Intrusion Detection in a Computer System:

Improving Gassata and EigenProfiles to Intrusion Detection

Yacine Bouzida and Sylvain Gombault

{Yacine.Bouzida, Sylvain.Gombault}@enst-bretagne.fr

Département RSM
GET/ENST BRETAGNE (Rennes)
02, RUE DE LA CHATAIGNERAIE
CS 17607
F-35576, CESSON SEVIGNE CEDEX

June 2002

Technical Report 1

Introduction

Despite the undeniable progress in the area of computer security that has taken place over the past two decades, there is still much to be done to improve security of today's computer systems.

In fact, during the past few years, significant progress has been made toward the improvement of computer system security. Unfortunately, the undeniable reality remains that all computers and computer networks are vulnerable to compromise. These systems are vulnerable to attacks from both non-authorized users (outsiders attacks) as well as attacks from authorized users who abuse their privileges (insider attacks). As a result to this situation, the need for user accountability is very important, both as a deterrent and for terminating abusive computer usage once it is discovered.

The security infrastructure provides several security services such as privacy, integrity, availability and authentication.

Confidentiality: consists of ensuring only authorized users can read or copy a given file or object,

Control: only authorized users can decide when to allow access to information,

Integrity: only authorized users can alter or delete a given file or object,

availability: the computer system remains working without degradation of access and provides resources to authorized users when they need it,

utility: fitness for a specified purpose.

There are three kinds of computer security applications: access control, system restoration and intrusion detection. The first ensures that the user is the one he claims to be and that he is a registered person. It mainly deals with cryptography and access control. The second consists of restoring the system to a working state after damages (physical or logical) have occurred. Both fields have been largely studied and have machine implementation on almost all operating systems. The third is somewhat new. In fact, research on intrusion detection started in early 1980's.

What is Intrusion detection?

An intrusion detection is defined by Heady et al. [1] as :*"any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource"*. The field of intrusion detection is currently some 21 years old. The seminal paper that is most often cited is James P. Anderson's technical report [2], where the author divides the possible attackers of a computer system into four groups:

- *External penetrator* The external penetrator has gained access to a computer for which he is not a legitimate user. Anderson uses this definition to include users that are, e.g. employees of some organizations, where they have physical access to the building that houses computer resources, even though they are not authorized to use them.

- *Masquerader* The masquerader —who can be both an external penetrator and an authorized user of the system— is a user who, having gained access to the system, attempts to use the authentication information of another user, becoming in effect him as far as the computer system is concerned. This is an interesting case since there is no direct way of differentiating between the legitimate user and the masquerader.

- *Misfeasor* The legitimate user can operate as a misfeasor, that is, although he has legitimate access to privileged information, he abuses this privilege to violate the security policy of the installation.

• *Clandestine user* The clandestine user operates at a level below the normal auditing mechanisms, perhaps by accessing the system with supervisory privileges. Since there is little, if any, evidence of this type of intrusive activity, this class of penetrator is difficult to detect.

Anderson [2] uses the term *"threat"* in this same sense and defines it to be the potential possibility of a deliberate unauthorized attempt to

- Access information,
- Manipulate information, or
- Render a system unreliable or unusable .

An intrusion is a violation of the security policy of the system. However, any definition of intrusion is, of necessity, imprecise, as security policy requirements do not always translate into a well-defined set of actions.

Intrusion detection systems help computer systems prepare for and deal with attacks. They collect information from a variety of vantage points within computer systems and networks, and analyze this information for symptoms of security problems.

Detecting intrusions can be divided into two categories: *anomaly intrusion detection* and *misuse intrusion detection*.

The former refers to intrusion that can be detected via anomalous behavior and use of computer resources. For example, if user *X* only uses the computer from his office from 9 AM to 5 PM, an activity on his account late in the night is therefore suspicious and might be an intrusion. Another user might use only text processing applications. A compiler use during his session might be considered as abnormal. Anomaly detection attempts to quantify the usual or acceptable behavior and flags other irregular behavior as potentially intrusive.

In contrast, misuse intrusion detection refers to intrusions that follow well-defined attack scenarios that exploit weaknesses in system and application software. Such patterns can be precisely written in advance. Therefore, from this prior knowledge about bad or unacceptable behavior, this technique seeks to detect it directly, as opposed to anomaly intrusion detection, which seeks to detect the complement of normal behavior.

Denning [3] presented another intrusion detection classification scheme based on intrusion types that are:

Attempted break-in: someone attempting to break into a system might generate an abnormally high rate of password failures with respect to a single account on the system as a whole.

Masquerading or successful break-in: someone logging into a system through an unauthorized account and password might have a different login time, location, or connection type from that of the account's legitimate user. In addition, the penetrator's behavior might differ considerably from that of the legitimate user; in particular, he might spend most of his time browsing through directories and executing system status commands, whereas the legitimate user might concentrate on editing or compiling and linking programs.

Leakage by legitimate user: A user trying to leak sensitive documents might log into the system at unusual times or route data to remote printers not normally used.

Denial of service: Often detected by atypical usage of system resources; an intruder able to monopolize a resource (e.g. network) might have abnormally high activity with respect to the resource, while activity for all other users is abnormally low.

Malicious use: Often detected by atypical behavior profiles, violations of security constraints, or use of special privileges.

Penetration of the security control system: usually detected by monitoring for specific patterns of activity.

This classification provides a grouping of intrusions based on the end effect and the method carrying out the intrusions. Irrespective on how intrusions are classified, the main techniques for detecting them are the same: the statistical approach of anomaly detection, and the precise monitoring of well-known attacks in the misuse detection approach. Both approaches make implicit assumptions about the nature of intrusions that can be detected by them.

• Report Statement and Outline:

This Report contributes to both ameliorating ***GASSATA*** "*Genetic Algorithms as Simplified Security Audit Trail Analysis*" and the possibility of using the Principal Component Analysis in anomaly intrusion detection.

- **Improving *GASSATA*:** GASSATA, which is an intrusion detection system based on genetic algorithms, presents many shortcomings such as its time consuming, its poor formalization which does not permit the detection of all actual attacks performed by an intruder and other drawbacks related to GASSATA described in chapter 3. The first part of this report introduces a new formalization which is used not only to overcome the different encountered problems with the current formalization of GASSATA but also permits to find all solutions provided by GASSATA in real-time.

- **Using PCA in anomaly intrusion detection:** we have imagined to apply the principal component analysis in anomaly detection so that a novel method may be added to the existing techniques in this intrusion detection kind. However, our experiment consist of considering only some behaviors' simulations. Experiments on a real network with real users shall be realized to demonstrate the robustness of this new method.

This report is organized as follows. Chapter 1 introduces some definitions of the audit trail and examines the different rules of the security audit trail analysis, and reviews some representative research efforts. Chapter 2 gives a brief overview of Genetic Algorithms, discusses the different operators of GAs and introduces the schema theorem. Chapter 3 examines the different formalizations of the different security audit trail problem introduced by Mé [18] and chapter 4 presents our formalization which brings many interesting ideas that resolve somehow the GASSATA's problems. Chapter 5 describes experiments in using our new formalization to solve the security audit trail problem and a comparison between this new formalization and GASSATA. Chapter 6, which represents the second part of this report, introduces a novel method in anomaly detection based on principal components analysis.
Chapter 7 summarizes the report and outlines ideas for future work.

Chapter 1

Security Audit Trail

This chapter focuses on examination of the security audit trail. We first introduce some terminology concerning the three activities of the security audit: The specification of the system activities to be audited, the audit events collection and the audit trail analysis. We then examine the different visions of the *"security bibles"* (the Orange Book (TCSEC), the White Book (ITSEC), the ECMA TR 46 and the ISO 7498-4). The two approaches to intrusion detection and their evaluation are described in the last paragraph.

1.1 Activities of the audit trail

Each operation performed on the target system is transformed into actions sequences. These actions are called system activities. An activity which holds in a specific time is called *"event"*. An audit trail is a time ordered sequence of actions that are audited on the target system. Each audit trail consists of one or more audit files.

The audit trail should answer the following questions:

- ***Which action has taken place?*** Operation performed by the subject on or with the subject, e.g., login, logout, read, execute.
- ***Who is the initiator of the action?*** Who is the subject? A subject is typically a terminal user, but might also be a process acting on behalf of users or groups of users, or might be the system itself.
- ***Who are the receptors of actions?*** It may include such entities as files, messages, terminals, printers, programs.
- ***When the action had been performed?*** The Unique time/date stamp identifying when the action took place.
- ***Where did the action take place?*** The server address if it is a distant one.
- ***In the failure case, why the operation has failed?***

To answer these question, the site security officer has to specify the system activities to be audited, insure the audit collection in the audit trail and regularly analyze this file. In addition, he has to restore the system to a working state after the damages have occurred.

1.1.1 Specification of the system activities to be audited

We list below some pertinent information to insure the target system security. The SSO may define the events to be audited according to the security level wanted to be reached.

System access information

It consists of determining the information which will constitute an investigation on an eventual security violation:

- Who has penetrated the system (user or process identifier),
- When (unique time/date of the system access),
- Where (terminal identifier, address),
- How (accessing mode: interactive, local batch, distant connection).

System usage information

It consists of determining which resources are used and how:

- Commands set –The commands used,
- CPU usage,
- I/O resources access,
- Memory occupation rate of a user.

File usage information

It is a sensitive point because these files include the information. For each file operation, we will be interested in:

- Time/date of the file access,
- The file mode access (opening, closing, reading, records adding, records modification,…),
- Access source (triplet (user, terminal, application)),
- Information volume exchanged during the system access.

Relative information to each application

Each application may influence the security of the target system. We can record the following events:

- Application running and halting,
- The actually executed modules,
- The commands executed and their results (success or failing),
- The input data,
- The outputs.

Information about the possible security violations

It consists of recording all the events which may attempt to access an unauthorized resource defined by the security policy of the target system. Among these events:

- attempting to execute an application in a privileged mode (e.g. rights access modification under UNIX),
- attempting to access an unauthorized file or introducing a wrong password for this access,
- attempting to use some system commands which are reserved for privileged users,
- access rights altering to some sensitive files,
-information on the system state.

We may have some information about the system performance by using the following indicators:
- Abnormal level of the denied system access.
- Abnormal level (high or low) of some system commands usage.

1.1.1 Audit collection

Audit data, from which to make intrusion detection decision, must be collected. Many different parts of the monitored system can be used as sources of data, keyboard input, command based logs, application based logs etc. However, typically, network activity, or host based security logs (or both) are used.

1.1.2 Audit trail analysis

The main goals of this analysis are: detecting every violation of the security policy, identifying the responsible of this violation, restoring the possible damages and introducing new changes to insure the system security .
Debar et al. [4] used a number of concepts to classify intrusion-detection systems, as presented in figure 1.1.

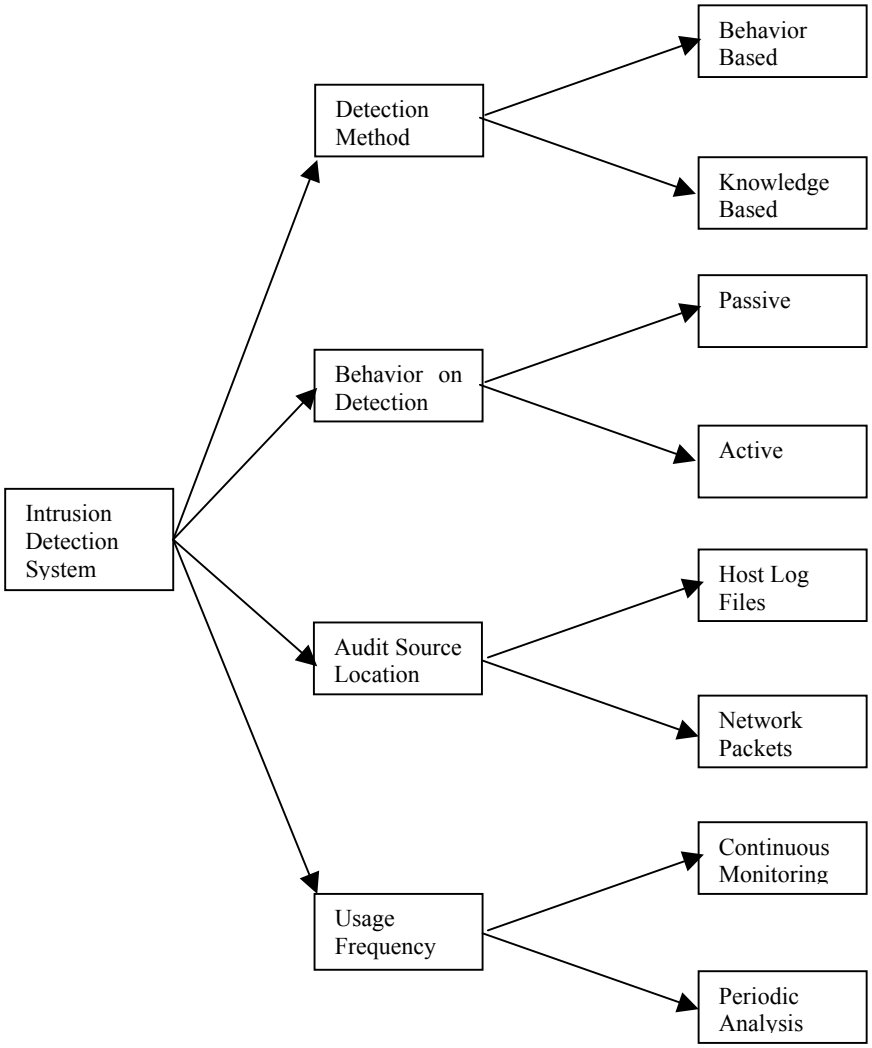


Figure 1.1.: Characteristics of intrusion-detection systems (from [4]).

The detection method describes the characteristics of the analyzer. When the intrusion-detection system uses information about the normal behavior of the system it monitors, it is qualified as behavior based.

When the intrusion-detection system uses information about the attacks, it is qualified as knowledge-based.

Behavior on detection describes the response of the intrusion detection system to attacks. When it actively reacts to the attack by taking either corrective (closing holes) or proactive (logging out possible attackers, closing down services) action, then the intrusion detection-system is said to be active. If the intrusion-detection system merely generates alarms, it is said to be passive.

The audit source location distinguishes among intrusion detection systems on the kind of the input information they analyze.

Usage frequency is an orthogonal concept. Certain intrusion-detection systems have real time continuous monitoring capabilities, whereas others must be run periodically.

Audit trail protection

The audit trail integrity and confidentiality is very important; a malicious user may not modify or erase the audit traces. The audit data shall be protected so that read access to it is limited to those who are authorized for audit data. Hence, an intrusion detection system has to perform and propagate its analysis as quickly as possible to enable the security officer to react before many damages have been done.

1.2 The security bibles and intrusion detection

We present in this paragraph the different visions to intrusion detection of the Orange Book (TCSEC) [5], the White Book (ITSEC) [6], the reports ECMA TR 46 [7] and 138 [8] and the ISO 7498-4 standard.

The Orange Book

The audit trail appears in the *Controlled Access Protection (Class C2)* where the *TCB (Trusted Computing Base)* shall be able to create, maintain and protect from modification or unauthorized user access or destruction an audit trail of accesses to the objects it protects. The TCB shall be able to record the following types of events:

- use of identification and authentication mechanisms,
- introduction of objects into user's space address (e.g. file open),
- deletion of objects,
- actions taken by computer operators and system administrators and/or system security officers and other security relevant events.

For each recorded event, the audit record shall identify:
- date and time of the event,
- user,
- type of event,
- success or failure of the event,
- origin of request (e.g., terminal ID) for identification/authentication events.
- name of the objects for events that introduce an object into a user's address and for object deletion events.

The security officer shall be able to selectively audit the actions of any one or more users based on individual identity.

The Labeled Security Protection (Class B1) which requires all the feature required for class (C2). In addition, an informal statement of the security policy of the model, data labeling and mandatory access control over named subjects and objects must be present. Thus, for events that introduce an object into a user's address space and for object deletion events, the audit record shall include the name of the object and the object's security level. The system administrator shall also be able to selectively audit the actions of any one or more users based on object security level. The TCB shall be able to audit any override of human readable output markings.

In the Structured Protection (Class B2), the TCB shall be able to audit the identified events that may be used in the exploitation of covert storage channels.

The Security Domain (Class B3) introduces the definition of a mechanism that is able to monitor the occurrence or accumulation of security auditable events that may indicate an imminent violation of security policy. This mechanism shall be able to immediately notify the security administrator when thresholds are exceeded, and if the occurrence or accumulation of these security relevant events continues, the system shall take the least disruptive action to terminate the event.

The audit requirements of the *Verified Design (Class A1)* are the same as those required in class *B2*.

The White Book

The White Book [6] provides less requirements about the audit trail than the Orange Book. It simply specifies the necessity to define the functions allowing the collection, protection and trace analysis. An *"on line "* analysis, permitting possible violations of the security policy before they occur, is proposed but is not mandatory as it is defined in Class B3 of the Orange Book.

ECMA TR/46 and ECMA 138

The audit trail is considered, by the ECMA, as a generic component of the computer system security. The events to record shall be specified, the recorded information should be protected and an information analysis has to be performed to obtain the security reports.
The analysis is done after the definition of the security criteria.
To obtain the report of all the computer system, information collection coming from all the sites of the system is necessary. This collection should be performed safely. The audited objects should not be known by the system users but the security officer. However, a user may have the audit of his own objects.

The ISO 7498-4 Standard

The security is one of the five functional fields of the network administration. However, it is not well developed.

1.3 Approaches to intrusion detection

A survey of the research in the field of computer and network intrusion detection is beyond the scope of this report because there are currently over seventy intrusion detection systems. However, we try, in this chapter, to describe approaches by reporting example systems that are based on these approaches.

There are two approaches, as described in the introduction, to intrusion detection: anomaly detection and misuse detection. An hybrid intrusion detection which uses these two approaches at the same time may also be considered as a third approach to intrusion detection systems.

1.3.1 Anomaly detection approach (Profiles)

This approach consists of establishing normal behavior profile for user and system activity and observing significant deviations of actual user activity with respect to the established habitual pattern. Significant deviations are flagged as anomalous and should raise suspicion. Ideally, the daily use of a computer system by a specific user follows a recognizable pattern, which can serve as a characterization of the user identity. The set of application programs, commands and utilities invoked by a user is often determined by the nature of the job assignment for that user.

For instance, a secretary is expected to use a document processing program, e-mail, or calendar management applications. By contrast, a programmer almost always restricts himself to editing source files, compiling and testing programs and so forth. Clearly, the fact that a compiler is suddenly invoked by a secretary should be considered as highly suspicious and hence should be immediately brought to the attention of the security officer who may conclude that an intruder is probably masquerading as a secretary. Moreover, even though a group of users may have the same job assignments, it is still possible to distinguish among their profiles based on their personal habits, such as their session location, normal working hours, or the set of commands they frequently use.

1.3.1.1 DENNING intrusion detection model

This model [3] has six main components:

• ***Subjects:*** they are the initiators of the actions in the target system. A subject is typically a terminal user, but might also be a process acting on behalf of users or group of users, or might be the system itself. All activity arises through commands initiated by subjects. Subjects may be grouped into different classes (e.g. , user groups) for the purpose of controlling access to objects in the system. Users may overlap.

• ***Objects:*** they are the receptors of actions and typically include such entities as files, programs, messages, records, terminals and user- or program- created structures. When subjects can be recipients of actions (e.g., electronic mail), then those subjects are also considered to be objects of the model. Objects are grouped into classes by type (program, text file, etc.). additional structure may also be imposed, e.g., records may be grouped into files or database relations; files may be grouped into directories. Different environments may require different object granularity; e.g., for some database applications, granularity at the record level may be desired, whereas for most applications, granularity at the file or directory level may suffice.

• ***Audit records:*** generated by the target system in response to actions performed or attempted by subjects on objects. They are 6-tuples representing actions performed by subjects on objects: <Subject, Action, Object, Exception-Condition, Resource-Usage, Time-Stamp>

- Action: operation performed by the subject on or with the object, e.g., login, logout, read, execute.

- Exception-Condition: denotes which, if any, exception condition is raised on the return. This should be at the actual exception condition raised by the system, not just the apparent exception condition returned to the subject.

- Resource-Usage: list of quantitative elements, where each element gives the amount used of some resource, e.g., number of lines or pages printed, number of records read or written, CPU time or I/O units used, session elapsed time.

- Time-Stamp: unique time/date stamp identifying when the action took place.

Profiles: an activity profile characterizes the behavior of a given subject (or set of subjects) with respect to a given object (or set thereof). Thereby as a signature or description of normal activity for its respective subject(s) and object(s). Observed behavior is characterized by terms of a statistical metric and model. A metric is a random variable x representing a quantitative measure accumulated

over a period. The period may be a fixed interval of time (minute, hour ,day, week, etc.), or the time between two audit related events (i.e., between login and logout, program initiation and program termination, file open and file close, etc.). observations (sample points) x_i of x obtained from the audit records are used together with a statistical model to determine whether a new observation is abnormal. The statistical model makes no assumptions about the underlying distribution of x; all knowledge about x is obtained from observations.
Here are some possible variables which may characterize a profile:
- number of password failures during a minute,
- CPU time consumed by a program as an interval timer that runs between program initiation and termination,
- number of times some command is executed during a login session,
- number of logins during an hour, … etc.

• ***Anomaly records:*** these records are generated whenever abnormal behavior is detected. Each anomaly record contains three components: event, time-stamp and profile.

• ***Activity rules:*** An activity rule specifies an action to be taken when an audit record or anomaly record is generated, or a time period ends. They have the common "*condition-action*" form. An example is with password failures, where the security officer should be notified immediately of a possible break-in attempt if the number of password failures on the system during some interval of time is abnormal.

The purpose of a statistical model, from n observation x_1, x_2,…, x_n for a random variable x, is to determine whether a new observation x_{n+1} is abnormal with respect to previous observations. Reference [3] proposes many models:

Operational model: this model is based on the operational assumption that abnormality can be decided by comparing a new observation of x against fixed limits. Although the previous sample points for x are not used, presumably the limits are determined from prior observations of the same type of variable. The operational model is most applicable to metrics where experience has shown that certain values are frequently linked with intrusions. An example is an event counter for the number of password failures during a brief period, where more than 10, say, suggests an attempted break-in.

Mean and Standard Deviation Model: This model is based on the assumption that all we know about $x_1,x_2,\ldots,x_n$ are mean and standard deviation as determined from its first two moments:

$$sum=x_1+\ldots+x_n$$

$$sumsquares=x_1^2+\ldots+x_n^2$$

$$mean=\frac{sum}{n}$$

$$stdev=\sqrt{\frac{sumsqures}{n-1}-mean^2}$$

The new observation x_{n+1} is defined to be abnormal if it falls outside a *confidences interval* that is d standard deviations from the mean for some parameter d:

$$mean\pm d\times stdev$$

This model is applicable to event counters, interval timers, and resource measures accumulated over a fixed time interval or between two related events. It has two advantages over an operational model.

First, it requires no prior knowledge about normal activity in order to set limits; instead, it learns what constitutes normal activity from its observations, and the confidence intervals automatically reflect this increased knowledge. Second, because the confidence intervals depend on observed data, what is considered to be normal for one user can be considerably abnormal for another.
A slight variation on the mean and standard deviation model is to weight the computations, with greater weights placed on more recent values.

Multivariate Model

This model is similar to the mean and standard deviation model except that it is based on correlation among two or more metrics. This model would be useful if experimental data show that better discriminating power can be obtained from combinations of related measures rather than individually- e.g., CPU time and I/O units used by a program, login frequency, and session elapsed time.

Markov Process Model

This model, which applies only to event counters, regards each distinct type of event (audit record) as a state variable, and uses a state transition matrix to characterize the transition frequencies between states (rather than just the frequencies of individual states -i.e., audit records- taken separately). A new observation is defined to be abnormal if its probability as determined by the previous state and the transition matrix is too low. This model might be useful for looking at transitions between certain commands where command sequences were important.

Time Series Model

This model, which uses an interval timer together with an event counter or resource measure, takes into account the order and inter-arrival times of the observations x_1, x_2,…, x_n as well as their values. A new observation is abnormal if its probability of occurring at that time is too low. A time series has the advantage of measuring trends of behavior over time and detecting gradual but significant shifts un behavior, but the disadvantage of being more costly than mean and standard deviation.

1.3.1.2 HYPERVIEW – A neural network component for intrusion detection

Introduction

Hyperview [9] is a system with two major components. The first component is an *"ordinary"* expert system that monitors audit trails for signs of intrusions known to the security community, the other is a neural network based component that adaptively learns the behavior of a user and raises an alarm when the audit trail deviates from this already *"learned"* behavior.

The designers of the system notes that the audit trail could emanate from a number of different sources, with different levels of detail. For instance; the keyboard level –the system observes every keystroke made by the user, the command level– the system records every command issued by a user, the session level– the system aggregates several commands issued from the time of login to the system to the time of logout, and finally, group level –where several users are grouped together and treated as a class of known users.

The authors then note that the more detailed data made available to the intrusion detection system, the better the chance of the system being able to correctly raise an alarm. However, the more data presented to the system the more problematic storage and processing becomes. The most aggregated level of data will not put such a strain on the intrusion detection system. For the purpose of Hyperview, the authors decided to provide the system with an audit trail on the command level.

Hypotheses about user behavior and the audit trail

The decision to attempt to employ a neural network for the statistical anomaly detection function of the system stems from a number of hypotheses about what the audit trail will contain. The fundamental hypothesis is that the audit trail constitutes a multivariate time series, where the user constitutes a dynamic process that emits a sequentially ordered series of events. The audit record that represent such an event consists of variables of two types; one, the values of which can be chosen from a finite set of values –for instance the name of terminal the command was issued on– the other, a continuous value, for instance CPU usage or some such.
The more detailed hypotheses that follow from the fundamental hypothesis are:

1. The user submits commands to accomplish a given task. These commands will be consistent over time, as the user requires preferences vis-à-vis which way the task should be performed. Among tasks, the actions of the user will be less predictable, or even unpredictable. Thus, we will observe patterns of usage in the audit trail, as quasi-stationary sequences, interspersed with periods of non-stationary activity.

2. The preferred behavior of the user follows a stochastic law, the audit trail belonging to which, is a projection of this law onto the variables of the audit record in question. The audit trail can thus be viewed as a set of samples of the quasi-stationary process. The authors note that it is difficult to express a law from asset of samples, even when the underlying process is quasi-stationary. This law will instead be treated as a black box, and it will be approximated by the neural network, without ever having been made explicit.

3. There are correlations between the various measures contained in the audit record. This is a common sense hypothesis, since there would for instance -always by necessity- be an effect on, for instance, cache hit ratio, with increased CPU usage. Since the authors do not make the parameters of the user model explicit they cannot express these correlations. The proposed neural network component must be able to take advantage of these correlations during the learning process.

The neural network component

The authors proposed a then untested approach of mapping the time series to the inputs of the neural network. At the time, the researched approach was to map N inputs to a window of time series data, shifting the window by one between evaluations of the network. The authors acknowledged that this would make for a simple model, easily trained. However, there would be a number of problems with this method:

- N is completely static, if the value of N were to change, a complete retraining of the network would be required.

- If N was not adequately chosen, the performance of the system would be dramatically reduced. Too low a value of N, and the prediction would lack accuracy because of a lack of older relevant information, too high a value of N and the prediction would be perturbed by irrelevant information.

- During the quasi-stationary periods of the usage, a large value of N would be preferred, to encompass this quasi-stationary process. During the transition periods, on the other hand, where the older data has no meaning, we would like as small a value of N as possible, to eliminate this irrelevant data quickly.

The authors then go on to state the correlations between input patterns are not taken into account with this model, since these types of networks learn to recognize fixed patterns in the input and nothing else. Other disadvantages are that they are slow to converge and the adaptability is low since partial retraining can lead to a network that forgets everything it has learned before.

Instead the designers of Hyperview choose to employ a recurrent network, where part of the output network is connected to the input network, as input for the next stage. This creates an internal memory in the network. Between evaluations, the time series data is fed to the network one datum at a time, instead of a shifting time window, the object of the latter being the same, namely to provide the network with a perception of the past. It is interesting to note that the recurrent network has long term memory about the parameters of the process in the form of the weights of the connections in the network, and short term memory about the sequence under study, in form of the activation of the neurons. These kinds of networks were at the time of the design much less studied than the recurrent ones.

System implementation

The design of the system as a whole is a complex and interesting one. The authors connected the artificial neural network to two experts systems. One monitors the operation, the training of the network -to prevent the network from learning anomalous behavior for instance- and evaluates its output. The other expert system scans the audit trail for known patterns of abuse, and together with the output from the first expert system (and hence from the artificial neural network) forms an opinion about whether to raise an alarm or not. The decision expert system also provides the artificial neural network with *"situation awareness"* data –data that audit trail itself does not contain- from the simple *"current time and date",* to the complex *"state of alert, or state of danger for the system"* defined by the site security officer. (See figure 1.2).

It becomes clear from the system graph that artificial neural network component of the system could be viewed as a filter of the audit record stream before it is presented to the decision expert system. This is perhaps not surprising, since artificial neural networks are often put to this use. The division of labor presented here has -according to the authors- the advantage that the numeric evaluation of the artificial neural network is an efficient process, that does not consume a lot of resources in terms of processing power, while the more intensive data processing done by the decision expert system is concentrated on a much reduced set of the audit trail. This could lead to the detection of intrusions in real time.

Experimental results

The designers put the neural network component of the system –the only part that was fully functional at the time of publication– to the test by feeding it an audit trail submitted by an anonymous user on a SUN3 UNIX work station. They used the accounting files as the source of the audit data, where each record contains the name of the command, the amount of CPU and core memory used, and the number of input/output performed. The beginning, and end, for each session was discernible from the audit trail.

The first experiment considered the input as an endless continuous sequential stream of events. The artificial neural network was given each audit record sequentially, and asked to predict the next command in the sequence. When the next one was presented the network was retained to reflect the new discovery. The commands, of which there *60* different given in the audit trail, where mapped onto one output neuron each, the optimal result being *1* neuron with a numeric value of *1.0*, and the other neurons with a value of *0.0*. Three important parameters define the success of the network's performance:

Confidence

The maximum activation is numerically large, and there exists a convincing difference to the second highest activation. If the prediction is correct, this is an ideal state of affairs, and very troublesome one if the prediction is, in fact, false. Then the network is overconfident in its ability to predict the correct behavior.

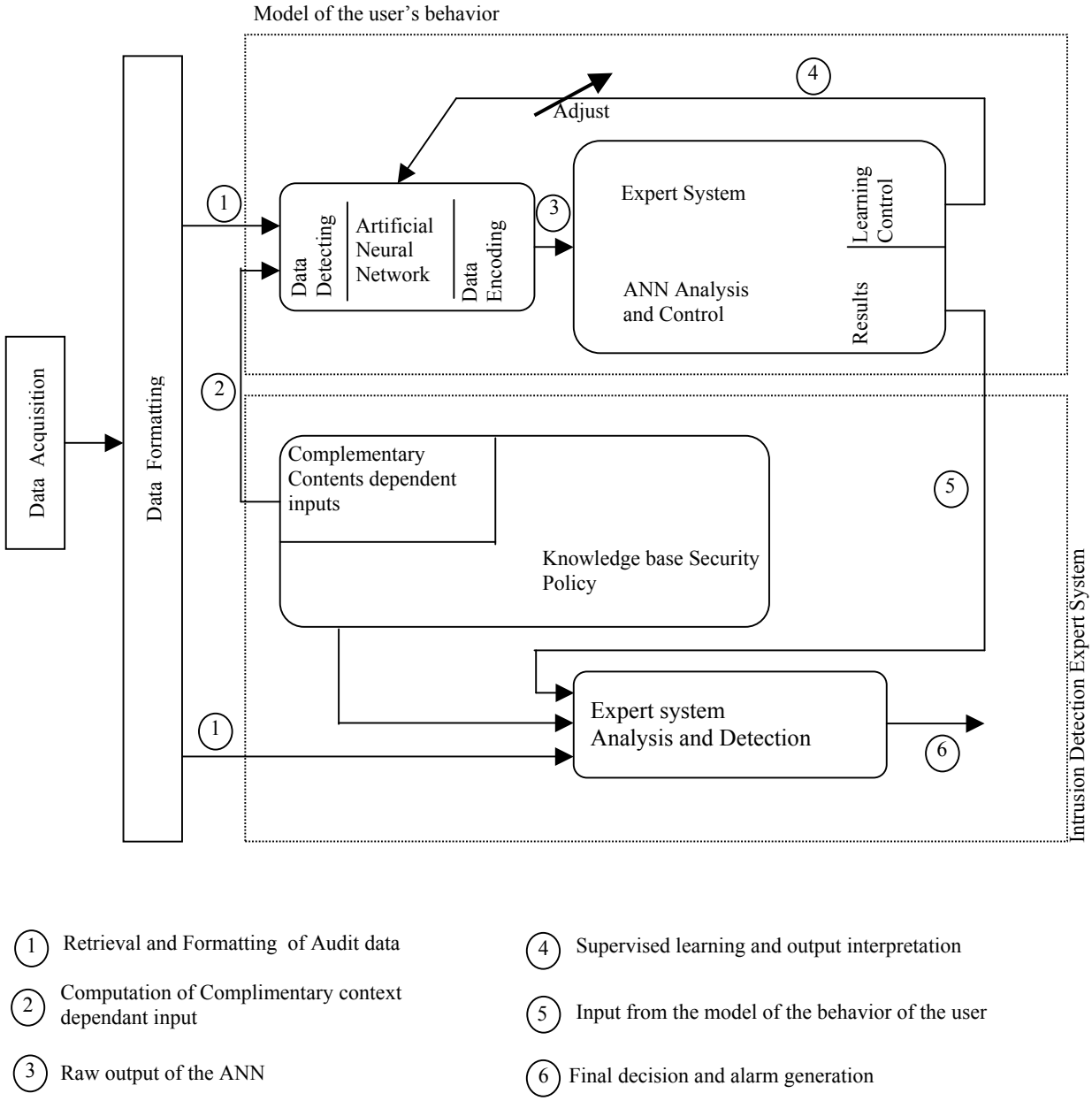


Figure 1.2.: Block diagram of the Hyperview system (from [9]).

Uncertainty

The largest activation is very low. The output of the neurons are in the same range, the network cannot discriminate from what it knows, to propose the next command. This is either from a lack of example, i.e. this time series has not been seen before, or from an overabundance of choices the time series could mean one of possibly many things.

Conflict

The largest activation is somewhere in the middle, and the difference to the second largest is too small. That means that either of the commands could be considered likely candidates, and the output of the network is only an indication of which is the more likely.

Debar [9] observed a sequence of 6550 commands, trained the network on half of that sequence, and then fed the network with the entire sequence. The results looked quite promising. Correctly predicted commands had a high degree of confidence and the farther away from the correct prediction the output of the artificial neural network was, the lower the confidence.

When looking closer at the results it became evident that some types of commands were often predicted in error, for instance the **date** command, that displays the current time and date. The network had learned however to classify this as an irrelevant command, not worth considering for inclusion in the user profile. Such commands could be characterized as noise in the deterministic sequence.

Other commands, such as those issued when dealing with a prototype of a database system (that crashed often, and at random intervals), where marked as very indicative of the usage of that particular user. The network also managed to automatically associate commands with similar actions, such as *sh*, and *csh*, often predicting a *sh* for *csh* and vice versa. The authors left it to the neural network control expert system to decide that "errors" like these were in fact not indicative of a security violation, but instead a more benign kind.

1.3.2 Misuse detection approach

In contrast, misuse intrusion detection refers to intrusions that follow well defined attack patterns that exploit weaknesses in system and application software. Such patterns can be precisely written in advance. Therefore, from this prior knowledge about bad or unacceptable behavior, this technique seeks to detect it directly, as opposed to anomaly intrusion detection, which seeks to detect the complement of normal behavior.

1.3.2.1 USTAT-Unix State Transition Analysis

USTAT [10, 11] is a mature prototype implementation of the state transition approach to intrusion detection. State transition analysis takes the view that the computer is initially in some safe state, and after a sequence of actions performed by the attacker leads the target system to a compromised state. USTAT reads specifications of the state transitions necessary to complete an intrusion, supplied by the SSO, and then evaluates an audit trail with respect to these specifications.

An example of a penetration scenario

Table 1.1 presents an example penetration scenario for 4.2 BSD UNIX that can be used illegally with which an attacker gains administrative privileges.

Step	Command	Comment
1.	**%cp /bin/csh /usr/spool/mail/root**	- assumes no root mail file
2.	**%chmod 4755 /usr/spool/mail/root**	- make setuid file
3.	**%touch x**	- create empty file
4.	**%mail root < x**	- mail root empty file
5.	**%/usr/spool/mail/root**	- execute setuid-to-root shell
6.	**Root%**	

Table 1.1.: Penetration scenario (from [10]).

In this scenario, the attacker exploits a flaw in the mail utility, in which ***mail*** fails to reset the setuid bit of the file to it appends a message and changes the owner. The attacker has thus gained administrative (or root/super-user) privileges.
Model this penetration as a sequence of state transitions, we must first define the initial requirement state and the compromised (goal) state.

To execute the first step over, *root's* mail file must not exist, or must be writable/ As we progress through the steps in the example, we find that also; the intruder must have write access to the mail directory, he must also have executed access to *cp*, *chmod*, *touch* and *mail*. Note that the nature of the penetration in this case is not the execution of the setuid-shell per se. Even if the intruder chose not to execute the command interpreter, there would still be a violation in that there now exists an executable setuid-to-root file on the system that the super-user (*root*) did not create.

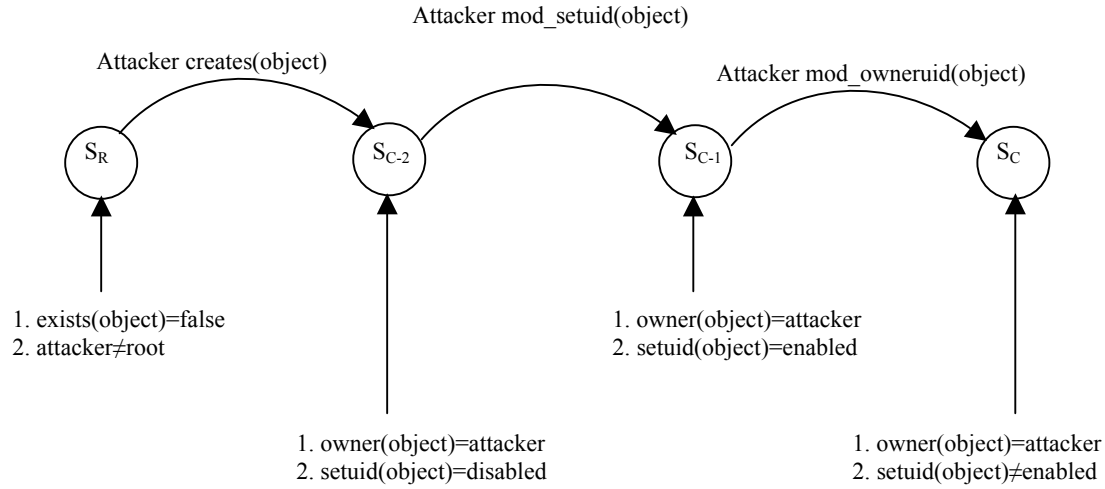


Figure 1.3.: State transition diagram (from [10]).

The intrusion described above leads to the state transition diagram in figure 1.3. note how the intrusion scenario above has been stripped of many assumptions about what the nature of the intrusion is, e.g. the fact that the file */usr/spool/mail/root* is a copy of the command interpreter csh. This information is not necessary to detect the violation. The first step, that of creating */usr/spool/mail/root* is paramount in detecting this intrusion however, it is not of vital importance how this file is created, or what it contains. Thus, the state transition diagram has obstructed away from the intrusion in such a way as to allow the diagram to represent variations of the same intrusion scenario, that a more straightforward, simplistic, signature based intrusion detection system may fail to detect.

The prototype system

The USTAT prototype is intended as a real-time expert system for detecting intrusions in real-time. The prototype runs under SunOS 4.1.1, with SunOS BSM (Basic Security Module) installed. This module provides USTAT with a C2 compliant audit trail. USTAT consists of the following components as described in figure 1.4.:

- The Preprocessor
- The knowledge-base
  - The fact-base
    - The Fact-base Initializer
    - 0The Fact-base Updater
  - The Rule-base
    - The State Description Table
    - The signature Action Table
- The Inference Engine
- The Decision Engine

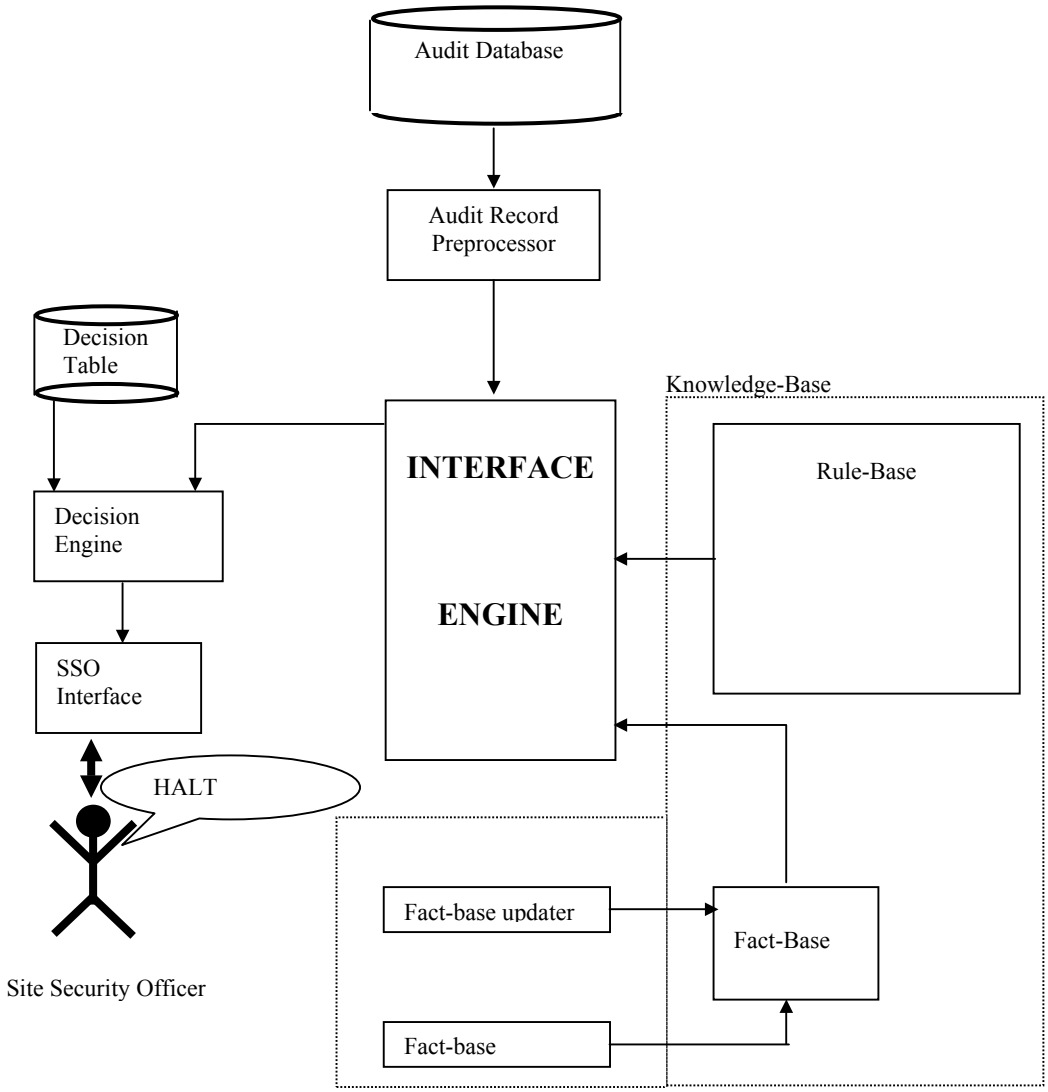


Figure 1.4.: USTAT Components (from [10]).

The Preprocessor

The preprocessor is responsible for reading, processing, and filtering the audit data to be used by the inference engine. The USTAT canonical format identifies the audit record according to the six-tuple: {*Subject Id, Subject Perm, Action, Object Id, Object Owner, Object Perm*}, where the normal BSM/UNIX audit record contents are mapped onto these fields.

The knowledge-base

USTAT knowledge-base consists of two major components: the fact-base contains system-specific information that is relevant to the success of the penetrations represented with the rule-base. It consists of groups of objects which are called *filesets*. These filesets are used by the rule-base to increase the generality of the penetration rules. The rule-base contains the state transition diagrams that describe a particular intrusion scenario. The latter information is stored in two files; the *state description table* and the *signature action table*. The state description table stores the state assertions, depicted below the circles in figure 1.3., and the signature action table stores the signature actions, placed above the arcs in figure 1.3.

Inference engine

The inference engine evaluates the new audit records, using information from the rule-base and the fact-base, and updating the fact base with state information. It forms the heart of the control mechanism. It consists of a collection of algorithms that make use of the other components of an expert system in search of new information [11]. The inference engine does not "*know*" hat rules and facts should be or could be in the knowledge-base. For any given inference step, the inference engine blindly uses all the relevant rules and facts that are available to it at that time. USTAT's inference engine use a forward chaining inference scheme. The forward-chaining scheme comes from the well-known logic rule called *modus ponens*.

Decision engine

The basic role of the decision engine is to send information to the SSO about how close a compromise is to being achieved or whether a compromise has been achieved [11]. This gives the SSO the ability to monitor the progress of the inference engine. Porras [12] suggests that a fourth mechanism could be added to make the decision engine respond actively to thwart the attack, as it progresses.

Results

The authors [10] put the prototype to two kinds of tests, function as well as performance were evaluated. The prototype was put against a number of possible intrusion scenarios, and variation thereof, where the attacks were performed by several attackers in unison, using hard links to files, instead of the original file names etc. These tests demonstrated that USTAT indeed managed to detect intrusions under these circumstances.

Performance-wise the prototype was run on a single workstation that also performed the audit collection, these tests indicated that under light load, USTAT kept up well with the stream of audit records, but when audit intensive applications such as *find* were run, USTAT did not fair as well. USTAT consumed approximately 13% of the CPU, and the bottleneck was identified as being the disk to which both the audit facility stored audit records, and USTAT attempted to read those same records from.

1.3.2.2 IDIOT-Intrusion detection using petri-nets

IDIOT "*Intrusion Detection In Our Time*" [13-16] is a system developed at COAST, Perdue University, IN, USA. Kumar[13-16] employed Petri Nets for signature based intrusion detection. Each intrusion signature is presented as a Colored Petri Net.

Presentation of the model

Kumar and al. suggested a layered model that divides the intrusion detection effort into three distinct abstraction layers:

The information layer

To isolate any machine/platform dependencies in the audit data, and provide the upper layers with a low-level data interface.

The signature layer

Describes the signatures indicative of intrusive behavior in a system independent fashion, by the use of a virtual machine model.

The matching engine

It matches the signatures in the preceding layer. This enables the use of any suitable matching technology as, when, it becomes available.

The proposed model has as its basis the notion of an auditable event. These events have tags, that hold data about the event. Intrusion signatures are specified with a "***follows***" rather than "***immediately follows***" semantics, in terms of the events that the matcher would see. Kumar et al. [13] classified some UNIX attack types into the following classes:

Existence

The fact that something ever existed is sufficient to detect the intrusion attack. Simple existence can often be found by static scanning of the file system.

Sequence

The fact that several things happened in strict sequence is sufficient to assert an intrusion.

Partial order

Several events are defined in a partial order (for example figure 1.5), i.e. many parallel or sequential preconditions must exist in order for the later part of the intrusion specification to hold.

Duration

This requires that something happened for not more than or no less than a certain interval of time.

Interval

Things happened an exact (plus or minus clock accuracy) interval apart. This is specified by the conditions that an event occurs no earlier and no later than *x* units of time after another event.

The authors [13] emphasis that the vast majority of known intrusion patterns fall into the first two categories listed above.

Applying Colored Petri Nets to the proposed IDS

The authors suggested to adapt the CP net model among other available techniques of pattern matching, such as attribute grammars, regular expressions and deterministic context free grammars, because of its generality, conceptual simplicity and graphical representability. In addition to this, CPNs do not suffer from a number of shortcomings common to the other techniques.

An illustrative example of a specification of an intrusion signature using the proposed model is presented in figure 1.5 below.

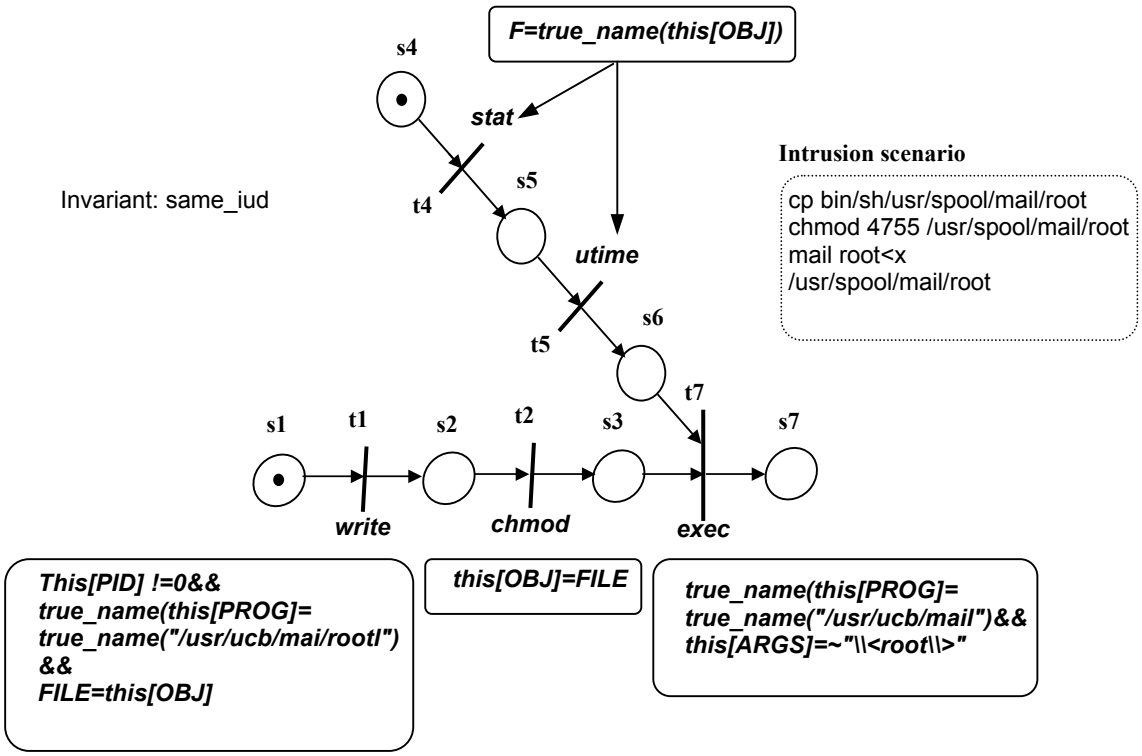


Figure 1.5.: IDIOT - a Petri net intrusion signature (from [16]).

The start states are **s1** and **s4**, with **s7** the final state. The evaluation begins by placing a token in each start state. These tokens (each token has a set of variables associated with it) "flow" through the Petri net. These transitions may be guarded by Boolean expressions that must evaluate to true for the transition to take place. There is a special operator ***this*** that is instantiated to the most recent event.

Each specification also has a set of pre-conditions, post-conditions, and invariants. These are similar to guards that must be *true* to be successful. Patterns that have no transitions can be specified using pre-conditions to an empty pattern. Post-conditions are provided for symmetry to allow the recursive invocation of the same pattern. Invariants are provided to allow the user to specify some condition that should not be true while another pattern is being matched. The authors mention that negative pattern specification in a CPA (Colored Petri Automaton) unnecessary clutters the description of the pattern.

In order to instantiate this generic model to a specific environment, e.g. UNIX, it is suggested that the SSO define the primitives supported in guard expressions. It might also include coding file test operations, set manipulation function, and other operations.

System overview

IDIOT contains four principle components:

audit trail Technically, the audit trail is not part of IDIOT, although IDIOT receives all its information about the system via the audit trail. The version of IDIOT described uses the Solaris 2.4, BSM (Basic Security Module) C2 audit mechanism as its source of input. However, IDIOT is designed to be easily portable to any other form of audit trail format.

Showaudit.pl This PERL script converts the audit trail to a canonical format, to be used by the rest of IDIOT. This division of labor is intended to ease porting of IDIOT to other platforms, with other forms of audit trails. *showaudit.pl* can be run either in batch mode, to convert an already existing file, or record-by-record mode, where the script watches the end of the audit file, and converts each record as it becomes available.

C2-server This is the heart of the intrusion detection system. Implemented as a C++ class, the C2-server, reads and audits records from *showaudit.pl*, steps through the different intrusion detection patterns (implemented by the pattern matching each) that request an event, and lets each pattern matching engine decide whether to update its state according to the event. Thus each pattern gets access to each event. The pattern matching engines can be dynamically added to an already running C2-server.

C2-appl The C2-appl provides the SSO with a user interface, from which to control IDIOT, he can start and add new pattern matching engines for example, and learn of the status of IDIOT.

The C2-engine and (where applicable) the patterns can be moved unchanged to the new platform, the y are intended to be platform independent. However, the audit trail and the *showaudit.pl* scripts need to be ported when moving IDIOT to a new platform.
The patterns are written in an ordinary textual language, resulting in a new pattern matching engine which can then be dynamically added to an already running IDIOT instance, via the user interface. Therefore, the user can extend IDIOT to recognize new audit events.

Conclusion

The work presented is thorough on the nature of the intrusions that the system is supported to detect. It is in fact by far the most thorough presentation of the existing IDS. The description of the pattern that describe the intrusions is based on theoretical foundations, and thus not ad-hoc in nature. The authors [13-16] furthermore stress the necessity of testing the effectiveness of intrusion detection, by building a set of standard test cases.

1.3.2.3 GASSATA- intrusion detection system with Genetic Algorithms

GASSATA "*Genetic Algorithms for Simplified Security Audit Trail Analysis*" [17-19]is a system developed at Supelec, University of Rennes, France. This system uses genetic algorithms to find potential attacks by analyzing the audited users. It runs under an IBM RS6000 11.7 Mflop [18] with the audit system AIX 3.2.4.

The architecture of this tool is presented in figure 1.6 below.

A thorough study and improvements of this system are detailed in the next chapters. An outline of genetic algorithms, which is the tool used to implement this IDS, is presented in chapter 2 and chapter 3 describes the IDS called GASSATA and its drawbacks. Finally, some enhancements to overcome the GASSATA's shortcomings are also discussed in chapter 3.

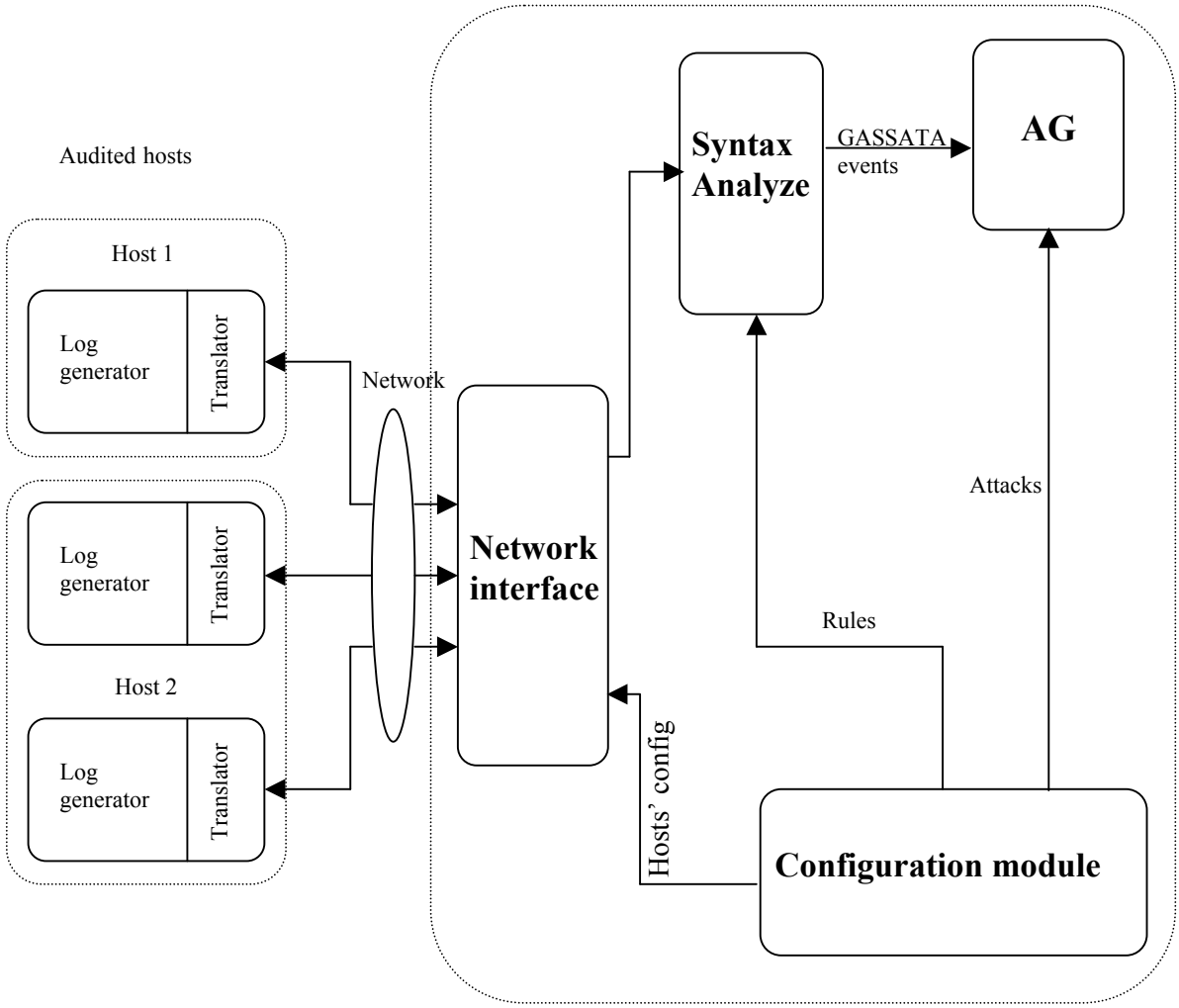


Figure 1.6.: Architecture of GASSATA (from [17]).

1.4 Summary

The security audit trail is essential in any implementation of an intrusion detection system. It is not credible (according to the Orange Book or White Book) to develop an IDS without using this principle function.

The interesting events to be collected are highly described in literature. However, the Site Security Officer has to point out the "*required*" events according to the current security policy and the system on which the audit is being developed.

Although many Intrusion Detection Systems tools are developed, there is still much to be done to enhance these systems and overcome the encountered drawbacks generated by them. In fact, a careful study of them can highlight their vulnerabilities that could be used unlawfully by criminals (why not by terrorists).

" When the only tool you have is a hammer, every problem you encounter tends to resemble a nail "
-Source unknown.

Part 1

Genetic Algorithms to intrusion detection

And Contributions to GASSATA

Chapter 2

Introduction to Genetic Algorithms

Genetic Algorithms (GAs) are adaptive methods which may be used to solve search and optimization problems. They are based on genetic processes of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and "*survival of the fittest*", first clearly stated by Charles Darwin in the Origin of Species. By mimicking this process, genetic algorithms are able to "*evolve*" solutions to real world problem, if they have been suitably encoded.

In this chapter, we present a state-of-the-art of genetic algorithms as an heuristic technique in optimization problems. A thorough description of the simple genetic operators, established by John Holland such as: selection, crossover and mutation, is also presented. These mechanisms are entirely stochastic. Based on these operators, genetic algorithms are a class of general-purpose search methods [21] combining elements of directed and stochastic search which can make a remarkable balance between exploration and exploitation of the search space. GAs use a direct analogy of natural behavior, they work with a population of "*individuals*", each representing a possible solution to a given problem. Finally, we spotlight the two common problems corresponding to the GAs' implementation: (a) coding which consist of representing a potential solution to a problem as a set of parameters which are joined together to form a string of values (mostly binary parameters are used) and (b) constructing a unique fitness function to this problem.

2.1 Introduction

Genetic Algorithms had been proposed by JOHN HOLLAND during the 1970's. They are based on the mechanism of natural selection and natural genetics. The usual form of genetic algorithms was described by Goldberg [20]. Genetic algorithms, differing from conventional search techniques, start with an initial set of random solutions called *population*. Each individual in the population is called a chromosome, representing a solution to the problem at hand. A chromosome is a string of symbols; it is usually, but not necessary, a binary bit string. The chromosomes *evolve* through successive iterations, called *generations*. During each iteration, the chromosomes are *evaluated*, using some measures of *fitness*. To create the next generation, new chromosomes, called *offspring*, are formed by either (a) merging two chromosomes from current generation using a *crossover* operator or (b) modifying a chromosome using a *mutation* operator. A new generation is formed by (a) selecting, according to the fitness values, some of the parents and offspring and (b) rejecting others so as to keep the population size constant. Fitter chromosomes have higher probabilities of being selected. After several generations, the algorithms converge to the best chromosome, which hopefully represents the optimum or sub-optimal solution to the problem.

Let *P(t)* and *C(t)* be parents and offspring in current generation *t*; the general structure of genetic algorithms [21] (see figure 2.1) is as follows :

Procedure: Genetic Algorithms

begin
  t← *0*;
  initialize *P(t)*;
  evaluate *P(t)*;
  while (not termination condition) **do**
    recombine *P(t)* to yield *C(t)*;
    evaluate *C(t)*;
    select *P(t+1)* from *P(t)* and *C(t)*;
    t← *t+1*;
  end while
end

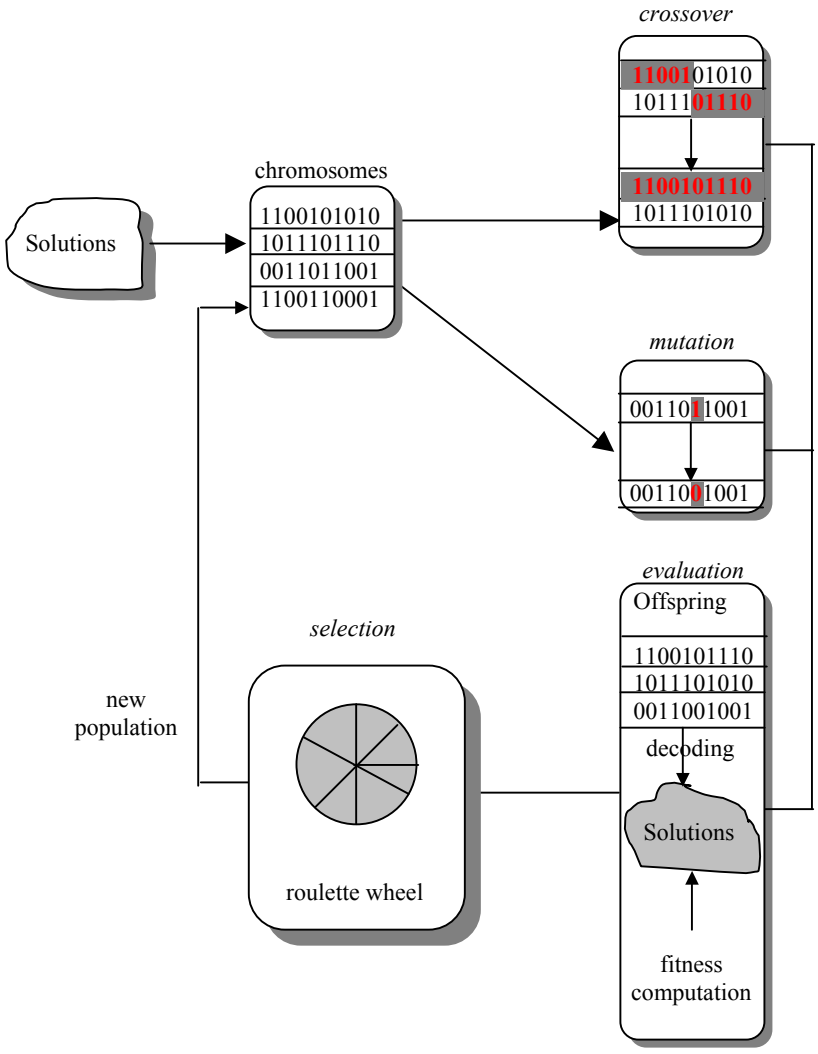


Figure 2.1.: A Traditional Genetic Algorithm.

Usually, initialization is assumed to be random. Recombination typically involves crossover and mutation to yield offspring. In fact there are two kinds of operations in genetic algorithms:

1. *Genetic operations*: such as crossover and mutation.
2. *Evolution operation*: selection.

The genetic operations mimic the process of heredity of genes to create new offspring at each generation. The evolution operation mimics the process of *Darwinian evolution* to create population from generation to generation.

How are Genetic algorithms different from other conventional optimization and search methods? Goldberg [20] provided an instructive answer to this question, noting the following four main differences:

1. GAs work with a coding of parameter, not with the parameters themselves,
2. GAs search from a population of solutions, not a single solution,
3. GAs use payoff (fitness function) information, not derivatives or other auxiliary knowledge, and
4. GAs use probabilistic transition rules, not deterministic rules.

2.2 Basic Principles

Before a GA can be run, an appropriate coding (or representation) for the problem must be devised. We require a fitness function, with a figure of merit to each coded solution. During the run, parents must be selected for *reproduction*, and recombined to generate offspring. These operators are described below.

2.2.1 Coding

It is assumed that a potential solution to a problem may be presented as a set of parameters. These parameters (known as *genes*) are joined to form a string of values (often called chromosome).
In genetic terms, the set of parameters represented by a particular chromosome is referred to as *genotype*. The genotype contains the information required to construct an organism-which is referred to as the *phenotype*.

2.2.2 Initialization

There are many ways to initialize and encode the initial generation: binary or non-binary, fixed or variable length strings, and so on. Holland's initial encoding method is as binary-fixed length string, although this is not vital and although not desirable or natural. At the initial stage, the system just randomly generates valid chromosomes and evaluates each one.

2.2.3 Reproduction

2.2.3.1 Generational Reproduction

With generational reproduction, the whole of the population is potentially replaced at each generation [22]. The most often used procedure is to loop $N/2$ times, where N is the population size, selecting two chromosomes each iteration according to the current selection procedure, producing two offspring from those two parents, finally producing N new chromosomes (although they may not all be entirely new).

2.2.3.2 Steady State Reproduction

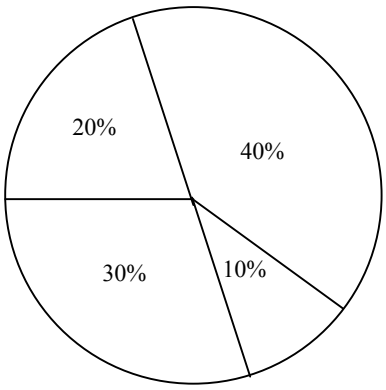
The steady state method selects two chromosomes according to the current selection procedure, performs crossover on them to obtain one or two offspring, perhaps applies mutation as well, and installs the result back into that population; the least fit of the population is destroyed [23]. The chromosome selection is not performed according to its fitness proportional probability but to its fitness value.

2.2.4 Selection

During the reproductive phase of the GA, individuals are selected from the population and recombined, producing offspring which will comprise the next generation. Parents are selected randomly from the population using a method which favors the fitter individuals; good individuals will probably be selected several times in a generation, the other ones may not be at all. Though this selection procedure is random, it does not imply that GAs employ a directionless search.

2.2.4.1 Fitness-based selection

The usual, original method for parent selection is *Roulette wheel* selection or fitness-based selection. In this category of parent selection, as presented in figure 2.2, each chromosome has a probability of selection that is directly relative to its fitness. The effect of this depends strongly on the range of fitness values in the current population.



Individual	Fitness value
I1	2
I2	4
I3	1
I4	3

Figure 2.2.: Roulette wheel

2.2.4.2 Tournament-based Selection

The standard tournament selection, as described by (Brindle, 1981), which selects *K* parents at random and returns the fittest one of these. Some other forms of tournament selection exist, for example, *Boltzmann* tournament selection [24] evolves a Boltzmann distribution across a population and time using pair-wise probabilistic acceptance and anti-acceptance mechanisms. The procedures are the following: generate a candidate solution uniformly at random in the neighborhood of the current solution and accept the new solution with logistic probability. Repeat this for some number of trials. Marriage tournament selection (Ross & Ballinger, 93) chooses one parent at random, has up to *K* tries to find one fitter, and stops at the first of these tries which finds a fitter one. If none is better than the initial choice, return that initial choice.

2.2.4.3 Rank-based Selection

Baker [26] was the first to report the rank-based selection method, in which selection probabilities are based on a chromosome's relative rank or position in the population, rather than absolute fitness. There are many possible forms of rank-based selection, depending on how the ranks are converted into objective fitnesses. Whitely [23] used a now popular method involving a bias scheme to control the selection pressure. That is, the larger the bias, the stronger the selection pressure. In Whitley's method, if the bias is 1.0, then the fittest of *N* chromosomes is *N* times more likely to be selected than the least fit. A higher bias gives somewhat more chance to the fittest and somewhat less to the unfit.

2.2.4.4 Spatially-oriented Selection

This selection category is a local selection scheme rather than a global one. That is, the selection competition is between several neighboring chromosomes instead of the whole population. This method is based on Wright's shifting-balance model of evolution (Wright 68, 69, 77, 78), which is adapted by several authors. Whitley termed it *Cellular Genetic Algorithms*. One of these methods consists of a two dimensional grid and maintains a roughly square population. This grid is consider rotoidal, that is leaving one edge will wrap around to the opposite edge. Two random walks of length *N* are taken from a selected starting location. The fittest found on each walk are taken as the two parents which produce one child. The child is installed at the start location if it is fitter than the one already there.

2.2.5 Crossover

Crossover and mutation are the basic operators in GAs. Crossover is a process yielding recombination of bit strings via an exchange of segments between pairs of chromosomes (see figure 2.1). It takes two individuals, and cuts their chromosome strings at some randomly chosen position, to produce two *head* segments and two *tail* segments. The tail segments are then swapped over to produce two new full length chromosomes. The two inherit some genes from each parent. This is known as *single point* crossover. There are a lot of kinds of crossover, here we just point up the concepts of four kinds of crossover.

Two-point Crossover: In *2-point* crossover, (and *multi-point crossover* in general), rather than linear strings, chromosomes are regarded as *loops* formed by joining the ends together. To exchange a segment from one loop with that from another loop requires the selection of two cut pints, as presented in figure 2.3. In this view, *1-point* crossover can be seen as 2-point crossover with one of the cut points

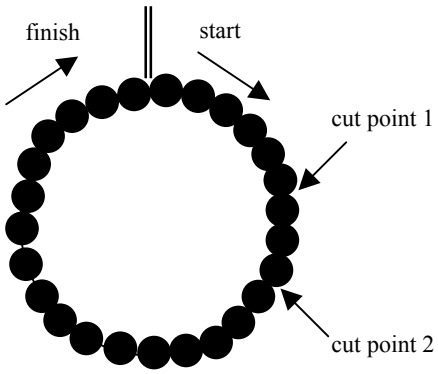


Figure 2.3.: A chromosome viewed as a loop.

fixed at the start of the string. Hence *2-point* crossover performs the same task as *1-point* crossover (i.e. exchanging a single segment), but is more general. Researchers now agree that *2-point* crossover is generally better that *1-point* crossover.

Uniform Crossover: In this kind of crossover, each gene of the first parent has a *0.5* probability of swapping with then corresponding gene of the second parent. One of the methods consists of using a *crossover mask*. Where there is a *1* in the crossover mask, the gene is copied from the first parent, and where there is a 0 in the mask, the gene is copied from the second parent, as shown in figure 2.4. The process is repeated with the parents exchanged to produce the second offspring. A new crossover mask is generated for each pair of parents.

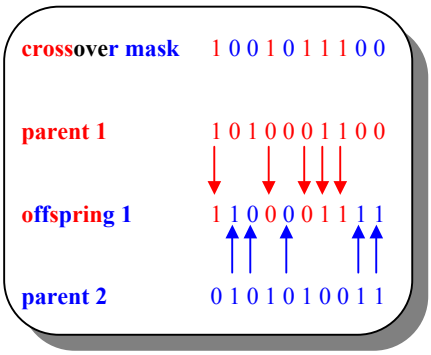


Figure 2.4.: Uniform Crossover.

2.2.6 Mutation

Usually, mutation is applied to each child individually after crossover. It randomly alters each gene with a small probability (typically 0.001). Figure 2.1 shows the sixth gene of the chromosome being mutated.

The traditional view is that crossover is the more important of the two techniques for rapidly exploring a search space. Mutation provides a small amount of random search, and helps ensure that no point in the search space has a zero probability of being examined.

2.2.7 Other Operators

2.2.7.1 Inversion

Three operators (crossover, mutation and inversion) are considered for causing offspring different from their parents. Inversion functions as a kind of reordering technique. It operates on a single chromosome and inverts the order of the elements between two randomly chosen points on the chromosome. While this operator was inspired by a biological process, it requires some additional overhead. Although there are some researchers who have investigated this operator, thy generally have not reported good results in GAs in practice.

2.2.7.2 Migration

Parallel or Island [27] GAs allow to run several populations at the same time in each processor respectively and occasionally alter a chromosome from one population to others, this is known as migration, a chromosome is chosen from one population (the first population is chosen for the first migration, the second for the second and so on) according to the current selection procedure and copied into all the other populations. The populations remain fixed in size, so this insertion causes the least fit in a population to be destroyed.

2.3 An illustrative example of GAs operators

Initialization

Let us consider the following problem:

$$\begin{cases} Max \sum_{i=1}^{n} x_i \\ s.t. \\ x_1 + 3x_2 + 10x_4 + 30x_5 \le 12 \\ 2x_3 + 4x_4 \le 17 \\ x_1 + x_3 \le 8 \\ 5x_2 + x_4 \le 8 \\ 2x_2 + 15x_6 \le 5 \end{cases} \qquad (2.1)$$

where $x_i \in \mathrm{N}$, $i=1..4$.

Now, we can use a non binary bit string representation to represent the chromosomes here, because it is easy to understand and represent. Our example has four positions representing four positive integer variables. We can generate the population randomly to assign each variable a positive value.

If we randomly generate four numbers *3, 1, 5, 0, 0, 0* as the values of x_1, x_2, x_3 , x_4 and the value of the objective function is *3+5+1+0+0+0=9*.

- Reproduction and Selection

In table 2.1, assuming that the fitness columns of each chromosome reflects the quality of that corresponding solution, which could be, for example, an inverse function of various violations of the constraints of the problem. If we apply fitness-based selection and randomly generate a number *0.4* and multiply by the sum of fitness *0.0509*, then get an accumulated fitness *0.02036* which is what we expect to select among our parents. We consult table 2.1 to get the accumulated fitness *0.02036* which is between the first accumulated fitness *0.0209* and the second accumulated fitness *0.0265*, so we select the second chromosome as our first parent *7 3 4 2 1 1*.

Similarly, using the same method to get another accumulated fitness *0.90*, we select the fifth chromosome *259623* as our second parent.

i	**Chromosome**	**fitness**	**Accumulated fitness**
1	*3 1 5 0 0 0*	*0.0209*	*0.0209*
2	*7 3 4 2 1 1*	*0.0056*	*0.0265*
3	*8 1 2 0 5 7*	*0.0164*	*0.0429*
4	*4 6 3 5 0 0*	*0.0074*	*0.0503*
5	*2 5 9 6 2 3*	*0.0006*	*0.0509*

Table 2.1.: Initial population and fitness.

- Crossover

- ***One-Point crossover:*** with the two parents selected above, we randomly generate 4 as the crossover position:

Parent1: *7 3 4 2 1 1*
Parent2: *2 5 9 6 2 3*

then we get two offspring:

child1: *7 3 4 2 2 3*
child2: *2 5 9 6 1 1*

where the first child has the first part from parent1 and the second part from parent2 and the second child has the first part from parent2 and the second part from parent1.

• ***Two-Point crossover:*** with the two parents above, we randomly generate two numbers 2 and 4 as crossover postions:

Parent1: *7 3 4 2 1 1*
Parent2: *2 5 9 6 2 3*

After crossover, we get two children:

child1: *7 3 9 6 1 1*
child2: *2 5 4 2 2 3*

• ***N-Point crossover:*** if we randomly generate a number *n* equal to *3*, then randomly generate *3* numbers *1,2,4* as crossover positions and swap even-part positions but let odd-part unchanged to produce two children:

Parent1: *7 3 4 2 1 1*
Parent2: *2 5 9 6 2 3*

After crossover, we get two children:

child1: *7 5 4 2 2 3*
child2: *2 3 9 6 1 1*

• ***Uniform crossover:*** for each position, we randomly generate a number between *0* an *1*, for example *0.3, 0.8, 0.6, 0.3, 0.7, 0.2*. if the number generated for a given position is less than 0.5, then child1 gets the gene from parent1, and child2 gets the gene from parent2. Otherwise, vice versa.

Parent1: *7 3 4 2 1 1*
Parent2: *2 5 9 6 2 3*

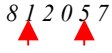
After crossover, we get two children:

child1: *7 5 9 2 2 1*
child2: *2 3 4 6 1 3*

- Inversion

If we randomly choose two positions *2, 5* and apply inversion operator:

8 1 2 0 5 7



Then we get the new chromosome:

8 5 2 0 1 7

2.4 The fundamental Theorem of Genetic Algorithms

In this section, we describe the fundamental theorem of GAs, called the Schema Theorem. Our purpose is not to investigate this theorem or do any kind of formal analysis of GAs. However, here we demonstrate the extraordinary and surprising power of Genetic Algorithms, where simple, genetically-inspired search operators seemed to quickly solve many ***difficult*** problems.

2.4.1 Implicit Parallelism

A schema [22] is a similarity template describing a subset of strings with similarities at certain string positions. Why are schemata important for Genetic Algorithms? Goldberg [20] says:

> "*In some sense we are no longer interested in strings as strings alone. Since important similarities among highly fit strings can help guide a search, we question how one string can be similar to its follow strings. Specifically we ask, in what ways is a string a representative of other string classes with similarities at certain string positions? The frame of schema provides the tool to answer these question*."

A schema is a string having the following form:

0 # 0 1 1 #

Each position presents 0, 1 or #, in which # is a do not care symbol, can 0 or 1. The more # it has, the less specific it becomes. In a binary string, if there are n "#", then a schema can describe 2^n strings. Equally, any string of 0s or 1s of length n is an instance of 2^n schemata. In our example there are 2 "#", therefore it can describe four strings:

0 0 0 1 1 0

0 0 0 1 1 1

0 1 0 1 1 0

0 1 0 1 1 1

The total number of different schemata of length n is 3^n, because each of the n positions can be 0, 1 or #. The total search space remains 2n since the # symbols are not involved in the real string processing; they only serve as a tool to describe similarity templates.

2.4.2 The Schema Theorem [20]

Let us consider a simple generational GA. The frequency $m(H, t)$ of a schema H at generation t will change at generation $t+1$ proportionally to the selection probability of strings for reproduction. There are two steps to consider. The first step is the selection step, in which we choose from the generation at time t the chromosomes that will be the parents for the generation at time $t+1$. Let $m^P(H,t)$ be the frequency of schema H among the *parents* selected to produce generation $t+1$. In the variation step, which we will consider later, we apply crossover and mutation to these parents with each other to produce the new generation $t+1$.

Let *f(H)* represent the average fitness of a string containing schema *H* in generation *t* and $\bar{f}$ be the average fitness of the entire population. So, *f(H)* depends on *t*, and so does $\bar{f}$. we can derive the following formula:

$$m^p(H,t)=m(H,t)\times\frac{f(H)}{\bar{f}} \qquad (2.2)$$

In the following, we assume we are using one-point crossover and P_c is the probability of a crossover. The defining length $\delta(H)$ is the distance between the first and last specific schema positions of *H*, and l is the total length of a string then the probability of a schema being disrupted because of crossover is as in equation *(2.3)*

$$P_c\times\frac{\delta(H)}{l-1} \qquad (2.3)$$

the probability that an instance of schema *H* is crossed with a schema different from *H* is in equation *(2.4):*

$$1-\frac{m^p(H,t)}{c} \qquad (2.4)$$

where c is the size of the population. If we multiply the probabilities from equation *(2.3)* and equation *(2.4)*, we get the probability that schema H will be destroyed by the crossover operation. This is as in equation *(2.5)*:

$$(1-\frac{m^p(H,t)}{c})\times P_c\times\frac{\delta(H)}{l-1} \qquad (2.5)$$

this is, however, usually a small number. Let us call it ε.

The mutation rate is usually very small in normal situations. If the probability of a mutation is P_m and the order of a schema *H* denoted by $o(H)$, is simply the number of positions (number of 0's and 1's in a binary string) present in the template, then the probability of a schema being disrupted because of mutation is as in equation *(2.6)*:

$$o(H)\times P_m \qquad (2.6)$$

Therefore, we know that short order schemata are more likely to survive than long order ones. Now, we can derive the following schema formula:

$$m(H,t+1)\geq(1-\varepsilon)\times(1-o(H)\times P_m)\times m(H,t)\times\frac{f(H)}{\bar{f}} \qquad (2.7)$$

The greater-than-or-equal sign is there because we actually overestimate the amount of schemas *H* that will be destroyed. This is because it is still possible to produce instances of schema *H* by crossover of schema H with a chromosome which has no schema *H* even if the crossover position cuts the schema. Also, it is possible to produce instances of schema *H* even if neither of the parent chromosomes contains schema *H*. As we can see, since $(1-\varepsilon)$ is near to 1, this variation step does not harm the selection step too much. This means we can still be sure of having good schemas increasing in future generations, and the bad ones die out, at the same time as producing new good schemas by crossover.

So, the effect of the selection step is only affected by the amount $(1-\varepsilon)\times(1-o(H)\times P_m)$, which is an overestimate of how much crossover and mutation destroy instances of schema *H*. now, we can conclude that highly fit, short defining length, lower order schemata are strongly propagated generation to generation. This propagation of good schemata is driven by the stochastic selection according to high fitness, and the above equation *(2.7)* means that we can crossover and mutate, and so produce new and possible much better schemata, with only small change to this effect (because the

amount of destruction of good schema is usually very small). This is very important conclusion in GAs and it has the special name: *the Schema Theorem or the Fundamental Theorem of Genetic Algorithms*.

2.5 Solving a problem by GAs

Resolving a problem by using Genetic Algorithms requires two fundamental conditions:

- ability to code the solutions of the given problem by a limited bit string,
- ability to assign a selecting value to each string (i.e. finding the fittest fitness function).

2.5.1 Coding

Coding, as defined in section 2.2.1, consists of presenting the potential solution of a problem by a set of parameters joined together to form a string of values. However, during the coding phase, Goldberg [20] suggested:

- use an alphabet of a minimal cardinality.

- favor a coding where short defining length, lower order schemata have a sense towards the problem.

2.5.2 Fitness Function

In this section, we just introduce the possible fitness function to be used when the given problem is a constrained one. However, many methods exist among them:

- repeat the reproduction process until an individual respecting the constraints is generated.

- apply genetic operators that generate only individuals respecting the given constraints of the problem.

- penalize the individuals that do not respect the constraints by lessening their fitness values. We may use a penalty function P that reduces the fitness values of the individual that do not respect the constraints.

2.6 Summary

Genetic Algorithms are good optimizers and were highly used to solve **difficult** problems during the past few years. However, when a **polynomial algorithm** can, **precisely**, solve a problem in a short time, it is not necessary to use Genetic Algorithms. Because, first they are heuristic techniques and provide random solutions on one hand, and they are time consuming when combinatory explosion occurred, on the other hand.

"Every sentence I utter must be understood not as an affirmation, but as a question "

- Neils Bohr.

Chapter 3

Formalization of the Security Audit Trail Problem

Many intrusion detection methods had been developed and implemented. Most of them consist of detecting intrusions or the risk of intrusion by using an audit trail which is provided by the target computer system.

The main goal of intrusion detection by examining the audit trail is to determine when a computer system is entered, or is likely to enter, a faulty or intruded state. This chapter describes some formalizations of the security audit trail problem by expressing attack scenarios with regular expressions. Thus, the (present) intrusion detection method consists of searching (finding) the known scenarios in the current audit trail. To perform that, three algorithm problems are described and discussed. These problems, namely, first, second, and third security audit trail analysis problem seemed to be *NP-Complete* and the use of classical algorithms is impractical because of the huge amount of the generated audit trail files (during the audited sessions). Hence, a simplification [18] is made on the third problem in order to reduce the search space exploration. It consists of elimination of events chronological order. However, the problem complexity still persists when using the third idea. To overcome this problem, an heuristic method based on genetic algorithms [18] is established. The second part of this chapter introduces a new formalization of the problem which resolves all the encountered problems when using GAs.

3.1 Introduction

An approach based on the use of the attack scenario was first proposed by Thomas Garvey and T. F. Lunt [28]. An attack scenario may be defined as activity command stream with which the intruder can gain administrative privileges or transform the target system to a certain compromised state. Applying these attack scenarios is very interesting and brings many advantages:

- The SSO can, himself, model an attack scenario from his experience and from the known vulnerabilities of the target system.

- A scenario modification is very easy to perform. In addition, a new scenario form (detected from an anomaly intrusion detection system for example) may be easily taken into account.

- An attack scenario ordering according to the cost of each model may be performed to use it as a decision function during the intrusion detection procedure.

- It is also possible to model some scenarios which exploit well known security failures and not yet corrected ones (because of the cost reasons for example). A user performing these attacks is actually a danger to the target system.

For these reasons, applying a method, based on attack scenario models, is very interesting in intrusion detection process. However, building a language expressing the scenario attacks is necessary to localize the potential attacks which could be present in the audit trail.

3.2 Attack scenario expressions

An overview of the different substring recognition techniques is important in order to apply one (or more) of them, if possible, on the generated audit trail files. In fact, each event may be considered as a letter (symbol) belonging to an alphabet composed of the audit events set. The audit file, on the other hand, may be considered as a character string and the attack scenarios as the substrings to be recognized from the principle string. Thus, a use of the regular expressions and the different regular expression operators is useful in order to reach the above-cited objectives.

Before introducing the different formalization problems of the scenario models, we begin with a review of some regular expression material.

Definition 3.1.

"*An attack model*" is a regular expression, with a back reference character, denoting an attack scenario.

1- the following characters are called meta-characters: |, ( ), *,%,

2- for each character a, $a \notin \{|, (\ ), *, \%\}$, a is a model denoted by the string a,

3- if r_1 and r_2 are models, then $r_1|r_2$ is a model corresponding to a string denoting either r_1 or r_2,

4- if r is a model, then (r) is a model corresponding to the same strings denoting r,

5- if r_1 and r_2 are models, then $(r_1)\ (r_2)$ is a model denoted by each string having the following form xy, where x corresponds to r_1 and y to r_2 (*the concatenation operator*).

6- if r is a model, then $(r)^*$ is denoted by the following pattern $x_1x_2...x_n$, $n \geq 0$, where x_i corresponds to r, for $1 \leq i \leq n$. (*kleene closure operator*).

For each symbol a, $a \notin \{|, (\ ), *, \%\}$, a corresponds to the symbol e_i, $1 \leq i \leq n$, which represents the i^{th} auditable event defined by the SSO.

Definition 3.2.

A *simple attack model*, is a model which is denoted by a simple regular expression (i.e. without a back reference symbol.

3.3 Applying security audit trail analysis to intrusion detection

Having the attack model described in the previous section, we may now define some security audit trail analysis problems.

3.3.1 Model by model analysis

The first thought is considering the models independently each one from the other. Thus, a model by model analysis is likely to be performed. This idea consists of considering only one model formed by all the other models as follows: $m_1|m_2|...|m_n$.

This problem is called the "*first security audit trail analysis problem*" PASFAS1 [18].

Problem 3.1.

Let *I* be an alphabet. Each symbol of *I* denotes an auditable event. Let *S* be a set of scenario attacks expressed by the attack model, let *F* be the audit trail, which is a pattern defined over *I*. Find each attack model for which there is a substring denoting this model.

Theorem 3.1. PAFAS1 is *NP-Complete***.**

Proof 3.1.

If *N* is the cardinal of S, a solution of PAFAS1 may be obtained by the resolution of *N* problems of regular (expression) pattern matching (PRERC). This problem is defined as follows:
Le *re* be a regular expression with a back reference character. Determine whether an input pattern is present in the principle pattern *F*.

PRERC is an *NP-Complete* problem. This may be proved from an hamilitonian circuit problem [14] which is known as an *NP-Complete* problem (which may be transformed to PRERC) and for which there is a transformation from PREREC to it. Hence, PAFAS1 is *NP-Complete*.

Remark.

A substring recognition problem of a simple pattern (i.e. there is not a back reference operator) in a principle string, i.e. determining whether a string is present or not in the input is desired, may be realized in a polynomial time.

Example.

Matching the pattern ba in the input abcaa may require the use of a DFA (Deterministic Finite Automaton) as follows:

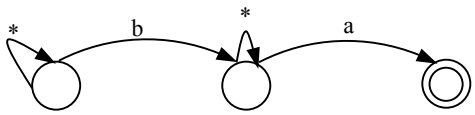


Figure 3.1.: A DFA to match the pattern ba.

Therefore, the problem of finding a simple attack model in the audit file is possible with a DFA.

Critics

The intrusion detection by analyzing the audit file is complex and is different from the substring recognition in a principle pattern. In fact, if we consider two different patterns: SS1 and SS2 appearing in the audit file. Assume that SS1 are not disjoined.?

This could be possible when the same auditable event appears in one or different attack models.
In this case, shall we consider this event as a present element in two attack scenarios? Or shall we consider only one scenario? In the second case, which scenario would be taken into account?

This situation can be of some high complexity when the same event(s) appears(s) in one or more different scenarios.

Example

Let m_1=abc, m_2=cde and m_3=def be three different scenarios corresponding to three different attacks a_1, a_2 and a_3. Consider that the principle audit string is abcdef.

If we consider the following audit string abcde, shall we consider the first attack present or only the second one. On the other hand, if we consider the audit string abcdef, shall we consider the presence of a_1 and a_2 or only a_2?

In real life, this situation may be very complicated especially when the same events appear in many different attack scenario models.

This first formalization is not interesting because the decision of the intrusion procedure, if there is more than one attack, is very complicated especially when the same events are present in the current attack scenario models.

For this reason, Mé [18] has proposed a new problem of the audit trail analysis by providing a corresponding weight to each attack scenario. It is called "*juxtaposed models*" and is described in the following section.

3.3.2 Juxtaposed models

This problem [18] is defined as the "*second security audit trail analysis problem*" PAFAS2.

Definition 3.3.

A model formed by arranging N attack models is called *meta-model*.

Problem 3.2 (PAFAS2)

Let E_m be a set of pairs (m_i, p_i), $E_m=\{(m_1, p_1), (m_2, p_2), \ldots, (m_N, p_N)\}$, where m_i is the attack scenario model and p_i its corresponding weight.

Which arrangement of the different E_m elements in the current audit trail maximizes the sum of p_i?

Theorem 3.2. PAFAS2 is *NP-Complete*.

Proof 3.2.

From N models of E_m, it is possible to assemble N_{mm} different meta-models of the following form $m_1.*m_2.*\ldots.*m_n$ $(1 \le n \le N)$.

where $$N_{mm} = \sum_{i=1}^{N} A_N^i = N! \sum_{i=0}^{N-1} \frac{1}{i!} \qquad (3.1)$$

To resolve PASFAS2, N_{mm} problems of the PRERC type shall be considered, adding to this the memorization of the meta-model that maximizes $\sum p_i$, for which there is a substring in the audit trail corresponding to it. PRERC is *NP-Complete*, then PAFAS2 is also *NP-Ccomplete*.

Remark

In PAFAS2, the meta-models are considered as if they were always adjacent, one following the other. However, this is not true in real life; the models may be intermingled (intermixed). An algorithm based on PAFAS2 is limited to successive meta-models and cannot find many other attacks. Hence, a third approach was proposed, called "*intermingled models*".

3.3.3 Intermingled Models

This approach is defined as the "*third security audit trail analysis problem*" PAFAS3.

Definition 3.4

A model formed by intermingling the events of N attack models (regarding their chronological order) is called the "*second meta-model form problem*".

Problem 3.3 (PAFAS3)

Let E_m be a set of pairs (m_i, p_i), $E_m=\{(m_1, p_1), (m_2, p_2), \ldots, (m_N, p_N)\}$
Where m_i is an attack model and p_i its corresponding weight. Which subset of E_m of the second meta-model form maximizes $\sum p_i$ and has its corresponding scenario in the audit trail?

Theorem 3.3 PAFAS3 is *NP-Complete.*

Mé [18] has proved that the number N_{mm2} of the second meta-model form which may be constructed by N models is given as follows:

$$N_{mm2}=\sum_{k=1}^{N}(V_k^N+V_k^N+\ldots+V_k^N) \tag{3.2}$$

where
$$V_k^N=\frac{(\sum_{i=1}^{k}N_j)}{\prod_{i=1}^{k}(N_j!)} \tag{3.3}$$

N_j represents the length of the attack model M_i.

It is clear that PAFAS3 is *NP-Complete* (with a similar proof as that of PAFAS2)
The following table represents the complexity of PAFAS3, where N is the number of the attack model of N_j events.

N_i	N=2	N=4	N=6	N=8	N=10
1	4	64	1956	109600	$9, 8641.10^{06}$
2	8	2920	$8, 2045.10^{06}$	$8, 73944.10^{10}$	$2, 50469.10^{15}$
3	22	376444	$1, 3824.10^{11}$	$3, 70863.10^{17}$	$4, 39762.10^{24}$
4	72	$6, 3202.10^{07}$	$3, 2485.10^{15}$	$2, 39099.10^{24}$	$1, 287.10^{34}$
5	254	$1, 17358.10^{10}$	$8, 88364.10^{19}$	$1, 89758.10^{31}$	$4, 91207.10^{43}$
6	926	$2, 30881.10^{12}$	$2, 67019.10^{24}$	$1, 71889.10^{38}$	$2, 22254.10^{53}$
7	3434	$4, 7252.10^{14}$	$8, 57226.10^{28}$	$1, 70775.10^{45}$	$1, 13266.10^{63}$
8	12872	$9, 95611.10^{16}$	$2, 88925.10^{33}$	$1, 81657.10^{52}$	$6, 30264.10^{72}$
9	48622	$2, 14528.10^{19}$	$1, 01097.10^{38}$	$2, 03653.10^{59}$	$3, 75237.10^{82}$
10	184758	$4, 70536.10^{21}$	$3, 64415.10^{42}$	$2, 38025.10^{66}$	$2,35707.10^{92}$
11	705434	$1, 04707.10^{24}$	$1, 34567.10^{47}$	$2, 87778.10^{73}$	$1, 54654.10^{102}$
12	$2, 70416.10^{06}$	$2, 35809.10^{26}$	$5, 0696.10^{51}$	$3, 57829.10^{80}$	$1, 05202.10^{112}$
13	$1, 04006.10^{07}$	$5, 36447.10^{28}$	$1, 94236.10^{56}$	$4, 55568.10^{87}$	$7, 37717.10^{121}$
14	$4, 01166.10^{07}$	$1, 23094.10^{31}$	$7, 54977.10^{60}$	$5, 91823.10^{94}$	$5, 30917.10^{131}$
15	$1, 55118.10^{08}$	$2, 84562.10^{33}$	$2, 97124.10^{65}$	$7, 82343.10^{101}$	$3,90748.10^{141}$
16	$6, 0108.10^{08}$	$6, 62123.10^{35}$	$1, 1821.10^{70}$	$1, 05003.10^{109}$	$2, 93265.10^{151}$
17	$2, 33361.10^{07}$	$1, 54947.10^{38}$	$4, 74804.10^{74}$	$1, 4283.10^{116}$	$2, 23922.10^{161}$

Table 3.1.: Number of meta-models of the second form for N models of N_i events (from [18]).

3.3.4 Summary

Table 3.2 presents the number, in the worst case, of comparisons with a simple regular expression to be performed when using PAFAS1, PAFAS2 or PAFAS3 according to *N*. Where *N* represents the number of attack scenarios. This substring recognition is a simple regular expression recognition where all the models have a length of 10 events.

N	PAFAS1	PAFAS2	PAFAS3
1	*1*	*1*	*1*
2	*2*	*4*	*184758*
3	*3*	*15*	*5, 551.10^12*
4	*4*	*64*	*4, 70536.10^21*
5	*5*	*325*	*4, 83348.10^31*
6	*6*	*1956*	*3, 64415.10^42*
7	*7*	*13699*	*1, 44565.10^54*
8	*8*	*109600*	*2, 38025.10^66*
9	*9*	*986409*	*1, 36166.10^79*
10	*10*	*9, 8641.10^06*	*2, 35707.10^92*

Table 3.2.: Number, in the worst case, of comparisons when using PAFAS1, PAFAS2 and PAFAS3 with 10 events (from [18]).

Although simple regular expression recognition is considered, combinatory explosion occurred when using PAFAS2 or PAFAS3.

Mé [18] has also tried to apply Myers' algorithms to PAFAS2 and PAFAS3. The combinatory explosion still persists. Thus, he has introduced another model which reduces the search space of the substring recognition.

3.4 Reducing the search space in PAFAS3

The idea of reducing the number of the possible cases in PAFAS3 is introduced [18] by not considering the events chronological order. In this case, the attack scenarios are not considered as events sequences but as a set of couples (E_i, N_i) where E_i represents an event type and N_i the number of audit events of type E_i generated by the corresponding scenario. Hence, constructing an efficient algorithm is useful to find all possible couples (E_i, N_i) in the generated audit file.

The following paragraph describes the simplified model to the security audit trail analysis. It is called the "*Simplified Audit Trail Analysis Problem*".

The Simplified Audit Trail Analysis Problem

The audit trail may be defined by an *Ne-dimensional* integer matrix. Let *O* (Observed) be this matrix. The i^{th} element O_i of *O* corresponds to the number of auditable events of type *i* currently present in the audit trail.

To each attack corresponds a weight, which is proportional to the risk that the target system enters an intruded state, whenever this attack is present in the audit trail.

Problem 3.4 (PASFAS)

Let *Ne* be the number of the different audit event types and *Na* the number of potential known attack scenarios. Each attack scenario is expressed as a set of *Ne* pairs (e_i, n_j) where e_i is the event type $(1 \le i \le Ne)$ and n_j its occurrence number in the attack scenario $(n_j \ge 0)$.

An *Ne* x *Na* dimensional matrix, denoted by *AE* (Attacks-Events matrix), is constructed where AE_{ij} denotes the number of events of type *i* generated by the scenario *j* ($AE_{ij} \geq 0$). Let *R* (Risk) be an *Na-dimensional* vector where R_i ($R_i > 0$) is the weight associated to attack *i* (R_i is proportional to the risk of intrusion whenever the scenario *i* is present). Let *O* (*O* for Observed) be an *Ne-dimensional* vector corresponding to the current audit trail. Let *H* (for Hypothesis) be a binary *Na-dimensional* vector, where $H_i=1$ if the attack *i* is assumed to be present and $H_i=0$ otherwise (*H* describes a particular subset). The simplified security audit trail problem consists of determining the hypothesis vector *H* which maximizes the *R* x *H* product subject to the constraints $(AE \times H)_i \leq O_i$ $(1 \leq i \leq Ne)$ (equation 3.4 from [18]).

$$\left\{ \begin{array}{l} \left(\begin{array}{ccc} 1 & i & Na \\ & R_i & \end{array} \right) \left(\begin{array}{c} \\ H_i \\ \\ \end{array} \right) = maximum \\ \left(\begin{array}{ccc} 1 & i & Na \\ 1 & & \\ j & AE_{ij} & \\ Ne & & \end{array} \right) \left(\begin{array}{c} \\ H_i \\ \\ \end{array} \right) \left(\begin{array}{ccc} 1 & 1 & \\ i & \leq & j \\ Na & Ne & \end{array} \right) \left(\begin{array}{c} \\ O_i \\ \\ \end{array} \right) \end{array} \right. \qquad (3.4)$$

This problem consists of finding the hypothesis vector *H* that maximizes the risk of intrusion according to the observed audit trail (i.e. finding *H* so that the corresponding risk is the greatest according to the observed audit *O*).

Theorem 3.4. *PASFAS is NP-Complete.*[18]

Proof 3.4.

There is a polynomial transformation from the *Zero-One Integer Programming Problem (ZOIPP)* to PASFAS.
Zero-one integer programming problem, defined as follows, is *NP-Complete* [29]:
Let *X* be a set of *m* couples (x_i, b_i) where x_i is an *n-uple* of integers and b_i a fixed integer. Let *C* be an *n-uple* of integers and *B* an integer.
Can we find an *n-dimensional* binary vector *Y* of integers such that
1- $x_i.Y < b_i$ *for all* $(x_i, b_i) \in X$ $(1 \leq i \leq n)$ *and*
2- $C.Y \geq B$?

Let

- $n=Na$ and $m=Ne$,
- $x_i=(AE_{i1}, AE_{i2}, ..., AE_{iNa})$,
- $b_i=O_i$,
- $C=R$,
- $B \in \{1,2,..., \sum R_i\}$ *and*
- $Y=H$.

From this, we deduce that zero-one integer programming problem may be transformed to PASFAS.
The ZOIPP is an *NP-Complete* problem , then we conclude that PASFAS is also *NP-Complete*.

The necessary execution time to resolve PASFAS by a classical algorithm

In this section, we will try to discuss whether an *NP-Complete* algorithm is feasible to be used in order to resolve PASFAS.

If *Na* is the number of attack scenarios, the number *M* of subsets in which each scenario is considered present *(1)* or not *(0)* (*M* denotes the whole search space) is given as follows:

$M=2^{Na}-1$ *(3.5)*

Each subset corresponds to a particular value of *H*.
For the moment (in our knowledge), there is not an efficient algorithm to resolve this problem. Therefore, we have to consider all the cases (there are $2^{Na}-1$ cases). For each value of *H*, the *AE* x *H* product should be evaluated (which corresponds to (*Na* x *Ne* multiplications, *Ne* x *(Na-1)* additions and *Ne* affectations) then comparing $(AE.H)_i$ *to* O_i $(1 \leq i \leq Ne)$. If $(AE.H)_i \leq O_i$, the *R* x *H* product shall be evaluated (it corresponds to *Na* multiplications, *(Na-1)* affectations and one addition). The greatest product value is stored corresponding in the worst case to *M-1* comparisons and *M* affectations.
If N_{mult}, N_{add}, N_{comp} and N_{affec} represent the number of multiplications, additions, comparisons and affectations, we then obtain for this algorithm:

$N_{mul\ t}= (2^{Na}-1)$ x *Na* x *(Ne+1)* *(3.6)*

$N_{add} = (2^{Na}-1)$ x (*Na-1)* x *(Ne-1)*
(3.7)

$N_{comp}=[(2^{Na}-1)$ x *(Ne+1)]-1* *(3.8)*

$N_{affec}=(2^{Na}-1)$ x *(Ne+2)* (3.9)

If the necessary times needed to perform a multiplication, an addition, a comparison and an affectation are respectively *0.1,0.3,0.3* and *0.5 µsec*, the execution time needed to resolve PASFAS with this algorithm, considering *Ne=20* different events, is presented in the following table with different values of *Na* (table 3.3).

Na	T_{PASFAS}
1	*33.6 µsec.*
5	*3.12 msec.*
10	*0.19 sec.*
15	*8.8 sec.*
20	*6.16 minutes.*
24	*1hour 57 minutes.*
30	*155 hours 20 minutes.*
40	*24 years.*
50	*305 centuries and 8 years.*

Table 3.3.: The necessary execution time of PASFAS (*Na=50*) (from [18]).

Is it feasible to use an NP-Complete algorithm to resolve PASFAS?
A short answer to this question may be "*yes*" but only for small values of *Na* (the number of attack scenarios). In real life, more than *50* attacks may be considered. Thus, the application of an *NP-Complete* algorithm to PASFAS is impractical in such a situation.

3.5 Using Genetic Algorithms to resolve PASFAS [17,18]

Mé [18] has chosen to use an heuristic approach, based on genetic algorithms, to solve PASFAS which is known as an *NP-Complete* problem. Thus, coding a solution and finding a robust fitness function to this problem are mandatory to define. The population, in this case, is the different values of *H* vector coding a potential solution.

3.5.1 Coding a solution

An individual, as defined in *chapter 2*, represents a possible solution to a given problem. PASFAS consists of defining the *H* vector that maximizes *R* x *H* product subject to the constraints *AE* x $H \leq O$. Finding a particular value of *H* is a potential solution to PASFAS. Thus, each individual corresponds to a particular *H* vector. The binary coding is therefore used and the length of each individual is equal to the number of attack scenarios *Na*.

3.5.2 Fitness function

The first function, one can use, is a function which may maximize the product $R \times H = \sum_{i=1}^{Na} R_i \times H_i$

However, if the fitness $= \sum_{i=1}^{Na} R_i \times H_i$, this does not take into account the constraints of the problem (i.e. *AE* x $H \leq 0$). To *penalize* the large number of individuals which do not respect the constraints $(AE.H)_i \leq O_i$, *i=1..Ne*, a penalty function *(P)* is introduced where $P=Te^p$ and *Te* is the events types number for which $(AE.H)_i > O_i$. A quadratic penalty function *(i.e. p=2)* allows a good discrimination among the individuals.

Then the proposed fitness function [18] is

$$F(H) = \alpha + \left(\sum_{i=1}^{Na} R_i H_i - \beta . Te^2 \right) \qquad (3.10)$$

β is a parameter used to modify the slope of the function and if we use a non binary coding, it is sufficient to find the corresponding value of β.

α sets a threshold making the fitness positive.

The developed tool used to solve this problem was called "GASSATA" (***G**enetic **A**lgorithm for a **S**implified **S**ecurity **A**udit **T**rail **A**nalysis)* in which $R_i = 1$ for *i=1..Na*.

3.6 Some critics to GASSATA

This paragraph presents some shortcomings resulting from the formalization of PASFAS:

(i) - by using a binary coding for the individuals, GASSATA cannot detect the multiple realization of a particular attack, (Mé [17] proposed to try to use non-binary GAs)

(ii) - if the same event or group of events occur(s) in several attack scenarios, a ***malicious*** intruder realizing these attacks simultaneously without duplicating this event or group of events, GASSATA fails to find the actual attacks, (Mé [17] has not a solution for this problem)

(iii) - by using Genetic Algorithms, if there is more than one optimum solution, GASSATA provides randomly one of them, (see the example in the first section of chapter 4)

(iv) - if an intruder knows the period of a session, he can perform an attack during two or more different sessions, GASSATA will also fail to detect this attack. (Mé proposed to execute GASSATA whenever possible (every 18 seconds for example) by considering the whole audit trail from the beginning of the user's session),

(v) - GASSATA does not locate attacks in the audit trail. It only gives a presumptive set of attacks present in a given audit session. (Mé [17] proposed that the audit trail must be investigated later by the SSO to precisely locate the attacks).

3.7 Conclusion

We have presented in this chapter the different formalizations of the audit trail analysis introduced by Mé [18]. This problem is NP-Complete even if we are considering the simple version PASFAS which seems to be NP-Complete. The following chapter presents a new formalization of the audit trail analysis problem called PASFAS-G (G for Generalized) which brings solutions to the first three shortcomings of GASSATA.

Chapter 4

PASFAS-G

Generalized-Simplified Security

Audit Trail Analysis Problem

4.1 Introduction

This chapter examines a new formalization called PASFAS-G of the audit trail analysis problem which is simple and robust to solve the maximum problems which may be encountered when using GASSATA. Its programming and implementation are very easy. This model may also find all the possible attacks as those of GASSATA by using the different lemmas and definitions discussed in the following sections.

4.2 PASFAS-G

We have used the same matrixes mentioned in the previous sections *(AE, I(for H) and O)* and have introduced a new formalization to the problem.

Our formulation is as follows:

$$\begin{cases} Max(\sum_{i=1}^{Na} I_i) \\ s.t.\ (I_i \times AE^i \leq O)_{i=1,..,Na} \end{cases} \quad (4.1)$$

Where AE^i is the i^{th} column of AE, which represents the i^{th} attack scenario.
I is an Na-dimensional positive integer vector.

It is clear that the system *(4.1)* is not an *NP-Complete* problem and its resolution is very easy:

$$I_i = \min_{j=1,..,Ne} \left\lfloor \frac{O_i}{AE_{ji}} \right\rfloor,\ i=1,..,Na \text{ for all } AE_{ji} \neq 0 \quad (4.2)$$

The following section introduces the different lemmas and definitions that may be used in order to obtain the same results as those of GASSATA in real time.

4.3 From PSFAS-G to GASSATA

4.3.1 Lemma 4.1.

Each detected attack scenario by GASSATA is also detected by PASFAS-G.

Proof 4.1.

Let AE_d $(d \in \{1,..,Na\})$*be an* attack scenario detected by GASSATA. In this case, assume that the optimum H_{opt} vector obtained by GASSATA is ${}^{t}H_{opt}=\left(\underset{1}{H_1}\ \underset{2}{H_2}\ \cdots\ \underset{d}{1}\ \cdots\ \underset{Na}{H_{Na}}\right)$

Therefore the following vector ${}^{t}H=\left(\underset{1}{0}\ \underset{2}{0}\ \ldots\ \underset{d}{1}\ \ldots\ \underset{Na}{0}\right)$ also verifies the constraints of the system mentioned in system (3.4) (i.e. $(AE.H) \leq O$).

So, by substituting H with its value in system (3.4), we obtain the following system:

$$\begin{cases} H_d \cdot AE_{id} \leq O_i \quad for\ i=1..Ne \\ H_d = 1 \end{cases} \qquad (3.13) \qquad \Rightarrow \left\lfloor \frac{O_i}{AE_{id}} \right\rfloor \geq 1 \quad AE_{id} \neq 0 \quad i=1..Ne \qquad (4.4)$$

so,

$\underset{i=1,...,Ne}{min}\left\lfloor \frac{O_i}{AE_{id}} \right\rfloor \geq 1 \quad AE_{id} \neq 0$ which means that $I_d \geq 1$

Hence, each detected attack scenario by GASSATA is detected by PASFAS-G.

Lemma 4.1 may also be announced as follows:
An attack scenario, which is not detected by PASFAS-G, may not be detected by GASSATA.

4.3.2 Lemma 4.2

If an attack scenario is present in the audit trail, it is detected by PASFAS-G.

Proof 4.2

Assume that there is a presence of an attack scenario in the audit trail. Let AE^d be this attack, then

$$AE_{id} \leq O_i \quad \forall i \in \{1,...,Ne\} \qquad (4.5)$$

$$\frac{O_i}{AE_{id}} \geq 1 \quad \forall i \in \{1,...,Ne\}\ and\ AE_{id} \neq 0 \qquad (4.6)$$

Therefore

$$I_d = \min_{i=1..Ne} \left\{ \left\lfloor \frac{O_i}{AE_{id}} \right\rfloor \right\} \geq 1 \qquad (4.7)$$

Which means that PASFAS-G detects any *present* attack in the audit trail.

4.3.3 Lemma 4.3

If there is not any attack in the audit trail, then GASSATA and PASFAS-G find the same optimal solution corresponding to an *Na-null* vector.

Proof 4.3

Consider that there are no attacks in the audit trail, hence

$$\forall j \in \{1,...,Na\},\ \exists i \in \{1,...,Ne\} \ such\ that\ AE_{ij} > O_i \qquad (4.8)$$

i.e.

$$\forall j \in \{1,...,Na\},\ \exists i \in \{1,...,Ne\} \ for\ which \left\lfloor \frac{O_i}{AE_{ij}} \right\rfloor < 1 \ and\ AE_{ij} \neq 0 \qquad (4.9)$$

we have O_i , $AE_{ij} \in \angle^2$

hence,

$$\left\lfloor \frac{O_i}{AE_{ij}} \right\rfloor = 0 \text{ then } I_j = \min_{j=1..Ne} \left\{ \left\lfloor \frac{O_i}{AE_{ij}} \right\rfloor \right\} = 0 \ for\ all\ j \in \{1,...,Na\} \qquad (4.10)$$

which means that $I_{opt}=0^{Na}$
So, when there are no attacks in the audit trail, PASFAS-G finds a null vector as an optimum vector and from lemma 4.1, GASSATA also finds the *Na-null-vector* as an optimum vector.

Definitions

Independent event

We call *independent event* each event appearing only in one attack scenario model. This event is called *proper event* of the attack scenario in which it appears.
A non-independent event is called *dependent event*.

Independent attack scenario

An *independent* attack scenario is an attack scenario AE^i verifying the following condition:

$$if\ i \in \{1,...,Na\} and\ if\ AE_{ji} \neq 0 \Rightarrow AE_{ki} = 0 \quad \forall j,k \in \{1,...,Ne\} and\ k \neq j \qquad (4.11)$$

if an attack scenario does not verify condition *(4.21)*, it is called *dependent* attack scenario.

Equivalent Attacks-Events matrix $AE^{(E)}$

An equivalent attacks-events matrix $AE^{(E)}$ is an Attacks-Events matrix where only dependent scenarios are present and their corresponding events. It is obtained from the Attacks-Events matrix AE by deleting the independent attack scenarios and their proper events.

Reducted Attacks-Events matrix

The reducted attacks-events matrix $AE^{(R)}$ is obtained from the attacks-events matrix AE by considering only all dependent events.

> ***Definition 2.*** The reducted attacks-events matrix $AE^{(R)}$ is a submatrix of AE containing only (all) dependent events and dependent scenarios.

Simple Attacks-Events matrix

Let U and V be two N-dimensional vectors. We introduce a new comparison operator $\prec\prec$:

$$U \prec\prec V \text{ iff } \forall i=1..N \text{ and } V_i \neq 0 \text{ then } U_i \leq V_i \qquad (4.12)$$

we call a simple attacks-events matrix a matrix A_{mn} where all its column vectors A^i $i=1, .., N$ verify the following conditions:

$$1)\ \text{for all } i\neq j \quad i,j\in\{1,..,N\}^2 \quad \text{we have} \quad A^i \prec\prec A^j \quad \text{or} \quad A^j \prec\prec A^i \qquad (4.13)$$

and

$$2)\ \text{for all } i\neq j,\ j\neq k \text{ and } i\neq k \quad i,j,k\in\{1,...,N\}^3 \text{ and } A^i, A^j \text{ and } A^k \text{ are dependent scenarios} \quad \text{if } A^i \prec\prec A^j \text{ and } A^j \prec\prec A^k \text{ then } A^i \prec\prec A^k \qquad (4.14)$$

4.3.4 Lemma 4.4

PASFAS may be reduced to the following problem

$$\begin{cases} Max \sum_{i=1}^{Na^{(E)}} R_i H_i^{(E)} & R_i = 1,\ i=1..Na^{(E)} \\ s.t. \quad AE^{(E)} \times H_i^{(E)} \leq O^{(E)} & \\ and \quad H_i^{(ind)} = \min_{j=1..Nz} \{O_j \div AE_{ji}^{ind}\} & i=1,...,Na^{ind} \text{ and } AE_{ji}^{ind} \neq 0 \end{cases} \qquad (4.15)$$

where

$$a \div b = \begin{cases} 1 \text{ if } a \geq b \\ 0 \text{ otherwise} \end{cases} \qquad a,b \in N^2 \qquad (4.16)$$

$Na^{(E)}$ is the number of dependent scenarios and $Na^{(ind)}$ is the number of independent scenarios ($Na^{(E)} + Na^{(ind)} = Na$). $AE^{(E)}$ corresponds to the equivalent attacks-Events matrix.

Proof 4.4

Let AE^{ind1} be an independent scenario of the attacks-Events matrix AE, and let $ep_{1..k}$ its proper events (k=number of these events).

PASFAS may be written as follows:

$$
\begin{cases}
max \sum_{i=1}^{Na} H_i \\
s.t \\
\sum_{\substack{i=1 \\ i \neq ind_1}}^{Na} AE_{1,i} \times H_i + 0 \times H_{ind_1} \leq O_1 \\
\sum_{\substack{i=1 \\ i \neq ind_1}}^{Na} AE_{2,i} \times H_i + 0 \times H_{ind_1} \leq O_2 \\
\quad \vdots \\
\begin{cases}
AE_{p1,ind_1} \times H_{ind_1} + \sum_{\substack{i=1 \\ i \neq ind_1}}^{Na} 0 \times H_i \leq O_{p1} \\
AE_{p2,ind_1} \times H_{ind_1} + \sum_{\substack{i=1 \\ i \neq ind_1}}^{Na} 0 \times H_i \leq O_{p2} \\
\quad \vdots \\
AE_{pk,ind_1} \times H_{ind1} + \sum_{\substack{i=1 \\ i \neq ind_1}}^{Na} 0 \times H_i \leq O_{pk}
\end{cases} \\
\quad \vdots \\
\sum_{\substack{i=1 \\ i \neq ind_1}}^{Na} AE_{Ne,i} \times H_i + 0 \times H_{ind_1} \leq O_{Ne}
\end{cases}
\qquad (4.17)
$$

Problem (4.17) is equivalent to the problem:

$$
\begin{cases}
\begin{cases}
\max \sum_{\substack{i=1 \\ i \neq ind_1}}^{Na} H_i \\
s.t. \\
AE^{\{-ind_1\}} \times H^{\{-ind_1\}} \leq O^{\{-ind_1\}}
\end{cases} \\
and \\
\begin{cases}
\max H_{ind_1} \\
s.t. \\
AE_{p1,ind_1} H_{ind_1} \leq O_{p1} \\
AE_{p2,ind_1} H_{ind_1} \leq O_{p2} \\
\quad \vdots \\
AE_{p2,ind_1} H_{ind_1} \leq O_{pk}
\end{cases}
\end{cases}
\qquad (4.18)
$$

where $AE^{\{-ind1\}}$ corresponds to the attacks-events matrix AE in which the scenario AE^{ind1} and its proper events are deleted.
$O^{\{-ind1\}}$ corresponds to the observed matrix in which the corresponding events of the scenario AE^{ind1} are omitted.
$H^{\{-ind1\}}$ corresponds to the hypothesis vector of all potential attacks but AE^{ind1}.
Problem (4.28) is equivalent to the problem:

$$
\begin{cases}
\max \sum_{\substack{i=1 \\ i \neq ind_1}}^{Na} H_i \\
s.t. \\
AE^{\{-ind_1\}} H^{\{-ind_1\}} \leq O^{\{-ind_1\}} \\
H_{ind_1} = \min_{j=1\ldots k} \left\{ O_{pj} \div AE_{pj,ind_1} \right\}
\end{cases}
\qquad (4.19)
$$

and so on, using the same procedure to the other independent attack scenarios we obtain the problem (4.15) i.e.:

$$
\begin{cases}
Max \sum_{i=1}^{Na^{(E)}} R_i H_i^{(E)} \qquad R_i = 1, \; i = 1..Na^{(E)} \\
s.t. \quad AE^{(E)} \times H_i^{(E)} \leq O^{(E)} \\
and \quad H_i^{(ind)} = \min_{j=1..Ne} \left\{ O_j \div AE_{ji}^{ind} \right\} \quad i = 1, \ldots, Na^{ind} \; and \; AE_{ji}^{ind} \neq 0
\end{cases}
\qquad (4.15)
$$

C.Q.F.D.

remark

If all attack scenarios are independent, problem (4.25) may be reduced to the following problem:

$$H_i = \min_{j=1..Ne}\{O_j \div AE_{ji}\} \quad i=1,...,Na \quad and \quad AE_{ji} \neq 0 \qquad (4.20)$$

4.3.5 Lemma 4.5

If $AE^{(R)}$ is a simple attacks-events matrix, then $AE^{(E)}$ and AE are also simple attacks-events matrixes.

Proof 4.5

a) If AE^i, $i \in \{1, .., Na\}$, is an independent attack scenario then from (4.11) condition (4.13) is verified.

b) If AE^i, $i \in \{1, .., Na\}$, is a dependent attack scenario then for all independent events appearing in $AE^{(E)}$, conditions (4.13) and (4.14) are verified for these independent events.

So from **a)**, **b)** and if $AE^{(R)}$ is a simple attacks-events matrix (i.e. dependent events verify conditions (4.13) and (4.14)) then AE and $AE^{(E)}$ are simple matrixes.

4.4 From PASFAS-G to GASSATA Summary

In this section, we suggest an efficient algorithm that finds the optimal solution of GASSATA in a bout (short period). This algorithm is using the optimal solution generated by PASFAS-G and the different lemmas introduced in the previous section.

We address the issue of using the solutions of PASFAS-G to generate those of GASSATA because the execution time of PASFAS-G is far behind that of GASSATA.

Algorithm "*From PASFAS-G to GASSATA*"

Step1: transform PASFAS to an equivalent problem as stated in lemma 4.4.

if all attack models are independent ***then***

the corresponding problem is $H_i = \min_{j=1..Ne}\{O_j \div AE_{ji}\} \quad i=1..Ne \quad and \quad AE_{ji} \neq 0$

where $a \div b = \begin{cases} 1 \; if \; a \geq b \\ 0 \; otherwise \end{cases} \qquad a,b \in N^2$

hence the solution is

$$H = \begin{pmatrix} \min\limits_{j=1..Ne}\{O_j \div AE_{j1}\} \\ \min\limits_{j=1..Ne}\{O_j \div AE_{j2}\} \\ \vdots \\ \min\limits_{j=1..Ne}\{O_j \div AE_{jNa}\} \end{pmatrix} \quad for \; all \; AE_{ji} \neq 0 \quad i=1..Na$$

go to end.

Step2: if the equivalent problem is:

$$\begin{cases} Max \sum_{i=1}^{Na^{(E)}} R_i H_i^{(E)} & R_i = 1,\ i = 1..Na^{(E)} \\ s.t. \quad AE^{(E)} \times H_i^{(E)} \le O^{(E)} & \\ and \quad H_i^{(ind)} = \min_{j=1..Ne} \left\{ O_j \div AE_{ji}^{ind} \right\} & i = 1,...,Na^{ind} \text{ and } AE_{ji}^{ind} \neq 0 \end{cases} \qquad (4.15)$$

H_i^{ind} are immediately identified.

The remaining problem consists of resolving the following linear program

$$\begin{cases} Max \sum_{i=1}^{Na^{(E)}} R_i H_i^{(E)} & R_i = 1,\ i = 1..Na^{(E)} \\ s.t. \quad AE^{(E)} \times H_i^{(E)} \le O^{(E)} & \end{cases} \qquad (4.21)$$

Step3: perform PASFAS-G to problem (4.21) with $\div$ division ($H_i \in \{0,1\}$)

where $a \div b = \begin{cases} 1 \text{ if } a \ge b \\ 0 \text{ otherwise} \end{cases} \qquad a,b \in N^2$

i.e.:

$$H_i^{(E)} = \min_{j=1..Ne} \left\{ O_j \div AE_{ji} \right\} \quad AE_{ji} \neq 0 \quad i \in \{1, ..., Na^{(E)}\}$$

if $I_i^{(E)} = 0$ *for all* $i = 1..Na^E$ *then from lemma* $3.2\,a)$ ${}^t H_{opt}^E = (\underset{1}{0}, \underset{2}{0}, ..., \underset{Na^E}{0})$ ***go to end***

step4: if only one element of $I^{(E)}$ is equal to 1

Let H_k be this element, i.e. ${}^t H_{opt}^E = (\underset{1}{0}, \underset{2}{0}, ..., \underset{k}{1}, ..., \underset{Na^E}{0})$

then ${}^t H_{opt}^{(E)\,GASSATA} = (\underset{1}{0}, \underset{2}{0}, ..., \underset{k}{1}, ..., \underset{Na^E}{0})$

go to end

step5:

consider only the attack scenarios that are detected by PASFAS-G (i.e. $H_i^{(E)} = 1$ by performing PASFAS-G to system (4.18) and there corresponding dependent events.
Let AE^{dep} ,H^{dep} and O^{dep} be, respectively, the attacks-events matrix containing these dependent scenarios and events, the corresponding H vector of the remaining dependent attacks scenarios and the observed audit matrix corresponding to the dependent events appearing in AE^{dep}.

if AE^{dep} is a simple attacks-events matrix ***then***
perform ***algorithm simple*** to AE^{dep}, H^{dep} and O^{dep}
else
perform GASSATA to AE^{dep} ,H^{dep} and O^{dep}
endif

end "From PASFAS-G to GASSATA".

Algorithm simple AE^{dep}, H^{dep} and O^{dep}

Let AE^{dep} be a simple matrix
Reorganize AE^{dep} so that for all $i<j$ then $AE^i \prec\prec AE^j$
$K=1$
While $k \neq Na^{dep}$ and $AE^{dep\,k} \leq O^{dep}$ ***do***
$O^{dep} \leftarrow O^{dep} - AE^{dep\,k}$
$H_k^{dep} \leftarrow 1$
$k \leftarrow k+1$
endwhile

end Algorithm simple

Remarks

In step five, we use GASSATA when the matrix AE^{dep}, containing potential attacks detected by PASFAS, is not a simple attacks events matrix. In the case where the number of columns of AE^{dep} is not great enough, it is preferable to test all possible values of H^{dep} from up to down (i.e. first, considering all attacks present (i.e. the unit vector), etc.).

On the other hand, when the number of potential attacks detected by PASFAS-G is great enough, the audited user to which corresponds the current observed matrix O is considered as a dangerous one. In this case, we suggest execute PASFAS-G in real time; i.e. execute PASFAS-G every time there is a new generation of audit records.

When using algorithm "*simple"*, if there is more than one optimum solution, the optimal solution generated by GASSATA may or may not be the same as that of *"From PASFAS-G to GASSATA"* algorithm.

4.5 Summary

We have presented, in this chapter, our new formalization (PASFAS-G) of the security audit trail analysis problem. This problem is not *NP-Complete* and its resolution is done in real time *(≈0sec.)*. However, its true test is an evaluation of its implementation running under *"live"* conditions. On the other hand, we have demonstrated with the different lemmas that we may find the same results, in real time, as those generated by GASSTA by using PASFAG-G.

The next chapter introduces some simulation results and shows that there is no need to use GASSATA (and then Gas) in the different simulation experiments performed by Mé [17-18].

" Je montre mon travail tout en sachant qu'il n'est qu'une partie de la vérité, et je le montrerai même en le sachant faux, parce que certaines erreurs sont des étapes vers la vérité "

- Robert Musil.

Chapter 5

Experimental Results

In this chapter, we describe our experiments on the scenario models defined in [18] using AIX system under IBM RS 6000 machines. Mé [18] has defined 28 attack scenario models which are possible and realistic under AIX. Among these models, there are some attacks considered as litigious (less dangerous) or abnormal. These attack scenarios are translated into event sequences generated by the AIX audit sub system. This will help the construction of the Attacks-Events matrix AE.

The audit trails we will analyze consist of some simulated behaviors [18] during 30 *mn* to which we have integrated some defined attacks scenarios. This will help us to test our new formalization PASFAS-G and the derived algorithm "from PASFAS-G to GASSATA" whether it can or cannot detect all present attacks and will allow us to make a comparison between our algorithm and formalization to those of Mé [18] according to execution time, number of real detected attacks … etc.

We have not tested our system on a real network with real users. This is due *(i)* to the non availability of this type of network in our environment, and *(ii)* till now, Mé [17-18, 37-46] has not reported any realistic experiments and all his work is based on simulated behaviors and so have we.

The following Attacks-Events Matrix [17-18] *AE*, consisting of 24 attacks and 28 AIX auditable events is used in our experiments. The construction of the different attack scenarios is defined in [17-18].

	Attack type																									
	a_2	l_1	l_2	l_3	l_4	a_3	a_4	a_5	l_5	l_6	l_7	l_8	a_6	a_7	a_8	a_9	l_9	l_{10}	l_{11}	a_{10}	l_{12}	l_{13}	l_{14}	l_{15}
User_login fail	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
User log (23h to 6h)	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Short_Session	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Use_SU OK	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
User_SU fail	.	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Who,w,finger,…	.	3	.	.	.	.	.	.	8	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
More,pg,cat,…	.	.	.	.	.	5	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	5	.	.
ls OK	.	.	.	.	.	30	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Ls fail	.	.	.	.	.	.	5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Df,hostname,uname	.	.	.	.	.	.	.	.	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Arp,netstat,ping	.	.	.	.	.	.	.	.	.	.	2	.	.	.	.	.	.	.	.	.	.	.	.	.
Ypcat	.	.	.	.	.	.	.	.	.	.	.	3	.	.	.	.	.	.	.	.	.	.	.	.
Lpr	.	.	.	.	.	.	.	.	.	.	.	.	.	.	10	1	.	.	.	.	.	.	.	.
rm, mv	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.
Ln	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.
Whoami, id	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	4
rexec,rlogin,rsh	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.
Proc_Execute	.	3	.	.	.	35	5	.	8	3	2	3	.	.	10	3	.	300	.	2	.	5	.	4
Proc_SetPetri	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	100	.	.	.	.	.
File_Open fail	.	.	.	.	.	.	.	5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
File_Open fail cp	.	.	.	.	.	.	.	.	.	.	.	.	.	10	.	.	.	.	.	.	.	.	.	.
File_Open .netrc	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.
File_Read lpr	.	.	.	.	.	.	.	.	.	.	.	.	.	.	10	.	.	.	..	.	.	.	.	.
File_Read passwd...	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	5	.	.
File_write passwd,… fail	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.
File_write cp Ok	.	.	.	.	.	.	.	.	.	.	.	.	30	.	.	.	.	.	.	.	.	.	.	.
File_Unlink rm	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	50	.	.	.	.	.	.	.
File_mode	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3	.	.	.

Figure 5.1.: The attacks-Events (from [17-18]) matrix used to validate the experiments.

5.1 Comparative study between GASSATA and PASFAS-G:

To show the robustness of PASFAS-G in resolving the first three GASSATA's shortcomings described in the previous chapter, we use a (*5x4; Ne=5, Na=4*) Attacks-Events matrix. This is also true when using the above *AE* (28x24) Attacks-Events matrix.

Example :

$$let\ AE=\begin{pmatrix}5&0&1&0\\3&6&2&0\\1&2&4&0\\2&1&0&8\\2&0&0&0\end{pmatrix}\ and\ O=\begin{pmatrix}10\\8\\5\\9\\4\end{pmatrix}$$

With *GASSATA* the optimal *H* vector is

$$H_{max1}=\begin{pmatrix}1\\0\\1\\0\end{pmatrix}\ \text{or}\ H_{max2}=\begin{pmatrix}0\\1\\0\\1\end{pmatrix}$$

while the solution to this problem with our model is

$$H_{max}=\begin{pmatrix}2\\1\\1\\1\end{pmatrix}$$

From equation (3) our example may be resolved easily as follows:

$$H_1=\min(O_1\div AE_{11},O_2\div AE_{21},O_3\div AE_{31},O_5\div AE_{51})\ \{AE_{41}=0\}$$
$$=\min(10\div5,8\div3,5\div1,4\div2)=2$$

$$H_2=\min(O_2\div AE_{22},O_3\div AE_{32},O_4\div AE_{42})\ \{AE_{12}=AE_{52}=0\}$$
$$=\min(8\div6,5\div2,9\div1)=1$$

and so on. We finally find $H_{max}=\begin{pmatrix}2\\1\\1\\1\end{pmatrix}$

This example shows that PASFAS-G not only detects all present attacks in the current audit trail (*O*) but also finds the number of duplicated attacks (attack 1 is duplicated).

However, GASSATA may find the first and the third attack or the second and the forth attack but can never find the four attacks simultaneously because there are some events that occur simultaneously in the four attacks scenarios (shortcoming *(ii)* mentioned in the previous chapter).

Assume that the user has, actually, duplicated the first attack, GASSATA not only cannot find this duplication but also cannot at all find this attack if the resulted optimum *H* vector is H_{max2}.

5.2 Efficient results with our simplified model

Figure 5.2 shows the execution time in seconds versus number of attacks in the Attack-Events matrix : a) when using *GASSATA*. b) when using our simplified model.

Number of attacks	Execution time
24	18 sec.
27	24 sec.
40	32 sec.
50	38 sec.
70	49 sec.
100	104 sec.
150	300 sec.
200	625 sec.

a)

Number of attacks	Execution time
24	≈ 0 sec.
400	≈ 0 sec.
1000	0.05 sec.
3000	0.06 sec.
5000	0.11 sec.
7000	0.17 sec.
10000	0.22 sec.
15000	0.38 sec.

b)

Figure 5.2.: Execution time **a)**when using *GASSATA* (from [17-18]). **b)** when using PASFAS-G.

This comparison is not performed on a real network with real users. It is only a simulation of some users' behavior after introducing some known attacks in the audited user's behavior because till now the author [1,2] has not reported a real experiment using GASSATA with real users. In addition, the audit trails we were provided do not present any attack defined in the used attacks-events matrix [1,2]. However, in the cases where there is not any attack in the audit trail, there are no duplicated attacks, or dependent attacks, PASFAS-G finds the optimal *Na-vector* in real time which is not true when using GASSATA (see Figure 5.2).

Figure 4.2 shows that the detection execution time when using GASSATA (a simple mean of 10 execution times from [18]) highly increases when the number of attacks augments. However, the execution time is somewhat stable, when using PASFAS-G, even if we consider thousands of attacks.

The optimal solution of PASFAS-G is different from that of GASSATA. However, all detected attacks by GASSATA are also detected by PASFAS-G (lemma 4.1 of chapter 4). To, really, appreciate the results of PASFAS-G, real experiments with real users shall be performed to compare the rate of false alarms between the two systems.

For this reason, we will try, in the following paragraph, to perform the "***From PASFAS-G to GASSATA***" algorithm with the Attacks-Events *AE* (28x24) matrix presented above.

5.3 Applying "*From PASFAS-G to GASSATA*" algorithm

Step 1,2: transform PASFAS to an equivalent problem as stated in lemma 4.4.

There are 11 independent scenarios: a_2 ,l_2 ,l_3 ,l_4 ,a_5, a_6, a_7, l_9, l_{11}, l_{12} and l_{14}, and 13 dependent scenarios: l_1, a_3, a_4, l_5, l_6, l_7, l_8, a_8, a_9, a_{10}, l_{13} and l_{15}.

The following formula detects whether one of the eleven independent attacks is present or not.

$$H=\begin{pmatrix} \min\limits_{j=1..Ne}\left\{O_j \div AE_{j1}\right\} \\ \min\limits_{j=1..Ne}\left\{O_j \div AE_{j2}\right\} \\ \vdots \\ \min\limits_{j=1..Ne}\left\{O_j \div AE_{jNa}\right\} \end{pmatrix} \quad for\ all\ AE_{ji}\neq 0\ \ i=1..Na$$

The Equivalent Attacks-Events matrix is the following:

$AE^{(E)}=$

	l_1	a_3	a_4	l_5	l_6	l_7	l_8	a_8	a_9	l_{10}	a_{10}	l_{13}	l_{15}
Use_SU OK	3	.	.	.	.	.	.	.	.	.	.	.	.
Who,w,finger,…	3	.	.	8	.	.	.	.	.	.	.	.	.
More,pg,cat,…	.	5	.	.	.	.	.	.	.	.	1	5	.
ls OK	.	30	.	.	.	.	.	.	.	.	.	.	.
Ls fail	.	.	5	.	.	.	.	.	.	.	.	.	.
Df,hostname,uname	.	.	.	.	3	.	.	.	.	.	.	.	.
Arp,netstat,ping	.	.	.	.	.	2	.	.	.	.	.	.	.
Ypcat	.	.	.	.	.	.	3	.	.	.	.	.	.
Lpr	.	.	.	.	.	.	.	10	1	.	.	.	.
rm, mv	.	.	.	.	.	.	.	.	1	.	.	.	.
Ln	.	.	.	.	.	.	.	.	1	.	.	.	.
Whoami, id	.	.	.	.	.	.	.	.	.	.	.	.	4
rexec,rlogin,rsh	.	.	.	.	.	.	.	.	.	.	1	.	.
Proc_Execute	3	35	5	8	3	2	3	10	3	300	2	5	4
File_Open .netrc	.	.	.	.	.	.	.	.	.	.	1	.	.
File_Read lpr	.	.	.	.	.	.	.	10	.	.	.	.	.
File_Read passwd...	.	.	.	.	.	.	.	.	.	.	.	5	.

Figure 5.3.: The Equivalent Attacks-Events matrix $AE^{(E)}$.

The Reducted Attacks-Events $AE^{(R)}$ matrix is the following:

$AE^{(R)}=$

	l_1	a_3	a_4	l_5	l_6	l_7	l_8	a_8	a_9	l_{10}	a_{10}	l_{13}	l_{15}
Who,w,finger,…	3	.	.	8	.	.	.	.	.	.	.	.	.
More,pg,cat,…	.	5	.	.	.	.	.	.	.	.	1	5	.
Lpr	.	.	.	.	.	.	.	10	1	.	.	.	.
Proc_Execute	3	35	5	8	3	2	3	10	3	300	2	5	4

Figure 5.4.: The Reducted Attacks-Events matrix $AE^{(R)}$.

This matrix is simple because it verifies the two conditions 4.*13* and 4.*14*.

The rearranged Reducted Attacks-Events matrix $AE^{(R)}$ (using simple algorithm described in the previous chapter) is:

Rearranged $(AE^{(R)})=$

	l_6	a_{10}	l_7	l_8	a_9	l_1	l_{15}	a_4	l_{13}	l_5	a_8	A_3	l_{10}
Who,w,finger,…	.	.	.	.	.	3	.	.	.	8	.	.	.
More,pg,cat,…	.	1	.	.	.	.	.	.	5	.	.	5	.
Lpr	.	.	.	.	1	.	.	.	.	.	10	.	.
Proc_Execute	2	2	3	3	3	3	4	5	5	8	10	35	300

Figure 5.5.: The rearranged Reducted Attacks-Events matrix $AE^{(R)}$.

By using the different matrixes cited above, and performing steps 3 and 4, then "*algorithm simple*" we get the same optimal solutions as those of [18] (pp. 97, 130) but in real time (≈ 0 sec.).

From this, we have demonstrated that Genetic Algorithms are not necessary to solve this problem, particularly, in the case of the Attacks-Events matrix (24x28) used in [18] which is a simple one (lemma 4.5).

5.4 Conclusion

In this chapter, we have applied the "From PASFAS-G to GASSATA " algorithm and have obtained all the optimal results as those of [18] in real time by using the different lemmas described in the previous chapter. However, to demonstrate the robustness of PASFAS-G, real experiment shall be performed to compare it with GASSATA especially according to false alarms.

" A new idea is delicate. It can be killed by a sneer or a yawn; it can be stabbed to death by a quip, and worried to death by a frown on the right man's brow "

- Charlie Brower.

Part 2

Chapter 6

Towards Using PCA

in Intrusion Detection

We introduce a novel anomaly intrusion detection method based on Principal Component Analysis (PCA). This approach functions by projecting user profiles onto a feature space that spans the significant variations among known user profiles. The significant features are known as *eigenprofiles* because they are the eigenvectors (principal components) of the set of user profiles. The projection operation characterizes a user profile by a weighted sum of the *eigenprofile* features, as so to detect whether a user profile is anomalous, it is sufficient to compare its weights to those of known user profiles. Some advantages of this new approach are that *(i)* it provides for the ability to learn and later determine whether a new profile does or does not correspond to those of known users, *(ii)* its implementation is very easy in any system having the audit mechanism, and *(iii)* it is robust and provides high rates of detection.

Principal Component Analysis (PCA) [30] has proven to be an exceedingly popular technique for dimensionality reduction and is discussed at length in most texts on multivariate analysis. Its many application areas include data compression, image analysis, visualization, pattern recognition and time series prediction.

The most common definition of PCA, due to Hotelling (1933), is that, for a set of observed *d-dimensional* data vectors $\{v_i\}, i \in \{1,...,N\}$, the q principal axes $w_j, j \in \{1,...,q\}$, are those orthonormal axes onto which the retained variance under projection is maximal. It can be shown that the vectors w_j are given by the q dominant eigenvectors (i.e. those with the largest associated eigenvalues) of the simple covariance matrix $C = \sum_i \frac{(v_i - \bar{v})(v_i - \bar{v})^T}{N}$ such that $Cw_j = \lambda_j w_j$ and where $\bar{v}$ is the simple arithmetic mean. The vector $u_i = C^T(v_i - \bar{v})$, where $W = (w_1, w_2, ..., w_q)$ is thus a *q-dimensional* reduced representation of the observed vector v_i .

6.1 Introduction

Anomaly detection, as described in chapter 1, consists of first establishing the normal behavior profiles for users, programs or other resources of interest in a system, and observing the actual activities as reported in the audit data to ultimately detect any significant deviations from these profiles[3]. (for more details of the current existing tools see [9, 10,12, 36, 31,32, 33, 34, 35]).

In this chapter, we investigate an *eigenprofile* approach, based on Principal Component Analysis, for anomaly intrusion detection which involves mainly in the following sections.

We first derive some computational feasible formula to find eigenprofiles, and then we describe the intrusion detection algorithm based on these eigenprofiles. The first preliminary experimental results using this algorithm are very interesting.

6.2 The Eigenprofiles Approach

Much of the previous work on anomaly intrusion detection has ignored the issue of just what measures of the user profile stimulus are important for intrusion detection. This suggested us that an information theory approach coding and decoding user behaviors may give new information content of user behaviors, emphasizing the significant features. Such features may or may not be directly related to the actual used metrics such as CPU consumed time, the number of commands passed by the user during a login session, the number of each event type audited during an interval time, …etc.

In the language of information theory, we want to extract the relevant information in a user profile, encode it as efficiently as possible, and compare one behavior encoding with a database of user behaviors encoded similarly. A simple approach to extracting the information contained in a profile is to somehow capture the variation in a collection of user behaviors and use this information to encode and compare user behavior profiles.

In mathematical terms, we wish to find the principal components of the distribution of the behaviors, or the eigenvectors of the covariance matrix of the set of user profiles, treating a behavior as a point (or vector) in a space of a dimension equal to the number of the different metrics used. The eigenvectors are ordered, each one accounting for a different amount of the variation among the user behaviors.

These eigenvectors can be thought of as a set of features that together characterize the variation between user behaviors. Each behavior location contributes more or less to each eigenvector which we call *eigenprofile*.

Each user profile can be represented exactly in terms of linear combination of the eigenprofiles. Each profile can also be approximated using only the best eigenprofiles-those that have the largest eigenvalues, and which therefore account for the most variance within the set of user profiles. The best *N* eigenprofiles span an *N-dimensional* subprofile – *profile space* – of all possible profiles.

A user normal profile, in our system, consists of a set of statistical measures such as that proposed by Denning [3] :

Account of some numerically quantifiable aspect of observed behavior. For example, the amount of CPU used, the number of failed login during an interval of time, the number of each type of commands passed by the user during an interval of time, number of password failures during a minute, number of files opened during a period, number of application executed during a session…etc.

The set of these measures are collected in an *N-dimensional* vector called a *profile vector* representing the profile of the user during the login session or a chosen interval of time, where *N* is the number of the statistical measures mentioned above. So if Γ is a profile that corresponds to a behavior of a certain user, then we can write

$$\Gamma = \begin{pmatrix} m_1 \\ m_2 \\ . \\ . \\ . \\ m_N \end{pmatrix} \qquad (6.1)$$

where $m_i, i=1..N$ corresponds to the metrics cited above.
In most cases, the users' profiles are somehow very similar and they can be described by some *"basic profile vectors"*.

This approach to anomaly intrusion detection involves the following initialization procedure:

1. acquire (audit) an initial set of user profiles (the habitual users of the network for example, this set is called the training set).
2. calculate the *eigenprofiles* from the training set, keeping only *M* profiles that correspond to the highest eigenvalues. These *M* profiles define the profile space. As new users are experienced or new functions are assigned to the usual users, the eigenprofiles can be updated or recalculated.
3. calculate the corresponding distribution in *M-dimensional* weight space for each known user profile, by projecting their profiles onto the "profile space".

This procedure can also be performed from time to time whenever there is free excess computational capacity.

Having initialized the system, the following steps are then used to detect intrusion from the audit trail whenever a new profile is introduced:

1. calculate a set of weights based on the input profile and the *M* eigenprofiles by projecting the input profile onto each *eigenprofile*.
2. Determine whether the profile corresponds to one of the training set by checking to see if the profile is sufficiently close to "*profile space*".
3. if there is a profile in the training set closest to the input one, verify if the current user is the one he claims to be or a masquerader.
4. (optional) update the eigenprofiles whenever it seems necessary.
5. (optional) if the same intrusive profile is encountered several times, verify if it corresponds to an actual user of the network and if it is true, calculate its characteristic weight and incorporate into the known behaviors.

6.3 Calculating the Eigenprofiles

Let the training set of profile vectors be $\Gamma_1, \Gamma_2, \ldots, \Gamma_M$. The average profile of this set is defined by:

$$\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i \qquad (6.2)$$

Each profile differs from the average by:

$$\Phi_i = \Gamma_i - \Psi, \quad i=1..M \qquad (6.3)$$

The eigenprofiles are the eigenvectors of the covariance matrix C

where

$$C=\frac{1}{M}\sum_{i=1}^{M}\Phi_i\Phi_i^T=A.A^T \quad (6.4)$$

and

$$A=\frac{1}{\sqrt{M}}\left[\Phi_1\ \Phi_2\ldots\Phi_M\right] \quad (6.5)$$

Let: U_k be the k^{th} eigenvector of C, λ_k the associated eigenvalue and $U=\left[U_1\ U_2\ldots U_M\right]$ the matrix of these eigenvectors (*eigenprofiles*). Then

$$CU_k=\lambda_k U_k \quad (6.6)$$

such that

$$U_k^t U_n=\begin{pmatrix}1 & if\ k=n\\ 0 & if\ k\neq n\end{pmatrix} \quad (6.7)$$

The feature vector of the k^{th} profile vector is:

$$\Omega_i=U^T\times\Phi_i=\begin{pmatrix}\omega_1\\ \omega_2\\ .\\ .\\ .\\ \omega_M\end{pmatrix} \quad (6.8)$$

The weights $\omega_i, i=1..M$ describes the contribution of each *eigenprofile* in representing the input profile, treating the eigenvectors as the basis set for user behaviors.

This feature vector may be then used in a standard behavior classification algorithm to find which of a number of predefined behavior classes, if any, best describes the behavior. The simplest method for determining which profile class provides the best description of an input profile vector is to find the *eigenprofile* class k that minimizes the Euclidian distance with the input profile described in the following paragraph.

If the length of the profile vector is N (number of considered metrics), the matrix C is N by N and determining the N eigenvalues and eigenvectors is an intractable task when considering hundreds of metrics. So, a computational feasible method is that, from equations (6.6) and (6.4), one can obtain

$$A.A^TU_k=\lambda_k U_k \quad (6.9)$$

$$A^TA(A^TU_k)=\lambda_k(A^TU_k) \quad (6.10)$$

let

$$Y_k=A^TU_k \quad (6.11)$$

then

$$A^T.AY_k=\lambda_k Y_k \quad (6.12)$$

From equation (5.12) Y_k is the eigenvector of $A^T A$ and λ_k is its corresponding eigenvalue.

Let $X_k = \alpha_k Y_k$

So X_k is also an eigenvector of $A^T A$.

From equation (6.11):

$$X_k^T X_k = (\alpha_k A^T U_k)^T (\alpha_k A^T U_k) = \alpha_k^2 \lambda_k U_k^T U_k \qquad (6.13)$$

We have $U_k^T U_k = 1$ *then*

$$X_k^T X_k = \alpha_k^2 \lambda_k \qquad (6.14)$$

In order to obtain a normalized vector X_k (i.e. $X_k^T X_k = 1$) we shall have:

$$\alpha_k^2 \lambda_k = 1 \Rightarrow \alpha_k = \frac{1}{\sqrt{\lambda_k}} \qquad (6.15)$$

From equations (6.11) and (6.9), we have:

$$AY_k = AA^T U_k \qquad (6.16)$$

$$AY_k = \lambda_k U_k \qquad (617)$$

$$U_k = \frac{1}{\lambda_k} AY_k = \frac{1}{\lambda_k \alpha_k} AX_k = \frac{1}{\sqrt{\lambda_k}} AX_k \qquad (6.18)$$

finally, we have:

$$\boxed{U_k = \frac{1}{\sqrt{\lambda_k}} AX_k} \qquad (6.19)$$

With this analysis, the calculation is highly reduced, from the order (of the number) of metrics to the order of the number of profiles considered in the training set, because X_k is an eigenvector of the matrix $A^T A$ having a reduced dimension *M*.

Class Organization

To perform realistic experiments, we assign to each known user *k* an *eigenprofile* class Ω^k which is a vector describing the k^{th} user profile. The profile classes Ω^l are calculated by averaging the results of eigenprofile representation over a small number of audited profiles (as few as one), for each user, that best represent the user behavior during a session or a pertinent interval of time. A user behavior is classified as belonging to class k when the minimum

$$\varepsilon_k = \left\| (\Omega - \Omega^k) \right\|^2 \qquad (6.20)$$

is below some chosen threshold θ_ε (where ε_k represents the Euclidian distance between the average profile Ω^k of class *k* and the feature vector Ω of the input profile). Otherwise, the new behavior is considered as anomalous (intrusive).

Classification Algorithm by PCA :

The *eigenprofile* identification process is the following :

step1: Audit a new user profile during a session (or an interval of time).

step2: Determine its corresponding different profile $\Phi_i = \Gamma_i - \Psi_i$

step3: project the different profile Φ_i onto the *eigenprofile space*. We then obtain the feature vector

$$\Omega_i = U^T \times \Phi_i = \begin{pmatrix} w_1 \\ w_2 \\ . \\ . \\ w_m \end{pmatrix} \qquad (6.21)$$

where U is the *eigenprofile* matrix.

step 4: Define the k^{th} class which minimizes the distance $\varepsilon_k = \left\| \Omega_i - \Omega^k \right\|^2$

if $\varepsilon_k < \theta$ ***then***

the new observed profile is that of the k^{th} user

else

the new observed profile is *anomalous*

endif

end Classification by PCA

6.4 Summary of Eigenprofile Procedure

To summarize, the eigenprofiles approach to intrusion detection involves the following steps:

1. collect a set of characteristic profiles vectors of the known users. This set should include a number of profiles, for each user, audited in different days and in an interval of time such as for example during the login session. These profiles should be as significant as possible and should be closest to the actual user behavior to well characterize it. (say for example four profiles for twenty users, so *M=80*)

2. calculate the (80 x 80) matrix L, find its eigenvectors and eigenvalues, and choose the M' eigenvectors with the highest associated eigenvalues. (let $M' = 50$ for this example).

3. combine the normalized training set of profiles according to equation (6.19) to produce ($M' = 50$) *eigenprofiles* U_k.

4. for each known user, calculate the class vector Ω^k by averaging the profile vector Ω calculated from the original (four) profiles of the user. Choose a threshold θ_ε that defines the maximum allowable distance from any profile class.

5. for each new profile to classify, perform the classification algorithm cited above (classification algorithm by PCA).

6. If a new profile is classified as a known user, this profile may be added to the original set of familiar profile vectors (steps1-4). This gives the opportunity to modify the profile space as the detection system encounters more instances of known users.

In our current (future) system, calculation of the eigenprofiles is done offline as part of the training.

6.5 Preliminary Experimental Results

In order to show the rapidity, the robustness and the simplicity of our approach, we have considered a very simple example (which may be considered as a small part of our system because it just considers the different occurrences of events and some commands) which consists of four kinds of users as those defined in [17,18]: inexperienced user, the novice developer, the professional developer and the UNIX intensive user. Our metrics in this primary experience are the different occurrences of events generated by the different commands[1] introduced by a user during an interval of time (30 minutes).

In the following paragraph, we apply the different *eigenprofile* technique steps on these users' behaviors and show that eigenprofiles technique really characterizes well the behavior of each user and this new method is very interesting in anomaly intrusion detection.

6.5.1 Definition of these Behavior Types (from [18])

The inexperienced user

A possible behavior of an inexperienced user (such as a secretary), using UNIX, is the abusive use of the electronic mail (*mail* or *xmh* command). This user may make many logins (5 to 6) short logins per day (of about 10 minutes) or spend all the day connected with long periods idle. This behavior may be characterized by the following commands sequence:

[x@ host]/u/perso/si/x>mail
[x@ host]/u/perso/si/x>lpr mail1
[x@ host]/u/perso/si/x>lpr mail2
[x@ host]/u/perso/si/x>ls
[x@ host]/u/perso/si/x>rm mail0
[x@ host]/u/perso/si/x> rm mail00
[x@ host]/u/perso/si/x>mail
[x@ host]/u/perso/si/x>lpr mail1
[x@ host]/u/perso/si/x>lpr mail2
[x@ host]/u/perso/si/x>ls
[x@ host]/u/perso/si/x>rm mail1
[x@ host]/u/perso/si/x> rm mail2

The novice developer

His behavior is limited to electronic mail and developing some small applications. The novice does not use some commands such as adb or xde. However, he may simultaneously open many windows. His behavior is characterized by the following commands sequence:

[x@ host]/u/perso/si/x>xterm
[x@ host]/u/perso/si/x>who
[x@ host]/u/perso/si/x>mail
[x@ host]/u/perso/si/x>ls
[x@ host]/u/perso/si/x>cd /perso/si/x/repertoire
[x@ host]/u/perso/si/x>ls
[x@ host]/u/perso/si/x>cp serveur.c serveur_old.c

[1] these commands are translated into AIX audit events.

```
[x@ host]/u/perso/si/x>vi serveur.c
[x@ host]/u/perso/si/x>make serveur
[x@ host]/u/perso/si/x>make serveur
[x@ host]/u/perso/si/x>make serveur
[x@ host]/u/perso/si/x>serveur
[x@ host]/u/perso/si/x>make serveur
[x@ host]/u/perso/si/x>serveur
[x@ host]/u/perso/si/x>serveur
[x@ host]/u/perso/si/x>lpr serveur.c
[x@ host]/u/perso/si/x>who
[x@ host]/u/perso/si/x>mail
```

The professional developer

This behavior is characterized [18] as follows:

```
[x@ host]/u/perso/si/x>xterm
[x@ host]/u/perso/si/x>xterm
[x@ host]/u/perso/si/x>xterm
[x@ host]/u/perso/si/x>who
[x@ host]/u/perso/si/x>mail
[x@ host]/u/perso/si/x>ls
[x@ host]/u/perso/si/x>cd /u/perso/si/x/rep1
[x@ host]/u/perso/si/x>ls
[x@ host]/u/perso/si/x>cd /u/perso/si/x/rep2
[x@ host]/u/perso/si/x>ls
[x@ host]/u/perso/si/x>cp serveur.c serveur_old.c
[x@ host]/u/perso/si/x>vi serveur.c
[x@ host]/u/perso/si/x>vi client.c
[x@ host]/u/perso/si/x>cat serveur.make
[x@ host]/u/perso/si/x>env
[x@ host]/u/perso/si/x>grep –F NBMAXCONNECT –f serveur.c
[x@ host]/u/perso/si/x>make serveur
[x@ host]/u/perso/si/x>make serveur
[x@ host]/u/perso/si/x>make client
[x@ host]/u/perso/si/x>xde serveur
[x@ host]/u/perso/si/x>make serveur
[x@ host]/u/perso/si/x>xde serveur
[x@ host]/u/perso/si/x>client
[x@ host]/u/perso/si/x>make serveurps – edf  | grep x
[x@ host]/u/perso/si/x>lpr serveur.c
[x@ host]/u/perso/si/x>lpstat
[x@ host]/u/perso/si/x>ftp
[x@ host]/u/perso/si/x>who
[x@ host]/u/perso/si/x>mail
[x@ host]/u/perso/si/x>xlock
```

Unix intensive user

This user type is considered as a person having a long experience with UNIX system. His behavior is modeled [18] as the following:

[x@ host]/u/perso/si/x>who
[x@ host]/u/perso/si/x>find . –type f –size +1000 –print
[x@ host]/u/perso/si/x>more serveur.c
[x@ host]/u/perso/si/x>ftp
[x@ host]/u/perso/si/x>chmod 700 toto.c
[x@ host]/u/perso/si/x>vi toto.c
[x@ host]/u/perso/si/x>lpr toto.c
[x@ host]/u/perso/si/x>ps –edf | grep x
[x@ host]/u/perso/si/x>finger @remotehost.xxx.fr
[x@ host]/u/perso/si/x>mail
[x@ host]/u/perso/si/x>who
[x@ host]/u/perso/si/x>rlogin guest @remotehost.xxx.fr
[x@ host]/u/perso/si/x>cd /u/perso/si/x/rep1
[x@ host]/u/perso/si/x>vi memoire.tex
[x@ host]/u/perso/si/x>latex memoire
[x@ host]/u/perso/si/x>xdvi memoire

let us apply the different steps cited in the eigenprofiles method above :

Initialization step:

1- The different profiles generated from these characterized behavior (the training set) are as follows:

	inexperienced user Γ_1	Novice developer Γ_2	Professional developer Γ_3	Unix intensive user Γ_4
User_login fail				
User log (23h to 6h)				
Short_Session				
Use_SU OK				
User_SU fail				
Who,w,finger,…		2	3	4
More,pg,cat,…		1	3	3
ls OK	2	2	3	
Ls fail				
Df,hostname,uname				
Arp,netstat,ping				
Ypcat				
Lpr	4	1	1	1
rm, mv	4			
Ln				
Whoami, id				
rexec,rlogin,rsh			1	2
Proc_Execute	16	18	55	17
Proc_SetPetri				
File_Open fail		4	2	4
File_Open fail cp				
File_Open .netrc				
File_Read lpr				
File_Read passwd...				
File_write passwd,… fail				
File_write cp Ok		1	1	
File_Unlink rm	4			
File_mode				1

Table 6.1: The different generated profiles (from [18])

2- the average eigenvector profile defined by these profiles is the following (according to equation (5.2)):

$$\Psi^T=\frac{1}{4}\sum_{i=1}^{4}\Gamma_i^T=(0\ 0\ 0\ 0\ 0\ 2.25\ 1.75\ 1.75\ 0\ 0\ 0\ 0\ 1.75\ 1\ 0\ 0\ 0.75\ 26.5\ 0\ 2.5\ 0\ 0\ 0\ 0\ 0\ 0.5\ 1\ 0.25)$$

3- calculate the feature vectors (according to equation (6.3)) for each user ($\Phi_1,\Phi_2,\Phi_3,\Phi_4$).

4- calculate the matrix A according to equation (6.5).

5- calculate the eigenvectors of the covariance matrix $C=\frac{1}{M}\sum_{i=1}^{M}\Phi_i\,\Phi_i^T=A.A^T$:

this may be done by using the result defined in equation (5.19)

$$A^T.A=\begin{bmatrix} 37.15 & 20.03 & -77.40 & 20.21 \\ 20.03 & 19.65 & -60.28 & 20.59 \\ -77.40 & -60.28 & 204.78 & -67.09 \\ 20.21 & 20.59 & -67.09 & 26.28 \end{bmatrix}$$

The eigenvalues λ_i , i=1..4 are of this matrix are $\begin{bmatrix} 273.792 \\ 12.249 \\ 1.833 \\ 0 \end{bmatrix}$

Their corresponding eigenvectors

$$X_1(273.792)=\begin{bmatrix} -0.329 \\ -0.254 \\ 0.865 \\ -0.282 \end{bmatrix},\ X_2(12.249)=\begin{bmatrix} -0.786 \\ 0.268 \\ -0.038 \\ 0.556 \end{bmatrix},\ X_3(1.83)=\begin{bmatrix} 0.157 \\ -0.783 \\ 0.026 \\ 0.601 \end{bmatrix}$$

We obtain the eigenvectors U_1, U_2, U_3 from equation (6.19).

In this case, we have considered only one profile for each user then this profile corresponds exactly to the class vector $\Omega^k, k=1,..,4$ (from step 4 of the summary of *eigenprofiles* procedure).

The following table presents the Euclidian distances (according to equation (6.20)), between the four users considered, in the new *eigenprofile space* .

	User1	**User2**	**User3**	**User4**
User1	0	4.095	19.927	4.795
User2		0	18.577	2.179
User3			0	19.111
User4				0

Table 6.2: Euclidian distances between (one from another) the four classes.

And if we try to represent the four profiles in the new *eigenprofile space*, the representation looks like the following (where U_1 , U_2 and U_3 are orthonormal axes):

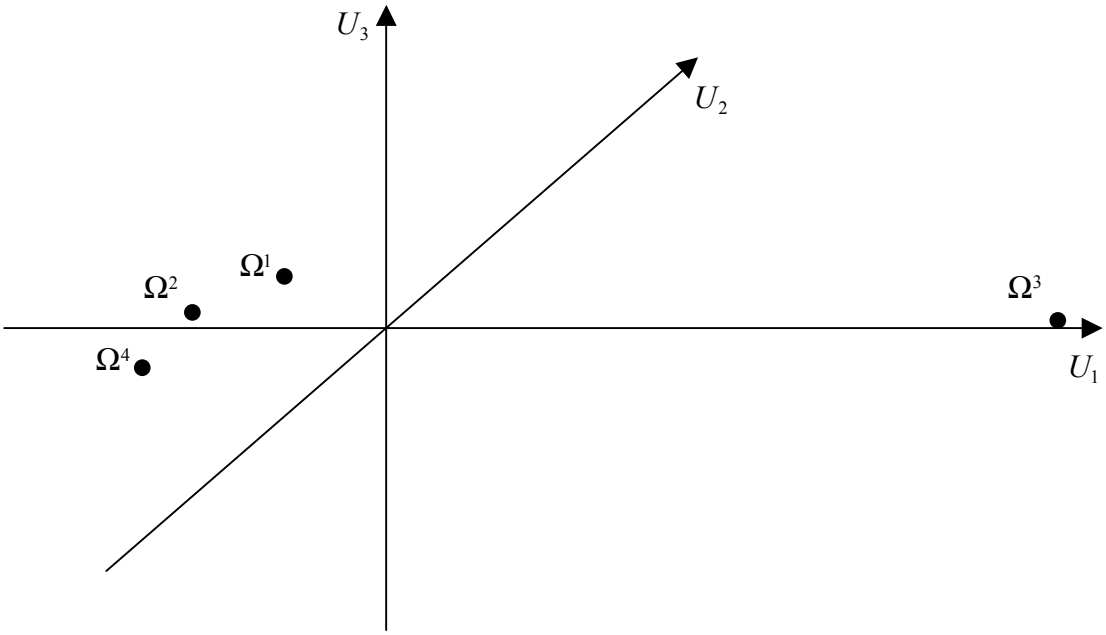


Figure 6.1: The projection of the users' profiles onto the new *eigenprofile space*.

The precedent paragraph describes a small application of this new method on a simple example, which shows that this eigenprofile intrusion detection method really classifies the behaviors of the different users, and really assigns a unique class to each user profile.

The reader may simulate some behaviors according to the different events and then apply the classification by PCA to these audited behaviors. He will certainly finds that this method is very interesting and if the simulated behaviors do not differ a lot from those of the training set then the calculated Euclidian distance (equation (6.20)) is small, otherwise these behaviors will be considered as anomalous (intrusive); i.e. the Euclidian distance from the different known users is great. (a threshold θ_ε shall be chosen. We have not defined this threshold, in this simple example, because we do not have many profiles of each user. In general, the threshold is defined by experience for example it is chosen equal to the distance between the farthest user profile of the training set from its average class + ε (a small distance)).

6.6 Conclusion

In summary, this chapter outlined the following ideas:

- It studied a novel method to anomaly intrusion detection which permits a good classification of the different unix users.
- It presented a simple engineering solution to the problem using Principal Components Analysis.
- The primary simulation results are very interesting. However, its true test is an evaluation of its implementation running under *"live"* conditions with hundreds of users (this will not increase the execution time because the detection consists only of projecting the new audited behavior onto the *eigenprofile space* and comparing the resulted feature vector with the different classes of the training set and the calculation of the different classes, eigenvectors and eigenvalues is done offline).

Conclusions and Future Work

The principal purpose of our work is to realize a system which will be able to detect any present intrusion in a computer system. The study of the different methods on intrusion detection allowed us to discover a new field full of scientific curiosities. We are conscious that the implementation of such a system in one or two years is impossible especially when there are no means (in our institution) of real users with a real network. Nevertheless, we have resolved, in the first part of this rport, many problems of GASSATA [16,17,18,41] by using a new formalization, which can be used in real time to detect all intrusions.

The second part of this report, on the other hand, proposes a new method in anomaly detection. This method is based on information theory (Principal Component Analysis) that is novel in the field of computer security. The first simulation results seemed interesting. However, its true test is an evaluation of its implementation running under *"live"* conditions with hundreds of real users.

We hope we will implement the model based on PCA and derive experimental results in the near future and then compare its results with other methods.

Bibliography

1. R. Heady, G. Luger, A. Maccabe, and M. Servilla. An Architecture of a Network Level Intrusion Detection System. Technical Report, Department of Computer Science? University of New Mexico, August 1990.

2. J. P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, James. P. Anderson Co., Fort Washington, Pennsylvania, April 1980.

3. D. Denning. An Intrusion Detection Model, IEEE Transactions on Software Engineering, Vol. 13 (2), 1987, pp. 222,232.

4. H.Débar, M.Dacier, A.Wespi. Towards a taxonomy of intrusion detection systems, Computer Networks 31(8) 805-822, N.H Elsevier, 1999.

5. U.S. Department of Defense DoD. Dep. Defense trusted computer system evaluation criteria (the orange book). Technical Report 5200.28-STD, DoD, 1985.

6. Service Central de la Sécurité des Systèmes d'Information SCSSI. Critères d'évaluation de la sécurité des systèmes informatiques (ITSEC). Technical report, SCSSI, 1990.

7. ECMA. Security in open systems: A security framework. Technical Report, TR/46, ECMA, 1988.

8. ECMA. Security in open systems: Data elements and service definitions.. Technical Report, TR/138 ECMA, 1989. Standard.

9. H. Debar, M. Becker, and D. Siboni. A neural network component for an intrusion detection system. Proceedings of the 1992 IEEE Symposium. On Research in Computer Security and Privacy, Oakland, CA, May 1992, pp. 240,250.

10. K. Ilgun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection approach. IEEE Transactions on Software Engineering, 21(3) 181-199, March 1995.

11. K. Ilgun. USTAT: A Real-Time Intrusion Detection System for UNIX. Master's Thesis, Computer Science Department, University of California, Santa Barbara, July 1992.

12. P. A. Porras. STAT : A State Transition Analysis Tool for Intrusion Detection. Master's Thesis, Computer Science Department, University of California, Santa Barbara, July 1992.

13. S. Kumar. Classification and Detection of Computer Intrusions. PhD Thesis, Purdue University, West Lafayette, Indiana, August 1995.

14. S. Kumar and E. H. Spafford. An application of pattern matching in intrusion detection. Technical Report CSD-TR-94-013, the COAST Project, Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907-1398, June 17th 1994.

15. S. Kumar and E. H. Spafford. A pattern matching model for misuse intrusion detection. In Proceedings of the 17th National Computer Security Conference, Baltimore MD, 1994, pp. 11,21. NIST, National Institute of Standards and Technology/ National Computer Security Center.

16. S. Kumar and E. H. Spafford. A Software architecture to support misuse intrusion detection. Technical Report CSD-TR-95-009, the COAST Project, Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907-1398, March 17th 1995.

17. L. Mé. GASSATA, A Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis. First international workshop on the Recent Advances in Intrusion Detection. September 14-16, 1998. Louvain-la-Neuve, Belgium. Web proceedings (http://www.zurich.ibm.com/~dac/Prog_RAID98/Table_of_content.html).

18. L. Mé. Audit de Sécurité par Algorithmes Génétiques, University of Rennes 1, PhD Thesis, Order N° 1069, July 7th 1994.

19. L. Mé. Un algorithme génétique pour détecter des intrusions dans un système informatique. *VALGO*, 95(1):68-78, 1995.

20. D.E. Goldberg. Genetic Algorithms in Research, Optimization and Machine Learning, Addison Wesley Editions, USA, 15767, 1991.

21. M.Gen, R.W.Cheng. Genetic Algorithms and engineering design, John Wiley and Sons, Inc., New York, NY 10158-0012, 1997.

22. J. H. Holland. Adaptation in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press, 1975.

23. D. Whitley. The GENETOR algorithm and selection pressure. In proceedings of the Third International Conference on Genetic Algorithms, pp. 116,121. San Mateo, Morgan Kaufmann, 1989.

24. D. E. Goldberg. A note on Boltzmann tournament selection for genetic algorithms and population oriented simulated annealing. Complex Systems, (4), pp. 445-460, 1990.
25. P. Ross and G. H Ballinger. PGA-Parallel Genetic Algorithm Testbed. Department of Artificial Intelligence, University of Edinburgh, 1993.

26. J. E. Baker. Adaptive selection methods for genetic algorithms. In J. J. Grefenstette, editor, Proceedings of the First International Conference on Genetic Algorithms and their applications, pp. 101, 111, San Mateo, Morgan Kaufmann, 1985.

27. D. Whitley. A genetic algorithm tutorial. Technical report, Colorado State University, March 1993.

29. R. Horst, P. M. Pardalos and N. V. Thoai. Introduction to global Optimization, Kluwer Academic Publishers, 1995.

30. I. T. Jolliffe. Principal Component Analysis. New York: Springer Verlag, 1986.

32. D. E. Denning, P. G. Neumann : Requirements and model for IDES, a real time intrusion detection expert system, Technical Report , Computer science Laboratory, SRI International, Menlo Park, CA 1985.

33. R. Jagannathan et al. : System design document : Next Generation Intrusion Detection Expert System (NIDES). Technical Report A007/…/A0014, SRI International, Ravenswood Avenue, Menlo Park, CA 94025, March 1993.

34. S. Smaha, Haystack : an intrusion detection system, *4th* Aerospace Computer Security Applications Conf. October 1988, pp. 37-44.

35. H. S. Vaccaro, G. E. Liepins, Detection of anomalous computer session activity, Proc. IEEE symp. On Research in Security and Privacy, 1989, pp. 280-289.

36. D. Anderson, T. Frivord, A. Tamaru, A.Valdes. NIDES: Next Generation Intrusion Detection Expert System. Software Users Manual, Beta-Update Release, SRI International, December 1st 1994.

37. B. Jouga and L. Mé. Un bref aperçu de l'histoire, des concepts et des problèmes de sécurité des réseaux informatiques. Conférence invitée au séminaire "Sécurité des réseaux" de l'Institut des Hautes Etude de la Sécurité Intérieure. Rennes, 6 et 7 janvier 1998

38. L. Mé and V. Alanou. Une expérience d'audit de sécurité sous AIX 3.1. *TRIBUNIX*, 8(43):30-38, 1992.

39. L. Mé. Security Audit Trail Analysis Using Genethic Algorithms. In *Proceedings of the 12th International Conference on Computer Safety, Reliability and Security*, pp329-340, Poznan, October 1993.

40. L. Mé. Méthodes et outils de la détection d'intrusion. In *Actes du congrès Intelligence Economique et Compétitive (IEC)*, Paris, novembre 1996.

41. L. Mé. Genetic Algorithms, a Biologically Inspired Approach for Security Audit Trails Analysis. Short paper presented at the 1996 IEEE Symposium on Security and Privacy, Oakland (CA), may 1996.

42. L. Mé and V. Alanou. Détection d'intrusion dans un système informatique : méthodes et outils. *TSI*, 96(4), pp429-450, 1996.

43. L. Mé, V. Alanou and J. Abraham. Utilisation de cartes de Kohonen pour détecter des intrusions dans un système informatique : une pré-étude. *VALGO*, septembre 1997.

44. L. Mé. Un complément à l'approche formelle: la détection d'intrusions. In *Actes de la journée CIDR97*, Rennes, octobre 1997.

45. L. Mé et C. Michel. La détection d'intrusions: bref aperçu et derniers développements. Actes de EUROSEC'99. Mars 1999.

46. L. Mé. Méthodes et outils de la détection d'intrusion. Conférence invitée au séminaire X-Aristote "Sécurité des réseaux", janvier 2000.