CHALMERS

Application Audit Trail Analysis

Master of Science Thesis at Computer Science and Engineering

BJÖRN SCHUBERG

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, October 2010

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Application Audit Trail Analysis

BJÖRN SCHUBERG

© BJÖRN SCHUBERG, October 2010.

Examiner: ANDREI SABELFELD

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden October 2010

Preface

This thesis was carried out as an internship during six months at Amadeus in Sophia Antipolis, France. The ideas presented herein are primarily based on research found in papers that were published on-line. Amadeus on-site library was also source for information on associated topics. I would like to thank the team leader of the group that the work was carried out in, Sylvain Jacob in Form of Payment, for constructive criticism during the elaboration of this thesis. I would also like to thank Luc Capanaccia in the same team for support and always being available for discussion. A big thank you is also expressed to my examiner, Andrei Sabelfeld, for promptly answering any questions that I've had.

Abstract

An application audit trail may be analyzed for a number of reasons; performance enhancement of networks and computers, identification of security incidents and operational problems as well as fraud tracking. In this thesis, focus lies on detecting intrusions into information systems. An approach for analysis of an audit trail from an arbitrary application is presented. It's extensible and allows for further methods of analysis to be incorporated. For the purpose of this thesis, two methods were implemented and used in parallel to find intrusions. A technique for labeling signaled intrusions with an indication of their certainty is developed. This allows for focus on alarms with a higher certainty, and will thus facilitate a faster response. The analysis system was tested against several scenarios. All which contained intrusions were detected. Among these, all but one was deemed as an intrusion with highest certainty. Further, the amount of false positives when analyzing the test datasets were only fraction of a percentage.

Contents

Chapter 1

Introduction

As the number IT services grows and we get more and more dependent on IT infrastructure to facilitate our day-to-day tasks such as banking, telecommunication (VOIP[1]), online shopping and traveling, the security of these systems have become paramount. You don't have to wait many days between headlines that mention denial of service attacks, worm propagation or data breaches and identity theft — and it is only a fraction of the incidents that are disclosed[10]. A commonly cited case is the Heartland data breach[14], where around 100 million credit cards might have been stolen from the credit and debit card processing service provider. Their network had been breached and code that stole credit card information was planted in their IT landscape.

Compromised systems may have its resources exposed to different extents, depending on how the network infrastructure around it is designed and configured. In the worst case, a compromised host inside a corporate network may be able to access all resources within the company, including branches that are on the other side of the globe.

Different security controls can be set up to serve as defensive shells to fend off intruders. Examples of preventative online measures are firewalls, authentication schemes, cryptography and antivirus software. There are also hardening, passive techniques such as secure software engineering methods. Many vulnerabilities are introduced by programmers during development of software — the introduction of buffer overflow vulnerabilities is such a common pitfall among developers that a special security control, ASLR[2], has evolved to mitigate its ramifications. It is now part of any modern operating system.

The level of security in a computer system often comes down to a trade off between security and usability. Tightening security too much will degrade the level of service and make the system tedious to use. Even if one considers a system using a solid security architecture and software developed in a sensible way, the complexity of most software will make it virtually impossible to make it free from vulnerabilities. Therefor it is prudent to assume that certain attacks will find its way through all these preventative measures that can be put in place.

Intrusion detection methods aims to catch such attacks by observing events that occur in an information system. There are two major techniques within the field; signature detection (also referred to as misuse detection) and anomaly detection. The first involves recognition of events that are previously known to constitute an attack. Anomaly detection[6] on the other hand, determines what constitutes normal behavior for a user, network, process etc and flags any deviations from this behavior. The power of this technique is that it can catch previously unknown attacks.

This thesis presents the author's work on an approach for analysis of audit trail data from an arbitrary application. An extensible technique is presented, where a set of methods are used in parallel to determine if an intrusion has occurred. The next chapter covers the background behind this thesis together with goals and limitations. In chapter three, the problem at hand, and the developed approach to handle it is presented. This is followed by test results and an evaluation of the solution in chapter four and five.

[1] Voice Over IP
[2] Address Space Layout Randomization

Chapter 2

Background

Amadeus develops IT solutions for the global travel and tourism industry. The solutions enable travel providers, such as airlines, hotels, car rental companies and travel agencies to easily publish information about schedules, availability, pricing and ticketing of their worldwide services. Amadeus solutions also facilitate inventory management, reservations, itinerary planning, fare searching, departure control and passenger check-in.

In January 2010, Amadeus was PCI DSS certified. This is a standard developed by the credit card industry to facilitate an adoption of consistent data security measures for companies that handle credit card information. In order to remain compliant, a certified company must pass a reevaluation every year. This ensures that certified companies continuously maintain and review their solution that handles credit card data.

Whenever a credit card payment is processed through Amadeus on behalf of a provider, for instance an airline, the PAN[1] is stored on a distributed application server. All access to this application server is monitored and stored in an audit trail, as imposed by section 10 in the PCI DSS requirements[13].

2.1 Objective

This thesis aims to find way to analyze the audit trail in question. The analysis should incorporate a sound method for analysis. It should also use a general approach as well as being interoperable. By keeping the analysis general, it should be possible to reconfigure, or do minor modifications to the analysis system for analysis of a different application audit trail.

Based on the results from the analysis, alarms shall be raised if any suspicious activity may have occurred.

2.2 Limitations

Finding a sound analysis method for audit trails from the distributed application is the main purpose of this thesis. Lower-level audit trails created by the operating system will not be considered. Analysis of raw network audit data is also out of scope. Correlation between events from the application server and other components throughout the IT-landscape will be left behind as well, the analysis will only cover an audit trail from one source. Further, a secure way of storing the audit trail has already been designed and implemented, and will not be discussed. The integrity of the audit trail is thus assumed to be intact. Also, reconstruction of events and root cause analysis of each event in the system is not being considered.

[1] Primary Account Number

Chapter 3

Method

This chapter presents the technique developed for analysis of the the distributed application's audit trail. First, a pre-study will be discussed that was intended to give different angles on how an audit trail may be analyzed. Using it as a foundation, a feasible technique for this one, and audit trails in general has been derived.

3.1 A Survey on Audit Trail Analysis

To find a viable approach for analysis of audit trails, a pre-study was made to get a picture of existing technology and methods in the field. The study resulted in a document that presents what an audit trail is, areas of use and briefly cover the aspects of analysis.

An open source security event management system called OSSIM[4] was tested as part of the study. Such a system facilitates collection of security events from network attached devices; firewalls, routers, switches, servers etc. A consolidated view of all events that occur across the IT landscape can then be monitored. Rules can be defined to give certain events higher priority, which will help addressing important matters early.

It was set up in a small lab network with a few machines running Linux. One of these machines posed as running the application that handles credit cards. Collection of security events from different sources proved to be very simple. Methods for getting events from a wide range of devices and services is bundled with the installation. To handle the audit trail that is of interest in this case, a plug-in had to be created.

For the purpose of testing, this was done by using netcat read the audit trail on the application server side and deliver the audit trail events to the host running OSSIM. A simple regular expression was then used to transform it into the format required; parsing date and the other attributes that each event consists of. Using this approach in an production environment is not wise, as neither confidentiality nor integrity is protected, but it is sufficient for testing.

As it turns out, analysis capabilities were rather limited. Even though cross correlation with events from other sources is possible, the way any event can be analyzed is basic. Essentially it comes down to writing if-implication rules like; if event a has a certain signature and event b has this characteristic, raise the priority of these events. OSSIM also provides a mechanism for threshold monitoring. That is, if a special event has been recognized x times within a certain amount of time, raise an alarm.

This is powerful if one knows what to look for. Attacks which follow a known pattern, like port scans and repeated login failures can easily be recognized. When it comes to an audit trail that only contains puts and gets of credit cards from different sources, these capabilities are not sufficient.

On the other hand, if the audit trail is first analyzed by a module that is specialized on the analysis process. The results can be handed to tool such as this one, which consolidates events from many sources and attaches priorities. As it seems, security event monitoring software is superb when it comes to collecting events, but analysis capabilities fall behind. See the pre-study in appendix for more on this subject and an overview of audit trails.

3.2 Design Rationale

As the analysis technique should not only be able to analyze the credit card handling application's audit trail, but any application's — little can be assumed about its contents. According to CEE[1], a drafted standard[3] for IT event logs, a *log*, *audit log* or *audit trail* is a set of *log entries* which is written to file or sent over the network. A log entry is a record with information about one or more events. An event is defined as situations or modifications within a system that occur over a time interval. As such, it may refer to a CPU instruction, system call, user logon or retrieval of a webpage — the grade of abstraction or granularity can be at any level.

Keeping in mind that one of the goals is to make the analysis method generic and interoperable, the approach taken must account for this. Hence, a very interesting concept is anomaly detection[6], which does not require availability of deep knowledge about a system that is to be analyzed. The foundation for this approach was introduced by Dorothy Denning in 1987. It is essentially based on monitoring normal system usage, which is considered to be free from intrusions. Profiles are created with the normal usage as basis. No known exploits or vulnerabilities is incorporated into the model. Intrusions are detected by recognizing usage that does not comply with the profile. The rational behind this is that intrusions into an information system must manifest as a change in usage and will thus make its behavior differ from what earlier has been observed.

3.2.1 Multiple Angles of Analysis

There are many different ways to determine what constitutes normal behavior. Some of these include statistical techniques, hidden markov models, k-means clustering, and neural networks[15][2]. Hence, different models will perceive normal behavior differently. What is considered normal by the first model, doesn't exactly conform to what is found to be normal by the next one.

As a simple example, let's consider a corporate network, with a rather loosely defined IT policy, where there are no restrictions on outgoing connections. That is, any host inside the network may connect to any service on the Internet (i.e. there are no firewall rules restricting access to remote ports). A profile of each host within the network is derived from an audit trail with time, used services (i.e. destination IP and ports) together with the amount of data transferred. From this information it is possible to paint a vivid picture of Internet and LAN usage on a per workstation basis.

Let's say that a workstation is normally used during office hours and that it is used to browse the web, editing documents on the local fileserver, and checking e-mail from the corporate e-mail server. All this behavior is captured in the workstation's profile and significant deviations should be flagged.

Now, if the workstation starts to connect to previously unknown services, lets say IRC (ports 6667–7000) — often used by operators of botnets to issue commands to zombies[2] — this should contribute to a departure from the workstation's profile, at least for some models of normal behavior. Further, if the workstation is issued a command to send e-mail (by acting as a relay) to a vast amount of different recipients, a huge amount of connections to port 25 (smtp service) will be made. This will most certainly trigger the indication of an anomaly for any model — be it a neural network or statistical— because this behavior differs significantly from the regular behavior of the the workstation. Further, if this is done during a period when there is usually no activity coming from the workstation in question, a deviation is even more obvious.

Perhaps the connection to IRC was completely benign, and just part of a new pattern of normal behavior. If so, any intrusion detection models that treated this as an anomaly has signaled a so called false positive — a false alarm. As for the latter deviation with remote connections to port 25, it is more probable to be malicious. Keeping this in mind, a very important aspect of the design rationale that will be presented has been demonstrated. That is, if an event is jointly indicated as anomalous by multiple analysis techniques, the certainty for an actual intrusion becomes elevated.

[1] Common Event Expression

[2] Computers that have been taken control of and included as part of a botnet

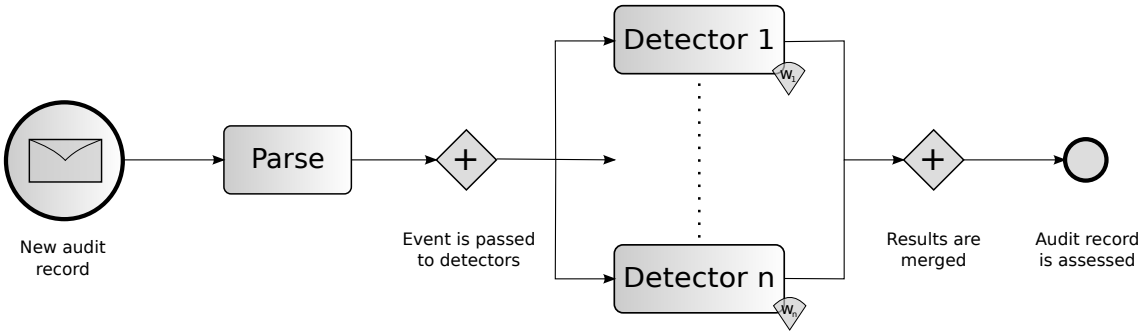

Figure 3.1: Detection chain

3.2.2 Event Flow

Few assumptions can be made on the audit trail format. However, what is known, is that an audit trail consists of several events. Each of these events has certain properties that depend on the application. These properties will follow a consistent format. Hence, each event can divided into its properties, or a set of n attributes $A = \{a_1, a_2, ..., a_n\}$. The only attribute that is assumed to be present for an event from an arbitrary audit trail is the date and time when it occurred.

For each application, the attributes of interest need to be determined and extracted from each event. The extraction is done by a parser, which translates each event into a uniform format, regardless of underlying audit trail, holding time and remaining attributes. Of course, which attributes that are of importance, depends on the application, and needs to be decided upon in each specific case.

After an event has been parsed, resulting attributes are passed to several detectors in parallel. More specifically, the parsed event is passed to a queue in each detector, holding events that are to be processed. Each detector is using a separate anomaly detection scheme. As an event arrives, it is removed from the queue and compared to the detector's behavioral profile. Either the event is deemed normal or anomalous, and the verdict is put in a result queue. The verdicts are drawn from the result queues and are merged into a joint evaluation. The evaluation is done by looking at how many of the detectors that deemed the event anomalous. If a large portion of the detectors classified the event as anomalous, an alarm with high certainty is signaled. In the case where only a few detectors evaluated the event as anomalous, an alarm will still be raised, but with low certainty. Figure 3.1 depicts this process. The certainty of an alarm is dependent on a weight that is associated with each detector. To control how much each detector impose the certainty of an alarm, this weight is adjustable.

Formally, for a set of detectors D, each detector $d \in D$ has a pre-defined weight w_d attached to it. For each event, a detector d produces a binary result $r_d = \{0, 1\}$, that designates an event as normal or anomalous, respectively. The joint certainty, c, of an anomalous event is calculated as

$$c = \frac{1}{W} \sum_{\forall d} r_d w_d \tag{3.1}$$

where

$$W = \sum_{\forall d} w_d \tag{3.2}$$

This model allows for simple tuning of how much a detector imposes the certainty of an alarm. As c will be in the interval $[0, 1]$, this is a good scale for the certainty of an alarm.

3.2.3 Class Design

In this section, a few classes which serve as a foundation for the detection process will be presented. Implementation has been done in Python, which facilitated fast development. Therefor, a few Python specific types like `list` and `dict` show up in the diagrams. Equivalents to these data structures exists in most languages (if not, can easily be derived), so the following can be implemented, with few modifications, in any object oriented language.

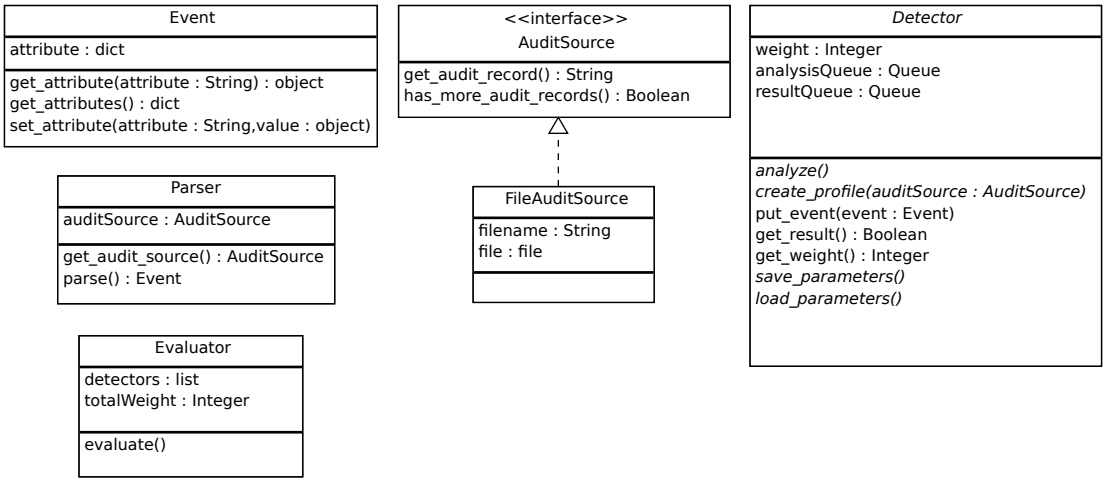

Figure 3.2: Foundation classes

Parser

An audit trail can be stored in different ways. It is often stored in a plain text file or transferred over a network. An interface called `AuditSource` has thus been introduced for a generic representation of any audit trail. At the moment, three classes implements this interface; `FileAuditSource`, `GzSource` and `Bzip2Source` for reading plain files, gzipped files and bzip2 compressed files respectively (the two latter are not shown i figure 3.2). They essentially read a file representing an audit trail, line by line, where each line represents an audit record.

The `Parser` class is specific to an audit trail. That is, if an audit trail from a different application is to be analyzed, this class must be derived and its methods overridden. An instance of `AuditSource` must be passed to the `Parser` class when it is instantiated. The `parse()` operation is called repeatedly to convert an audit record from the `AuditSource` into an `Event` object. The `Event` object holds each attribute of the audit record, including date and time.

Detectors

For implementation of a new detector that models normal behavior, the abstract `Detector` class must be derived. The `analyze()` operation needs to be overridden and will run in its own thread. Events that are to be analyzed must be consumed from the `analysisQueue` and their corresponding results put in the `resultQueue`. The `put_event(event)` operation is used to add events for analysis to in the `analysisQueue`. `save_parameters()` and `load_parameters()` operations supply functionality for saving and loading profiles to/from disk. In the detectors that were implemented, concerned data structures were simply serialized using Python's pickle module.

When instantiated, the detector's associated weight, together with a list of the event attributes that are to be analyzed, must be passed. The attributes are split into two lists. If it is known which outcomes or values that are possible for an attribute, these can be defined during configuration and will be stored in one of the attribute lists. For instance, if an audit trail contains records regarding users and which networks they connect to, the *static* user attribute would contain an exhaustive list of all users in the system. The other list contains attributes which values are hard to specify, and called *dynamic* attributes. These are gradually discovered during the training process when building a profile of normal behavior. Concerning the aforementioned audit trail, a dynamic attribute could be the networks that the users connect to.

Two detectors that derives the abstract Detector class has been implemented. One is based on statistics to determine what normally happens during a time window. The second uses a multivariate statistics approach, where an audit trail is modeled as a process[16]. Each is described in the subsequent sections.

Evaluator

The `Evaluator` class collects analysis results from the detectors. Using these as a basis, each event is given a certainty according to equation 3.1. The `detectors` list contains all detectors in the analysis chain. Results are obtained from the detectors by calling `get_result()`. This operation is called for each detector, by sequentially going through the detector list. If no result is available, the operation is blocking until one arrives. The certainty of an analyzed event can then be calculated by getting each detectors' weight through the `get_weight()` operation.

3.3 Sliding Window Detector

The basic idea behind this detector is examination of what has happened during a time window. As time progresses, this window follows, and thus "slides" with time. Using the notion first introduced by Denning[6], that an intrusion into a system will impose abnormalities in the characteristics of its usage, it is assumed that events occurring during the examined time window should conform to previously observed behavior.

To measure these characteristics, a representative, existing audit tail that serves as training data is needed. Also, a set of attributes, $A = \{a_1, a_2, ..., a_n\}$, associated with each event in the audit trail needs to be monitored. Which attributes that are of interest depends on the field the application that is to be analyzed is used in. Each attribute has a finite number of possible values. So, for each $a_i \in A$, there is a set of possible values, $V_i = \{v_{i1}, v_{i2}, ..., v_{im}\}$. A univariate statistical model is built for the occurrences of attribute values over time. In that way, it is possible to see how event characteristics, in terms of their attributes, vary over time.

More specifically, a univariate model is created for each element in the n-ary cartesian product of the values corresponding to the attributes in A. This set of tuples is introduced as P. In general, many of these will have no events corresponding to them, and can be omitted from the profile.

To illustrate this, let's say that attributes of interest in an audit trail containing information on which applications that are executed in an OS, is the user that executed the application, and the name of the binary. This translates into $A = \{user, application\}$. Hence, V includes all users and all applications in the system. If we have a small system with two users and three applications, this yields: $V = \{V_{user}, V_{app}\} = \{\{alice, bob\}, \{cat, cd, ls\}\}$. The cartesian product of the elements in A yields $P = user \times application = \{(alice, cat), (alice, cd), (alice, ls), (bob, cat), (bob, cd), (bob, ls)\}$. Thus, there will be a maximum of six univariate statistical models that describes this system. When an event arrives as a consequence of a user executing a binary, let's say that *alice* executes *cd*, this will be reflected in the corresponding univariate model, $(alice, cd)$, by storing the time that it occurred.

3.3.1 Model Parameters

To build a profile of a system, a few parameters need to be introduced. First, the size of the time window, w, that is to be examined needs to be established. A second is time step, Δ, which determines in how big steps the window is pushed forward.

Now, having grouped events by their attributes and noted when they occur, a univariate statistical model is built for each tuple in P. This is done by slicing the time line in equal sizes for each attribute tuple. The width of each slice is defined by the time step parameter, yielding s time slices per day (and attribute tuple) in the training data. These are now combined into s time windows. For each window i, a corresponding occurrence set is introduced. This set is created by adding the occurrence count in time slice $i - \Delta$ through i for each day, resulting in n sums. These sums constitute occurrence set O_i.

The window is repeatedly pushed forward an interval of the size Δ until s slices have been covered. This yields s occurrence sets, $\{O_1, O_2, ..., O_s\}$, for each tuple in P. The time window must be a multiple of the time step, otherwise a window won't cover a number of complete time slices.

As an example, if the time line is sliced into one hour pieces, and a time window spans four hours, the event occurrences in the time span 21:00 – 01:00 will be put in the first window, the

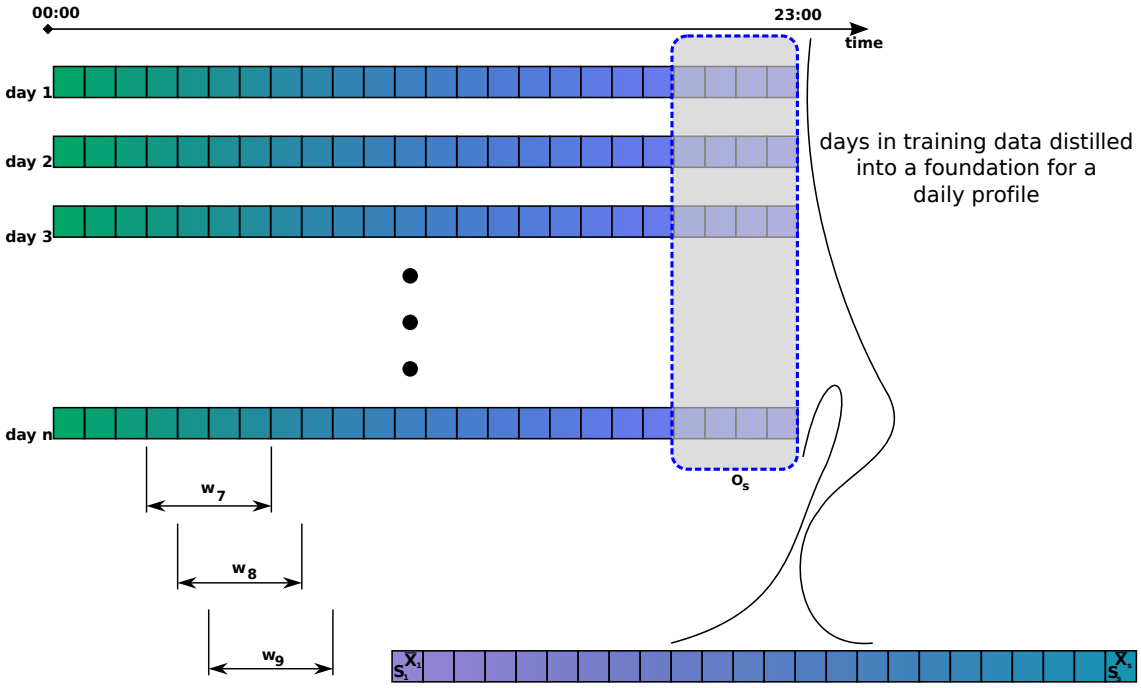

Figure 3.3: Merge of time slices into a daily profile for an attribute tuple

event occurrences between 22:00 – 02:00 will be put in the next, and so forth up until 20:00 – 00:00, totaling in 24 windows.

The last step in building the set of univariate statistical models for the profile — one model for each tuple in P — is deriving the mean and variance of event occurrences in each of the s time windows. This is rather straight forward, as the occurrence sets contains the number of events for each window. Having an audit trail with n days of training data, the occurrence sets corresponding to each window, will be of length $|O_i| = n$. This is due to the fact that each window has as many samples as there are days in training data. The sample mean $\overline{X}$ and sample variance, S^2, for a window, t, can be obtained in the following way:

$$\overline{X}_t = \frac{1}{n} \sum_{i=1}^{n} X_i \tag{3.3}$$

$$S_t^2 = \sum_{i=1}^{n} \frac{(X_i - \overline{X}_t)^2}{n-1} \tag{3.4}$$

where each X_i corresponds to occurrence sum i in occurrence set O_t.

An important assumption is made here, which is that the events in the audit trail are following a normal distribution. According to the central limit theorem[11], this is approximately true if the events are random and independent and if the sample space is large enough. Empirical studies show that a bell shape starts to arise when the number of drawn samples exceed 25, regardless of the underlying population's characteristics.

Another assumption, which can over-generalize the profile, is that system usage varies in a cyclic manner over 24 hours. This is probably true in many cases, but if an office workstation is considered, it will probably run according to a certain pattern during work days, and another during weekends where there is little or no activity. Therefor, it might be prudent to create two different profiles, one for work days, and another for holidays.

3.3.2 Detection Process

The principle for profile creation has now been introduced. What remains is to establish if new events in the audit trail conform to the previous behavior as recorded in the profile.

During the detection process, the same attributes as established during the profile creation are looked at. When a new event arrives, its attribute values are examined and the occurrence count for the corresponding tuple is increased — note that the profile is not altered, a new occurrence count for each tuple in P is introduced. After the event's contribution has been added, the newly recorded number of occurrences during the latest time span, as defined by the window size parameter, w, is compared to the profile as follows:

Let's denote the new occurrence value during the latest window, t, as c, and that the this count corresponds to tuple p_i. The mean and variance recorded in the profile for tuple p_i in time window t is compared to c. Based on the normal probability rule[11], which implies that when sampling from a normal distribution, 99.8% of all observations should lie within 3 standard deviations from the mean, an occurrence count above this limit is considered an anomaly. Counts that fall below 3 standard deviations from the mean are definitely rare, but not considered an anomaly. Activity below this limit is hard to infer as intrusions, as they tend to manifest as an increase in activity. Hence, if

$$c > \overline{X}_t + 3S_t$$

the event is considered an intrusion. This gives a theoretical 0.1% of false positives. Table 3.1 contains a summary of the normal probability rule.

Normal probability rule
$P[-\sigma < X - \mu < \sigma] = 68\%$
$P[-2\sigma < X - \mu < 2\sigma] = 95\%$
$P[-3\sigma < X - \mu < 3\sigma] = 99.8\%$

Table 3.1: X is normally distributed with mean μ and standard deviation σ

To illustrate this process, consider the small system previously mentioned with alice and bob as users. Let's assume that a profile has been created from an audit trail with training data and that the assumption about normally distributed events applies. Further assume that the window size is 4 hours and that the time step is 1 hour. This yields a profile with 24 windows, each having an associated sample mean and sample standard deviation.

Now, Alice works hard during one morning and reads a substantial amount of files by using cat. As it turns out, between 8:00 and 12:00 during the morning, she has executed cat 120 times. This count corresponds to window 11 in the profile for attribute tuple {alice, cat}, which has a sample mean, $\overline{X}_{11} = 60$, and a sample standard deviation, $S_{11} = 22$. The occurrence count of 120 for {alice, cat} in the "live window" is clearly above the mean, but still below $\overline{X}_{11} + 3S_{11}$, so no false alarm will be signaled.

3.3.3 Previously Unknown Events

In the event that training data does not contain any prior information regarding certain event characteristics, a policy need to be introduced that handles how to take action. This often happens when a new attribute value is brought into the picture. In the example of the audit trail for the small system, it can be a new user or application.

There are basically three different ways to handle this. The easiest is just to disregard any events that falls into this category. This might be a viable approach if profiles are rebuilt often, so that new attribute values are incorporated.

Another approach is to flag it as an anomaly, which can give a high amount of false positives. Consider the case when a new user is created and starts to use the system.

A more prudent approach is to derive a generic profile for this case. One way to do this is to look at average system usage as a whole and obtain its sample mean an standard deviation. However, because prior knowledge of the actual characteristics of these events does not exist it is

impossible to know a priori if this is even close. Although — as just mentioned — when the new event data is incorporated into the profile, this will cease to be a problem. This approach can serve as something in between the two extremes.

Which policy to use is hard to give any pointers on without knowing the underlying field of usage. If new attribute values are rare — which is seldom the case — going with the second approach will not yield many false positives and is a good choice.

On the other hand, if system usage is changing often, a better alternative is the first, unless drowning in false positives is not a problem. Perhaps the third is policy feasible, but may still give some false positives. All this comes down to the nature of the audit trail.

3.4 Multivariate Statistics Detector

This detector is based on a paper presented in 2001 by Nong Ye and Qiang Chen[16]. The approach essentially treats the audit trail as a process, of which its properties are measured for each new audit record. These measurements are evaluated to see if the process operates within parameters. If it is out of control, the event that put it out of control is considered an anomaly. This technique, based on a chi squared distance, was compared to a different approach, based on Hotelling's T^2 test, in a paper[17], published in IEEE Transactions on Computers in 2002. From this comparison, the chi-squared test technique was chosen, because it shows much better performance and signaled less false positives.

The technique is based on statistical quality control, often used during the manufacture of products to make sure that quality is maintained. As an example, let's consider the production of a specific steel beam. In the specification of the beam, its dimensions and weight, among other details are listed. During production, these are measured of each new beam, and if any, or a combination of the measurements are differing too much from the specifications, the production process is deemed to be out of control.

3.4.1 Model Representation

In the same way as for the previous detector, a set of attributes that are to be monitored must first be established. This is done in the same way, and a set of attribute values, P, is obtained by the n-ary cartesian product of the values corresponding to the attributes in A.

A set of column vectors of length $|P| = p$ are introduced to hold observations of the process (i.e. audit trail). These are denoted $\mathbf{X_i} = (X_{i1}, X_{i2}, ..., X_{ip})$, where i is the i:th event in the audit trail. If the small system with alice and bob as users is considered, this vector would be of length 6 with each position corresponding to an attribute tuple. When a new audit record is examined, let's say that *alice* executes *cat*, the corresponding observation vector will have a 1 in the position that corresponds to this event, and zeros the other. Thus, each vector will be sparse, and only the index corresponding to a specific observation needs to be stored.

When the process is in control (i.e. no intrusions are in the audit trail), $\mathbf{X}$ follows a multivariate normal distribution with mean vector μ. From a sample space of size n, the sample mean vector $\overline{\mathbf{X}}$, can be obtained:

$$\overline{\mathbf{X}} = \begin{pmatrix} \overline{X}_1 \\ \overline{X}_2 \\ \vdots \\ \overline{X}_p \end{pmatrix} = \frac{1}{n} \sum_{i=1}^{n} \overline{\mathbf{X}}_\mathbf{i} \tag{3.5}$$

From the set of observation vectors, a set of chi-square test statistics are calculated. There are p attribute tuples, and X_i corresponds to the observation of the i:th tuple at a certain time.

$$X^2 = \sum_{i=1}^{p} \frac{(X_i - \overline{X}_i)^2}{\overline{X}_i} \tag{3.6}$$

where $\overline{X}_i$ is the sample mean of the ith tuple.

The X^2 test statistic indicates the distance of a data point from the center of the data population. According to the central limit theorem, when p is large the X^2 test statistic follows a normal distribution. By obtaining the mean and standard deviations of the X^2 values, a good indication of when the process is in control can be determined.

Before this is done, the observation vectors are transformed to emphasize recent events. Using an exponentially weighted moving average, the vectors depend not only on the current observation, but also the previous ones. A new observation gives a contribution of weight λ in the position that corresponds to the observed attribute tuple. The others are given a decay of $1 - \lambda$ from the previous observation vector's values. Formally, event i in the audit trail gets the following transformed observation vector, T_i:

$$\begin{array}{ll} T_1 = \lambda X_1 & \text{first event} \\ T_i = \lambda X_i + T_{i-1}(1-\lambda) & \text{further events} \end{array}$$

As an example, let's consider the small system with *alice* and *bob* as users. Let $\{(alice, cat)$, $(alice, cd)$, $(alice, ls)$, (bob, cat), (bob, cd), $(bob, ls)\}$ correspond to the first through the last positions of an observation vector. Then suppose the following events are observed: $\{(alice, cd)$, $(alice, ls)$, (bob, cd), (bob, ls), (bob, cd), $(alice, cd)\}$. This yields the following observation vectors:

$$\mathbf{X_1} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \mathbf{X_2} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \mathbf{X_3} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \mathbf{X_4} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \mathbf{X_5} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \mathbf{X_6} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Using a smoothing factor of $\lambda = 0.3$, which is the recommended value in the paper[16] that this detection technique is based upon, the following is obtained after application of the exponentially moving average technique:

$$\mathbf{T_1} = \begin{pmatrix} 0 \\ 0.3 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \mathbf{T_2} = \begin{pmatrix} 0 \\ 0.21 \\ 0.3 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \mathbf{T_3} = \begin{pmatrix} 0 \\ 0.147 \\ 0.21 \\ 0 \\ 0.3 \\ 0 \end{pmatrix}, \mathbf{T_4} = \begin{pmatrix} 0 \\ 0.1029 \\ 0.147 \\ 0 \\ 0.21 \\ 0.3 \end{pmatrix}, \mathbf{T_5} = \begin{pmatrix} 0 \\ 0.07203 \\ 0.1029 \\ 0 \\ 0.447 \\ 0.21 \end{pmatrix}, \mathbf{T_6} = \begin{pmatrix} 0 \\ 0.37203 \\ 0.07203 \\ 0 \\ 0.3129 \\ 0.147 \end{pmatrix}$$

Older observations will thus contribute less as new events arrive. An observation that was made i events ago has decayed a factor of $\lambda(1-\lambda)^i$.

Once all transformed vectors from the training data has been acquired, a sample mean vector can be obtained as per equation 3.5. To see how far from the population center each of these are located, their chi-squared distances are calculated using equation 3.6. A small X^2 value indicates a close proximity to the center. In order to decide whether new events comply with previous behavior from the training data, a base line for what is close need to be derived. Therefor, the sample mean, $\overline{X}$, and variance, S, are calculated from the X^2 values in the training data. From the normal probability rule, see table 3.1, 99.9% of all samples drawn from a normal distribution should fall below $\mu + 3\sigma$.

Using this as a control limit, when a new event is found that has a greater X^2 distance than $\overline{X^2} + 3S$, the process will be treated as out of control. An anomaly will not be flagged yet, however, as it is in the paper that this technique is based upon. A new parameter, tolerance, has been introduced.

During testing, it was found that the process went out of control for one event, and then turned in, below the control limit, right away. For other chains of events, it continued to depart from the control limit as events were processed. This parameter was introduced to set a lower limit on how many events in a row that must continuously put the process out of control, before an anomaly is signaled. That is, if tolerance is set to 2, and event i puts the process out of control. Event $i + 1$

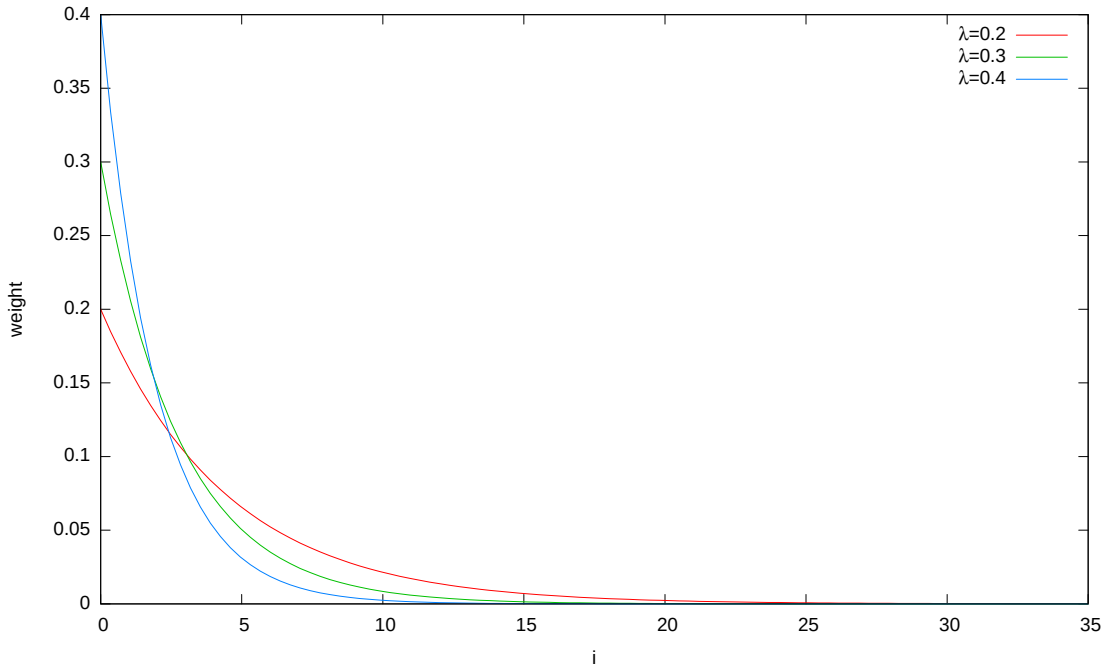

Figure 3.4: Three different decay factors — plot of $\lambda(1-\lambda)^i$

must have a greater X^2 value than event i for an anomaly to be indicated. See the next chapter for how false positives were eliminated by altering this parameter.

As with the previous detector, events with attributes corresponding to tuples not yet discovered in the training data may be observed during analysis. To handle this, the event may just be discarded and not signaled. The alternative is to define a low pseudo number for this event's sample mean, so that this will be reflected in calculation of its X^2 distance (equation 3.6). Using this strategy will bump up the distance significantly. To illustrate this by example, consider the following; having $\lambda = 0.3$, and setting the attribute tuple's sample mean to 10^{-5} yields a contribution of $\frac{(X-\overline{X})^2}{\overline{X}} = \frac{(0.3-10^{-5})^2}{10^{-5}} = 8.9 \times 10^3$ to the X^2 distance sum.

Chapter 4

Results

This chapter reveals the tests that were made to determine how well the intrusion detection system is able to catch intruders. Each detector was first tested individually. From these tests, tunable parameters have been altered to lower the amount of false positives without reducing detection rates. The derived parameters are kept for testing of the entire system, where all detectors will run in parallel.

4.1 Analysis Configuration

The audit trail analysis system is configured through a configuration file. Which event attributes that are to be analyzed is specified here. Also, specific parameters for each detector, like where profiles are stored, and how many standard deviations to use for signaling anomalies can be set. Tables 4.1 and 4.2 shows a summary of configuration settings that affects the detection process.

Parameter	Description
Window size	Specifies the size of the time window that the frequency of events are observed.
Time step	Determines in how big steps the time window is pushed forward.
Threshold	Anomalies will not be signaled unless the number of observations within a time window is above this limit.
Standard deviations	The number of standard deviations that must be exceeded for an anomaly to be flagged. See table 3.1.

Table 4.1: Summary of configuration directives for the Sliding Window Detector

Parameter	Description
Tolerance	The number of subsequent events that continuously puts the process out of control before an anomaly is flagged.
Decay	Imposes the weight of each new event observation. A higher value will give recently observed events more weight. Figure 3.4 shows the decay of an observation for three different values.

Table 4.2: Summary of configuration directives for the Multivariate Statistics Detector

4.2 Audit Trail Format

The audit trail is stored in a gzipped or plain text file, with one audit record per line. Each audit record contains time and date the event occurred, as well as details regarding each credit card

Parameter	Setting
Dynamic attribute	city
Static attribute	messageType
Window size	180 minutes
Time step	30 minutes
Threshold	1
Standard deviations	3

Table 4.3: Initial test Parameters for the Sliding Window Detector

transaction. Many of these are related to Amadeus' IT infrastructure, and its entire format will not be disclosed. All are however not needed for revealing the test results. The `Parser` class was adopted to extract each of the event attributes from the audit records and create corresponding `Event` objects.

From each audit record, the office, from which the transaction originates can be obtained. Each office is located in a city, which can be derived from the office attribute. Another important attribute is the message type, which indicates if a transaction concerned a retrieval of a credit card number, or an addition of a new one. Accounting staff usually retrieve credit card numbers, and airlines and travel agencies often add new ones upon selling a trip. Other attributes of interest includes the sign, or user who initiated the transaction, as well as originating application — there are many ways to access the credit card application.

4.3 Sliding Window Detector

The detector testing profile was built using audit data from four weeks, 2010-06-07 to 2010-07-04, totaling in 141 096 126 events. The profile has however been split on a daily basis. Instead of having a generic profile for any day, it has been split into several sub-profiles; one for Mondays, a second for Tuesdays and so forth. The rationale behind this is that transactions are from cities world wide, and not all will have a working week between Monday and Friday. In order to come closer to a sub-profile for working days, and another for holidays, it has been split in this way. Thus, a Monday in the test data will be compared to the Monday subprofile and so forth.

The attributes chosen are messageType and city, i.e. $A = \{messageType, city\}$. With about 1650 different cities and 3 message types in the audit trail, this yields $|P| = |messageType \times city| \approx 5000$.

As a first step to see if any false positives are signaled by the Sliding Window Detector, the same dataset used to create the profile was analyzed. The policy for previously unknown event attributes has been set to disregard any such events. See table 4.3 for initial test settings. The threshold has been set to 1 to include most events in the analysis; if a city has less than one transaction per 30 minutes over three hours, that specific time window will not be compared to the profile.

When analyzing each of the 141 096 126 events from the dataset used to create the profile, no events were indicated as anomalous. This suggests that when an analyzed audit trail conforms well to the profile, there are no false positives. It is however well known that future audit trails is going to have some variability and will deviate from the profile to different extents, even though there are no intrusions.

To get an indication of how much this natural variability in the audit trail data will reflect as false positives, audit trails from the three days following the profile dataset range were analyzed. The settings were left unchanged, see table 4.3. If we assume that the audit trails from these three days do not contain any intrusions, all alarms are false positives. Table 4.4 shows figures over the number of alarms signaled.

When looking closer at the anomalies, it is easy to see that the first set of test data roughly contains four to six times more anomalies than the second and third test data set. Looking at the cause for this, it was discovered that a big deviation from the profile had been signaled, see figure 4.1.

Dataset	Number of events	Events deemed anomalous	False positive ratio (%)
10-07-05	5936116	3680	0.061993
10-07-06	5962288	892	0.014961
10-07-07	5154466	647	0.012552
Total	17052870	5219	0.030605

Table 4.4: Test Results for the Sliding Window Detector using initial parameters

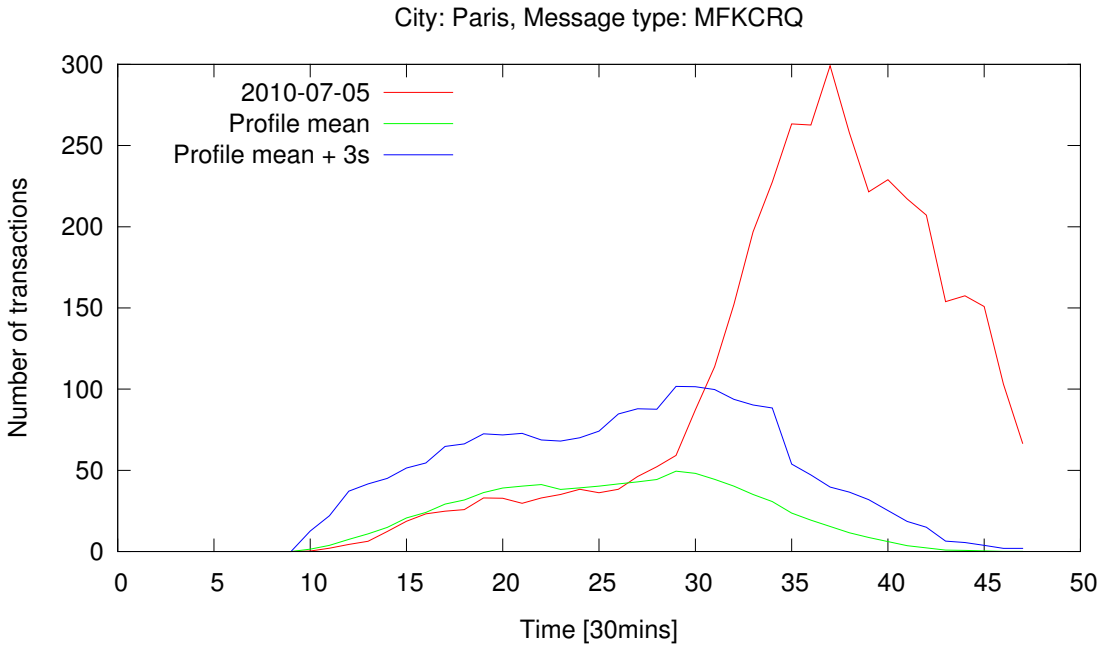

Figure 4.1: Substantial deviation from the test profile

A substantial part of the anomalies are signaled due to the mean and standard deviation being rather small for many of the attributes. As seen in figure 4.2, the mean is close to 0 during the whole day and even though the standard deviation vary more, it doesn't give much room between half-hour 5 and 15.

Signaling an anomaly when the profile limit is so close to zero is not very prudent. The lower boundary for the number of transactions over time that need to be exceeded for an anomaly to be signaled is therefor increased (currently set to 1, see table 4.3). In order to see how this threshold affect false positives, the same three datasets were analyzed with the threshold parameter set to eight different values, see 4.5. Figure 4.3 shows the impact on the number of anomalies by changing the threshold parameter. The graph clearly shows that most false positives are reduced when the threshold is starts to depart from one; when it is set to 5, the false positives are nearly halved for the 10-07-07 dataset. For the 10-07-06 dataset however, the false positives continue to decline substantially until the threshold reaches 20.

Anomalies triggered when the transaction rate is greater than $1/60s$ (threshold $= 30$) may start to get interesting; a city with only a few transactions per day that suddenly goes above this limit is prudent to indicate. On the other hand, if the number of transactions increase from $0.5/60s$ to one per minute, it is not as interesting and will hopefully be caught within the limits of the $\overline{X} + 3s$ in the profile. Of course the threshold level is very specific to the audit trail that is being analyzed and need to be tuned for each new application.

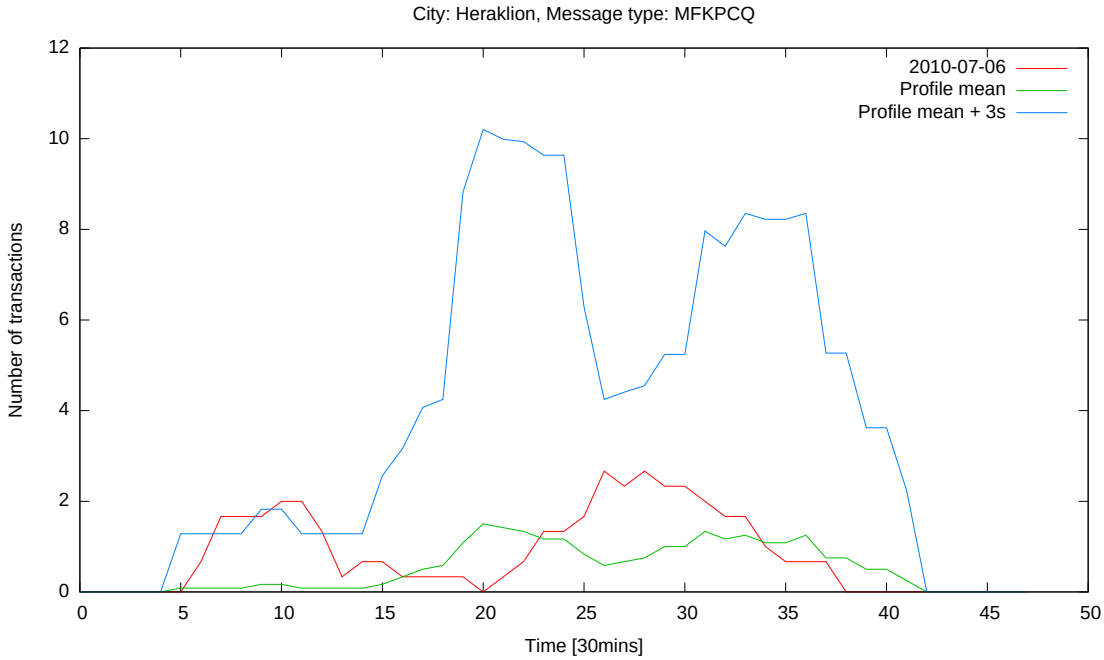

Figure 4.2: Minor fluctuation being signaled as an anomaly

Dataset	thr = 1	thr = 5	thr = 10	thr = 15	thr = 20	thr = 30	thr = 40
10-07-05	0.061993	0.059163	0.057950	0.057327	0.056721	0.056451	0.056451
10-07-06	0.014961	0.009426	0.006793	0.005132	0.003723	0.002717	0.001409
10-07-07	0.012552	0.006751	0.006635	0.006635	0.006247	0.005083	0.003919

Table 4.5: False positive ratios for different thresholds (%)

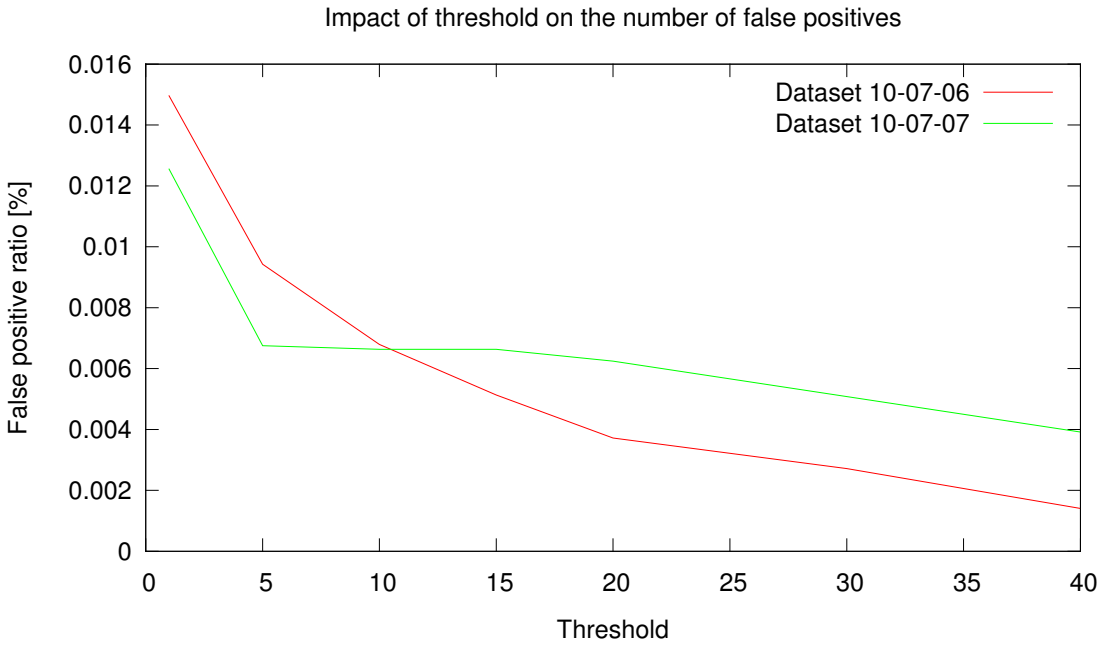

Figure 4.3: Threshold affecting the false positive ratio for the test profile

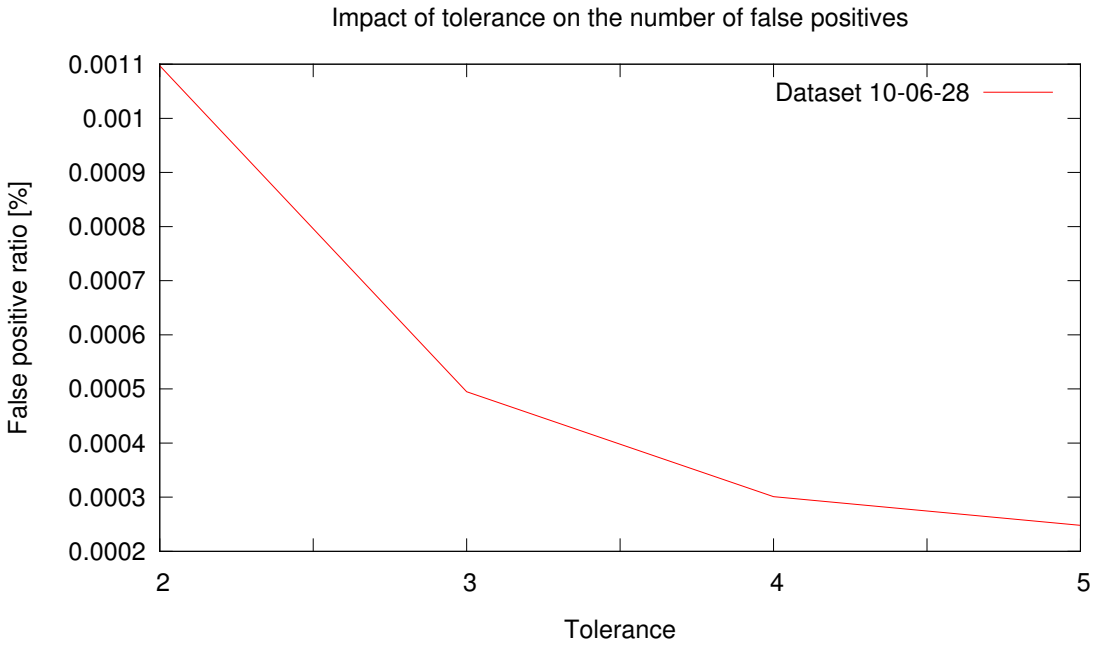

Figure 4.4: Tolerance impact on false positives

4.4 Multivariate Statistics Detector

The dataset used to build a profile for the multivariate statistics detector consists of 5652649 events, corresponding to data from one day. This is due to time constraints. Using a longer period is possible — creating a profile from one day takes about one hour when analyzing by the message type and city attributes. Using these attributes yields an observation vector of a dimension that is approximately 5000.

As initial indication of the amount of false positives, the same data used to create the profile was analyzed. The analysis was run with the tolerance parameter set to 2, resulting in 62 anomalies, or a false positive rate of $62/5652649 = 0.001097\%$.

To lower the amount of false positives, the tolerance parameter was increased in steps of 1 until the ratio decrease was no longer very significant. Figure 4.4 shows the ratio of false positives compared to the tolerance level.

4.5 Detectors in Parallel

In this section, test results from runs with both detectors in place in the analysis chain will be presented. Each detector has first been tuned with the parameters found in the previous sections of this chapter, see table 4.6.

Sliding Window Detector	
Parameter	Setting
Window size	180 min
Time step	30 min
Threshold	30
Standard deviations	3

Multivariate Statistics Detector	
Parameter	Setting
Tolerance	4
Decay	0.3

Table 4.6: Detector settings

The test data used has been produced by enquiring the designers of the system on how it is

ID	Case	Expected outcome	Motivation
1	A previously unknown city puts a credit card.	Deemed normal.	Activity from previously unknown cities should not be indicated.
2	A transaction is made from a city outside working hours.	Deemed anomalous.	Transactions from offices outside working hours is suspicious.
3	Transactions from a city consisted only of puts of credit cards, now a substantial amount gets occur.	Deemed anomalous.	Major change in system use should be flagged.
4	A city normally has 200 credit card puts per day, it is now increased to 1000.	Deemed anomalous.	Major behavioral changes should raise an alarm.
5	A city normally gets 2 credit cards per day, it is now increased to 6.	Deemed normal.	A 200% system usage increase should not trigger alarms if the amount of transactions is low.

Table 4.7: Summary of test cases

supposed to operate under normal conditions. Getting representative test results is hard without test data that includes real intrusions, but with no such data, this is seems like a viable approach. A few cases have been manufactured that should either the deemed intrusive or not, these can be found in table 4.7. The same profiles that were used to tune the detectors are used to test these cases.

That is, the Sliding Window detector will have a profile based on data from four weeks, 2010-06-07 to 2010-07-04 and the Multivariate Statistics detector will use a profile created from data corresponding to one day, 2010-06-28. The test data is created by taking all the events from 2010-07-05, which is not part of any profile, and injecting events corresponding to each test case into this data. The actual data used in the Sliding Window detector's profile will correspond to four days from the four training weeks. As the test data happens to be a Monday, the data used will be the four Mondays in the profile.

4.5.1 Test case 1

The first case was created by introducing a transactions from an office in a pseudo city in the test data. None of the introduced transactions were deemed anomalous by either detector.

4.5.2 Test case 2

Case 2 was tested by adding 489 transactions outside a city's working hours to the test data. These events were distributed over a period ranging from 19:45 to 20:25. After running the test case, 14 events were jointly flagged with full certainty. The Sliding Window detector deemed 186 of the injected events as anomalous, and 95 were flagged by the Multivariate Statistics detector.

4.5.3 Test case 3

The initial construction of case 3 test data was done by adding just about enough data to have it signaled by the Sliding Window detector. This corresponds to the addition of at least $threshold/time\ step * window\ size = 1\ event/min * 180min = 180$ events, distributed randomly over three hours to breach the threshold. Therefor, the first data set was given 200 events, and 20 of these events were deemed anomalous by the Sliding Window detector, as expected. However, none of them were deemed anomalous by the Multivariate Statistics detector.

To see if an increase of malicious events in the test data would make this case detectable, 2000 events were added randomly over three hours. This did not affect the Multivariate detection results, 1820 of the events were however deemed anomalous by the Sliding Window detector.

A third alteration of the data to see if detection rates could be imposed, was to cluster the events over a shorter period and thus increase the intensity of the malicious activity. The 2000 events were instead distributed randomly over five minutes. This resulted in 60 events deemed anomalous by the Multivariate Statistics detector and 1820 events being signaled by the Sliding Window detector. Both detectors flagged 58 of the 2000 events jointly.

4.5.4 Test case 4

This case's test data was created by injecting another 850 events for a city with 177 events in the 2010-07-05 audit trail, totaling in just above 1000 events. These were injected to proportionally to the system usage by city. That is, the events were only added during working hours in order to avoid skewing when the system is used.

The Sliding Window detector deemed 33 of these as anomalous, and the Multivariate Statistics detector found 7 anomalous. None were however jointly flagged as anomalous.

To see if the tolerance of the Multivariate Statistics detector imposes the detection rate in this case, it was set to 2 instead of 4, and the same dataset was rerun. This increased the amount of detected anomalies to 17, but none were jointly detected. As in the previous case it is possible to group the added events under a shorter interval and thus increase the intensity of the event flow.

4.5.5 Test case 5

The fifth case was tested by introducing a few more transactions for a city with a low transaction count and thus yielding 200%+ increase in system usage. None of these transactions were however deemed anomalous by any detector.

Chapter 5

Evaluation

An approach for audit trail analysis from an arbitrary application has been presented. Two anomaly detection techniques have been implemented and their results correlated to sieve out anomalies with higher certainty of being an intrusion. Evaluating the actual results has been complicated by the fact that no real intrusion sessions have been available during testing. The intrusion scenarios were manufactured and audit data was injected into copies of production data to reflect these. False positives can however be more easily determined than the actual detection capability, because no intrusions are needed for a false positive to be signaled.

5.1 Testing and Datasets

Due to the fact that audit trail data from production have been used to derive the test profiles, it has simply been assumed to be free of intrusions. If this is not the case, and intrusions in fact are in the data that the profiles are based on, the same nature of intrusions could be in the data that the detectors are tested against. Intrusions in the profile will thus be classified as part of normal behavior and mask the same intrusive behavior during testing. Misses, or false negatives, of this nature is a weakness that all anomaly detection schemes are troubled by.

There is also the issue of finding the right amount of training data to fit a detector's profile to training data. If too much training data is used, the model might become too generic and deem a true anomaly as normal. Have too little data, on the other hand, will cause over-fitting, and have the adverse effect (i.e. yielding a higher amount of false positives).

5.1.1 False Positives

Each of the detectors yields rather low false positive ratios. Especially when tuned further. Regarding the Sliding Window Detector, increasing the threshold parameter proved very effective. The largest portion of false positives were removed just when the threshold parameter started to increase. When it was pushed beyond 20, the effect was no longer as distinct.

As for the Multivariate Statistics Detector, the tolerance could effectively be used to yield a decrease in false positives. Figure 4.4 clearly shows this. By altering the detectors' parameters in this way, a decrease in false positives of up to 80% was shown for some datasets.

Setting these parameters too excessively risks that real intrusions will be masked, and false negatives to be introduced. From the test cases, this indication is however not very present. This is probably due to their careful settings. It is also important to keep in mind that these parameters are specific to the audit trail that in question. The settings derived in the previous chapter may be too high for a different application.

Trying to reduce a false positive rate that is a fraction of a percent might seem like a little gain. When analyzing millions of events per day, these small fractions will translate into hundreds of false positives. Therefor, being able to decrease the false positives to around a fifth of the initial, untuned amount, is not that insignificant.

Further, when looking at how events were jointly flagged by the detectors, no false positives were flagged by both detectors. This gives all false positives a lower certainty according to the

rationale used to evaluate detection results.

5.1.2 True Positives

As was discussed in the preamble of this chapter, the audit trail data for the test cases were synthesized. Probable intrusion scenarios have been derived together with designers of the distributed application. Thus, the merit of the actual capability to catch intrusions may be hard to uphold.

Even so, the test results show that all intrusion scenarios were detected. All, but test case 4 was given the highest intrusion certainty, but was still detected by both detectors.

A problem when test audit data is injected is that the results depend on how the events are distributed. Test results can thus be skewed. As indicated by test case 3, the Multivariate Statistics Detector is very sensitive to the distribution of events. This is due to the fact that the intensity of events affect the X^2 values significantly. A burst of subsequent events with the same observation vector will yield a big chi-squared value if the mean vector, $\overline{\mathbf{X}}$, does not reflect this as being a common phenomenon.

An interesting observation when it comes to the Multivariate Statistics Detector is that it is sensitive to mean shifts. This can potentially detect cooperative attacks from multiple cities. Consider the case where multiple cities increases their intensity of transactions to different extents. This will contribute to several departures from the mean vector. These mean shifts will result in a big chi-squared value that could potentially breach the control limit and signal an anomaly.

The univariate analysis performed by the Sliding Window Detector can not pick up this kind of attack. This holds as long as transactions from a city that is involved in the attack does not individually breach the control limit.

In general, for both analysis techniques, attacks that hide in the noise will be hard to catch. That is, if an attack is not intensive enough to breach the control limits, it will go unnoticed. Because all audit trails are different, the variability, and thus the noise level, will vary with the application and its environment.

5.2 Future Work

As mentioned previously, the audit trail data that is used to create the profiles of normal behavior can not be guaranteed to be free from intrusions. An approach for sanitizing training data is presented in a paper[5] by five researchers at Columbia University. Incorporating this scheme before training of the detectors could potentially lower the amount of false negatives.

To have any indication of false negatives, the detection chain would have to be tested on datasets with known intrusions. The modifications needed to do this, will be to derive the Parser class, so that the events in the new audit trail can be partitioned into its attributes. A new AuditSource class may need to be implemented as well, if it differs from the currently assumed file format with one audit record per line.

Having this in place, ROC[1] curves that display false positives versus true positives could be constructed. Finding representative datasets is however not very easy. One that is commonly used to evaluate intrusion detection systems has been produced by the Lincoln Laboratory at MIT. Criticism[9] has however been given to the way the datasets have been derived, and questions if they are appropriate for evaluating intrusion detection systems.

[1] Receiver operating characteristic

Appendix A

Pre-study

Introduction

The aim of this document is to give an overview of the general use audit trails and some insights as to how they can be analyzed. Several areas of use and different levels of logging will be discussed. In the last part, each of the requirements in section 10 of the PCI DSS standard will be mapped to the different applications of audit trails. Finally, the mapped-out topics will be examined a bit more closely.

Definition

Very briefly, an audit trail is a log of any events that occur in a computer system. However, without a clear definition of the terminology, this discussion will not be very fruitful. According to CEE, the Common Event Expression standardization[3] that defines how computer events are described, logged and exchanged,

"*Events* are observable situations or modifications within an environment that occur over a time interval."

As such, an event can be very granular; i.e. an instruction executed by the CPU, or a bit more coarse such as execution of an application or the opening of a file. Further, a *log entry* is defined as:

"A single record involving details from one or more events. A log entry is sometimes referred to as an event log, event record, alert, alarm, log message, log record, or audit record."

Finally, a collection of log entries constitute a *log* or an *audit trail*:

"A log is the collection of one or more log entries typically written to a local log file, a database, or sent across the network to a server. A log may also be referred to as a log file, an audit log, or an audit trail."

Many different levels of auditing is thus possible, some audit trails concern the packets transmitted on a network and others log what happens locally on a workstation.

Areas of Use

Logs have been around since the dawn of computer systems and were originally designed to track down technical problems. Nowadays, the information found in audit trails is used in many different areas. Analyzing logs accommodates optimizing computer and network performance, identifying security incidents, policy violations, tracking frauds and operational problems. It is also of interest to other initiatives, such as demonstration of compliance with regulations.

When looking at the computer security aspect of audit trails, parts from many different logs have potential bearing depending on the situation. Sources of direct interest from this perspective are network devices, such as routers, switches and network monitoring software, as well as servers and workstations. According to the computer security handbook[12] released by NIST, the following classifications cover the computer security objectives regarding audit trails.

Accountability

An audit trail can be scrutinized to find illegitimate use of resources. A user, for instance, might have authorization to use a resource in order to be able to do his/her job, but the audit trail can show any misuse of such resources (e.g. for personal gain). This aspect of audit trails mainly involves mapping events to users.

Reconstruction of Events

Audit trails can be used to reconstruct a previous event — or a chain of events, depending on the granularity — to aid the task of finding the cause of a problem. For instance, it can help determining if it was an application malfunction or a human mistake that caused a system crash. It may also help forensic teams to track down the source of a security incident by indicating how the system was breached, and when.

Intrusion Detection

An audit trail can include sufficient information to aid the detection of intruders. Intrusions can either be detected in *real-time*, as the audit records are created, or *after-the-fact*, at a later time from analysis of the trail (often as a batch job at regular intervals). Attention can then be given to any raised alerts and the audit trail can used to assess the seriousness of any intrusions (or attempted) and facilitate the review of any security controls that were subverted.

Monitoring

The audit trail may also be reviewed on-the-fly — not to find intrusions — but to find problems with the operation of critical systems. A real-time analysis of the audit trail can show if a system is operating as expected. Component failures, such as disk crashes and increasing use of resources (e.g. network bandwidth) may be caught by monitoring.

Levels of Auditing

As mentioned in the introduction, an audit trail can target many different layers of a computer system. It is of course possible to have multiple audit trails present in a system concurrently, each creating a trail at a specific level. The NIST handbook does not cover network auditing, and differentiates between system and user level audit. The latter two can be consolidated into one level, and focus can be put on operating system auditing, network auditing and application auditing.

Keystroke Monitoring

There is one exception to the event-based approach, it involves monitoring each keystroke made on a system and the corresponding response during a user session. Examples of keystroke monitoring include capturing characters as they are typed by users, reading users' electronic mail, and viewing other recorded information typed by users.

OS Level Auditing

An OS audit trail might capture any logon attempts, logoffs, devices used, applications executed and files opened. It may also record the success of any event (e.g. if the logon was successful or not and if access to any files were permitted or not). Events relating directly to the internals of application execution are often not captured at this level of logging. Further, this level of auditing may also capture statistics such as I/O load that is not directly security related.

Network Level Auditing

Any network connected device may record data that is relevant to this level of auditing. Devices include, but are not limited to, firewalls, routers, switches, authentication servers and web proxies. A firewall for instance, might log if a packet was rejected or accepted, protocols encapsulated, source and destination addresses.

Application Level Auditing

Application level audit trails may capture anything of interest that happens at the application level. This includes capturing the userid that runs the application, which files are opened, access to devices and so forth. As an example, consider a database being monitored — it can be useful to track who accessed which tables and any changes that have been made.

One could argue that network level auditing, mentioned above, is a special case of application level auditing. However, logging events occurring on the network is a vast area, and is therefore mentioned in its own section.

Preserving the Benefits of Audit Trails

The data in an audit trail must be reliable, its integrity and availability are key concerns. If the data in an audit trail cannot be trusted, its purpose is completely undermined. Further, it must be analyzed, in real-time or at regular intervals, or a combination of both. Confidentiality might also be of concern if sensitive information is captured.

PCI DSS Concerns

In this section, each requirement of section 10 in the PCI DSS standard will be discussed. Every requirement concern different topics mentioned above.

Req. #	PCI DSS Requirement	Context and Motivation
10.1	Establish a process for linking all access to system components (especially access done with administrative privileges such as root) to each individual user.	This requirement involves the topics of accountability and reconstruction of events. The ability to track each user's actions carefully will facilitate the forensics after a security incident. It can also be related to intrusion detection.
10.2	Implement automated audit trails for all system components to reconstruct the following events:	These requirements primarily concern what kind of events to audit. The events relate to both the operating system and application level auditing. By analyzing the events created by the system components listed, it is possible to detect suspicious activity. Audit trails from these components will also aid post-incident forensics.
10.2.1	All individual user accesses to cardholder data	
10.2.2	All actions taken by any individual with root or administrative privileges	
10.2.3	Access to all audit trails	
10.2.4	Invalid logical access attempts	
10.2.5	Use of identification and authentication mechanisms	
10.2.6	Initialization of the audit logs	
10.2.7	Creation and deletion of system-level objects.	
10.3	Record at least the following audit trail entries for all system components for each event:	This requirement involves what details to log for each event, and thus relates to all levels of auditing. Each of these relate to the events described in the previous requirement.

10.3.1 10.3.2 10.3.3 10.3.4 10.3.5 10.3.6	User identification Type of event Date and time Success or failure indication Origination of event Identity or name of affected data, system component, or resource	
10.4	Synchronize all critical system clocks and times.	In order for the audit trail to be reliable, times must be consistent. It is also important to note the time zone of each audit trail.
10.5 10.5.1 10.5.2 10.5.3 10.5.4 10.5.5	Secure audit trails so they cannot be altered. Limit viewing of audit trails to those with a job related need. Protect audit trail files from unauthorized modifications. Promptly back up audit trail files to a centralized log server or media that is difficult to alter. Write logs for external-facing technologies onto a log server on the internal LAN. Use file-integrity monitoring and change detection software on logs to ensure that existing log data cannot be changed without generating alerts (although new data being added should not cause an alert).	To maintain the reliability of an audit trail, these requirements imposes controls that should prevent an audit trail from being tampered with.
10.6	Review logs for all system components at least daily. Log reviews must include those servers that perform security functions like intrusion detection system (IDS) and authentication, authorization, and accounting protocol (AAA) servers (for example, RADIUS).	This requirement is perhaps the most important. Without any review of the audit trails, they serve no purpose.
10.7	Retain audit trail history for at least one year, with a minimum of three months immediately available for analysis (for example, online, archived, or restorable from back-up).	Since it is not uncommon for a security breach to be uncovered a while after it occurred, the audit trails must be kept for a reasonable amount of time. This requirement relates to reconstruction of events with the intent of aiding post-incident forensics to track down any involved users and affected systems.

Table A.1: PCI DSS requirements from section 10 with motivation

Having the proper events in an audit trail combined with the right detail, as described in requirement 10.2 and 10.3, is paramount in order to be able to obtain viable information from the logs.

This is only true as long as the information can be relied upon. An intruder will most likely try to cover its tracks by altering the audit records (requirement 10.5). Another way of attempting this could be to alter the system clock as mentioned in requirement 10.4. Being able to do a sound analysis of the audit trails generated at both the OS and application level will only be possible if

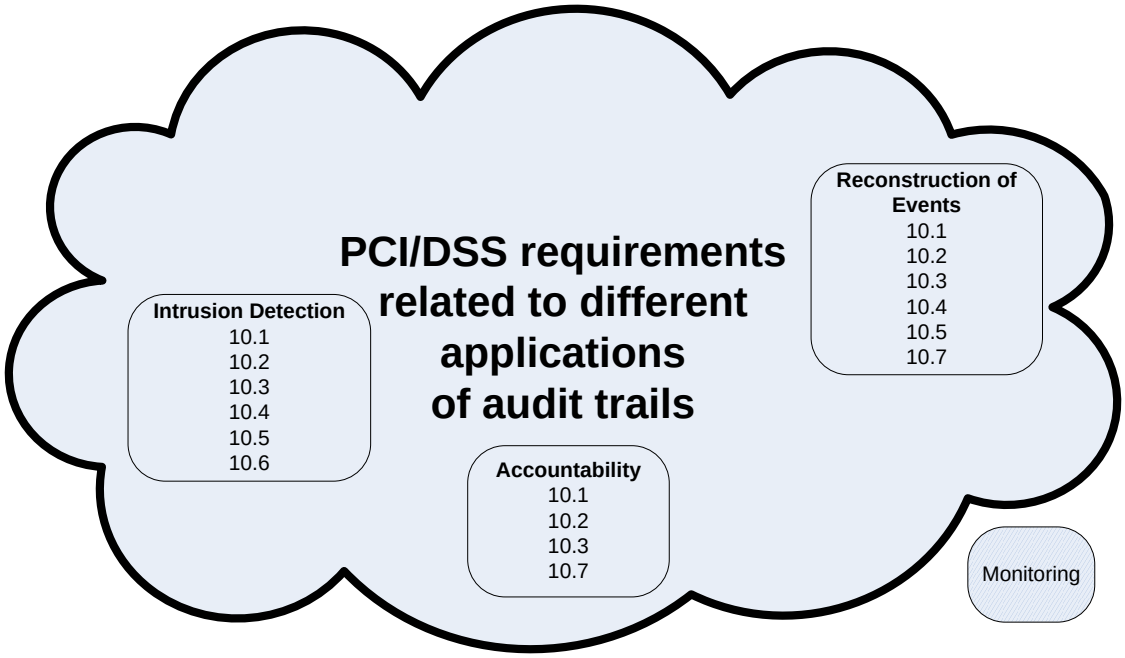

Figure A.1: PCI DSS Requirements in relation to the areas of use for audit trails

these key issues are considered.

To obtain a complete picture, all parts of the systems must be included in the analysis. That is, audit trails from the operating systems must be correlated with application logs and network trails.

Figure A.1 shows a summary of how the PCI DSS requirements relate to the different applications of audit trails.

Further Discussion

Now that the areas of use for audit trails have been established, this study will further explore those related to the PCI DSS requirements mentioned above. After key points of an area have been discussed, a few tools aimed at addressing these will be mentioned. This should in no way be treated as an exhaustive analysis of the different audit trail applications. The aim is to find out what they concern and how tools may facilitate management of these issues.

Accountability and Aggregation of Events

As mentioned previously, accountability involves the process of mapping events to actions performed by users. Accountability thus answers the question "who, what and when". By advising users that their actions are monitored can work as a control to refrain them from using resources improperly. It is however important to note that if a user appears to be using privileges in a questionable manner, someone might be masquerading as the user in question. This issue of non-repudiation can be remedied to some extent by using solid access controls, such as biometrics; it will be much harder for an intruder to coerce an employee to log on, and hand over the computer for the assailant to do whatever he/she chooses, than for the intruder to do some garbage diving in order to find a password.

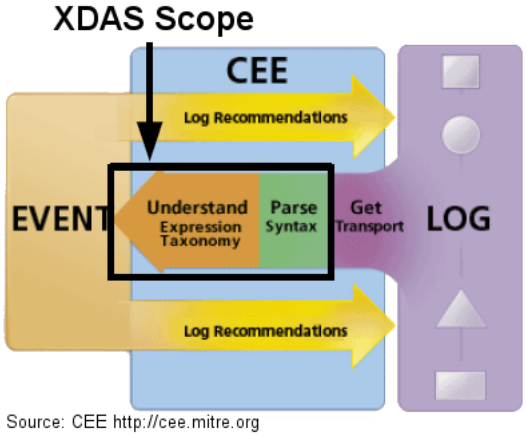

Figure A.2: XDAS in relation to CEE initiative

The issue can also be addressed by use of digital signatures, as well as keeping software up to date — to minimize exploits of vulnerable software that could allow an attacker to execute arbitrary code. This will be discussed further in the next section. It is not easy to determine, but by correlating multiple audit trails and making a sound analysis, it is possible to uncover a masquerader.

Assembling Audit Trails

There is currently no standard that describes how a log entry is to be formatted. Every operating system/syslogger and application uses its own format. This presents a problem when it comes to gathering audit trails into a uniform format for analysis. Without a uniform format, it will be very hard to do correlation between logs and make inferences on — for instance — what constitutes an attack. In an environment with a vast number of applications and different operating systems this poses a big problem for administrators trying to discern what is going on.

However, there are log management tools that remedy this issue[7]. The latest idea within the field goes under the acronym SEM[1], also known as SIM[2], or SIEM[3]. A tool like this essentially takes input from devices like firewalls, routers, switches and security applications like virus guards as well as server applications in order to normalize the data into a common format. Once normalized, the data is correlated and grouped. The tool delivers a real-time overview of the most relevant events (determined by severity) and periodic reports of activity. Its central idea is to ease the amount of data that has to be analyzed by personnel. This is done by building threads of related events and informing when unusual events occur which may need more investigation.

These tools however, are rather focused on the collection of the audit data itself and on aiding the understanding of events in an audit trail. Analysis of the audit data is left behind[1].

Understanding an event is often straight forward. Though in some cases, the meaning of an event is hard to discern without its proper context. The event has to be correlated with other events from different logs and perhaps even non-log sources such as an application's configuration to make sense. A logged event can also be hard to interpret because of its obscure description; it might simply be an error code that only makes sense to the vendor. SEMs are generally bundled with a vast set of vendor error codes that makes the interpretation of an event smoother.

A different effort aimed at solving the irregular landscape of audit trails and giving a uniform accountability mechanism is XDAS[4], an open standard. It was initially released in 1998 as a distributed services auditing standard and is now being reworked to fit today's need for a uniform interoperable event reporting format. The goal is to provide a standardized way to represent events and facilitate consolidation. In this way, events can be easily normalized and gathered. Many of

[1]Security Event Management
[2]Security Information Manager
[3]Security Information and Event Manager
[4]Distributed Audit Service

the people involved in this project are also working on the more ambitious CEE project (referred to in the introduction). XDAS could very well fit as a piece in the wider CEE approach. There is currently a java framework called xdas4j that is implemented using the current draft of XDAS.

Forensics and Reconstruction of Events

In order to facilitate the investigation of a security incident, reconstruction of events from audit trails is very important. Before incidents can be analyzed from audit trails, the integrity of any audit trails included in the analysis must be ensured (e.g. by recalculating a message digest).

As an example, let's say that an application on a corporation's public web server has been compromised, and that the attacker has been able to further penetrate the corporate network from this server. By reconstructing the chain of events that constitutes this attack from the audit trail, investigators will be able to determine the attack vector. Using the information obtained, system administrators can address the issues that made the attack possible (e.g. hardening the server by creating a chroot environment around the web server and pointing out the vulnerable application to developers).

Usually, the event reconstruction is made manually by a team of forensic investigators that propose different scenarios that may have led to the security incident. The difficulty of acquiring enough relevant information from the audit trail varies with the task at hand and of course with the auditing policy — what to audit. It is vital to be able to reconstruct the chain of events that led to an incident in order to respond with the right measures.

As mentioned in the previous section, tools such as SEMs can facilitate the analysis of intrusions and can help discerning the chain of events. There are also tools such as BackTracker and Forensix that analyze system calls on a single host to detect how an application or service has been exploited. However, according to a study[8] of these reconstruction tools, they are prone to have a very rate high false positives, so using them can be very tedious work.

Intrusion Detection

When it comes to intrusion detection, two different techniques are used to find intrusions. One of the approaches is signature based and essentially looks for a specific pattern, and if such a pattern is found, an alarm is raised. The other technique used is anomaly based and analyzes behavior in order to detect any unusual activity. As an example, consider a workstation that suddenly starts to send a huge number of emails to different recipients. From analysis of the audit records, the IDS should detect this anomaly and raise an alarm. Booth techniques are often used in synergy to detect a wider range of threats and lower the rate of false alarms. Further, any suspicious activity is logged by the IDS for later review and is in turn also often forwarded to a SEM tool for further correlation and perhaps visualization.

Modeling Intrusion Detection Techniques

In this section, a few of the modeling approaches used to design the intrusion detection techniques described above will be mentioned.

Starting off with signature detection — rules describing patterns that identify intrusions can be based on a single event, or a chain of specific events. These rules may be defined by using simple if-then implications that matches unique attributes of an attack.

A more interesting approach is by use of machine learning to learn from raised alarms and automatically create patterns that are suspicious. The problem with this technique is that training data is needed, and it is not feasible until intrusion scenarios can be fed to the model so that it can learn to pin point attacks.

The basic idea of anomaly detection is to create a baseline of what is considered normal by analyzing passed behavior. Deviations from this behavior is flagged. The are many different approaches when it comes to modeling this kind of behavior, most of which concern statistical models.

A very simple approach is threshold monitoring. It involves defining metrics for acceptable ranges of system attributes. If any of the monitored attributes (e.g. number of MAC addresses on

the network) happens to be outside the allowed range, an alarm is raised. This is indeed a very blunt instrument and is only useful for a very limited amount of attributes across the IT landscape.

Profiling the behavior of users or resource usage across the environment is a more sound approach. The normal behavior of a user is defined by analyzing the use of applications, files, network resources and devices such as printers. Detecting abnormal behavior comes down to finding peculiar use patterns. Consider for instance a user that usually logs on during office hours and use CAD applications. The user now suddenly logs on in the middle of the night, reads files that deviates from the regular ones and attaches a flash drive to the computer in order to copy files. Looking at passed history, this kind of use deviates a great deal from normal use and should raise an alarm to advice that a masquerader might be in the system.

One of the challenges with this kind of profiling is to avoid raising alarms when a legitimate user changes its pattern of normal behavior (e.g. starts to use a new application or maps a new network drive).

Classes of Intrusion Detection Systems

One often differentiates between classes of intrusion detection systems; those focused on detecting intrusions across the network and those developed to detect intrusions on the local host. They are referred to as NIDS[5] and HIDS[6] respectively. One widely used system of each class is mentioned below:

The most widely deployed NIDS is Snort, an open source project. It combines booth signature and anomaly-based detection techniques to catch intruders on the network. Thanks to the open source model of the project, it benefits greatly from its large user base that reviews and tests the functionality of the engine and rule sets.

Another open source project within the field is OSSEC, an HIDS with support for a wide range of platforms. It is designed as a client-server architecture where agents on the client machines accumulate log data from both the OS and applications. This data is sent in real-time to the server where audit trails from multiple clients are correlated and analyzed.

Conclusion

To maximize the benefit from audit trails, a well defined plan should be established regarding how audit data is to be collected, organized, analyzed and stored. Making a sound decision about what events to log, and what to capture for each event is important. Having too much information about each event might impair performance of the OS/device/application that produces the logs as well as making organization of the audit trails harder. Keeping only relevant information in the audit may be tricky, but initiatives such as CEE will ease this task.

Ideally, one should make use of the drafted XDAS framework for generation of uniform audit trails when developing new applications. Such investments will pay off when more applications eventually adopt the XDAS standard. Should a wide adoption of the standard finally pan out, it would deem much of the functionality provided by SEM applications unnecessary and could divert attention from aggregation and understanding of events to focus on correlation and analysis.

As for the analysis of the audit data, tools such as SEMs will help by providing an overview of the systems. The SEMs are however only as good as the data that they are provided with. It is important to keep security software up to date, such as intrusion detection systems and virus guards in order to get the best possible overview of what is happening in the IT environment.

Finally, the integrity and storage of audit trails over time must also be well defined. Write-once storage solutions and message digests can be utilized to ensure integrity. If the audit trails must be kept for longer periods of time, the right type of storage media and backups are also of concern.

[5]Network Intrusion Detection System
[6]Host Intrusion Detection System

Bibliography

[1] XDAS standard aims to empower it audit trails from across complex events. `http://interarbor.libsyn.com/index.php?post_id=521463`, 2009. BriefingsDirect podcast.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, October 2004.

[3] The CEE Board. *Common Event Expression*, June 2008. The MITRE Corporation.

[4] Jeramiah Bowling. Alienvault: the future of security information management. *Linux Journal*, issue 191, March 2010. `http://www.linuxjournal.com/magazine/alienvault-future-security-information-management`.

[5] Gabriela F. Cretu, Angelos Stavrou, Michael E. Locasto, Salvatore J. Stolfo, and Angelos D. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *IEEE Symposium on Security and Privacy*, pages 81–94, May 2008.

[6] Dorothy E. Denning. An intrusion-detection model. In *IEEE Transactions on Software Engineering*, volume 13, no. 2, pages 222–232, February 1987.

[7] Nick Hutton. *Preparing for Security Event Management*. 360 Information Security Ltd., 2007. White paper.

[8] Sundararaman Jeyaraman and Mikhail J. Atallah. An empirical study of automatic event reconstruction systems. In *Digital Investigation*, volume 3, pages 108–115. Elsevier, 2006.

[9] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. In *ACM Transactions on Information and System Security*, volume 3, No. 4, pages 262–294, New York, NY, USA, November 2000. ACM.

[10] Robert McMillan. Most retailer breaches are not disclosed, Gartner says. *CIO*, May 2008. `http://www.cio.com/article/367763/Most_Retailer_Breaches_Are_Not_Disclosed_Gartner_Says`.

[11] J. Susan Milton and Jesse C. Arnold. *Introduction to Probability and Statistics*. McGraw-Hill, fourth edition, 2003.

[12] National Institute of Standards and Technology. *An introduction to Computer Security: The NIST Handbook*, 1995.

[13] PCI Security Standards Council. *Navigating PCI DSS*, October 2008. Version 1.2.

[14] Jaikumar Vijayan. Heartland data breach could be bigger than TJX's. *Computerworld*, January 2009. `http://www.computerworld.com/s/article/9126379/Heartland_data_breach_could_be_bigger_than_TJX_s`.

[15] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, May 1999.

[16] Nong Ye and Qiang Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. In *Quality and Reliability Engineering International*, volume 17, pages 105–112, 2001.

[17] Nong Ye, Syed Masum Emran, Qiang Chen, and Sean Vilbert. Multivaraite statistical analysis of audit trails for host-based intrusion detection. In *IEEE Transactions on Computers*, volume 51, No. 7, pages 810–820, 2002.