LUARM – An audit engine for insider misuse detection

George Magklaras Steven Furnell and Maria Papadaki

Center for Security, Communications and Networks Research,
School of Computing and Mathematics, University of Plymouth,
Plymouth, Devon, PL4 8AA, United Kingdom
e-mail: georgios.magklaras@plymouth.ac.uk

Abstract

'Logging User Actions in Relational Mode' (LUARM) is an open source audit engine for Linux, although it can be easily ported to other Unix based systems. It provides a near real-time snapshot of a number of user action data such as file access, program execution and network endpoint user activities, all organized in easily searchable relational tables. LUARM attempts to solve two fundamental problems of the insider IT misuse domain. The first concerns the lack of insider misuse case data repositories that could be used by post-case forensic examiners to aid an incident investigation. The second problem relates to how information security researchers can enhance their ability to specify accurately insider threats at system level. This paper presents LUARM's design perspectives and a 'post mortem' case study of an insider IT misuse incident. The results show that the prototype audit engine has a good potential to provide a valuable insight into the way insider IT misuse incidents manifest on IT systems and can be a valuable complement to forensic investigators of IT misuse incidents.

Keywords

insiders, misuse, detection, auditing, logging, forensics

1. Introduction

The problem of insider IT misuse (the term ‘misuse detection’ or ‘misuse’ is also used in the literature) is a very real threat for the health of IT infrastructures and requires its own domain in the information security world (Furnell, 2004). The term 'threat' in an IT infrastructure can be regarded as a “set of circumstances that has the potential to cause loss or harm” (Pfleeger et al, 2003). As a result, in legitimate user context, these circumstances might involve intentional IT misuse activities such as targeted information theft, introducing or accessing

inappropriate material, and accidental misuse (e.g. unintentional information leak).

Numerous studies have tried to define the term “insider” in the context of Information Security. This is because there are many possible sub-contexts that are applicable to shedding light on different aspects of what an insider is and what she can do. A generic definition focusing on the attributes of the insider, from an organizational trust point of view is the following (Probst et al, 2009):”An insider is a person that has been legitimately empowered with the right to access, represent, or decide about one or more assets of the organization's structure”. This definition has a wide perspective and emphasizes a key aspect of an insider: that of trust. Trust is a property that goes beyond an IT system oriented view (system credentials, actions, indications). Whilst people who constitute direct threats might not have access to IT access credentials, they still can decide on policies, equipment procurement and other issues that can affect the well being of an IT infrastructure. A good example is an IT director that spends millions on a state of the art security system but does not bother to emphasize or make policies that dictate the flow of information inside the organization (employ that bypasses the system with a simple USB key, intentionally or accidentally).

Providing a way to detect threats and sense vulnerabilities is vital for the process of insider threat mitigation. One way to capture the essence of insider threats is to look at the way they occur in the real world. There are two sources of information to help us derive conclusions. One consists of insider case reports, as they are reported by the press. The other source of information is an established information security survey which provides a quantitative view of the problem manifestation.

The most widely known insider misuse cases are usually about intellectual property theft. The arrest of Lan Lee and Yuefei Ge by FBI agents (Cha, 2008) is a classic case. The arrested men were engineers of NetLogic Microsystems (NLM) until July 2003. During the time of

their employment, they were downloading trade sensitive documents from the NLM headquarters into their home computers. These documents contained detailed descriptions of the NLM microprocessor product line and funded a startup made by the two engineers. Eventually, their ties to the Chinese government and military were discovered by investigators.

A last example of how important are the consequences of insider actions is the recent Wikileaks case (Greenmeier et al, 2010), where hundreds of thousands of classified government documents were circulated to the public and caused even widespread diplomatic problems. Irrespective of motives and whether revealing this kind of information is right or wrong, the fact is that part of the Wikileaks material came as a result of whistle-blowing insiders, which misused their privileges to obtain the material in electronic form.

The previous case examples capture the extent to which insider actions (intentional or not) can be devastating. The actual frequency of insider IT misuse manifestation is revealed by information security surveys. The CSI 2009 survey (Richardson, 2009) has some interesting quantitative findings about insider IT misuse. The survey states that 43.2 percent of its 443 respondents stated that at least some of their losses were attributable to malicious insiders, whereas 16.1 percent of respondents estimated that nearly all their losses were due to the non-malicious, merely careless behavior of insiders. A comparable figure from the Information Security Breaches Survey 2010 (Potter et al, 2010) mentions that the 46% of their large organization respondents had data stolen or lost as a result of insider actions, placing an emphasis on the lack of security culture and its contribution to accidental insider IT misuse.

However, both mass media case descriptions and surveys do not provide the tools nor the methodology to systemically study and mitigate the problem. Insider IT misuse is a multi-factorial problem and one of the things insider misuse researchers really need is a repository of more detailed case descriptions with a focus on the

impact insider misuse actions have at computer system level (NSTISSAM). These case repositories could provide a clearer picture of how a threat realizes into a misuse act. This is the area of Insider Threat Specification, the core concept behind the proposed logging engine which is examined in the next section.

2. Insider Threat Specification and modelling

Threat specifications follow the principles of intrusion specification, a concept which is not new in the information security world. Techniques to describe threats exist for an entire range of information security products, from anti-virus software to several intrusion detection/prevention systems (IDS/IPS) (Bace, 2000), where threats are specified by anomaly detection, pattern matching (also known as misuse detection) mechanisms or a heuristic-based combination of the two.

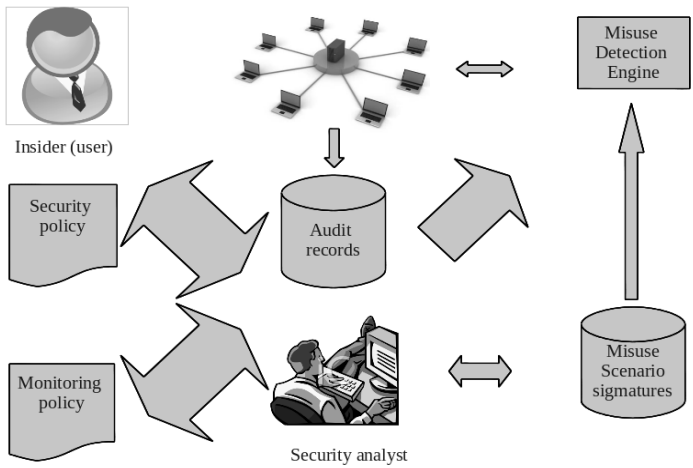

Figure 1: Information flow in an insider misuse detection system

Insider Threat Specification is the process of using a standardized vocabulary to describe in an abstract way how the aspects and behavior of an insider relate to a security policy defined misuse scenario. Figure 1 shows the information flow of a typical IT misuse

detection system that uses insider threat specification, illustrating the relations between user entities, the security and monitoring policies and the various components of the IT infrastructure.

The security specialist translates the Security (and resulting monitoring policy) into a set of misuse scenario signatures, standard descriptions of IT misuse acts that describe the behavior of a user at process execution, filesystem and network endpoint level (Magklaras et al, 2006). The involvement of a specialist is necessary as the process of translating from security to monitoring policy is error prone. The misuse scenario signatures and collected audit data (Bace, 2000) from the IT infrastructure are fed into a misuse detection engine. For every unique userid in an authentication domain and signature of the 'Misuse Scenario signatures' repository, the engine produces a triplet encoded association in two distinct ways:

- In a threat detection context, where the specialist is interested to detect a particular type of misuse, the output of the misuse engine is a triplet of the form { Yes/No,user,signature-id}, i.e. we have a misuse occurrence or we do not.

- In a threat prediction context, where the specialist is trying to predict the occurrence of a particular misuse incident, the output triplet looks like {EPT, user, signature-id}, where EPT stands for Evaluated Potential Threat, indicating the likelihood of the occurrence of an incident as dictated by the signature.

Insider threat detection and prediction are important mitigation techniques. However, in the case of threat prediction, a threat model is needed to produce the metrics that evaluate the EPT score. Insider threat modeling is a large topic outside the scope of this paper and the reader can refer to reviews of models, to gain a better understanding of how EPT scores are derived by various modelling approaches (Magklaras et al, 2010).

One important point needs to be emphasized. Vital to insider threat specification is the structure and content of the audit record, at the center of Figure 1. If the audit record is incomplete, in terms of the

type of information we need to log or unavailable, because the data are vanished due to bad system design or intentional data corruption, the specification of insider threats is useless. This is where LUARM comes into the game.

3. Insider misuse detection auditing and its requirements

Bace (Bace, 2000) discusses intrusion detection (and hence misuse detection) as an audit reduction problem. Audit reduction is the process of filtering the relevant information out of the audit records, in order to infer a partially or fully realized threat and excluding information that is irrelevant or redundant. In the process of designing an insider threat detection system, a great deal of emphasis should be placed on the content and structure of audit records, as they constitute the source of information of every misuse detection system.

The balance between too little and too much information on the audit record is a difficult one. Providing too much information makes the task of audit reduction difficult and not scalable, as the number of monitored system grows. In addition, redundant or irrelevant information might be difficult to relate to language semantics. In contrast, collecting data at a too coarse a level of detail can exclude vital information for the presence of a threat, cause false negative assessments and reduce the expressiveness of a misuse detection language.

The structure of an audit record is also important for a misuse detection system. A good structure has well defined fields that can be easily parsed. Moreover, the structure of the audit record should easily facilitate relational type (Codd, 1990) queries. It is necessary for the information to be applied on the disjunction (OR), conjunction (AND), and negation (NOT) operators, in order to increase the query versatility and speed of response.

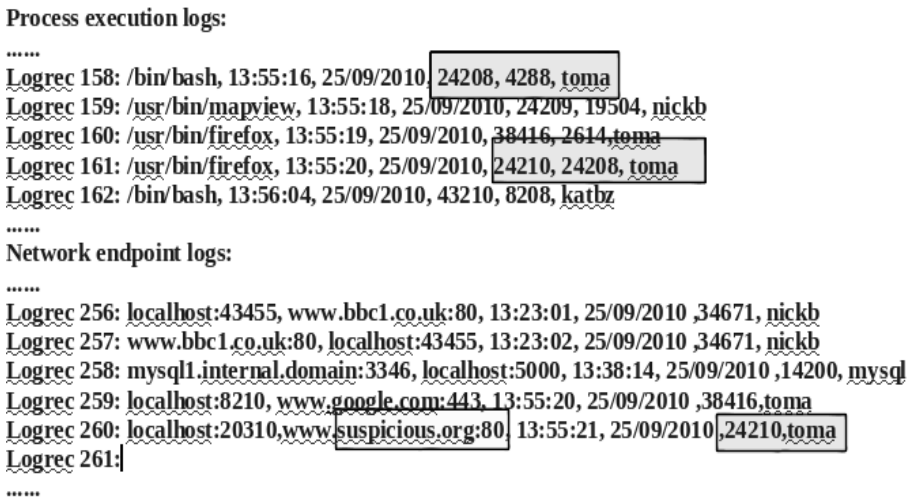

Figure 2: Events-to-user correlation in plain audit records

A final desired aspect of a suitable crafted audit record format for insider misuse detection is clear user accountability. This means that the audit record should be able to reliably and easily associate user entities to recorded actions. This desired mapping means that each recorded action could also be correlated to other actions of the same user, so that a set of actions can be related to a threat and the query language has enough information to perform step instance selection. Figure 2 illustrates such a correlation, by showing parts of the process execution and network endpoint creation log of a hypothetical audit record engine.

Let us assume that we wish to find whether user 'toma' has accessed the website 'www.suspicious.org', via a web browser, between 13:40 and 14:00 hours on the 25th of September 2010. In order to find such an event, we need to intercept the launching of a web browser process by user entity 'toma'. We would assume that the web browser will generate a network connection to 'www.suspicious.org'. The way one can relate the two events as part of a complex event (step instance selection) is to match the two events against a set of common identifiers, such as the process ID (PID), parent process ID (PPID) and username. This assumes that the process ID launching record has both

PID and PPID data inside each process execution record. It also assumes that the corresponding network endpoint log structure has a corresponding PID and username field that could be correlated.

The log snapshot of Figure 2 shows that there are two instances of user 'toma' executing a web browser. Only one of them is relevant and the correlation can be performed because the PID and username are recorded in the network endpoint and the process execution audit records. In particular, only PID 24210 has connected to www.suspicious.org, which was started by a shell process (PPID 24208) of user 'toma'. The wealth and replication of vital information in various types of audit records is a requirement for proper event correlation and step instance selection.

Another important issue of audit record engines is that of referencing time. In large IT infrastructures that span several networks and time zones, audited systems might report in different time formats. They can also experience 'clock skew', a difference in time recorded amongst computer systems due to computer clock hardware inaccuracies, especially when an NTP (Mills et al, 2010) server is not available to provide a reliable time source. Clock skew is common amongst mobile components of the IT infrastructure, as well as amongst operating systems that run in virtual mode (Matthews et al, 2008). An audit record engine should resolve that problem and make sure that every record is entered into the log set by having a correct time stamp.

Finally, audit record engines should provide a scalable storage system to keep a large number of audit records available for future reference. Modern IT environments that consist of a large number of multi-user serving devices of different kinds can easily produce a large amount of data. If the stored information is consolidated to a single place, a natural choice for data availability and correlation, the amount of data can quickly overwhelm traditional file based storage approaches.

One of the earliest and most commonly referenced works that concern the format of audit records is the US Government's Trusted Computer System Evaluation Criteria (TCSEC – 'Orange Book') (DOD 5200.28-std). This was a structured evaluation process (Trust Levels) that specified the features and assurances required for operating systems and application software to contain and process classified information. As part of these assurance features, the 'Orange Book' specified extensive lists of events that audit systems should monitor. However, these lists were provided without guidelines for selection, so that an analyst could abstract what is being monitored and choose a set of them. Moreover, the 'Orange Book' audit requirements did not provide any specification for the structure and storage of audit records.

These omissions, as well as the age of the drawn requirements led to the cancellation of the Orange Book by the US Department of Defense. The work is now purely a historic evidence of the need to draw audit requirements for operating systems. Instead, the Common Criteria for Information Technology Security Evaluation (Common Criteria Portal, 2009) standards have taken over the Orange Book's role. The Common Criteria (CC) effort does not fully address the previously mentioned audit record requirement omissions of its predecessor, the Orange Book. Despite enjoying an impressive industry product certification scheme and some criticism over the feasibility of implementing the listed requirements due to complexity (Jackson, 2007), the CC effort has still to produce a comprehensive array of audit requirements. In comparison to the Orange Book, the CC provide a more structured audit functional requirement list, but still, no substantial discussions with regards to the content, format and storage of audit records.

However, we do take note of some of the high level functional audit requirements of their 321 page document (Common Criteria Portal, 2009). In particular, CC requirement 88 of section 8.2 states that: “At FAU_GEN.2 User identity association, the TSF shall associate auditable events to individual user identities.” In CC terminology TSF stands for Target of evaluation Security Functionality, meaning essentially the software and hardware under evaluation.

The CC effort also states the minimum requirements for the content of an audit record by stating in requirement FAU_GEN.1.2: “The TSF shall record within each audit record at least the following information: a)Date and time of the event, type of event, subject identity (if applicable), and the outcome (success or failure) of the event; ...”. This is in-line with the previously discussed issues about user accountability and temporal information. The outcome of the event might be a tricky to implement, depending on the context of the event. For some types of events that are atomic (i.e., an attempt to execute a file), logging success or failure is meaningful (i.e. to log that an attempt was made to execute a file might be an interesting fact) and feasible (this can be easily performed by monitoring for exit codes or testing for the execution of the program by using the userid credentials) . For other types of events that are more complex and concern many intermediate steps (binary program that performs many actions that do not always follow the same order/execution path) this is less trivial to implement, as it requires tapping to the actual system calls level or other proprietary application logs.

The second CC requirement that concerns audit record storage is that of FAU_STG (section 8.6) (Common Criteria Portal, 2009). Actually, this is a set of requirements that concern various aspects of the audit record storage. We quote from the requirements text:“At FAU_STG.1 Protected audit trail storage, requirements are placed on the audit trail. It will be protected from unauthorised deletion and/or modification. FAU_STG.2 Guarantees of audit data availability, specifies the guarantees that the TSF maintains over the audit data given the occurrence of an undesired condition. FAU_STG.3 Action in case of possible audit data loss, specifies actions to be taken if a threshold on the audit trail is exceeded. FAU_STG.4 Prevention of audit data loss, specifies actions in case the audit trail is full. “. Once again, the requirements are given in high-level terms, specifying that:

- unauthorized deletion and/or modification of audit records
- any other condition that could cause storage failure.

should be mitigated.

This section concludes the list of 'must have' properties of an audit engine for insider misuse auditing. The next section discusses how most of today's audit engines fair against these requirements.

4. Existing audit record engines

Audit record engines have been around for a long time, since the very early days of operating systems. The following paragraphs will review a number of existing audit record engine specifications and solutions. The goal is to show that they do not fit all the requirements of misuse detection engines, as discussed in the previous paragraphs, and hence justify why LUARM was built from scratch, as an audit engine solution for insider threat specification.

The point behind the Orange Book (DOD 5200.28-std) and CC (Common Criteria Portal, 2009) is that they are both specifications and not implementations of audit record engines. However, several audit record engine implementations can be found in the literature.

The most common variety of audit record engines uses information that comes directly from the Operating System. Characteristic examples of this category of engines are Oracle's Basic Security Module (BSM) auditing system (Oracle Corporation, 2010) and its open source implementation OpenBSM (Trusted BSD Project portal, 2009), the psacct audit package (psacct utilities, 2003), as well as the syslogd (Gerhards, 2009) and WinSyslogd (Monitorware's website, 2010) applications (the latter runs on Windows operating systems).

The BSM audit system has seen widespread deployment in commercial server grade operating systems. It structures its audit records in binary (non human plain text readable) files. Audit trail management commands are then used to decode the binary form of these files and produce human readable output.

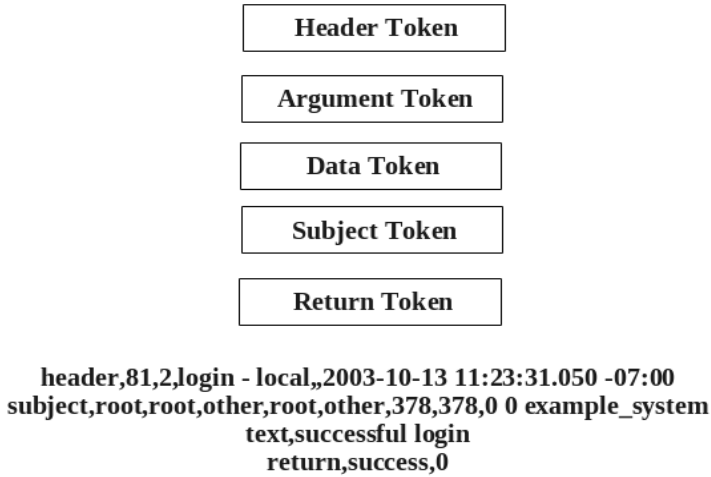

Figure 3: The BSM audit record format

Each BSM audit record is a series of byte encoded tokens. Figure 3 shows a typical structure of a BSM audit record and a corresponding decoded plain text example of an audited successful login entry. Actual audit records might vary in terms of the type and order of tokens. The Header token marks the start of the audit record. Argument and Data tokens normally contain data about the command and the arguments that caused an event. The Subject token states which process triggered the generation of the audit record. Finally, the Return token contains values that are returned by the process execution and can help the audit reviewer determine the success or failure of a command (the reader might recall CC (Common Criteria Portal, 2009) requirement FAU_GEN.1.2). The OpenBSM initiative (Trusted BSD Project portal, 2009) has similar audit record structure with minor differences in the encoding of the different token types.

Pssact (psacct utilities, 2003) is another audit system that generates operating system based audit trails. Although psacct can be used for security purposes (system administrators can check login attempts and user activity per user), its facilities are oriented towards resource usage

accounting. Thus, a system administrator can employ psacct to produce nice reports about the number of CPU hours spent per command or user. Figure 4 shows sample psacct output from the Linux operating system.

```
#Displays resource statistics per executed shell command
#    * sum of system and user time in cpu second
#    * "real time" in wall clock minutes
#    * sum of system and user time in cpu minutes
#    * cpu-time averaged core usage, in 1k units
#    * command name
1034   17545.52re      0.25cp    7754k
16       0.25re            0.21cp    68656k  yum-updatesd-he
23    5055.87re      0.01cp    6667k  ***other*
8    3099.25re      0.01cp    16921k  sshd*

# sa -m
#Displays resource statictics per system user
root                 987    8263.81re      0.23cp    7467k
toma                  45    9289.05re      0.01cp    11965k
nickb                 18       0.03re      0.00cp    15088k
```

Figure 4: psacct statistics

```
Oct  6 14:27:50 cn1 yum[16568]: Updated: totem-pl-parser-2.30.3-1.fc13.x86_64
Oct  6 14:27:51 cn1 yum[16568]: Updated: gnome-settings-daemon-2.30.1-8.fc13.x86_64
Oct  6 14:27:52 cn1 yum[16568]: Updated: shared-desktop-ontologies-0.5-1.fc13.noarch
Oct  6 14:28:15 cn1 yum[16568]: Updated:  selinux-policy-3.7.19-62.fc13.noarch
```

Figure 5: Syslog based audit record aggregation on a plain text file

Syslogd (Gerhards, 2009) and Winsyslog (Monitorware's website, 2010) are examples of widely employed Security Event Manager (SEM) applications. An SEM aggregates various types of audit records into a single interface. Audit record aggregation means that information is accepted not only from operating system audit trails, but also from third part sources such as security tools or even software application logs. The interface could be as simple as a human readable text file (Figure 5) or it can have its own sophisticated Graphical User Interface (GUI), as shown in Figure 6.

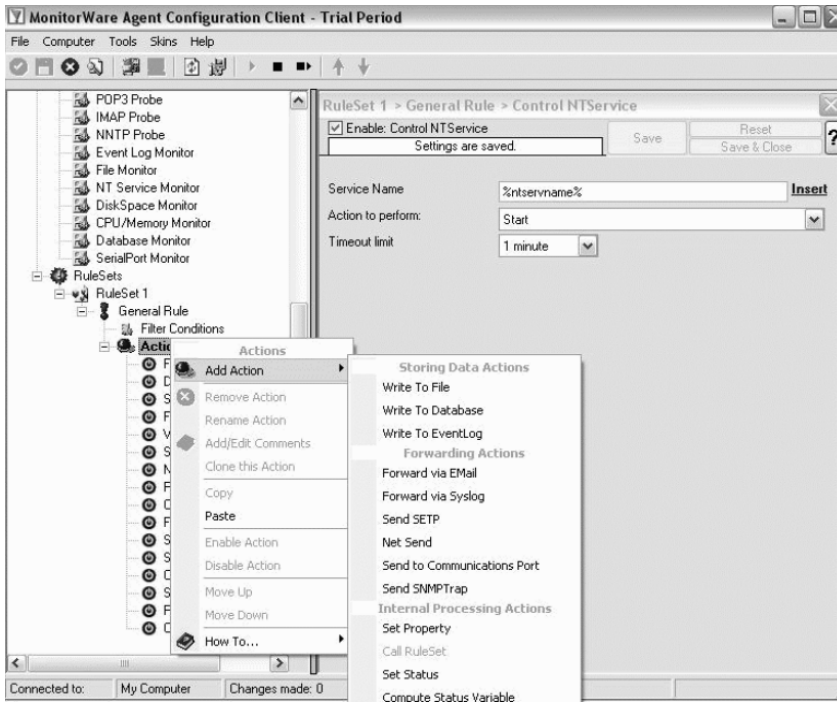

Figure 6: Winsyslog Graphical Console

One obvious thing to observe from all the previously described audit paradigms is that there is not a consistent audit record format amongst these log engines. This format diversity might suit specific operating system environments but it creates many problems, especially when one needs to devise a mechanism to consolidate logs from different operating systems and resources. Bishop (Bishop, 1995) was one of the first to discuss these issues in the context of distributed audit record engines and to propose various solutions for standardized audit record formats.

Figure 7 shows a sample of this proposed standard audit record format (Bishop, 1995) , together with special purpose plain text ASCII based (ANSI, 1986) field separators. The purpose of these field separators is to maintain compatibility amongst the character encoding sets of different operating systems. Whilst many characters encoding

differences have now been addressed by the Unicode standard (Unicode, 2006), Bishop's work is an interesting reminder of the need for a standard audit record format.

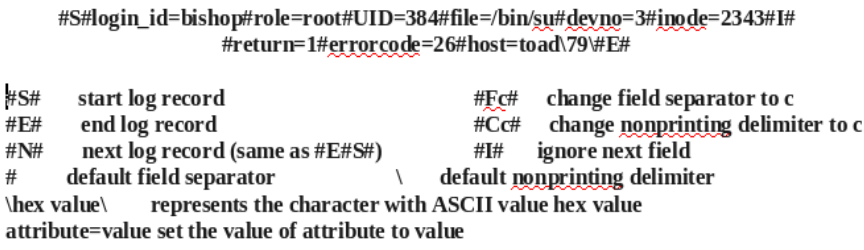

Figure 7: Standard audit record format by Bishop

Looking back at the previously discussed audit system approaches, serious deficiencies can be located in terms of using them for insider threat prediction. Firstly, we have issues that concern the bridging of the format variability (structure and content) across various operating systems. Modern SEMs might consolidate information from various different devices and operating system vendors, but they are far from describing sufficiently issues in an operating system agnostic way.

In addition, process accounting tools might not cover sufficiently the variety of different system level information (file, process execution and network level). In fact, some of them might miss data as described in (HP Portal, 2003). A logging engine that cannot facilitate the description of both static and live forensic insider misuse system data at the network, process and filesystem layer could hinder a forensic examination of an IT misuse incident. Static digital forensic analysis is employed by most forensic tools and reveals an incomplete picture of the system in question. It cannot portray accurately the non-quiescent (dynamic) state of the system under investigation.

Information such as active network endpoints, running processes, user interaction data (number of open applications per user, exact commands), as well as the content of memory resident processes may not be recorded accurately on non-volatile media. (Hay et al, 2009) discuss the shortcomings of static digital forensics analysis in detail. In order to overcome the barriers of static analysis, Adelstein (Adelstein et al, 2006) discusses the virtues of non-quiescent or live analysis, which essentially gathers data while the system under-investigation is operational. A proper IT misuse logging engine has to offer a combination of static and dynamic data in it logs.

Several audit record systems do not report consistently the timing of audit record generation. For instance, many implementations of the syslog audit standard and psacct tools generate the audit record by entering the time stamp of the client system. If the client system does not have a reliable time source, this generates inaccurate information and could seriously hinder event correlation.

An additional issue some audit record engines might not meet the scalability and data integrity requirements set by CC requirement FAU_STG.1 (Common Criteria Portal, 2009). Syslog will not always consolidate data in a central location away from the audited client, as its default configuration leaves the data on the monitored host. The same can be said for the BSM standard (Trusted BSD Project portal, 2009), leaving the integrity of audit data at risk. In addition, storing data in binary or text files might raise issues of storage efficiency and scalability.

Finally, one of the most serious drawbacks of existing audit approaches is the inability to store the audit information in a form that can utilize relational queries. Section 3 discussed the reasoning behind this requirement. In one sense, some people might argue that this is an audit management feature rather than an audit log design issue. However, as section 3 discussed the advantages of using a relational schema to form audit queries in a structured log record, the author's view is that everything that increases the expressive power of an audit

log query should be incorporated in the structure of the audit log, rather than being left as an 'add-on' feature.

For all these reasons, LUARM was designed and built a prototype audit record engine for insider IT misuse from scratch. The next section presents the architecture of the proposed audit engine.

5 The LUARM audit engine

LUARM is a prototype Open Source audit record engine (LUARM portal, 2010) that uses a Relational Database Management System (RDBMS) (Connoly T. et al, 2004) for the storage and organization of audit record data.

The employment of an RDBMS system is a core design choice for the LUARM engine. It offers the necessary data availability, integrity and scalability features, because most RDBMS tools are explicitly designed to organize and store large amounts of data. However, the main reason of placing an RDBMS engine at the core of LUARM is the ability to have a tremendous flexibility in the process of querying audit records in a standard manner. The Structured Query Language -SQL (ISO/IEC, 2008) is a declarative computer language used to query and process the data stored by RDBMS systems, adhering to the relational model. In particular, features such as the disjunction, conjunction and negation operators are part of the language. SQL calls these predicates and it used them to specify conditions in an accurate manner. Boolean (true/false/unknown) truth values are used to limit the effects of statements and queries. In addition, step instance selection and completion, as well as data correlation can be performed by using SQL clauses such as 'FROM' and 'WHERE'. Latter paragraphs will provide LUARM examples using standard SQL queries.

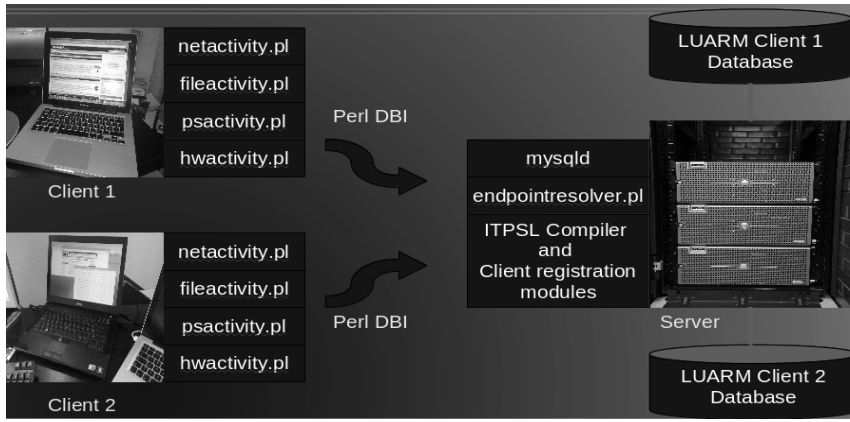

Figure 8: The LUARM architecture

Figure 8 displays the module client-server architecture of the LUARM audit engine. On the left of the figure, we can see a set of audited computer clients. Every client is running a unique instance of a set of monitoring scripts. Each of the client scripts audits a particular system level aspect of the operating system: 'netactivity.pl' audits the addition and creation of endpoints, 'fileactivity.pl' records various file operations, 'psactivity' provides process execution audit records and 'hwactivity.pl' keeps a log of hardware devices that are connected or disconnected from the system. The right hand side contains the centralized server part of the architecture where audit data are stored, maintained and queried in a MySQL (Oracle MySQL portal, 2010) based RDBMS (other RDBMS systems could be used as well). The Perl programming language is used to implement the modules and the communication between client and server is performed via a Perl DBI (CPAN-DBI, 2010) interface.

The client-server architecture avoids leaving the data in vulnerable clients. To prevent issues that affect the scalability of operations and satisfy data access control isolation (addressing the CC requirements

FAU_SAR.1 and FAU_STG.1). The central host MySQL server has its own authentication system responsible for controlling who has access to the audit data. By authenticating audit reviewers against the RDBMS authentication system, we de-couple the users being audited from the auditors, a desirable property that ensures that audited insiders cannot easily manipulate audit data. Furthermore, by assigning a separate database instance per audited client, we reduce the likelihood of compromising the data for all clients. If the database access credentials of one client are compromised, the damage is limited to the audit data for that client only. Standard RDBMS mirroring procedures can also ensure data availability on the server side.

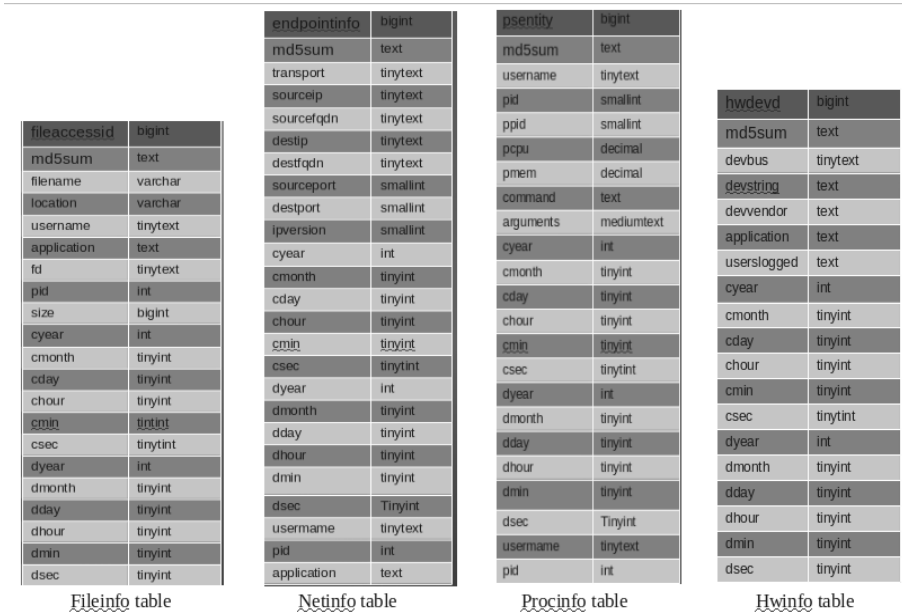

Figure 9: LUARM relational table structure

Figure 9 displays the relational table format for the four main types of recorded audit data in LUARM: fileaccess, process execution, network endpoint and hardware device information. Temporal information is provided by event creation time stamps (cyear, cmonth,

cday,chour,cmin,csec) and respective event destruction time stamps (dyear,dmonth,dday,dhour,dmin,dsec). The combination of the two types of timestamps can pinpoint exact time intervals for events in a consistent format for all recorded event types. In contrast, most audit systems may provide only event creation time references without hinting for the duration of an event.

The sampling of events is done at 100 ms intervals. This was an intentional decision. At first, this might seem problematic as many attack steps can occur much faster than that amount of time. However, in an event sampling loop, one has to account for the time delay to update the database, which can vary from 10ms to 60-70 ms intervals on heavily loaded clients and servers. In addition, time resolution varies amongst operating systems. In Linux, the finest granularity of timing for most computing devices is measured at approximately 10 ms (Love, 2005). The Windows 7 operating system (and its various derivatives) has a timer granularity of 15.6 ms (Microsoft Portal, 2009). For these reasons, LUARM relies on the Perl Time::HiRes module (CPAN-HiRes, 2010) to bridge the gap between the different operating system timer implementations. A time granularity of 100 ms is also a good compromise between accuracy and scalability. The more granular the time resolution, the greater the computational load for both the client and the server LUARM parts.

Each audit record of an event table is identified by a unique table key of bigint MySQL type. In version 5.1 of the MySQL RDBMS, a 'bigint' numeric type can create up to 18446744073709551615 unique keys, a number large enough to archive a useful number of events in each LUARM event table.

Another important design decision that concerns the format of the audit table was to include common attributes amongst different event tables for the purposes of increasing the ability to correlate events and provide user entity accountability, as mentioned by CC requirements FAU_GEN1.2 and FAU_SEL1.1. For instance, fields such as 'username' (user entity), pid (numeric process ID of the program responsible for the event creation) and application (string that represents the name of the application that matches the pid) can be

found in most of the event tables. This enables the audit reviewer to use SQL and relate events, so he can form queries of the type “Find the network endpoint created by program x of user y” in an easy manner.

The use of the MD5 cryptographic hash function (Rivest et al, 1992) (md5sum field) is used on all event tables for performing audit record updates in an efficient manner. In particular, every time LUARM inserts an audit record in a table, it calculates an MD5 sum of several relevant table fields, in order to uniquely identify the event and keep track of the record being inserted in the database. On the next audit record insertion cycle, LUARM generates an MD5 sum of the live records and compares them to the stored MD5 sums of every active stored record (a record that has a NULL value for the d* timestamps). If the MD5 sums do not match the record is inferred as a new one and is inserted to the database. This is a more efficient way than comparing multiple fields, in order to perform record updates.

The 'fileinfo' table stores file access related events. The filename specification consists of two parts. The 'filename' field which holds the filename with the file extension (i.e. data.txt) and the 'location' field which contains the absolute path of the file. The fact that the two are divided in separate fields makes it easier to search by location or by field name only, increasing the versatility of mining file data. In order to populate the data on this table, LUARM relies on the 'lsof' utility (Pogue et al, 2008). The utility is versatile and can record a variety of events including file and network endpoints in real time. It exists for an entire range of UNIX/Linux and MACOSX operating systems, covering a large spectrum of computing devices.

The 'netinfo' table logs the creation and destruction of network endpoints. In the context of LUARM, the term 'network endpoint' refers to the operating system data structures employed to facilitate network connectivity via the TCP/IP protocol suite (Socolofsky, 1991). Network endpoint activity is considered as live forensic data. A series of table fields are used to record endpoint details ('sourceip',

'destip', 'sourceport' , 'destport' and 'transport' record source and destination IP addresses, source and destination port and transport protocol respectively). The fields 'sourcefqdn' and 'destfqdn' hold the DNS (Mockapetris, 1987) resolved Fully Qualified Domain Name (FQDN) for the source and destination hosts.

A small LUARM implementation detail concerning the 'sourcefqdn' and 'destfqdn' fields is that they are not populated by the client LUARM routines. In contrast, they are populated on the LUARM server side. Due to the criticality of correct DNS data for the audit records, the frequent DNS configuration errors (Barr, 1996), aspects of DNS operational security (Bauer, 2003) and client performance, the endpoint name resolution is left on the server side. This provides a greater control on DNS derived data and does not rely on vulnerable clients (malicious insiders or software vulnerabilities) for auditing network connections.

Process execution activity is recorded in the 'psinfo' table (Figure 9). This table records 'live' forensic data. The table includes both the proces ID ('pid') and parent process id ('ppid'), so that process execution flow can be traced back to the original process. In order to speed up process execution searches, the LUARM engine also separates the executed command ('command') from its arguments ('arguments'). One might like to search them separately in the process of mining process execution data. The 'pcpu' and 'pmem' fields address process over-utilization issue. 'pcpu' contains the CPU time used divided by the time the process has been running (cputime/realtime ratio), expressed as a percentage. 'pmem' is the ratio of the process's resident set size to the physical memory on the machine, expressed as a percentage. The 'ps' UNIX/Linux utility (Pogue et al, 2008) is used to collect process information. For all active processes (whose d* temporal fields are NULL), LUARM updates in near real time these two fields.

The 'hwinfo' table logs 'live' device connection and disconnection events. All events generated by devices that connect to the Peripheral

Component Interconnect (PCI and PCI-Express) and Universal Serial (USB) buses (Mueller, 2006). These two buses are commonly found on a large array of computing devices, interconnecting various peripherals such as portable storage media, as well as sound and video interfaces amongst others. For instance, an audit reviewer or forensics analyst might be interest to correlate file activity to a portable storage medium connection, as part of an intellectual property theft scenario. In that case, the 'hwinfo' table logs information in various fields that help identify the attached device ('devstring', 'devvendor'), the bus the device was connected to ('bus') and correlate the device attachment event against a number of users that are logged into the system at the time of the device attachment ('userslogged').

LUARM contains a small number of additional tables for house-keeping functions that ar beyond the scope of this paper. The next section demonstrates LUARM usage.

6 LUARM in action

Having a proposed structure and content for the various categories of audit events as described in the previous section, we can now issue sample SQL statements to illustrate how audit data mining is performed. Figure 10 displays sample queries that demonstrate the expressiveness of LUARM's audit record content and structure.

There are a few important observations to make about the example LUARM SQL queries. The first one concerns the embedding of system specific knowledge inside the statement. In essence, the third example of Figure 10 defines a step of an insider trying to transfer a sensitive file to a portable medium. One has to know the name of the sensitive file 'prototype.ppt' and also the fact that '/media' is used as a mount point for portable media for that host. Additional possible destination locations could be specified by means of OR operators. The use of the 'RLIKE' operator (RLIKE RegExp, 2008), always in

relation to the second and third examples of Figure 10. The operator implements a regular expression (Friedl, 2002) type of match. Apart from the conjunction operator (OR), regular expressions give the specification polymorphic properties (one specification string, many matching results), a desirable property for compact misuse detection language statements.

Find all accesses of the file 'prototype.ppt' by users 'toms' OR 'georgem' between 9:00 and 14:00 hours on 23/10/2009.

```
SELECT * FROM fileinfo WHERE filename='prototype.ppt' AND ((username='toms') OR
(username='georgem')) AND cyear='2009' AND cmonth='10' AND cday='23' AND chour >= '9' AND
chour <= '13' AND cmin >= '0' AND cmin >= '59';
```

Find all USB devices that were physically connected to the system when users 'toms' OR 'georgem' were logged on 23/10/2009.

```
SELECT * from hwinfo WHERE devbus='usb' AND ((userslogged RLIKE 'toms') OR (userslogged
RLIKE 'georgem')) AND cyear='2009' AND cmonth='10' AND cday='23' AND chour >= '9' AND chour
<= '13' AND cmin >= '0' AND cmin >= '59';
```

Find whether users 'georgem' or 'toma' have tried to move or copy a file called 'prototype.ppt' (irrespective of location) under the directory '/media' between 9:00 and 13:00 hours on the 23rd of October 2009.

```
select * FROM psinfo WHERE ((command='cp') OR (command='mv')) AND (arguments RLIKE
'prototype.ppt' AND arguments RLIKE '/media') AND ((username='georgem') OR (username='toms'))
AND cyear='2009' AND cmonth='10' AND cday='23' AND chour >= '9' AND chour <= '13' AND cmin >=
'0' AND cmin >= '59';
```

Figure 10: Using SQL to mine data in LUARM

We have tested LUARM on a variety of simulated insider misuse scenarios. The scenarios were derived by real world LUARM captured data. However, permission to publish the original audit data was not obtained by the organizations in question. Thus, we had to reconstruct the misuse incidents by means of writing down a text based description of each incident and ask a team of users to re-enact it under a controlled IT infrastructure. The following paragraphs will present one of these incidents and demonstrate how the correlational versatility of the LUARM relational audit log structure can shed forensic light into the actions of a malicious insider. The scenario is provided below:

'Autobrake' Corp is a company designing car braking systems. Their engineering department is the most information sensitive work area. The braking system design process takes place, in high performance Linux workstations, one for each design engineer. The engineers have normal user rights to the workstations. Superuser rights (root) is given only to the IT admin. The designs reside on the local hard drives of the workstations and the company's IT policy forbids any transfer of

sensitive data to portable media. Autobrake's system administrator has requested a salary raise various times. This has been denied by management, as the company faces a declining car manufacturing market. The system administrator is lured by a competing company that asked him to deliver schematics of the new and revolutionary Autobrake's RGX9 SUV braking system in return for a large amount of money. Enjoying the trust of everyone and having full control of the engineering CAD workstations, the system administrator decides to take the offer of the competing company. He performs the intellectual property theft by following a well designed approach which is summarized below:

He carefully chooses the user account of a mechanical engineer (username 'engineer3') that had some disputes over work issues with management. He aims to avoid detection by means of masquerading as the engineer in question.

After successfully masquerading as the engineer in the IT system he uses a portable USB key to obtain the commercially sensitive RGX9 schematic, leaving only the traces of the engineer “actions”.

Assuming that a third party auditor manages the audit process and monitors the logging (ensuring that the logging infrastructure works) and that all Engineering workstations are monitored by LUARM, we are now tasked to find the offender and clear the name of 'engineer3'. The reader should consult the LUARM relational table structure (Figure 9), in order to follow the SQL queries presented below.

A post-mortem forensic examination of an incident is a tedious process. Due to the lack of space, we present here the basic reasoning. We begin our investigation from the most important file, that of RGX9, and the people that work on it. From the audit record of the workstations with name 'proteas', we utilize LUARM to find out who has been using the file:

mysql> select username,pid,cday,chour,cmin,location,filename from fileinfo where filename RLIKE 'RGX9' OR location RLIKE 'RGX9' \G

From the many hits we get from the data base, we focus our attention on the following ones:

**************************** 111. row ****************************

username: engineer3

pid : 8301

cday: 4

chour: 15

cmin: 30

location: /storage/users/engineer3/work/designs

filename:RGX9.jpg

...

**************************** 118. row ****************************

username: engineer3

pid: 28538

cday: 4

chour: 15

cmin: 32

location: /media/U3SAN03-12

filename: RGX9.jpg

The reason these file access patterns looked suspicious is that they were different than the normal pattern of accessing the file by the staff engineer. Normally, user 'engineer3' would access the file by means of certain design and image editing applications, under its usual directory (/storage/users/engineer3/work/designs). This time, however, things look a bit different, if one follows the association of file access to process execution, in order to confirm which programs performed the file transaction. The following SQL queries achieve the desired association:

mysql>select username,pid,command,arguments,cyear,cday,chour,cmin from psinfo where username='engineer3' AND pid='8031' AND cyear='2011' AND cday='4' AND chour='15' AND cmin='30;

**************************** 1. row ****************************

username: engineer3

pid: 8031

command: /bin/cp

arguments: work/designs/RGX9.jpg /tmp/

cyear: 2011

cday: 4

chour: 15

cmin: 30

mysql>select username,pid,command,arguments,cyear,cday,chour,cmin from psinfo where username='engineer3' AND pid='8031' AND cyear='2011' AND cday='4' AND chour='15' AND cmin='30;

**************************** 1. row ****************************

username: root

pid: 28538

command: mv

arguments: RGX9.jpg /media/U3SAN03-12

cyear: 2011

cday: 4

chour: 15

cmin: 32

Essentially, the previous results verify that the file was first copied from the normal directory to /tmp and then was moved to the /mnt/usb. At this point, a little bit of system specific knowledge comes into light, as /mnt/usb is the usual mount point where Linux links portable storage media to the filesystem. Hence, the question to raise is whether a portal storage medium was connected to the workstation, prior to the 'mv' file transaction. The query result yields a positive answer:

mysql> select * from hwinfo where cyear='2011' AND cmonth='01' AND cday='04' AND chour='15'\G

**************************** 1. row ****************************

hwdevid: 71

md5sum: a16e7386f14de769a7a9491da2071f5b

cyear: 2010

cmonth: 12

cday: 4

chour: 15

cmin: 30

csec: 28

devbus: USB

devstring: Cruzer Micro U3

devvendor: SanDisk Corp.

userslogged: engineer3,root

dyear: 2010

dmonth: 1

dday: 4

dhour: 15

dmin: 33

dsec: 38

This database hit seems to be in line with the actions of engineer3, as it indicates a device connection before the execution of the 'mv' command and a disconnection well after the mv command.

Thus, everything seems to point out that 'engineer3' violated the company policy and transferred a sensitive file to a USB medium, against the company IT regulations. However, this had been categorically denied by the actual person. A good but non IT based alibi for the staff engineer was that he exited the building with his security card token around 14:50, returning back to his desk at 15:50, a wide gap for him. Clearly, something else was going on and the clue was the 'userslogged' field of the last LUARM result. This 'hwinfo' LUARM table field contains the usernames for accounts that are

logged into the workstation at the time of the device connection. Apart from 'engineer3' we note the root account being active, which is clearly the only other choice that, under the circumstances, could have performed the mount procedure.

Based on the time stamp of the mv operation, a careful investigation of the root account actions reveals a key command execution, derived from the 'psinfo' table:

mysql> select * from psinfo where pid='27865' AND cyear='2011' AND cday='4' AND cmonth='1' AND chour='15' AND cmin >= '20' AND cmin <='33' \G

**************************** 1. row ****************************

psentity: 97654

md5sum: 7067284f2e1aefc430339ef091b4e41b

username: root

pid: 27865

ppid: 26407

pcpu: 0.0

pmem: 0.0

command: su

arguments: - engineer3

cyear: 2011

cmonth: 1

cday: 4

cmin: 28

chour: 15

csec: 36

dyear: 2011

dmonth: 1

dday: 4

dhour: 15

dmin: 28

dsec: 39

The 'su' command is used routinely by administrators to switch user credentials, in order to test environment settings and perform system tasks (Garfinkel et al, 1996). However, it can be easily used as a masquerading tool to covertly perform actions using the credentials of somebody else.

A further investigation also found the USB key on the desk of the IT administrator with the RGX9.jpg file. The hwinfo table device identifier data ('devstring', 'devvendor') as well as the mount point identifier (/media/U3SAN03-12) from the psinfo commands contributed towards strengthening the final piece of the puzzle.

This case shows the versatility of the relational structure of the LUARM record that showed the way from simple file operation to related program execution and other events that can provide strong evidence and lead to the misuser. In addition, LUARM has also been used successfully to provide evidence about security incidents of external origin (Magklaras, 2011). Thus, it offers a valuable complement of existing logging mechanisms.

The only serious drawback of the prototype we encountered was the fact that the prototype sampling interval of 100 ms is to slow and actually misses executed processes. This occurred during our controlled experiment and during the real incident recording phases. Most commands that are not typed interactively (in batch mode) could actually be a very important way to bypass the logging of execution events, as they can occur in sub 100 ms cycles. Nevertheless, most of the important evidence was logged in the majority of the cases.

The LUARM audit engine prototype is currently under constant development as an open source audit engine, in order to improve its operational security and audit speed performance. The authors welcome feedback and participation to the development of its code base. The prototype is not yet ready for production deployment, but it

should be suitable for experimentation and has already proved its value on a number of insider IT misuse incidents.

7 Conclusions

The insider IT misuse problem is a real substantial threat to the health of IT infrastructures. A very important tool to mitigate this type of threat is an audit record which is specifically designed to address the various needs of insider IT misuse detection, as well as complement existing forensic tools when security specialists perform a post-mortem incident examination. These specifications are not met by traditional audit engines and dictate a detailed log of user actions at file, process execution and network endpoint level stored in a Relational Database Management System.

The file, process and network endpoint data provide a dynamic forensic view of the system, a useful complement to existing forensic tools that offer only static data in their majority. The relational storage layer increases the correlational versatility amongst the different types of audit data, as it is vital to be able to perform various associations during the investigation of an incident (process to file, process to network activity) and reliably relate actions to user entities. The results are promising, showing a much better way to examine a system than looking at static text files which are difficult to parse and even more difficult to correlate.

8. References

Adelstein F. (2006), "*Live Forensics: Diagnosing Your System without Killing it First*", Comm. ACM, vol.49, no.2, 2006, pp. 63-66.

ANSI (1986), "*American National Standard for Information System: Coded Character Sets — 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)*", ANSI X3.4-1986, American National Standards Institute, Inc., March 26, 1986.

Bace R. (2000), "*Intrusion Detection*", Macmillan Technical Publishing, Indianapolis, USA, ISBN: 1-578701856, pp. 38-39 discuss the terms 'misuse

detection' and 'anomaly detection' in an intrusion specification context, pp. 47-66 discuss various audit record issues.

Barr D. (1996), "*Common DNS Operational and Configuration Errors*", Internet Engineering Task Force (IETF) Request For Comment (RFC) 1537, February 1996.

Bauer M. (2003), "*Building secure servers with Linux*", O'Reilly & Associates, ISBN: 0-596-00217-3: Chapter 6, pages 154-196.

Bishop M. (1995), "*A Standard Audit Trail Format (1995)*", In Proceedings of the 1995 National Information Systems Security Conference, pp. 136-145.

Cha A.E. (2008), "*Even spies embrace China's free market.*", Washington Post, February15,2008:,www.washingtonpost.com/wpdyn/content/article/2008/02/14/AR2008021403550.html (Accessed 03/03/2011)

Codd E. F. (1990), "*The Relational Model for Database Management*", Addison-Wesley Publishing Company, 1990, ISBN 0-201-14192-2.

Common Criterial Portal (2009), "*The Common Criteria for Information Technology Security Evaluation*", Version 3.1, Revision 3, July 2009. Part 2: Functional security components,www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R3.pdf (Accessed 03/03/2011)

Connoly T., Begg C. (2004), "*Database Systems: A Practical Approach to Design, Implementation and Management*", Fourth Edition, International Series in Computer Science, Addison-Wesley, ISBN-13: 978-0321210258.

CPAN-DBI (2010), "*The Perl Database Interface (DBI) module*" at the Comprehensive Perl Archive Network (CPAN),search.cpan.org/~timb/DBI-1.615/DBI.pm (Accessed 03/03/2011)

CPAN-HiRes (2010), "*The Perl High Resolution Timer module*" at the Comprehensive Perl Archive Network (CPAN),search.cpan.org/~jhi/Time-HiRes-1.9721/HiRes.pm (Accessed 03/03/2011)

DOD 5200.28-std (1985), "*Department of Defense Trusted Computer System Evaluation Criteria*", National Computer Security Center: Orange Book, DOD 5200.28-std, December 1985.

Friedl J. (2002), "*Mastering Regular Expressions*", O'Reilly, ISBN 0-596-00289-0.

Furnell S. (2004), "*Enemies within: the problem of insider attacks*", Computer Fraud and Security, Volume 2004 Issue 7, pp. 6-11.

Garfinkel S, Spafford G. (1996), "*Practical UNIX and Internet Security*", Second Edition, O'Reilly and Associates, Sebastopol, CA, ISBN: 1-56592-148-1

Gerhards R. (2009), "*The Syslog Protocol*", Internet Engineering Task Force (IETF), Request for Comment (RFC) 5424, March 2009.

Greenemeier L., Choi C. (2010), "*WikiLeaks Breach Highlights Insider Security Threat*",Scientific American,www.scientificamerican.com/article.cfm?id=wikileaks-insider-threat (Accessed 03/03/2011)

Hay B., Nance K., Bishop M. (2009), "*Live Analysis Progress and Challenges*", IEEE Security & Privacy, Volume 7, Number 2, pp. 30-37.

HP Portal (2003), "*psacct process accounting misses some commands*", HP IT ,forums11.itrc.hp.com/service/forums/questionanswer.doadmit=109447626+1286381845785+28353475&threadId=1413576 (Accessed 02/02/2011)

ISO/IEC (2008), Information Technology-- Database Languages – SQL: ISO/IEC 9075,www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45498 (Accessed 03/03/2011)

Jackson W. (2007), "*Under Attack: Common Criteria has loads of critics, but is it getting a bum rap*", Government Computer News, date: 10/08/2007, gcn.com/articles/2007/08/10/under-attack.aspx (Accessed 03/03/2011)

Love R. (2005), "*Linux Kernel Development*", Second Edition, Sams Publishing, ISBN: 0-672-32720-1: pp 98-107 discuss low-level details of the timer implementation.

LUARM portal (2010),luarm.sourceforge.net/ (Accessed 03/03/2011)

Magklaras G., Furnell S., Brooke P. (2006), "*Towards an Insider Threat Prediction Specification Language*", Information Management & Computer Security, (2006) vol. 14, no. 4, pp. 361-381.

Magklaras G., Furnell S. (2010), "*Insider Threat Specification as a Threat Mitigation Technique*", book chapter in "*Insider Threats in Cyber Security*", Advances in Information Security, Probst C.W., Hunker J., Gollman D., Bishop M. (Ed), Springer, ISBN: 978-1-4419-7132-6, pp. 219-244.

Magklaras G. (2011), "*Catching an undesired guest in the penguin /tmp room*", Epistolatory Blogspot, epistolatory.blogspot.com/2011/02/catching-undesired-guest-in-penguin-tmp.html (Accessed 03/03/2011)

Matthews J., Dow E., Deshane T., Wenjin H., Bongio J., Wilbur P. ,Johnson B. (2008), "*Running Xen, A Hands-On Guide to the Art of Virtualization*", Prentice Hall Pearson Education, ISBN-13: 978-0-13-234966-6: pp. 1-26 are a comprehensive introduction to the concept of virtualization.

Microsoft Portal (2009), "*Timers, Timer Resolution, and Development of Efficient Code*",download.microsoft.com/download/3/0/2/3027D574-C433-412A-A8B6-5E0A75D5B237/Timer-Resolution.docx (Accessed 03/03/2011)

Mills D., Delaware U., Martin J., Burbank J., Kasch W. (2010), "*Network Time Protocol Version 4: Protocol and Algorithms Specification*", Internet Engineering Task Force (IETF) Request For Comment (RFC) 5905, June 2010.

Mockapetris P. (1987), "*Domain Names – Implementation and Specification*", Internet Engineering Task Force (IETF) Request For Comment (RFC) 1035, November 1987.

Monitorware's website (2009), www.winsyslog.com/en/product/ (Accessed 03/03/2011)

Mueller S. (2006), "*Upgrading And Repairing PCs*", 17th Edition, Que Publishing, ISBN: 0-7897-3404-4: Chapter 4 describes the PCI and PCI-Express buses, pages 372-378. Chapter 15 describes the USB bus. Pages 980-989.

NSTISSAM (1999), "*The Insider Threat To US Government Information Systems*", U.S. National Security Telecommunications And Information Systems Security Committee, NSTISSAM INFOSEC /1-99.

Oracle Corporation (2010), "*System Administration Guide:Security Services*", Solaris 10 Operating System, Part No: 816–4557–19 , September 2010, pp. 559-672,dlc.sun.com/pdf/816-4557/816-4557.pdf, (Accessed 03/03/2011)

Oracle MySQL portal (2010), www.mysql.com (Accessed 03/03/2011)

Pfleeger C., Pfleeger S. (2003), "*Security in Computing*", Third Edition, Prentice Hall, ISBN:0130355488: Page 6 contains the definition of the term "threat" in an information security context.

Pogue C., Altheide C., Haverkos T. (2008), "*Unix and Linux Forensic Analysis DVD Toolkit*", Syngress, 2008, ISBN: 978-1-59749-269-0.

Potter C., Beard A. (2010), "*Information Security Breaches Survey 2010 technical report*",PriceWaterhouseCoopers/InfoSecurity,www.infosec.co.uk/files/isbs_2010_technical_report_single_pages.pdf (Accessed 03/03/2011)

Probst C., Hunker J., Bishop M., Gollman D. (2009), "*Countering Insider Threats*", ENISA Quarterly Review Vol. 5, No. 2, June 2009, pp. 13-14.

Psacct utilities (2003), Utilities for process activity monitoring, linux.maruhn.com/sec/psacct.html (Accessed 03/03/2011)

Richardson R. (2009), "*2009 CSI Computer Crime and Security Survey*", Comprehensive Edition, gocsi.com (Accessed 03/03/2011)

Rivest R. (1992), "*The MD5 Message-Digest algorithm*", Internet Engineering Task Force (IETF) Request For Comment (RFC) 1321, April 1992.

RLIKE RegExp (2008), "*String Regular Expression Operator*", MySQL 5.1 Manual, Oracle Corporation,dev.mysql.com/doc/refman/5.1/en/regexp.html (Accessed 03/03/2011)

Socolofsky T. (1991), "*A TCP/IP Tutorial*", Internet Engineering Task Force (IETF) Request For Comment (RFC) 1180 (Informational), January 1991.

Trusted BSD Project portal (2009), "*OpenBSM: Open Source Basic Security Module (BSM) Audit Implementation*",www.trustedbsd.org/openbsm.html (Accessed 03/03/2011)

Unicode (2006), "*The Unicode Standard*", Version 5.0, Fifth Edition, The Unicode Consortium, Addison-Wesley Professional, 27 October 2006. ISBN 0-321-48091-0.