actually use UDP port forwarding to redirect packets sent to port 514 to the port where the **wrapper** listens.

³ For example OpenBSD kernel audit data written to `/dev/klog` cannot be intercepted because the OpenBSD **syslogd** is hard-coded to read kernel audit data from `/dev/klog`.

priate scenarios and asks **shared** for pseudonyms for the features wrt. these scenarios. **shared** provides the cryptographic primitives, such as symmetric encryption and secret sharing. For encryption **shared** uses the OpenSSL crypto library [17] and the threshold secret sharing scheme is implemented using the GNU Multiple Precision Arithmetic library (GMP) [18]. **shared** may run on the same machine as the **pseudonymizer**, or it may run remotely. This allows several distributed **pseudonymizers** to use a central **shared** with the result that pseudonyms for a given identifying feature are linkable across the corresponding distributed audit components. The protocol used for communication between the **pseudonymizer** and **shared** transfers identifiable features. Since the PPO might not control the network between the **pseudonymizer** and **shared**, the channel is protected using the OpenSSL SSL/TLS library [19] when **shared** runs remotely. The **pseudonymizer** replaces the identifying features with the pseudonyms delivered by **shared**. The pseudonymized audit data is then handed to **syslogd** (via the **wrapper**). In the recommended setup **syslogd** sends the pseudonymized audit records to a remote **syslogd** running under the control of the SSO. Note that using the recommended setup audit data is not written into disk files before it is pseudonymized and received in the SSO domain.

In compliance with Sect. 3.1 the **syslogd** receiving pseudonymized audit records from the **pseudonymizer** as well as the **reidentifier** and **combined** are controlled by the SSO. **syslogd** receives the pseudonymized audit records and handles them according to the SSO's requirements. If the SSO wishes to be able to recover identifying features some time later, **syslogd** needs to be configured to write those audit records to a log file (see Fig. 1). When needed, the **reidentifier** is started manually or automatically on an alarm. It reads the log file, determines which pseudonyms satisfy their scenario and asks **combined** to recover the respective identifying features. **combined** provides cryptographic primitives, such as symmetric decryption and Lagrange interpolation, the reverse operation for the threshold secret sharing scheme we use. For decryption **combined** also uses the OpenSSL crypto library [17] and the Lagrange interpolation again is implemented using the GMP library [18]. **combined** may run on the same machine as the **reidentifier**, or it may run remotely. The protocol used for communication between the **reidentifier** and **combined** transfers identifiable features. Since identifiable features are transferred only if a scenario is satisfied, i.e. the SSO is allowed to see the features, there is no need for a channel providing confidentiality.⁴ The **reidentifier** replaces all recoverable pseudonyms with the corresponding identifying features delivered by **combined**.

4 Performance

The **pseudonymizer** processes audit data as soon as it has been generated. It is thus required, that it is able to pseudonymize audit records sufficiently fast to avoid a bottleneck, which might impair overall system performance. In contrast,

⁴ If the SSO does not control the network between **combined** and the **reidentifier** a confidential channel can be provided.

the `reidentifier` is sparsely used and needs not to satisfy real-time requirements.

The performance measurements were conducted on a single processor Pentium III 650MHz machine with 256MB RAM and a 100Mbps Fast-Ethernet NIC, running OpenBSD 2.7. For all the details about the presented performance measurements and for additional results refer to [1].

4.1 Microbenchmarks for the Cryptographic Components

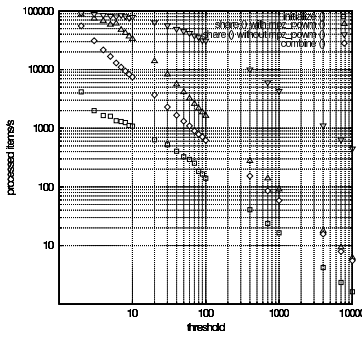
We measured the performance of the cryptographic primitives used in `shared` and `combined`. For secret sharing and Lagrange interpolation they operate over $GF(P)$ where P is a prime number of 128 bits. For encryption we also use symmetric 128 bit keys. Thresholds used in the measurements default to 5.

The first time `shared` processes a given feature it uses the `initialize()` routine, which generates a pseudo-random polynomial, encrypts the feature and inserts the results in an AVL tree. We found that the Blowfish encryption in the OpenSSL crypto library operates at about 15370 encryptions per second for features of eight characters. It slows down slightly to about 15150 encryptions per second for features of 32 characters. We expect most features to be pretty short strings. The encryption speed is independent of the number of bits used for the symmetric keys. Depending on the threshold of the scenario associated with a feature, `initialize()` generates a number of pseudo-random coefficients for the corresponding polynomial. The higher the threshold, the more pseudo-random numbers are generated (see `initialize()` in Fig. 2a). `share()` basically evaluates the polynomial `initialize()` provides. Hence, the higher the threshold, the more expensive the exponentiations `share()` calculates using the GMP library call `mpz_powm()` (see `share()` with/without `mpz_powm()` in Fig. 2a).

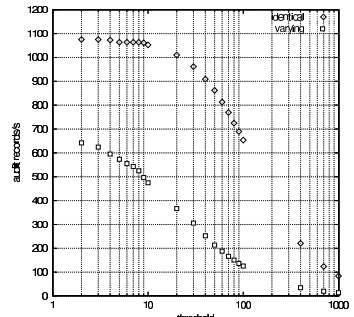
4.2 Macrobenchmarks for the Pseudonymizer

We measured the performance of the `pseudonymizer` for varying parameters using synthetic audit data and configurations. `shared` handles features differently depending on whether they have already been seen in the audit data or not. If a feature is processed for the first time, `initialize()` generates a pseudo-random polynomial and stores it together with the encrypted feature (see Sect. 4.1). In any case a polynomial is used to generate a fresh pseudonym using `share()` (see Sect. 4.1). Owing to these different cases of feature handling we used two kinds of audit data to measure the best and the worst case performance. In the best case all features associated with a given scenario were *identical*. `shared` then calls `initialize()` only once per scenario. In the worst case we had *varying* features associated with a given scenario. As a result `shared` calls `initialize()` for every feature in the audit data.

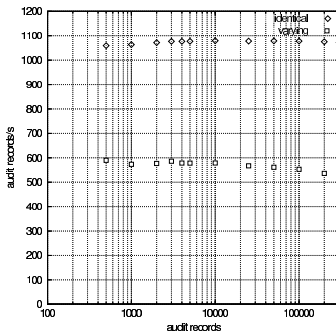
When increasing the number of audit records the `pseudonymizer` processes, we observe for varying features a mild performance penalty due to the growth of the AVL trees storing encrypted features and their corresponding polynomials (see Fig. 2c). A growing number of audit components, event types and scenarios



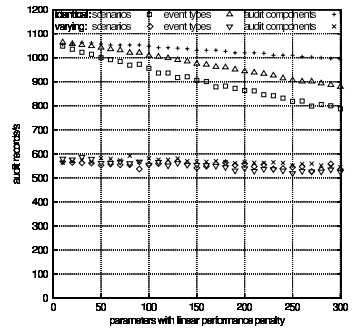
(a) Microbenchmarks



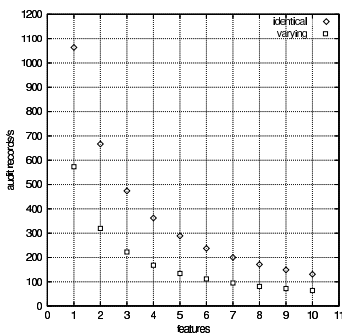
(b) Threshold



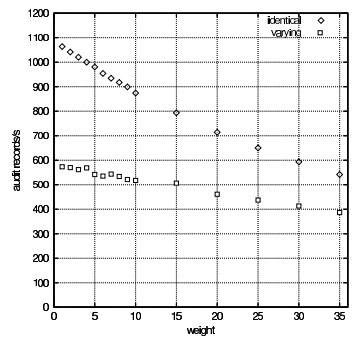
(c) Audit records



(d) Linear influence



(e) Features



(f) Weight

Fig. 2. Performance measurements of the pseudonymizer

has a linear influence on the performance since these objects are managed using lists (see Fig. 2d). The more features per audit record are pseudonymized, the more often the `pseudonymizer` asks `shared` for pseudonyms. In addition to the communication overhead more feature type contexts need to be matched by the `pseudonymizer` (see Fig. 2e). The pattern matching is the main cause for the performance loss when many features need to be pseudonymized per audit record. In contrast to this the performance loss caused by generating pseudonyms with larger weights is moderate (see Fig. 2f). There is no additional pattern matching or communication overhead involved. Increasing the threshold does not only slow down the generation of pseudonyms using `share()`, but also the initial generation of pseudo-random numbers for each distinct feature in a scenario using `initialize()` (see `initialize()` and `share()` with `mpz_powm` in Fig. 2a in comparison to Fig. 2b). However, in practice the parameters will only in rare cases reach values as high as shown in Fig. 2. In the next section we show that even in these rare cases the performance of the `pseudonymizer` is sufficient to cope with the audit record volume of a large site.

4.3 How Fast Is Fast?

To be able to judge whether our tools perform fast enough in a real world server environment we evaluated *syslog* and *Apache* audit records from a central server at the Center for Communication and Information Processing at the University of Dortmund. The machine hardware consists of a SUN Ultra Enterprise 4000 with 3GB RAM, six Ultra SPARC 168MHz CPUs, three disk arrays totaling 396GB and a 100Mbps full-duplex Fast-Ethernet uplink. The server runs Solaris 7 and about 1050 users are registered, which are employees and students of the University of Dortmund. During working hours an average of 25 users is logged on simultaneously. We took into account the following services provided by the machine: 37 world accessible *Apache* web servers, one FTP server with about 112000 FTP transfers per month with a volume of 12GB, and email services (SMTP, IMAP, POP) with about 45000 emails per month.

We evaluated all *Apache* access audit records collected over a period of 13 days for all 37 web servers. We have also evaluated all *syslog* audit records collected over a period of four weeks. We observed, that the `pseudonymizer` is able to keep up with the number of audit records generated even in situations where some audit component generates an abnormally high number of audit records. See Table 1 for the maximum numbers of audit records we measured. For more details about the evaluation and for statistics refer to [1].

5 Related Work

Our tools can be applied to server log data in order to hide information that may directly or indirectly identify the users that accessed the server. There are other well known services available to achieve this at the price of installing some client-side software or requiring the user to perform additional steps in order

Table 1. Maximum number of audit records generated per second

audit component	number of audit records	
	per hour	per second
<i>Apache</i>	33506	9.31
<i>syslog</i>	4956	1.38
Σ	38462	10.68

to achieve anonymity. More importantly, many of these anonymity tools do not provide for conditional identity recovery.

On the web the P3P framework helps users to match their privacy requirements against the privacy policies of the sites they visit. While this approach allows for fine grained control over what personal data is revealed at the application layer, identifying information from lower layer protocols is not protected. For this purpose there are also many different anonymizing services for accessing the web, such as the *Anonymizer*, *Anonymouse*, *Proxymate* formerly known as *LPWA*, *Crowds*, *Onion Routing* and *JAP*. In addition to filtering the transmission wrt. personal data, these services usually involve one or more intermediaries and aim at decoupling the client from the target server. Thus, identifying data is hidden before it reaches the target server. Similar examples for anonymous mail transfer are *Anonymouse* or remailers such as *Cypherpunk* and *Mixmaster*. Since the techniques employed, f.i. MIXes, are entirely different from the ones used for pseudonymizing log data, we do not go into the details here. These services are surveyed and compared in [20,21,22].

The techniques mentioned above are conducted under the control of the user himself and/or at least one third party trusted by the user wrt. anonymization. Thus, no trust is required in the target service provider. On the other hand these approaches require the user to take various actions to be able to communicate anonymously.

Pertinent legislation in various western countries requires providers of some classes of target services to support anonymous use of their service. To comply with these requirements, service providers need technical means, independent from actions their clients may take to protect their privacy. Existing services based on Unix systems often record personal data in log files. Reconfiguring the system such that it does not record certain kinds of log events is not always an option. The information supplied by some log events may be mission critical and may also contain personal data. One straightforward and easily deployable approach is to anonymize or pseudonymize identifying data after it has been collected, but before it is recorded in log files.

The *Anonymouse log file anonymizer* [20] is implemented as a customizable Perl script that replaces identifying information by default values or more coarse values. Anonymized log files may still be analyzed. In case some event happens, which requires further investigation wrt. the user's identity, it would be useful to be able to recover the original data, which has been anonymized. This feature is not supported by the *Anonymouse log file anonymizer*. The situation is similar

for the aforementioned MIX technologies. Indeed, even in the case of misuse of a service by an anonymous user it is infeasible for the provider or for investigating authorities to establish accountability. Contrary to this our tools provide for recoverability of pseudonymized data under certain conditions.

Due to this property our work is strongly related to privacy enhanced intrusion detection systems. Most of these systems use pseudonymization to hide identifying information after it has been generated and recorded in audit data. Some approaches have been published for the *Intrusion Detection and Avoidance* system (IDA) [23], the *Adaptive Intrusion Detection* system (AID) [23], a Firewall audit analyzer [24] and for the *Aachener Network Intrusion Detection Architecture* (ANIDA) [25]. These approaches have been reviewed and compared to our work in [2]. Intrusion detection systems (IDS) do not only give warnings in case of attacks, they also record evidence for further investigation. In case an attacker is to be prosecuted, it is desirable to extract identifying information from the recorded data. Privacy enhanced IDS support the recovery of pseudonymized information. A common weakness of existing privacy enhanced IDS is that they do not technically bind the recovery of identifying information to a specific purpose such as the investigation of a proceeding attack. Some privacy enhanced IDS require a third party to cooperate in the recovery process. The users have to trust that party to allow recovery only for given legitimate purposes. Other privacy enhanced IDS provide no strong protection against recovery for other purposes [2]. In contrast to this our tools instantly allow for recovery of pseudonymized data if predefined conditions bound to a specific purpose are met, but not otherwise.

Similar mechanisms are found in fair offline electronic payment systems. Such systems allow for anonymous payment as long as no payment unit has been spent more often than permitted. In the case of double spending of electronic coins, the perpetrator can be traced [26]. Some of the recovery techniques used are similar to the scheme our tools employ. Others use group signatures to achieve recoverability [27].

Payment units in fair offline electronic payment systems are also similar to anonymous credentials, which can be used for anonymous authentication. Many schemes for anonymous authentication support identity recovery while employing different techniques. Some schemes have been related to our approach in [4]. Since the resources of the target service are accessed using anonymous and pseudonymous credentials or payment units, the target service does not learn identifying information about the users during authentication. The users do not need to trust the target service provider regarding their anonymity. Also, malformed credentials, which do not exhibit the properties required by the target service can be rejected, avoiding further damage. On the other hand, before they can access any target service anonymously, users need to register with some third party to generate credentials and payment units. Since no such infrastructure has yet been widely adopted, in the meantime service providers can pseudonymize log data to offer privacy enhanced service access.

6 Conclusion

We motivated the need for technical enforcement of privacy principles and surveyed related work on privacy enhancing technologies. Though many of the related approaches allow for a strong trust model, they require the user to take additional steps and/or install client-side software. Also many approaches require a widely accepted infrastructure for issuing and managing credentials and payment units. None of these infrastructures is yet in widespread use, i.e. service providers cannot adopt widely accepted standards.

In contrast to this situation service providers are expected to respect and protect the privacy of their clients. We supply tools that help to retrofit privacy protection into existing Unix systems by means of pseudonymization of identifying features in audit data. Different from existing log file anonymizers or privacy enhanced intrusion detection systems, our approach balances requirements regarding anonymity and accountability. That is, pseudonymized identifying features can be revealed only if given definitions of scenarios requiring to do so are satisfied, but not otherwise. A trusted third party ensures that these scenarios meet the requirements regarding anonymity and accountability. This party controls the technical components that enforce the proper pseudonymization, such that later on identifying features can be recovered.

We described the trust model and architecture of the tools, and we presented performance measurements of the pseudonymizer. Finally we demonstrated measurements of the audit data volume generated by a real world system, indicating that the performance of the `pseudonymizer` should be sufficient for application at sites with even larger audit data volume.

Acknowledgments. We gratefully acknowledge the efforts of Hans Bornemann, Michael Heyer and Dieter Stolte in providing data about their networks and machines. Most of all Sven Bursch, Kai Grundmann, Stefan Magerstedt and Dennis Real deserve credit for their tireless work on the code as well as the measurements of the tools.

References

- [1] Ulrich Flegel. Pseudonymizing Unix log files. Technical report, Dept. of Computer Science, Chair VI Information Systems and Security, University of Dortmund, D-44221 Dortmund, May 2002. Extended version of this paper.
<http://ls6-www.cs.uni-dortmund.de/issi/archive/literature/2002/Flegel:2002a.ps.gz>.
- [2] Joachim Biskup and Ulrich Flegel. On pseudonymization of audit data for intrusion detection. In Hannes Federrath, editor, *Proceedings of the international Workshop on Design Issues in Anonymity and Unobservability*, number 2009 in LNCS, pages 161–180, Berkeley, California, July 2000. ICSI, Springer.

- [3] Joachim Biskup and Ulrich Flegel. Transaction-based pseudonyms in audit data for privacy respecting intrusion detection. In Hervé Debar, Ludovic Mé, and S. Felix Wu, editors, *Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection (RAID 2000)*, number 1907 in LNCS, pages 28–48, Toulouse, France, October 2000. Springer.
- [4] Joachim Biskup and Ulrich Flegel. Threshold-based identity recovery for privacy enhanced applications. In Sushil Jajodia and Pierangela Samarati, editors, *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 71–79, Athens, Greece, November 2000. ACM SIGSAC, ACM Press.
- [5] Privacy survey results, January 2002.
<http://www.cdt.org/privacy/survey/findings/>.
- [6] Louis Harris & Associates Inc. IBM multi-national consumer privacy survey. Technical Report 938568, IBM Global Services, 1999.
- [7] Jarek Rossignac et al. Gvu's 10th WWW User Survey, December 1998.
http://www.cc.gatech.edu/gvu/user_surveys/survey-1998-10/graphs/graphs.html#privacy .
- [8] Steven R. Johnston. The impact of privacy and data protection legislation on the sharing of intrusion detection information. In Lee et al. [28], pages 150–171.
- [9] National Computer Security Center. US DoD Standard: Department of Defense Trusted Computer System Evaluation Criteria. DOD 5200.28-STD, Supercedes CSC-STD-001-83, dtd 15 Aug 83, Library No. S225,711, December 1985.
<http://csrc.ncsl.nist.gov/secpubs/rainbow/std001.txt>.
- [10] National Computer Security Center. Audit in trusted systems. NCSC-TG-001, Library No. S-228,470, July 1987.
<http://csrc.ncsl.nist.gov/secpubs/rainbow/tg001.txt>.
- [11] Common Criteria Implementation Board. *Common Criteria for Information Technology Security Evaluation — Part 2: Security functional requirements, Version 2.1*. Number CCIMB-99-032. National Institute of Standards and Technology, August 1999. <http://csrc.ncsl.nist.gov/cc/ccv20/p2-v21.pdf>.
- [12] C. Lonvick. *RFC 3164: The BSD syslog Protocol*, August 2001.
<http://www.ietf.org/rfc/rfc3164.txt>.
- [13] *syslog.conf*. Manual Page.
- [14] Martin Roesch. Snort – lightweight intrusion detection for networks. In *Proceedings of LISA '99: 13th Systems Administration Conference*, pages 229–238, Seattle, Washington, November 1999. The Usenix Association, Usenix.
- [15] Giovanni Vigna, Richard A. Kemmerer, and Per Blix. Designing a web of highly-configurable intrusion detection sensors. In Lee et al. [28], pages 69–84.
- [16] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.
- [17] *OpenSSL cryptographic library*, December 2001.
<http://www.openssl.org/docs/crypto/crypto.html>.
- [18] Torbjörn Granlund. *The GNU Multiple Precision Arithmetic Library*. GNU, 3.1.1 edition, September 2000. <http://www.gnu.org/manual/gmp/index.html>.
- [19] *OpenSSL SSL/TLS library*, December 2001.
<http://www.openssl.org/docs/ssl/ssl.html>.
- [20] Claudia Eckert and Alexander Pircher. Internet anonymity: Problems and solutions. In Michel Dupuy and Pierre Paradinas, editors, *Proceedings of the IFIP TC11 16th International Conference on Information Security (IFIP/Sec'01)*, pages 35–50, Paris, France, June 2001. IFIP, Kluwer Academic Publishers.

- [21] Simone Fischer-Hübner. *IT-Security and Privacy: Design and Use of Privacy-Enhancing Security Mechanisms*. Number 1958 in Lecture Notes in Computer Science. Springer, 2001.
- [22] Oliver Berthold, Hannes Federrath, and Marit Köhntopp. Project “Anonymity and unobservability in the internet”. In *Proceedings of the Workshop on Freedom and Privacy by Design / Conference on Freedom and Privacy*, pages 57–65, Toronto, Canada, April 2000. ACM.
- [23] Michael Sobirey, Simone Fischer-Hübner, and Kai Rannenberg. Pseudonymous audit for privacy enhanced intrusion detection. In L. Yngström and J. Carlsen, editors, *Proceedings of the IFIP TC11 13th International Conference on Information Security (SEC'97)*, pages 151–163, Copenhagen, Denmark, May 1997. IFIP, Chapman & Hall, London.
- [24] Emilie Lundin and Erland Jonsson. Anomaly-based intrusion detection: privacy concerns and other problems. *Computer Networks*, 34(4):623–640, October 2000.
- [25] Roland Büschkes and Dogan Kesdogan. Privacy enhanced intrusion detection. In Günter Müller and Kai Rannenberg, editors, *Multilateral Security in Communications*, Information Security, pages 187–204. Addison Wesley, 1999.
- [26] George Davida, Yair Frankel, Yiannis Tsiounis, and Moti Yung. Anonymity control in e-cash systems. In R. Hirschfeld, editor, *Proceedings of the First International Conference on Financial Cryptography (FC'97)*, number 1318 in Lecture Notes in Computer Science, pages 1–16, Anguilla, British West Indies, February 1997. Springer.
- [27] Jaques Traoré. Group signatures and their relevance to privacy-protecting offline electronic cash systems. In J. Pieprzyk, R. Safavi-Naini, and J. Seberry, editors, *Proceedings of the 4th Australasian Conference on Information Security and Privacy (ACISP'99)*, number 1587 in Lecture Notes in Computer Science, pages 228–243, Wollongong, NSW, Australia, April 1999. Springer.
- [28] Wenke Lee, Ludovic Mé, and Andreas Wespi, editors. *Proceedings of the Fourth International Workshop on Recent Advances in Intrusion Detection (RAID 2001)*, number 2212 in LNCS, Davis, California, October 2001. Springer.

Annex – Example Application

Both, the `pseudonymizer` and the `reidentifier` leverage apriori knowledge provided in configuration files to locate identifying features and pseudonyms in audit data, respectively. Feature types are located using the audit component name (see the `FACILITY` keywords in Fig. 3), the event type context (see the `EVENT` keywords in Fig. 3) and the feature type contexts (see the `LEFTFEATURE` and `RIGHTFEATURE` keywords in Fig. 3). The apriori knowledge associates each feature type and corresponding pseudonyms with one or more scenarios (see the `GROUP` keywords next to the `RIGHTFEATURE` keywords in Fig. 3) as well as (a) corresponding weight(s) (see the `WEIGHT` keywords in Fig. 3), specifying the number of pseudonyms generated for the feature type. Also for each scenario a threshold is defined (see the `THRESHOLD` keyword at the top of Fig. 3), specifying the number of pseudonyms required for associated features to be recoverable from the corresponding pseudonyms.

In the following example we pseudonymize audit records generated by `tcplog`. The `tcplog` program logs TCP flag combinations that should not occur

in regular TCP traffic. Additionally `tcplog` is able to detect `queso` OS fingerprinting scans. `queso` is a reconnaissance tool used to determine the operating system version using the OS fingerprinting technique, offering a subset of the functionality of `nmap`.

We suppose that the PPO of a site considers IP addresses being sensitive information since they have the potential to identify the person behind the associated activities. The PPO thus decides to pseudonymize IP addresses.⁵ The SSO knows that for an intruder the information provided by `queso` is useful for determining if a given service provided by a host might be vulnerable to a particular exploit. Hence, the SSO wants to be able to identify the IP addresses originating `queso` scans against his site.

The PPO analyzes, which audit components generate event types containing IP address features, such as the sample audit records in Fig. 4. The SSO provides the PPO with information about scenarios that require feature recovery, such as the `queso` scenario. For the `tcplog` audit component the PPO in cooperation with the SSO extends the apriori knowledge stored for our tools in the configuration files as shown in Fig. 3.

```
GROUP I1 THRESHOLD 6
FACILITY tcplog EVENT 'QUESO'
  LEFTFEATURE ' from ', RIGHTFEATURE ' port' GROUP I1 WEIGHT const(1)
FACILITY tcplog EVENT ' from '
  LEFTFEATURE ' from ', RIGHTFEATURE ' port' GROUP I1 WEIGHT const(1)
```

Fig. 3. Sample configuration for pseudonymizing `tcplog` audit records, such as in Fig. 4

```
02:23:37 tcplog[1567]: FIN SYN RST          URG          : port 41067 from 217.82.199.102 port 42312
13:42:06 tcplog[1040]: SYN                RES2         : ftp from 192.168.1.4 port 45691
13:42:06 tcplog[1040]: FIN SYN          PSH         URG          : ftp from 192.168.1.4 port 45693
13:42:06 tcplog[1040]: FIN SYN          PSH         URG          : ftp from 192.168.1.4 port 45693
13:42:06 tcplog[1040]: FIN              PSH         URG          : port 34513 from 192.168.1.4 port 45697
13:42:06 tcplog[1040]: FIN              PSH         URG          : port 34513 from 192.168.1.4 port 45697
13:42:06 tcplog[1040]: QUESO: port 34513 from 192.168.1.4 port 45697
```

Fig. 4. Sample unpseudonymized audit records generated by `tcplog`, which contain a `queso` OS fingerprinting scan

When the tools have been configured for the `queso` scenario the audit records in Fig. 4 are pseudonymized by the `pseudonymizer`. The resulting audit records are shown in Fig. 5. For the sake of presentation we moved the additional records generated by the `pseudonymizer` to the bottom of Fig. 5 while retaining the relative order of the `tcplog` records and of the additional records. Note that in the `tcplog` records the IP addresses have been replaced by pointers to the additional

⁵ The PPO additionally might want to pseudonymize the port numbers. This also is perfectly possible, but for the sake of the brevity of the example we refrain from doing so.

records. The additional records provide the actual pseudonyms, i.e. the cryptographic material and other information required for the recovery of the replaced IP addresses. Note that the actual pseudonyms for IP address 192.168.1.4 are linkable using the identifier `d2petb50CX0un4CuLPhluM8`. Generally there may occur audit records describing activity that is definitely not related to any scenario, but where features need to be hidden. These audit records can be pseudonymized while omitting the additional records, such that recovery is infeasible later on.

```
02:23:37 tcplog[1567]: FIN SYN RST          URG          : port 41067 from a1 port 42312
13:42:06 tcplog[1040]:          SYN          RES2         : ftp from a2 port 45691
13:42:06 tcplog[1040]: FIN SYN          PSH          URG          : ftp from a3 port 45693
13:42:06 tcplog[1040]: FIN SYN          PSH          URG          : ftp from a4 port 45693
13:42:06 tcplog[1040]: FIN          PSH          URG          : port 34513 from a5 45697
13:42:06 tcplog[1040]: FIN          PSH          URG          : port 34513 from a6 45697
13:42:06 tcplog[1040]: QUES0: port 34513 from a7 port 45697
02:23:37 pseudonymizer: Short=a1 Long=Rw4gGpc2dWVL0wXgF20ENg:10:ECmA!_ph6k0hKLZBsMw!H7H2ztc Group=I1
13:42:06 pseudonymizer: Short=a2 Long=5QgfV3!umUu_Fj8MWI2yoA:5b:d2petb50CX0un4CuLPhluM8 Group=I1
13:42:06 pseudonymizer: Short=a3 Long=sW7b167o0o1G5eiBpmu7hg:5c:d2petb50CX0un4CuLPhluM8 Group=I1
13:42:06 pseudonymizer: Short=a4 Long=wdEey!pfsM4jGtz9IyejZw:5d:d2petb50CX0un4CuLPhluM8 Group=I1
13:42:06 pseudonymizer: Short=a5 Long=F17o9GJU!A5SswY0!dAcxsA:5e:d2petb50CX0un4CuLPhluM8 Group=I1
13:42:06 pseudonymizer: Short=a6 Long=rog6EDbIHE3V5moFB1Ar8Q:5f:d2petb50CX0un4CuLPhluM8 Group=I1
13:42:06 pseudonymizer: Short=a7 Long=it0SIYe5EYysleQNF0haNg:60:d2petb50CX0un4CuLPhluM8 Group=I1
```

Fig. 5. Sample pseudonymized audit records from Fig. 4

```
02:23:37 tcplog[1567]: FIN SYN RST          URG          : port 41067 from a1 port 42312
13:42:06 tcplog[1040]:          SYN          RES2         : ftp from 192.168.1.4 port 45691
13:42:06 tcplog[1040]: FIN SYN          PSH          URG          : ftp from 192.168.1.4 port 45693
13:42:06 tcplog[1040]: FIN SYN          PSH          URG          : ftp from 192.168.1.4 port 45693
13:42:06 tcplog[1040]: FIN          PSH          URG          : port 34513 from 192.168.1.4 port 45697
13:42:06 tcplog[1040]: FIN          PSH          URG          : port 34513 from 192.168.1.4 port 45697
13:42:06 tcplog[1040]: QUES0: port 34513 from 192.168.1.4 port 45697
02:23:37 pseudonymizer: Short=a1 Long=Rw4gGpc2dWVL0wXgF20ENg:10:ECmA!_ph6k0hKLZBsMw!H7H2ztc Group=I1
```

Fig. 6. Sample pseudonymized audit records from Fig. 5 with recovered features

The audit records in Fig. 5 are transferred to the SSO. When the SSO detects the `queso` scan contained in the audit records in Fig. 5 he uses the `reidentifier` to recover the IP address features associated with the satisfied `queso` scenario. The resulting audit records are shown in Fig. 6. Note that the pseudonym in the first audit record in Fig. 6 is not revealed since it corresponds to an IP address not involved in the `queso` scan originated at IP address 192.168.1.4.