Research in Intrusion-Detection Systems: A Survey*

Stefan Axelsson
Department of Computer Engineering
Chalmers University of Technology
Göteborg, Sweden
email: *sax@ce.chalmers.se*

TR: 98–17
December 15, 1998
Revised
August 19, 1999

* This work was funded by The Swedish National Board for Industrial and Technical Development (NUTEK) under project P10435.

Abstract

There is currently need for an up-to-date and thorough survey of the research in the field of computer and network intrusion detection. This paper presents such a survey, with a taxonomy of intrusion detection system features, and a classification of the surveyed systems according to the taxonomy. The conclusion is reached that current research interest should lie in the study of the effectiveness of intrusion detection and how to handle attacks against the intrusion detection system itself.

Contents

List of Figures

List of Tables

Chapter 1

Intrusion detection, introduction and survey

1.1 Introduction

This paper is a survey of the research in the field of computer and network intrusion detection. Some of the previous surveys of the field are [13, 40, 43, 45]. Most of these are somewhat dated,[1] and/or superficial, and the growing number of people taking interest in the field calls for an up-to-date and thorough survey. This survey is indeed intended to be thorough, with the surveyed systems described in some detail and classified according to a number of interesting features.

There are several ideas in the literature about how to perform intrusion detection, such as [5, 16, 27, 44] to name a few. These have not been covered since the emphasis here is on intrusion detection *systems*. We wish to survey substantial research efforts that have generated a prototype that can be studied, both quantitatively, and qualitatively. No slight towards the systems not covered, or its authors, intended. That said, the line drawn between surveyed systems, and those that were excluded, is somewhat arbitrary, since the distinction can be difficult to make.

1.1.1 Introduction to intrusion detection

To introduce the concept of intrusion detection we draw the analogy to the common "burglar alarm," [2] to instrument a computer system or network in such a way as to enable it to detect possible violations of a security policy, and raise an alarm to notify the proper authority. (This authority is henceforth referred to as the SSO, short for Site Security Officer). Some of the same problems, "false alarms" and circumvention of the alarm system, are common to both types of intrusion detection systems.

[1] A proposed taxonomy of intrusion detection systems that recifies many of these shortcommings was published after the conclusion of this survey [8]. [2] Unfortunately, there is a clash in terminology, in that the scientific term for "burglar alarm/intrusion alarm" is the same as in our case—intrusion detection system. We will use the latter term when referring to the computer systems version, to avoid confusion. My apologies to those in the security field at large who might feel slighted by the term "burglar alarm."

However, the analogy unfortunately breaks down quickly after the above similarities are noted. In comparison, even the most sophisticated "burglar alarms" operate under a much simpler security policy. Typically, no normal activity is performed on the premises while the monitoring is enabled, and thus *any* (human) activity can be construed as suspicious. If this were true of the computer system and network intrusion detection, the problem could be more readily disposed of. Unfortunately, we demand that intrusion detection systems operate in an environment where (often considerable) normal activities take place (whatever they may consist of), and the problem becomes one of being able to sort out the few rotten apples from a (substantial) barrel full.

1.1.2 Intrusion detection in a wider context

Several methods are available to protect a computer system or network from attack, a strong perimeter defence being only one of them. A good introduction to many such methods is [20], which this section borrows heavily from. The paper lists six general, non-exclusive approaches to anti intrusion techniques (Figure 1.1 depicts the various approaches):

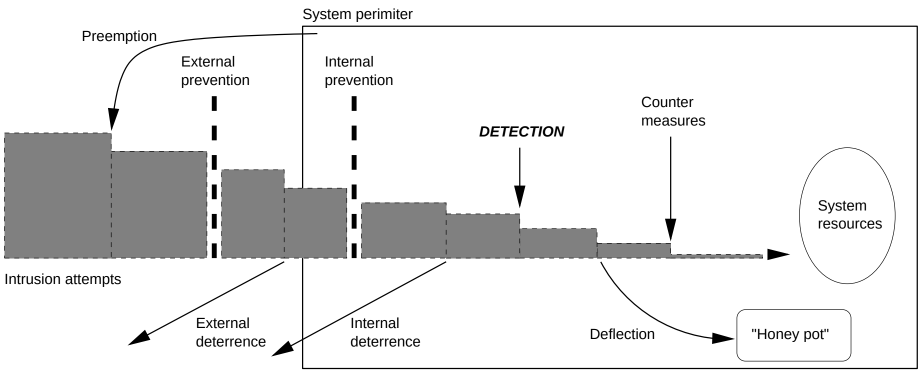

Figure 1.1: Summary of anti-intrusion techniques (from [20])

1. *Prevention* To preclude or severely handicap the likelihood of a particular intrusion's success. One can for instance elect to not be connected to the Internet if one is afraid of being attacked via it. Unfortunately, this can be an expensive and awkward approach, since it is easy to "throw the baby out with the bath water" in one's attempts to prevent attacks. Internal prevention is under the control of the system owner, while external prevention takes place in the environment surrounding the system, such as a larger organization, or society as a whole.

2. *Preemption* To strike against the threat before it has had a chance to mount its attack.[3] In a civilian setting, this is a dangerous (and probably illegal) approach, where innocent[4] bystanders may be harmed.

3. *Deterrence* Persuade an attacker to hold off his attack, or to break off an ongoing attack. Typically accomplished by increasing the perceived risk

[3] Popularly, and in jest, referred to as "Do unto others, before they do unto you." [4] And not so innocent...

of negative consequences for the attacker. Of course, if the value of the protected resource is great, the determined attacker may not "scare off" easily. Internal deterrence could take the form of login banners warning potential internal, and external attackers of dire consequences should they proceed. External deterrence could be effected by the legal system, making laws against computer crime, and the strinct enforcement of same.

4. *Deflection* Lure an intruder into thinking that he has succeeded when, in fact, he has been shunted off, or tricked away, from where he could do real damage. The main problem is that of managing to fool an experienced attacker, at least for a sufficient period of time....

5. *Detection* This is where the subject under discussion fits into the greater scheme of things. Detection aims to find intrusion attempts, so that the the proper response can be evoked. This is most often a notification to the proper authority. Problems include the obvious; difficulty of defending against a hit-and-run attack, and problems with false alarms, or failing to sound the alarm when someone surreptitiously gains, or attempts to gain, access.

6. *Countermeasures* To actively and autonomously counter an intrusion as it is being attempted. This can be done without the need for detection, since the countermeasure does not have to[5] discriminate between legitimate users that perform a mistake, and an intruder that sets off a predetermined response (a "booby trap" if you wish).

In light of the above taxonomy, it is straightforward to put intrusion detection into perspective. Current intrusion detection systems fall (almost) exclusively in the category of detection, although recently more interest has been shown in the question of how to provide an automated response to the detected intrusion. However, the discussion is then focused around the quality of the detection (or rather, lack thereof), and the perceived risk of having the intrusion detection system mistakenly striking down on benign activity. This being somewhat in contrast with the definition of "countermeasures" above.

1.1.3 Early research in intrusion detection systems

The field of intrusion detection is currently some eighteen years old. The seminal paper that is most often cited is James P. Anderson's technical report [3], where he divides the possible attackers of a computer system into the four groups:

External penetrator The *external penetrator* has gained access to a computer that he is not a legitimate user of. Anderson uses this definition to include users that are, e.g. employees of some organisation, where they have physical access to the building that houses the computing resource, even though they are not authorised to use it.

Masquerader The *masquerader* is a user who, having gained access to the system—the masquerader can be both an external penetrator, and another authorised user of the system—attempts to use the authentication information of another user, in effect becoming him, as far as the computer

[5] Although it is preferable if it does...

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 the legitimate user can operate as a *misfeasor*, that is, although he[6] 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 machine with supervisory privileges. Since there is little, if any, evidence of this type of intrusive activity, this class of perpetrator can be difficult to detect.

While this problematisation in itself does not open the field of intrusion detection, Anderson goes on to state in reference to the *masquerader* class that:

> Masquerade is interesting in that it is by definition extra use of the system by the unauthorised user. As such it should be possible to detect instances of such use by analysis of audit trail records to determine:
>
> a. Use outside of normal time
>
> b. Abnormal frequency of use
>
> c. Abnormal volume of data reference
>
> d. Abnormal patterns of reference to programs or data
>
> As will be discussed in the subsequent section, the operative word is "abnormal" which implies that there is some notion of what "normal" is for a given user.

This statement is the first in literature that presents the idea of (semi)-automatic intrusion detection in computer systems, in terms of of anomalies encountered. Furthermore, later in the paper the author expands the idea to also include the detection of outright violations of some security policy.

The paper that really opened the field was published some seven years later. Dorothy Denning [10] presented the idea that intrusions in computer systems could be detected by assuming that users of a computer system would behave in a manner that would lend itself to automatic profiling, i.e. that some model of the behaviour of a particular user could be constructed by the intrusion detection system, and that subsequent behaviour of a presumed user could be verified against that user's model, with the intention that behaviour that deviated sufficiently from the norm would be flagged as anomalous, and hence indicative of a possible intrusion. Denning mentioned several such models, based on the use of statistics, Markov chains, time-series, etc. Denning stressed that the work presented gives the basis for performing these functions in real-time, or near real-time. This paper has its base in the earliest prototype of IDES, on which Peter Neumann worked with Denning [9].

Another early system, that was influenced by the work of Denning and Neumann, was MIDAS [50]. The design of MIDAS centered around an expert system with rules concerning anomalous behaviour, but also, predetermined rules codifying the security policy of the installation. This is one of the earliest instances

[6] While the present author does not wish to stereotype, it feels appropriate to refer to the computer criminal with the third person masculine pronoun, since the overwhelming majority of computer criminals (as is true of most other criminals in society), belong to that gender.

of the idea to process audit data for manifestations of already known intrusive behaviour.

About the same time it was suggested [19, 40] that the two complimentary approaches of seeking anomalous activity based on some historic data, and searching for signatures of already known intrusions, should be employed in the same intrusion detection system, to better complement the relative strengths and weaknesses of the two approaches. One of the papers ([19]) also suggested that this system be autonomous enough to be trusted to respond unsupervised to detected intrusions. Although the author of that paper recognised that much research was yet to be done before this goal could be attained.

In summary: early research concerned itself with the question of whether profiles of normal subject behaviour could be constructed, and used for intrusion detection purposes. A split occurred with the advent of the principle of specifying known intrusion signatures so that audit data could be efficiently scanned for these signatures, and later the two ideas were combined into the hybrid approach.

1.1.4 Summary of early findings—anomaly versus signature detection

The early research uncovered several features of the two major approaches, *anomaly based* and *signature based* intrusion detection. The problems and advantages of the approaches can be summarised as:

Anomaly detection

Advantages The operator need not configure the system, it automatically learns the behaviour of a large number of subjects, and can be left to run unattended. Since it contains no knowledge, some would say prejudice, about how an *intrusion* would manifest itself, it has the possibility of catching novel intrusions, as well as variations of known intrusions.

Disadvantages By definition it only flags unusual behaviour, not necessarily illicit behaviour per se. This can be a problem when the two types of behaviour do not overlap. A system that learns to accept dangerous behaviour as "normal" for a particular user, that slowly changes his behaviour over time, will not find anything out of the ordinary when that user finally mounts his attack. The updating of the subject's profiles, and the correlation of current behaviour with those profiles is typically a computationally intensive task, that can tax the available computing resources hard.

Signature detection

Advantages The system "knows" for a fact, either suspect behaviour, or how normal behaviour should manifest itself. This leads to simple and efficient processing of the audit data. The rate of false positives (benign activity classed as an intrusion) can also be kept low.

Disadvantages Specifying the detection signatures is a highly qualified, and time consuming task. It is not something that "ordinary" operators of the system would do. Depending on how these signatures are specified, subtle variations of the intrusion scenarios can lead to them going undetected. Of course, the method has limited predictive powers. It cannot detect intrusions that are novel to it, especially not those of a fundamentally new *class* of intrusions.

As previously stated is was hoped that by combining these approaches into a hybrid approach, the best of both worlds could be attained.

1.2 A generic architechtural model of an intrusion detection system

Since the publishing of the early papers, several intrusion-detection systems have seen the light of day. Thus, today there exists a sufficient number of systems in the field for one to be able to form some sort of notion of a "typical" intrusion detection system, and its constituent parts. Figure 1.2 depicts such a system. Please note that not all possible data/control flows have been included in the figure, but rather the most important ones.

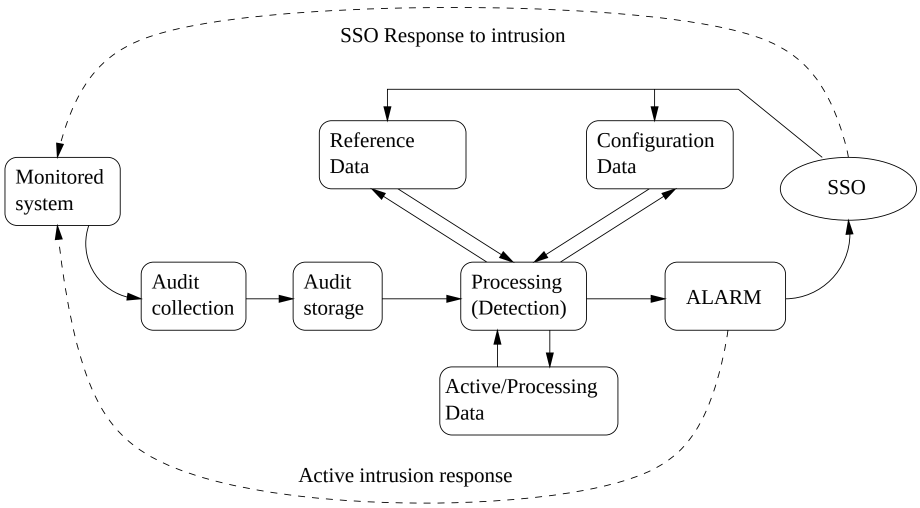

Figure 1.2: Organisation of a generalised intrusion detection system

The generalised model of an intrusion detection system would contain at least the following elements:

Audit collection Audit data from which to make intrusion detection decisions 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.

Audit storage Typically, the audit data is stored somewhere, either indefinitely[7] for later reference, or temporarily awaiting processing. The vol-

[7] Or at least for a long time—perhaps several months/years—compared to the processing turn around time.

ume of data is often exceedingly large[8] i.e., this is a crucial element in any intrusion detection system, and this has led some researchers in the field to view intrusion detection as a problem in audit data reduction [14].

Processing The processing block is the heart of the intrusion detection system. It is here that one or many algorithms are executed to find evidence (with some degree of certainty) of suspicious behaviour, in the audit trail.

Research has to date uncovered three principles of performing intrusion detection:

1. *Anomaly based intrusion detection.* The system reacts to deviations from normal behaviour. "Normal" is defined in relation to previously observed subject behaviour, and is typically updated as new knowledge about subject behaviour becomes known. This update is periodic and automatic in nature, the machine "learns" new behaviour profiles.

   Note that "subject" is to be interpreted loosely. Not only user behaviour, but also host parameters, network parameters, etc. can be monitored for deviations from the set norm.
2. *Signature based intrusion detection.* The system tries to find evidence in the data that matches known signatures of intrusive or suspect behaviour. These signatures are constructed off line, manually, as new types of intrusions becomes known to the security community. Note that even though these signatures can encode behaviour that is only "suspicious" in nature, and not *prima facie* evidence of known intrusive activity, it is still not anomaly detection as above, since the self learning component is missing in the system.
3. *Specification based intrusion detection.* A special case of signature based intrusion detection, where the system is fed with signatures not of intrusive behaviour, but instead of benign behaviour. Every action that deviates from the set norm is then flagged as indicative of an intrusion attempt.

We reduce the above classification into two classes:

1. *Anomaly based detection* As per the definition above.
2. *Policy based detection* Where the detection is based on some *security policy* external to the system. In the case this policy is specified in a *default permit* manner, the detection principle becomes that of signature based detection. In the case the policy is of the *default deny* variant, the detection principle is clearly specification based.

Configuration data This is the state that affects the operation of the intrusion detection system as such. How and where to collect audit data, how to respond to intrusions etc. etc. This is thus the SSO's main means of controlling the intrusion detection system. This data can grow surprisingly large, and complex for a real world intrusion detection installation. It is furthermore quite sensitive, since access to this data would give the

[8] The problem of collecting enough, but not too much audit data has somewhat humorously been described as; "You either die of thirst, or you're allowed a drink from a fire hose..."

competent intruder information about which avenues of attack are likely to go undetected.

Reference data The reference data storage stores information about known intrusion signatures and/or profiles of normal behaviour. In the later case the processing element updates the profiles as new knowledge about the observed behaviour becomes available. This update is often performed at regular intervalls, in a batch oriented fashion.

Stored intrusion signatures are most often updated by the SSO, as and when new intrusion signatures becomes known. The analysis of novel intrusions is a highly qualified task. More often than not, the only realistic mode of operation of the intrusion detection system is one where the SSO subscribes to some outside source of intrusion signatures. These are then proprietary, it is difficult, if not impossible, to make intrusion detection systems operate with signatures from an alternate source.

Active/Processing data The processing element frequently must store intermediate results, e.g information about partially fulfilled intrusion signatures. The space needed to store these active data can grow quite large.

Alarm This part of the system handles all output from the system, whether that be an automated response to the suspicious activity, or which is most common, the notification of some site security officer.

In a hybrid system—containing both anomaly and policy based detection elements—there will be two processing elements, and two sets of configuration and active data storage. The *alarm* module must then make a decision based on outputs from both (or more) detection modules, either a simple *and/or* type decision, or a more complex one, weighing other factors into the equation.

Of the parts described in figure 1.2, to date, the processing part has been most thoroughly studied. Other parts are less well studied, for example little emphasis has been placed on data collection (e.g. what data to collect to be able to ascertain that an intrusion has taken place, how to perform this efficiently)[9], how to store that data efficiently. Another question that remains largely unadressed is that of how to handle the intrusion, especially the interaction between the alarm component, and the SSO.

1.3 A simple taxonomy of intrusion detection systems

The surveyed intrusion detection systems can be classified according to many different features. The most obvious is the classification according to the detection principles previously mentioned:

Anomaly detection The system reacts to anomalous behaviour, as defined by some history of the monitored subjects previous behaviour, or by some previously defined profile of that subject. (Note that subject could mean user, host, network, etc.) In order to differentiate anomaly detection from

[9] We have published one paper that concerns itself with such a study [4].

policy based detection, the present author requires that the system automatically learns from past example. If a human operator where to draw the same conclusions from past data, and codify this knowledge into rules for an expert system, for example, then we would call that system policy based instead.

Policy based detection The system reacts when some policy is violated. This policy can be specified either in a *default permit*, or a *default deny*[10] fashion. I.e. the SSO either specifies some kind of signature that describes illicit behaviour, or he specifies, the normal, security benign, operation of the system, deviations from the set norm are viewed as an attempted intrusion by the intrusion detection system.

Hybrid As previously stated, most researchers believe that both approaches above should be combined when designing intrusion detection systems, to reap the benefits of both, avoid the weaknesses, and accomplish synergistic effects.

Having drawn a major line between groups of systems, based on their approach of detecting an intrusion in audit data, a closer study brings forth the following (albeit sometimes weak) dichotomies:

Time of detection Two major groups can be identified, those that attempt to detect intrusions in *real-time*, or near real-time, and those that process audit data with some delay (*non-real-time*). The latter approach would in turn delay the time of detection. Without any real exceptions the surveyed systems that fall into the real-time category, can also be run, off-line, on historic audit data. This is most likely for reasons of being able to simplify the verification process, as the system is being developed, but of course, it can sometimes be valuable to run an otherwise real-time capable system on previous saved data to establish past security critical events.

Granularity of data-processing This cathegory contrasts systems that process data *continuously*, with those that process data in *batches*, at some regular interval. This category is linked with the *Time of detection* category above, but note that they do not overlap, since a system could process data continuously with (perhaps) considerable delay, or process data in (small) batches in "real-time".

Source of audit data The two major sources of audit data in the surveyed systems are *network data*, typically data read directly off of some multicast network (Ethernet), and *host based* security logs. The host based logs can include operating system kernel logs, application program logs, network equipment (e.g. routers, and firewalls) logs, etc. etc.

Response to detected intrusions *Passive* versus *active*. Passive systems respond by notifying the proper authority, they do not in themselves try to mitigate the damage done, or actively seek to harm or hamper the attacker. Active systems could be further subdivided into two classes:

[10] Named *specification based* intrusion detection by its authors [29].

1. Those that exercise control over the *attacked* system, i.e. they modify the state of the attacked system to thwart or mitigate the effects of the attack. Such control could be in the form of terminating network connections, increase the security logging, kill errant processes etc.
2. Those that exercise control over the *attacking* system, i.e they in turn attack the attacker to try and remove his platform of operation. Since this approach is difficult to defend in court, we do not envision much interest in this approach outside of military/law enforcement circles.

Of the systems surveyed one sewers network connections in response to suspected attacks, and one blocks suspect system calls, terminating the process if that option fails. This mode of defence is generally difficult to field, in that it opens up the system to obvious denial of service attacks.

Locus of data-processing The audit data can either be processed in a *central* location, irrespective of whether the data originates from one—possibly the same—site, or is collected and collated from many different sources in a *distributed* fashion.

Locus of data-collection Audit data for the processor/detector can be collected from many different sources, i.e. in a *distributed* fashion, or from a single point, the *centralised* approach.

Security The ability to withstand hostile attack against the intrusion detection system itself. This is a little studied area. The classification would naively be on a *high—low* scale. The surveyed systems, with one exception, all fall in the latter category.

Degree of interoperability The degree to which the system can operate in conjunction with other intrusion detection systems, accept audit data from other sources etc, etc. This is not the same as the number of different platform the intrusion detection system itself runs on.

In fairness it should be said that not all of the above categories are dichotomies in the true sense of the word. However, the author believes that many of the surveyed systems display sufficient difference that it is meaningful to speak of a dichotomy.

1.4 A classification of the surveyed systems

When applying the above taxonomy to the surveyed systems the classification in table 1.1 is arrived at.

Table 1.1: Classification of the surveyed systems

Name of system	Publ. year	Detection principle	Time of detection	Granularity	Audit source	Type of response	Data-processing	Data-collection	Security	Inter-oper.
Haystack [51]	1988	hybrid	non-real	batch	host	passive	centralised	centralised	low	low
MIDAS [50]	1988	hybrid	real	continuous	host	passive	centralised	centralised	low	low
IDES [41]	1988	anomaly[a]	real	continuous	host	passive	centralised	distributed	low	low
W&S [54]	1989	anomaly	real	continuous	host	passive	centralised	centralised	low	low[b]
Comp-Watch [11]	1990	anomaly[c]	non-real	batch	host	passive	centralised	centralised	low	low
NSM [21]	1990	hybrid[d]	real	continuous	network[e]	passive	centralised	centralised[f]	low	low[g]
NADIR [25]	1991	policy	non-real	continuous	host[h]	passive	centralised	distributed	low	low
Hyperview [7]	1992	hybrid	real	continuous	host	passive	centralised	centralised	low	low
DIDS [52]	1992	hybrid	real	continuous	both[i]	passive	distributed	distributed	low	low[j]
ASAX [18]	1992	policy	real[k]	continuous[l]	host	passive	centralised	centralised	low	higher[m]
USTAT [24]	1993	policy	real	continuous	host	passive	centralised	centralised	low	low[n]
DPEM [30]	1994	policy[o]	real	batch	host	passive	distributed	distributed	low	low
IDIOT [34]	1994	policy	real[p]	continuous	host	passive	centralised	centralised	low	higher
NIDES [1]	1995	hybrid	real[q]	continuous	host[r]	passive	centralised	distributed	low[s]	higher[t]
GrIDS [53]	1996	hybrid[u]	non-real	batch	both[v]	passive	distributed	distributed	low	low
CSM [58]	1996	policy	real	continuous	host	active[w]	distributed	distributed	low	low
Janus [17]	1996	policy	real	continous	host	active[x]	centralised	centralised	low	low
JiNao [15]	1997	hybrid	real	batch	“host”[y]	passive	distributed	distributed	low	low
EMERALD [47]	1997	hybrid	real	continuous	both	active	distributed	distributed	moderate	high
Bro [46]	1998	policy	real	continuous	network	passive	centralised	centralised[z]	higher	low

[a] IDES pioneered the anomaly detection approach. IDES was a pure anomaly based system, signature based detection was added later. [b] The authors clearly intend the method to be generally useful. [c] The anomaly detection is really performed by the SSO himself, the system collects statistics, and then present them to the SSO. [d] With a strong tendency towards signature based detection. [e] The first system to utilise the raw network traffic as a source of audit data. [f] One central network tap. [g] The use of network protocols common to several computer and operating system architectures lends some interoperability from that perspective. [h] The hosts record information about network events, but the network is not used directly as a source of audit data. [i] DIDS has components that monitor both individual hosts (Hyperview), and network traffic (NSM) for audit data. [j] The network monitoring component lends some interoperability from a platform and/or operating system perspective. [k] Deduced from the proposed architecture. [l] Deduced from the proposed architecture. [m] The authors discuss the issue in some detail, and present a proposed architecture to somewhat remedy the problem. [n] The prototype presented is platform specific, the authors discuss the general applicability of the method. [o] This was the first system to employ a default deny policy in a consistent manner. [p] The user of the system can make the choice between real or non-real time processing. [q] Both types of processing are available. [r] NIDES can use host-based logs of network activity as input. [s] However, the authors are the first to discuss the problems with attacks against the intrusion detection system, at length. [t] NIDES contains user support for easing the conversion of proprietary audit trails to NIDES format. [u] The author's state indirectly that the system could be made to visualise the data, and hence help in performing the same anomaly detection, as in [11] above. [v] Primarily a network monitoring tool. [w] The prototype version is passive, but the authors envision a future version to sever the connection that the intruder appears to be utilising. [x] The most active system surveyed. [y] The audit logs originate from network *equipment*, not the network directly through network “sniffing,” hence the “host” classification. [z] One central network tap.

1.5 Trends and constants in intrusion detection research

When studying the historical development of a field of research, both in terms of the research done, and the research prototypes that has resulted, it is interesting to note trends, and also what has been constant over the years. A closer study of the classification in section 1.4—the surveyed systems spanning nearly fifteen years of research—and the references describing said systems, the following few trends and constants come to light.

1.5.1 Trends

From host to network

A shift from host-based intrusion detection to network based detection. This correlates with the shift from single multi-user systems to networks of workstations. However, recent network technology (switching networks, faster network communication speeds) has made it more difficult to monitor the network for audit data. Furthermore, the problem of what to do with encrypted data on the network has presented a problem, that yet remains to be solved. The latter question it is starting to be addressed, mainly by considering the hybrid approach, see [47].

From centralised to distributed

Another shift that correlates with the shift from multi-user systems to networks of workstations is the shift from centralised intrusion detection to distributed intrusion detection.

We see the trend most clearly in the case of data collection. Host based security logging must by it's very nature be distributed in order to operate in a network of workstations scenario. In the case of network data, it is conceiavable that one could monitor a network of workstations from a central network tap, and indeed the two purely network monitoring systems surveyed, has taken this approach, however, others that also monitor host based security logs have also distributed the network monitoring taps, there is thus no absolute concencus on the matter.

In the case of data processing, the trend towards distribution seems to lag behind, which is only natural when one considers the general difficulties of distributed data processing. However, since it is probably the only solution to the problem of how to make intrusion detection systems scale, there is a clear interest in the matter, and recent attempts have been made. It is interesting to note that even though as the processing is distributed, the reliance of a central SSO to receive and act on the alarms is often maintained.

Towards interoperability

While most of the early systems were closely linked to one specific platform, a recent trend is to move towards more and more open and interoperable intrusion detection systems. The perceived benefits are to be able to leverage different methods from different suppliers, capitalizing on their respective strengths and weaknesses, and to be able to operate an intrusion detection system in a heterogeneous environment.

One effort in the line with the former argument is [27], another in line with the later is [47]. The latter authors claim that the thorough specification of a framework in which several smaller agents can cooperate, allows them to do one well defined task efficiently and effectively, and leads to an architectural integrity that is paramount in a system that is envisioned to be very large, covering, and protecting infra structure scale investments.

More resistant to attack

Resistance to attack against the intrusion detection system itself is also an active topic, which previous research did not actively address. The trend is clearly towards systems that can withstand attack against themselves, as well as the monitored system. One recent system that attempts to address this issue is Bro [46].

However, there is still very little study of the nature of the attacks an intrusion detection system could realistically be able to withstand. One recent paper, that addresses some issues regarding evasion of detection is [49], but others still remain unaddressed.

1.5.2 Constants

The following issues seem to have remained largely constant over the period covered by the survey.

The hybrid between anomaly and policy

There seems to still be general agreement that in order to make an effective intrusion detection system, one must employ both anomaly and policy based intrusion detection methods, even though this is one of the original results.

It is interesting to note the relative shift—trend if you will—in concentration of the research where newer research often stress policy based detection, at the expense of anomaly based detection. There are probably practical reasons for this, in that it is more difficult to perform experiments that say anything conclusively about the coverage of anomaly based techniques, even though it must be said, that the discussion of coverage is somewhat lacking in other work based on policy detection techniques, as well.

Real-time detection

Early research systems performed non-real time detection, it was realised that this was an imperfection of the systems, neccesitated by then current technology limitations. Of the more recent systems, only one claim non-real time performance from a more philosophical standpoint.

While it is clear that real-time detection has desirable properties, the present author would not rule out the usefulness of non-real time detection altogether. There are many cases where after the fact assessment of the situation, to be able to accurately depict events as they transpired, is perhaps more desirable than being given an immediate warning that something may be amiss, and nothing more.

Such situations arise in cases where law enforcement is involved, where the accuracy, and traceability of events are more important than real-time performance. Another similar case is when the security policy of an organization states accountability, rather than preemptive control, such as in medical emergencies. Medical personnel need unrestricted access to possibly sensitive data quickly, the security repercussions of which can be dealt with later.

As previously mentioned there are links between the *time of detection* cathegory and the *granularity of data processing*, and it is clear from studying table 1.1 that real-time detection correlates well with continous data processing, and that non real-time detection correlates well with batch data processing. There are a few exptions to this rule, however, so the overlap is not perfect.

Few active responses

There is some discussion whether to allow systems to respond more actively, for instance by terminating the connections that appear to be causing the attack. The opinion is clearly in favour of more active systems, but research is, perhaps not surprising, still immature in this field. Difficult questions regarding the accuracy of the detection, the possibility of opening the system to a denial-of-service attack, and liability, remain to be solved before intrusion detection system can be trusted to respond on their own.

The consumption of resources

As computers and networks get faster, we can process more audit data per unit time, but that same computer or network unfortunately produce (some) audit data at a much higher rate as well.[11] Hence, the total ratio of consumed resources to available resources is, if not constant, at least not decreasing at a sufficiently fast pace, that the performance of the intrusion detection system becomes a non-issue. Quite the contrary, network communication speeds for instance, seems to be one of those obstacles that the research community seems to never be able to quite clear.

There is also little study into the question of how to collect, store, and prune, these vast amounts of audit data, even though the present author feels that this area hides contains some interesting problems to be researched.

Little study of coverage

There is still a lack of study in the field of coverage,[12] of the intrusions the system can realistically be thought to handle.[13] The problems are both that of incorrectly classifying benign activity as intrusive, a so called false positive,[14] and that of classifying intrusive activity as non-intrusive, a false negative. These mis-classifications lead to different problems. The term *coverage*, borrowed from the field of dependability, could in our field be defined as the ratio of correctly

---

[11] This cathegory has not been tabulated, even though it is conceivably a feature of the surveyed intrusion detection systems. The reason is mainly that few authors make solid claims in this area, and especially in relation to some usability scenario, i.e. as a percentage of how much a system owner would be willing to let intrusion detection cost him. [12] Since the first version of this paper, there has been some activity in this area, most notably [12, 39, 57].
[13] This cathegory has not been tabulated as it is not a feature of the surveyed systems. It becomes clear when studying the surveyed references however. [14] "False alarm," if you will.

classified intrusions (true positives) to the number of intrusions incorrectly classified as non-intrusive, (false negatives) plus the number of true positives, i.e. the fraction of intrusions that can be detected.[15] Even though the term would be a useful measure on the effectiveness of a proposed intrusion detection system, there are few references to it in the published literature.

The nature of computer security intrusions, from an intrusion detection perspective

Closely linked to the study of *coverage* is the lack of study of the nature of the intrusions the system should be able to classify, and the nature of the intrusions the intrusion detection system itself should be able to withstand.[16] Papers that *do* address the question of the nature of the computer security intrusion are [36, 37], and more specifically [38], and [32]. A paper that concerns itself with the nature of attacks against intrusion detection systems themselves, is [49].

The role, and capabilities of the SSO

The reliance on some SSO to handle the final arbitration, and response to the intrusion.[17] The specific role of the SSO has not been well studied, how results should be reported to him, how many results he can realistically be expected to handle, his abilities to respond etc. Of course, if one fails to address the issue of the number of false positives, for instance, both in relative terms, i.e. not more than 0.5% false alarms, and in absolute terms—0.5% could well mean 5000 alarms—then the difficulty of putting the function of the SSO in perspective follows.

1.6 Open research questions

In summary then, the most obvious shortcomings in the research performed to date, is that it fails to thoroughly address the following questions:

- What is the nature of the intrusions that the system is trusted to detect,
- to what degree can the system correctly classify these intrusions, and can the system correctly classify intrusion to such a degree that it can be trusted to respond actively to them? The reason we ask these questions is that we would like our intrusion detection system to be able to respond as accurately, quickly, and hence, with as little human intervention as possible. The use of active response also raises questions about the possibility of a denial-of-service attack, which compounds the problem.
- What audit data do we need to make a sound decision from an intrusion detection perspective? How do we collect, store, prune, and transmit this audit data, efficiently and effectively?

[15] Mathematically: $\textit{True positives}/(\textit{False negatives} + \textit{True positives})$. This is often the most convenient way of calculating $P(\textit{Intrusion indicated}|\textit{Intrusion existing})$. [16] This cathegory has not been tabulated as it is not a feature of the surveyed systems. It becomes clear when studying the surveyed references however. [17] This cathegory has not been tabulated as it is not a feature of the surveyed systems. It becomes clear when studying the surveyed references however.

- We also need to have more knowledge about the nature of the intrusions to be able to draw sound conclusions when it comes to the issue of *coverage*; have we found all possible types of intrusions, can we find all possible types, have we found all possible intrusions of this particular type? etc. etc.

- What is the nature of the attacks against the intrusion detection system itself,

- to what degree can it be trusted to continue correct operation in the face of opposition, and

- when it can no longer correctly perform its duties, how can graceful degradation of service be ensured? Can the system fail in such a way that security is not compromised and what charaterises such a failure?

  Since there is more and more commercial interest in intrusion detection, we will likely see more and more attackers become aware of the threat that intrusion detection poses. It is probably prudent to assume that those attackers that are motivated enough, will seek ways to attack the intrusion detection system itself, in order to avoid detection. This raises the questions above as well as others.

- What of the run-time efficiency of the intrusion detection systems? One criticism that is often raised is that intrusion detection systems consume too many resources to be fielded effectively. To date, very little has been done to study the execution efficiency of intrusion detection systems.

These are fundamental, interesting, and difficult, questions, and while we have started to address them [4, 36–38, 49], much work still needs to be done before any sort of major conclusion can be reached. This is especially true of the latter questions regarding attacks against the intrusion detection system itself, where research to date has been scant.

1.7 Remaining contents of this survey

The remaining paper consists of a detailed overview of each of the surveyed systems. The systems are presented in roughly chronological order. Each presented system is followed by the surveyor's opinion of the work presented.

Chapter 2

Details of the surveyed systems

As previously mentioned, this chapter consists of a detailed overview of each of the surveyed systems. The systems are presented in roughly chronological order. Each presented system is followed by the surveyor's opinion of the work presented. In the following the term "authors" is to be taken to mean the authors of the work currently being surveyed, while we will refer to ourselves as the "present author."

2.1 Haystack

2.1.1 Introduction

The Haystack prototype [51] was developed for the detection of intrusions in a multi-user Air Force computer system, then mainly a Unisys (Sperry) 1100/60 mainframe running the OS/1100 operating system. This was the standard Air Force computing platform at the time.

Haystack was primarily designed to detect six different types of intrusions (more or less verbatim from [51]):

1. Attempted break-ins: When an unauthorised user tries to gain access to the computing system.

2. Masquerade attacks: When an authorised user makes an unauthorised attempt to assume the identity of another authorised user.

3. Penetration of the security control system: Where a user attempts to modify the security characteristics of the system.

4. Leakage: Moving potentially sensitive data from the system.

5. Denial of service: Denying other users the use of system resources, making the resources unavailable to the other users.

6. Malicious use: Miscellaneous attacks such as deletion of files, resource hogging, etc. etc.

In order to detect these six types of intrusions the system employs two methods of detection: anomaly detection, and signature based detection. The anomaly detection is organised around two concepts; a per user model of how that user has behaved in the past, and pre-specified generic user group models, that specify generic acceptable behavior for a particular group of users. The combination of these two methods solves many of the problems associated with the application of any one of them in intrusion detection systems.

The authors explain that even though the US Air Force has well defined security policies—that may be lacking in the civilian sector—there are still many problems associated with the application of these policies in intrusion detection systems. e.g. there is no consensus on formal specifications for security policies, there is a lack of understanding of how intrusions are made, etc.

2.1.2 System organisation and operation

The system was divided into two platforms. The Unisys (Sperry) operating system was responsible for the audit data collection. The Unisys part of Haystack then converted this audit trail into a unified audit trail, the so called Canonical Audit Trail format (CAT for short), and parsed it with respect to the abstract elements that constituted a generalised audit trail event, possibly selecting records pertaining to certain users, etc. as per the SSO's instructions. This canonical format audit trail was then written to (then) standard 9 track ANSI tape.

The CAT audit trail was then processed on a Zenith Z-248, 80286 PC-AT clone, with at least 4 MB main memory. This platform ran the MS-DOS operating system. The PC part of the Haystack system constituted most of the code of the system. The PC would read the 9 track tape, detect and log obvious breaches of security, according to the policy based part of the intrusion system, aggregate several audit records for each user together into a session record for that user. This session record would then be inserted into the session record database, a commercial DBMS, using the standard SQL command set.

All security relevant events are listed separately for the benefit of the SSO, these typically account for less than 0.5% of the total number of audited events, and can thus be perused manually by the SSO. The event horizon, i.e. the amount of previous audit records that the processing mechanism has to consider when processing the current audit record, is set to one, i.e. the processing mechanism only considers the current audit record when searching for intrusions/anomalies in this phase of the processing. This design decision limits the amount of processing done in this step significantly.

The intrusion detection system then processes the new session records that comprise the database, using both statistical and pattern-based techniques looking for evidence of predefined "bad" behavior, and atypical or suspicious behavior. The pattern-based techniques assess multivariate characteristics of the sessions compared against expected characteristics of particular types of intrusions.

Should the SSO decide to look for evidence of a user that attempts to "learn" the anomaly based part of the system—that his, in fact, suspicious behavior, is actually normal, and nothing out of the ordinary—the SSO can choose to process past user sessions to look for trends that could indicate this. This also handles the case where a user gains savvy in operating the system, and thus deviates from his "normal" behavior.

2.1.3 Future research

The authors identify a number of areas for future research, among them:

- How do we test an intrusion detection system, and measure it's effectiveness?
- Could we implement a real-time intrusion detection system with sufficient security and reliability to be entrusted with the ability to shut down an offending user or even the entire system?
- What visual metaphors are most effective for presenting computer security information to the SSO? Is there a security metaphor that is analogous to the spreadsheet for financial analysis?
- What are the relevant privacy and legal issues? What are the effects on employee morale? Could heavy handed auditing reduce the perceived usefulness of the target computer system for exploratory or research work?

2.1.4 Survey conclusions

The questions identified under future research are of course still valid today, despite the fact that the research presented is more than ten years old. More recent research (see section 2.7) indicate that the assumption that all users would interpret auditing as something negative could perhaps be overly pessimistic.

It is interesting to note that one important question above—how to present the information to the SSO—has not really been addressed since the paper was published, excepting perhaps the cursory treatment in [56].

2.2 MIDAS—Expert systems in intrusion detection: A case study

2.2.1 Introduction

MIDAS [50] was developed by the National Computer Security Centre, in cooperation with the Computer Science Laboratory, SRI International, to provide intrusion detection for the NCSC's networked mainframe, Dockmaster, a Honeywell DPS-8/70. This computer was primarily used for electronic communication within the employee community at NCSC, and affiliated agencies. The authors acknowledge previous work by Denning et. al., and work at Sytek, as their main source of inspiration.

2.2.2 Expert knowledge in intrusion detection

MIDAS is built around the concept of heuristic intrusion detection. The authors make the example with the human site security officer, and how he would go about analysing audit logs manually, to find evidence of intrusive behaviour. He could for instance reason that most intrusions probably occur late at night/early morning, when the system is unattended. That would narrow the search somewhat. He could go on to hypothesise that most intruders, in an attempt to cover their tracks, would vary their points of attack from different locations on the network. Combining these two criteria he could well have narrowed the search to the point of him being able to peruse log records for individual user sessions. The seasoned security officer, could well find cause for suspicion simply by looking at the records for a particular session, that wouldn't "feel" right, and close that account, pending further investigation.

From this (imagined) process, the authors identify that successful (manual) intrusion detection, involve knowledge, and symbolic reasoning, with a measure of uncertainty. This leads to the conclusion that a rule-based expert system could be employed as a means of performing intrusion detection.

The authors note that the requirement that the expert system provide the knowledge of an "expert" security officer, is a minimum requirement, considering how abysmally small rate of success human security experts have when trying to find evidence of intrusive behaviour by manually examining audit records, that may either be too numerous to examine, or too sparse to contain enough information to draw the correct conclusions from.[1]

2.2.3 Application of the expert system

MIDAS applies the Production Based Expert System Toolset (P-BEST) for intrusion detection. P-BEST is a forward chaining expert system shell, in which the introduction of a new fact in its fact base, triggers the reevaluation of the rule base. This in turn can introduce new facts into the fact base, and processing stops with the conclusions drawn when no new rules fire.[2] P-BEST is written in Lisp, and produces Lisp code, that can be compiled and run on a dedicated Symbolics Lisp machine. The compilation of the expert system code into object code, provides for efficient execution of the expert system shell.

In MIDAS, P-BEST's rule-base is populated with rules in three distinct categories:

Immediate Attack The immediate attack heuristics, operate without any knowledge of the (statistical) history of the system, on a very narrow time-window of audit records, typically only one. Furthermore, the immediate attack heuristics are static, they do not change to reflect new trends in input data, other than as a direct result of site security officer action. The idea behind the immediate attack heuristic is that they would be able to find activity that is exceptional in and of itself, in effect searching for already known indications of intrusions.

User anomaly The user anomaly class of rules make use of statistical profiles of previous user behaviour to be able to detect sufficient deviations from those statistics. Two levels of user profiles are kept, statistics pertaining to the current session (session profile), and statistics pertaining to a longer period of time concerning the user in question (user profile). The session

[1] However, the present author feels that it is perhaps not in the field of reasoning, but rather in the department of sheer force of labour that the human SSO falls behind his computerised counterpart. [2] Contrast this with a backward chaining expert system, in which inference is triggered by the posing of some question, such as: "Is X true, for some statement X?" The system then evaluates the rules until all (necessary) facts have been processed.

profile is updated at time of login, from the user profile, which in turn is updated by the session profile, at time of logout. The updating of the profiles thus form a cycle.

System state The system state heuristics maintain knowledge about the statistics of the system as a whole, without concern for individual users. For example, the total number of failed login attempts, for a given period of time, as opposed to the number of failed login attempts for a particular user.

The structure of the rule base is two tiered. The first, lowest, layer, handles the immediate deduction about certain types of events, such as "number of bad logins" and asserts a fact to the effect that some threshold of suspicion has been reached when they fire. These suspicions are then processed by second layer rules, that decide whether to actually raise an alarm based on the suspicion facts asserted by the lower level rules, e.g. "This user is a masquerader because, he has made 40 command errors in this session, and he has tried the invalid commands *`suid`*, and *`priv`*, and he is logged in at an unusual time." This put together would be a strong indication that something is amiss, and the second level rule—representing a masquerader—would trigger, alerting the site security officer.

2.2.4 Threat model

The authors adapt the threat model from Denning [10], by collapsing some of her categories into one. The threat categories remaining, and how MIDAS attempts to counter them can be summarised as:

Attempted break-ins Countered by immediate attack heuristic, the rules that deal with this situation is mainly, but not exclusively, concerned with login failures.

Masquerade The assumption made by the authors is that this is a straightforward category of intruders to detect, by nature of their actions deviating from established statistical use by the authorised user. Anomaly heuristics are used.

Penetration The direct violation of system security policy. This type of attack is handled by all three heuristics, targeted towards access, or attempted access, of system sensitive programs or data.

Misuse Misuse of computer resource by an authorised user, detected using both anomaly, and system heuristics. The direction of printer output to some unusual location can be detected, for instance.

Trojan horse/virus Anomaly and system heuristics are applied to this area of concern. The authors do not differentiate between these two areas since they have not been able to separate the two using the available audit data. The key factors to consider are access violations on system sensitive objects, and execution statistics that violate norms established for given commands.

2.2.5 System organisation, performance, and conclusions

MIDAS itself runs on a symbolics list station, that receives it's audit data via the network from the Dockmaster mainframe, having been preprocessed to massage the audit data into a common format, and having had some simplistic reductions applied. The mainframe sends both operating system provided audit logs, and command logs from a specialised MIDAS module that runs on Dockmaster. The Symbolics Lisp machine maintains a rule-base, a fact-base that reflects the converted audit data, and a database of user and system statistics. MIDAS is provided with a graphical user interface, where the site security officer can view results from the processing, as well as parameters pertaining to the speed of processing itself, in real-time.

MIDAS was tested on a subset of the users, the data was feed via tape, in simulated real-time, and was found to execute more efficiently than the authors had previously expected. It managed to keep up with full scale operation of the system, some 1200 users, in real-time. The authors state as problems that even though MIDAS seems to detect, at least naive, simulated intrusion attempts, it gave too many false alarms. As a question for future research the authors mention the employment of other algorithms for the detection of anomalies, Markovian analysis being one such technique. The authors state further that some means of validating performance of the rule base should be developed. Furthermore, the authors hope to employ results in (then) ongoing expert system research, to validate the rule base itself, for completeness, and consistency.

2.2.6 Survey conclusions

Aside from the fact that the system presented is "seminal"—MIDAS was the first published system to employ signature based detection—it is interesting in that the authors clearly defines what type of problems MIDAS was designed to handle, and how MIDAS would handle them. Furthermore, the performance of the system was tested, and results published. While this is interesting data to obtain, more recent research often fails in this respect.

Also, the authors have begun by studying the situation of a senior SSO, and even though the demands on him made by the system is not clearly stated, that he is part of the system, other than as an ill defined recipient of the output of the system, is of course interesting.

2.3 IDES—A real-time intrusion-detection expert system

2.3.1 Introduction

IDES is one of the classic intrusion detection systems [41, 42], and to date one of the best documented. It is difficult to write about one IDES system however, since the IDES project went on for a number of years, continuing into the Next-Generation Intrusion Detection Expert System, or NIDES, after the IDES project was officially finished. Thus, there is really no one IDES system of which to speak, since the system underwent (sometimes) fundamental change as the research project progressed. This survey will focus on the earlier stages of the project, around 1988, and describe differences between the systems presented in the earlier, and later stages of the project, where appropriate.

The basic motivation behind IDES is that users behave in a consistent manner from time to time, when performing their activities on the computer system, and that the manner in which they behave can be summarised by calculating various statistics for the user's behaviour. Current activity on the system can then be correlated with the calculated profile, and deviations flagged as (possibly) intrusive behaviour.

IDES intended to detect intrusions in all of Anderson's categories, even the misfeasor category (i.e. a user that is authorised to access both the system, and its data, but who abuses this privilege.) IDES performs this detection by constructing a profile for a group of users, who should behave in the same manner, by virtue of their organisational status, and attempt to correlate behaviour for a particular user, not only with past behaviour for that user, but also with the behaviour that is recorded as "normal" for that group.

2.3.2 The prototype

The 1988 prototype of IDES differed from the original prototype in many respects. It runs on two Sun-3 workstations, one Sun-3/260 that maintains the audit database and the profiles, and one Sun-3/60 that manages the SSO user interface. The audit database is implemented using a COTS Oracle DBMS. The monitored system is a DEC-2065 that runs a local version of the TOPS-20 operating system. The audit data is transmitted (securely) to IDES via the network, one record at a time, and processed to provide a real-time response.

Later in the project IDES was run on faster hardware, and monitored a network of workstations. The Oracle database was discontinued, in favour of a locally developed audit database, while it was felt that the feature set of Oracle was not well suited to the access patterns of IDES. A signature based detection (sub)system based on the P-BEST expert system shell was also incorporated, since it was felt that signature based detection was necessary to provide a complete set of detection capabilities.

2.3.3 Profile data

IDES monitors three types of subjects; users, remote hosts, and target systems. Some, in total, 36 different parameters, called measures by the authors, are monitored for the subjects, 25 for users, 6 for hosts, and 5 for target systems. These measures fall into two categories:

Categorical measure A measure that is discrete in nature, and the values of which are members of a finite set, e.g. the commands that a user has invoked during a session is a categorical measure, with the set of possible values being the set of all possible commands, that that user can issue to the system.

Continuous measure A measure that is a real valued function of some parameter, say for instance, the number of lines printed for this session, or the length of a user session.

The prototype under discussion measures these measures during each user session, defined as the time from login to the system, to the time the user exits the system, either normally or abnormally, as the result of a systems malfunction for example.

Some of the measures that the system monitors for a user are:

CPU usage The number of CPU seconds consumed during this session, continuous measure.

Command usage Categorical measure, that records the commands used. Examples of other related measures are "Mailer usage", "Editor usage", and "Compiler usage," where by IDES keeps track of particular classes of commands.

Command usage (binary) A categorical measure that records whether a particular command was used during this session. The value is restricted to "true" or "false," respectively.

Network activity A continuous measure that keeps track of the number of "network activity" audit records received during a session. There are several other measures related to network activity as well.

These measures are kept in a real valued vector as summarised statistics for the session, the categorical measures having been converted to continuous measures first. Other data such as the standard deviation etc. is also kept in different vectors for a particular subject.

These statistical profiles are typically updated to reflect new user behaviour once a day, after the original profile has been "aged." This aging process ensures that newer behaviour plays a larger part in the detection of anomalies, than older behaviour. The rational being that subjects behaviour slowly change over time. The "half-life" (i.e. after this amount of time, the data contributes only half as much to the profile values as the new data) of the profile is approximately 50 days in the early prototype, 30 days in the later. The nature of the stored anomaly profiles, and what data to store, changed between the different versions of IDES.

2.3.4 Anomaly detection

IDES process each new audit record as it enters the system, and verifies it against the known profile for both the subject, and the group of the subject, should it belong to one. IDES also verifies each session against known profiles, when the session completes. In order to further differentiate between different but authorised behaviour, the prototype was extended to handle two sets of profiles for monitored subjects depending on whether the activity took place on an "on" or "off" day. The site security officer defines which days are (in effect) "normal" workdays for a particular subject, mostly users, and which are not. This further helps to increase the true detection rate since a finer notion of what is "normal" for a particular subject, based on real-world heuristics can be developed.

In order to detect anomalous behaviour during the session, when all session statistics are not yet present, IDES extrapolates the current session statistics and compares this extrapolation with the profile for the subject, otherwise, all subjects would report an abnormality until roughly half-way through the session, since some (continuous) measures would not yet have reached even their mean value for a session.

In the case the user is a new user, and not yet known to the system, IDES uses a default profile, to start off the monitoring of that user.

When an anomaly is detected IDES reports what measures that contributed the most to the classification, so that the site security officer can make his own judgement as to the validity of the reported anomaly.

2.3.5 User interface

IDES has a well thought out user interaction model, which defines three different classes of users. Each of these has his own user interface, tailored to their specific needs. The interfaces are graphical in nature, and provide the site security officer(s) with both; plots of anomaly data, as well as text based interaction, explaining why IDES found an activity anomalous. From the user interface, the user can also control many aspects of IDES behaviour, for instance, IDES has a feature where by which the site security officer can "roll-back" an updated profile, when he suspects that that session may be "tainted" by intrusive behaviour, that should not have been learnt as "normal" for that subject. He can also enable/disable monitoring for individual subjects etc.

2.3.6 Future work

The papers list several future enhancements, many of which where addressed in the later versions of IDES, and NIDES.

2.3.7 Survey conclusions

As previously stated, IDES falls in the category of "seminal" systems. It was the first that utilised anomaly based detection, and the rational for, and implementation of the statistical methods, and what parameters are used, is very well documented. Which makes it relatively easy to follow the thoughts, and work, of the authors, which is important since research continues in the field.

It is interesting to note that the session statistics in themselves do not adequately handle multi-modal distributions of data, but that some (somewhat crude) effort has been made—the introduction of "on" and "off" days—to simulate such a capability. This problem would come to be redressed in NIDES, where the statistical routines were altered to accommodate multi-modal distributions. See section 2.14 on page 56 for a survey of NIDES.

2.4 Wisdom & Sense—Detection of anomalous computer session activity

2.4.1 Introduction

W&S [54] is another seminal anomaly detection system. Development started as early as 1984, with the first publication in 1989. It is interesting to note that W&S was at first not intended for the computer security application but "a related problem in nuclear materials control and accountability."[3] W&S is unique in its approach to anomaly detection. W&S studies historic audit data and produces a forest of rules describing "normal" behaviour, this is the "wisdom" part of W&S. These rules are then fed to an expert system, that evaluates recent audit data for violations of the rules, and alerts the SSO when the rules indicate anomalous behaviour, the "sense" part of W&S.

The design criteria was for W&S to:[4]

- Reduce audit data to more usable forms.
- Build its own rule base without human guidance.
- Store and use very large, instantiated rule bases efficiently.
- Tolerate conflicting rules.
- Deal with uncertain and erroneous knowledge.
- Continue to learn from experience, and adapt to transient conditions.
- Accept human modifications to its rule base, but not be overly dependent on scarce human expertise.
- Make real-time, graded decisions regarding anomalous behaviour.
- Provide human-readable feedback on anomalies to aid in anomaly resolution.
- Create minimal interference with the real functions of its host system.
- Be portable to different applications, operating systems, and hardware.

The authors claim that most of these design criteria have been met, but that they would need more experience with operating environments, and simulated intrusions, to design additional evaluation tools, and to fine tune W&S to increase the precision of its classifications.

2.4.2 System operation

W&S reads historic audit records from a file. The authors state that more is better when it comes to the creation of rules from previous audit records, up to about 10000 records per user. A figure of around 500–1000 audit records per user is a good target to aim for according to the authors. The audit records that are used, typically record one event for each process execution, at the end of execution of that process.

The natural unit of processing for W&S is the audit record. However, to make an observation on the statistics of an occurrence some sort of aggregation has to be made. W&S creates a *thread class* for each aggregation of audit records. This thread is defined in terms of specific data values of the audit records. The authors give the example of a user/terminal thread, where each audit record that pertains to user "bob" on terminal `TTA1` are grouped together

[3] The authors were at Los Alamos National Laboratory, and Oak Ridge National Laboratory at the time of publication. These facilities have strong ties with the US nuclear weapons program. [4] Verbatim from [54].

in one thread, which is a member of the class. Each thread class has a number of functions associated with it that typically compute statistics, or perform other actions for the thread as each new record is added to the thread.

The authors note that the rule generation mechanism generates everything from very general rules, "the valid terminals are T1, T2, etc." to very specific ones, such as "on Tuesdays between 6:00 am and 7:00 am, when the user has system operator privileges, and is using terminal T3, only commands that generate little direct disk activity are used."

The "sense" part of the system reads audit records, evaluate the thread class that they are part of against the rule base, and triggers an alarm if enough of the rules report enough of a discrepancy—a high enough score—with the profile. The score of the thread (named Figure-of-merit, FOM by the authors) is a time-decayed sum of the FOM's for that thread's audit records. Thus several events, across several sessions, that are slightly anomalous, will eventually accumulate to an anomaly for that thread.

2.4.3 Rule generation

The rule base is generated in form of a tree, where each branch of the tree contains more specific rules pertaining to the same measured behaviour, closer to the leaf. The rules themselves contain a right-hand-side (RHS), and left-hand-side (LHS). The RHS, named "restriction" by the authors, describes the conditions under with the rule applies. The RHS specification can take one of three basic forms:

1. A list of acceptable categorical values for a particular audit record field.
2. A list of acceptable ranges for a continuous, metric audit record field.
3. A list of user-defined functions to be executed until either, one of them returns true, or the list is exhausted.

There are two types of nodes in the tree. The first type designates data fields in the audit record, and can have a maximum of 32 branches. The other type designates acceptable field values, and can have at most 255 branches. Together these two types compose a rule base "level." The tree is typically pruned to a maximum of 4–5 levels. The rules themselves are condensed into an average of 6–8 bytes of storage each[5] and a typical rule base can contain between 10^4 and 10^6 rules. This enables W&S to store the entire rule-base in memory, which leads to efficient evaluation of the rule base by the "sense" part of the system.

W&S views data mainly as categorical, and generates rules by first sorting continuous data into variable size bins with approximately the same number of points in each bin, by an ad-hoc method—this in effect clusters the data—and then including more and more bins into the rule, until a target ratio of "normal" data has been reached. The remaining data values are then regarded as "abnormal" by the rule. The authors claim that this clustering reaches good results with data that is binomially, normally, and multi-normally distributed. Purely categorical data are clustered in the same manner until a target percentage is reached.

[5] This is accomplished by using a data dictionary, that the rules can reference. This dictionary typically ends up around the the 10 KB-mark.

The rule base is then combined, where similar rules are condensed into one, and other rules, because they are not deemed significant, or because of concerns about tree size, are pruned from the tree.

Each rule is then assigned a grade measuring the "seriousness" of the rule. This grade is composed of two parts. The first part takes the historical accuracy of the rule into consideration, and the second part favours the rule with the longer RHS, i.e. the more specific the rule is, the more weight it is assigned by the grade measuring algorithm.

The rule base is not automatically updated, since the authors feel that this is a process that should run under the supervision of a trained SSO, perhaps once every 1–4 weeks.

2.4.4 Anomaly detection

The "sense" part of W&S then reads the rule-base, dictionary, and new audit records, either in batch, or as they become available. The inference engine then processes each audit record, finding the rules that apply, and computes a transaction score for that audit record. In doing so, the inference engine basically sums all contributions from the different failed rules—remember that we are searching for anomalies—and that the rules describe "normal" behaviour, taking the thread that the audit record is a part of into account. The thread score is updated, and aged by the previous process. W&S then reports an anomaly whenever the thread score, or individual audit record score, surpasses an operator defined threshold.

W&S aids the SSO in the task of anomaly resolution, it can help the SSO identify which data in the audit record was considered anomalous, it can list the rules that triggered the anomaly detection, provide a thread history, and suggest what data specific fields would have avoided the anomaly determination.

2.4.5 Results and future work

The process of detecting anomalous activity is quite fast. The authors report that W&S hhandles rule-bases of up to 500,000 rules averaging 6.0–9.0 bytes per rule, and 20,000–40,000 rule firings per second. Firing approximately 1% of the rules for a more ordinary rule-base of 100,000 instantiated rules, this gives a performance of between 20–40 transactions (audit records) per second. Searching the rule base can be done in under 0.05 seconds. These performance figures are for an IBM RT Model 6151-125, with an advanced floating point accelerator. The operating system is IBM's AIX version 2.1.

The authors recognise that even though they have performed sufficient tests, both in vitro, and using staged intrusion attempts, to ascertain the usefulness of the methods W&S use, more research into the nature of the computer security threat is needed.

The authors further state that W&S could probably be applied to other areas, both pertaining to security, and other fields, where the detection of anomalous data is of interest.

2.4.6 Survey conclusions

W&S is an interesting system in that it is one of the earliest systems, useful on a wide range of problems—not only in computer security—utilises anomaly detection, and does this using a novel approach to the subject. This approach would probably make for time efficient use of the detection resources, even though it is difficult to determine how effective the detection would be. (This is of course still an open question in the field of intrusion detection.)

The fact that the ideas behind W&S originated when studying security problems in another domain, that of nuclear materials control, makes W&S both unique, and interesting, and it supports the claims of the authors that W&S could probably be applied to many other fields of security—as well as other fields, that are not directly security related—such as the supervision of biological systems, for example.

2.5 The ComputerWatch data reduction tool

The ComputerWatch [11] data reduction tool was developed as a commercial venture by the Secure Systems Department at AT&T Bell Laboratories, as an add on package for use with the AT&T System V/MLS. System V/MLS is a B1 evaluated version of System V UNIX that provides multi-level security features that comply with the NCSC orange book B1 security criteria.

2.5.1 Introduction

The ComputerWatch tool operates on the host audit trail to provide the system security officer with a summary of system activity, from which he can decide whether to take further action, i.e. investigate particular anomalous looking statistics further. The tool then provides the SSO (Site Security Officer) with the necessary mechanisms for making specific inquiries about particular users, and their activities, based on the audit trail.

The B1 certification requires that the operating system provides the SSO with an audit trail that incorporates all security relevant events that have taken place on the system. Since the mere collection of this potentially voluminous amount of data can significantly affect system operation, the System V/MLS operating system has gone to some length to optimise the process of collecting audit data. The processing overhead of collecting data is less than 4%, this has been achieved by clever use of buffering to optimise disk traffic, and the use of a binary audit format, that reduces the individual records to an average length of 16 bytes. The latter minimises both disk traffic, and processing time. A problem with buffering of audit data, is of course potential loss of audit data in the event of a system crash, whether benign or malign. Another problem is that the binary format has to be translated to the database format before processing

ComputerWatch would normally be used by the SSO, with some periodicity, in an attempt to establish an overview of the type of system activity that has occurred. The tool provides the SSO with a summary report of this activity. This report can be perused as is, or certain entries can be automatically highlighted by the system according to a set of predefined rules, to provide a form of threshold highlighting capacity. The SSO then decides what type of activity, if any, merits further study, and can then make specific enquiries about users, and their activity to the audit trail database. Figure 2.1 provides an overview of the ComputerWatch system organisation, and the data-flow between the components.

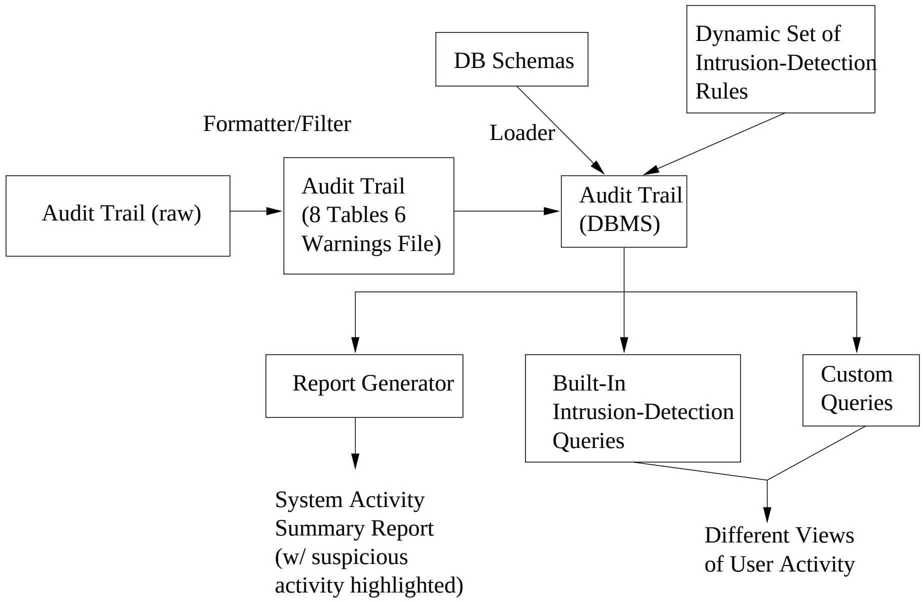

Figure 2.1: Dataflow diagram of ComputerWatch components (from [11])

2.5.2 DBMS

The database manager was especially designed for the task at hand. It is simple, small, and supports a very reduced set of SQL like operations, such as *project*, *select* etc. It contains no commands for data modification, since this could affect the integrity of the data negatively.

2.5.3 Report generator

The report generator generates a statistical summary of the system's activity since the last time the report generator was run. These statistics include items such as; number of failed logins, number of file accesses, whether these where reads or writes, successful or not, superuser activity, file declassification etc, etc.

The report generator module can operate in one of two modes, detection "on," or "off." In the detection "on" mode, the report generator applies a set of predefined, modifiable set of simple threshold type rules that may fire when applied to a certain statistic generated by the report generator. e.g. "The number of failed logins (58%) is too HIGH." In the other mode, this simple kind of detection is turned off. This saves processing time, and enables the SSO to study the statistics generated.

The authors claim that although there is some controversy over whether viewing statistics at a system level can help detect intrusions, they have found that such statistics can be useful in determining what kind of activity does not merit further investigation. i.e. if little or no file declassification is evident, this type of operation does not merit further investigation.

2.5.4 Queries Module

The queries module was designed to help detect "simple" breaches of security. These consist of a predefined set of possible queries that can be made to the audit database, either per user, or about the whole system. The set of predefined rules can be augmented with specific queries tailored to the situation at hand. The predefined queries are simple threshold like queries, they are designed to reflect the same kind of statistics that is evident in the system wide report, but at a user level. The SSO defined queries are made in the special SQL like query language of the DBMS. These can be made more complex than the simple predefined queries.

2.5.5 Rules Module

The SSO has the ability to augment and modify the rules applied to the system level statistics reported by the report generator. These rules can be of a simple set of predefined types, e.g. $value > threshold$, $value1 - value2 >= threshold$ etc. It is possible for the SSO to specify preconditions, i.e. other rules that must have fired for the equation of this rule to be considered. Error checking logic prevents the creation of rules with a preconditions list that would result in a loop.

2.5.6 Survey conclusions

ComputerWatch is one of a precious few systems that put the SSO in the focus of activity. ComputerWatch as a tool tries to visualise, and present data to the SSO, in a manner that enables him to operate more efficiently, and effectively in his role as monitor (guard if you will) of the system. This line of research has largely been ignored since this work was presented. (But, see [56], for a recent, cursory stab at the problem.)

Otherwise ComputerWatch feels slightly dated. Its merit would be that it is a very simple system, which would lend itself to efficient implementation. However, it is doubtful whether such a simple system could be effective today.

2.6 NSM—Network security monitor

2.6.1 Introduction

This section describes the latest published version of NSM [21, 45]. NSM is the first system to use network traffic directly as the source of audit data. NSM listens passively to all network traffic that passes through a broadcast LAN, and deducts intrusive behaviour from this input. This approach stems from the observation that even though several other intrusion detection systems try to ameliorate the problems of having several different forms of audit trails available from different platforms, the network traffic between these systems typically take place on broadcast networks, utilising standard protocols, such as TCP/IP, `telnet`, `ftp`, etc. Thus NSM can monitor a network of heterogenous hosts without having to convert a multitude of audit trail formats into some canonical format.

The authors identify several other benefits from using this approach:

1. The broadcast nature of these networks make the audit data almost instantaneously available to NSM. This in contrast with some host based audit trails that can delay the writing of audit records by several minutes, according to the authors. Furthermore, there is no need to transfer the logs to a separate computer for analysis, since they will already be available at the analysing computer.

2. The passive listening nature of NSM makes it impervious to (direct) attack, there is no possibility of an intruder corrupting the logs.[6]

3. Since NSM does not consume any resources on the monitored host, its performance will not degrade as a result of auditing, or analysis of audit trails. Furthermore, there will not be any loss of network bandwidth as audit trails are transmitted via the network to a centralised analysis machine, and the effectiveness of the intrusion detection is not affected by the administrative corroboration of the monitored hosts. I.e. the administrator of those hosts does not have to cooperate.

4. Finally, the authors are of the opinion that most of the serious intrusions involve the use of a network at some time, many attackers attack the system from a remote location, via a network for instance. The authors recognise however, that if the attack targets the host without accessing the network, NSM can do little to detect the intrusion.

2.6.2 System organisation

NSM follows a layered approach, called the Interconnected Computing Environment Model (ICEM). There are six layers in the ICEM:

1. The *packet layer*. This layers takes a bit stream from a broadcast LAN, i.e. an Ethernet, and divides it into proper Ethernet packets. These packets are timestamped and forwarded to the next layer.

2. The *thread layer* takes the packets from the packet layer, and correlates them into unidirectional data streams. These streams represent the data—sans packet headers—that are transmitted from one host to another, using a particular protocol and a particular set of ports. This stream is called a *thread*, and is passed as a set of *thread vectors* to the next layer.

3. The *connection layer* takes the tread vectors from the thread layer and attempts to pair them to form a bidirectional communication channel between sets of hosts. These connections are condensed into a *connection vector*, with some of the data gained from the lower layers pruned, and the reduced vector is sent on to the next layer.

[6] This is a moot point, see[49] for a detailed account of how an intruder could go about to avoid detection, or attack the IDS, in this scenario.

4. The data from the connection layer is accepted as input by the *host layer*, that condense several connection vectors belonging to a particular host together, to form a *host vector*, that represents the state of network activity of that host.

5. The host vectors are then combined—in the *connected-network layer*—into a graph, G, by treating the host-to-host information of the host vectors as an adjacency list. This layer can also build the sub-graphs of this graph, and compare those sub-graphs against historical connected sub-graphs. Furthermore, the user can ask questions about the graph to this layer. The authors make the example where a user (SSO) asks if there is a connection between two hosts—via any number of intermediate hosts—by a specific set of protocols. The graphs are passed on to the final layer as a set of *connected-network* vectors.

6. The final layer, the *system layer*, condenses the connected-network vectors into a single vector, the system vector, that describes the state of the entire system.

In the system described, only the host vectors, and the connection vectors are used as input to a simple expert system that analyses the data for intrusive behaviour. The expert system takes several other inputs, such as the profiles of expected traffic behaviour. These profiles consist of expected data-paths, which systems are expected to communicate with which systems, using what protocols. Another type of profile is constructed for each kind higher-level protocol, e.g. what does a typical `telnet` session look like.

Other types of input is the knowledge about the various capabilities of the protocols—e.g. `telnet` is a powerful protocol that enables the user to perform a variety of tasks—and knowledge about how well these protocols authenticate their requests. `Telnet` authenticates its requests, while `sendmail` requests identification, but does not authenticate this identification.

Furthermore NSM requests the level of security, as defined by the SSO, for each host, this figure could come from running a security analysis tool on the host. Finally, the last type of input that NSM requires is a collection of signatures of past attacks.

The data from these sources is combined to make a decision about the likelihood that a particular connection represents intrusive behaviour, based on anomaly reasoning. This is combined into a concept of the *security state* of the connection. This security state consists of four different factors:

Abnormality The abnormality of the connection is a function of the probability of the connection occurring, and the nature of the connection. This is based on the knowledge of the relative occurrence of the type of connection, for that pair of hosts, at that particular time, and the profile of the protocols involved, e.g. is it an `ftp` session with an unusual number of bytes transmitted or received?

Security level The security level of the connection is based on information about the capabilities of the protocol, and the authentication it requires. For instance, `tftp`, a very capable protocol, with no authentication, would rate high on the security level scale.

Connection sensitivity level Or rather the direction of the connection sensitivity level, i.e. which host in the pair initiated the connection, and what are the hosts' respective sensitivity levels.

Signatures of attack To what degree does the data transmitted over the connection match signatures of known attacks. These signatures are stored as simple strings, and a simple string match is made against the data transmitted.

The default presentation of the data to the SSO, is in the form of a sorted list, where each row in the list consist of a connection vector, and the computed suspicion level. The results are also stored in a database, whereby the SSO can make queries, about specific events he would like to take a closer look at.

2.6.3 Results

The authors report that the prototype system was deployed at UC Davis, Lawrence Livermore National Laboratory, and other DOE (Department of Energy), and US Air Force sites. In one two month period, NSM monitored more than 111,000 connections at UC Davis, and it correctly identified more than 300 of these as indicative of intrusive behaviour. These incidents spanned more than 40 different computers, four hardware platforms, and six different operating systems. Only about one percent of these attacks, (intrusions, and attempted intrusions) were detected by the system administrators of these systems. The system administrators operated in parallel with the evaluation, and without the benefit of utilising NSM.

2.6.4 Survey conclusions

The system presented is interesting in that it was the first to utilise network data directly, as a source of input. While not all the presented benefits of such a decision still hold true today—network technology has changed since the work presented was performed—many of them are still valid. The basic idea has still merit today, but paradoxically, more secure network technology, such as encryption, may thwart the effectiveness of this approach. How the switch to more secure network technology should be handled by intrusion detection system is a hot research topic today, but no real results have yet been presented. The authors also present *some* performance data, which is something that is all to often overlooked in recent research.

2.7 NADIR—An automated system for detecting network intrusion and misuse

2.7.1 Introduction

NADIR [22, 25] was developed at the Los Alamos National Laboratory, for use by the Laboratory to aid in its internal computer security effort.[7] As such NADIR was conceived with the problems and organisational needs of the Los

[7] It is not known what influence W&S (see section 2.4) had on the development of NADIR.

Alamos National Laboratory in mind. NADIR was designed to counter four different types of intrusive behaviour:

Disclosure Where someone (legitimate or an intruder) discloses information in violation of the security policy.

Integrity violation Where data or programs are subjected to unauthorised modification.

Denial of service Where the computer system is rendered temporarily or permanently unusable.

Unauthorised access Even though none of the above criteria are met, someone may use the computer system without authorisation. The authors note that many outsider attacks of course take their origin in this type of attack, where the intruder has not yet had time to perform an action that could be classified in the first three categories.

The authors stress that the first defence against any such violation is the "institution of formality of operations" and that such actions includes promoting safeguards, accountability, user training, and physical security measures.[8] The authors then go on to declare that intrusion detection has a place as a second line of defence, whereby intrusive behaviour can be detected and the proper authorities be notified. The authors have solid experience with manual audit review, and recognises that this practice has no real merit.

2.7.2 Overview of the computer installation

Since NADIR is closely tied to the computer installation it is put to protect, a discussion of said installation is not out of place.

The target system is the Integrated Computing Network (ICN) that is (was) Los Alamos main computer network, serving nearly 9000 users, and including six Cray-class supercomputers, many smaller computers, file servers, terminals etc. The ICN is divided into four partitions, each of which processes data at one defined security level, according to the US "military" security classification. All access to the ICN is through "ports" each of which connects to one partition. A computer that has connected to a partition through a port, can access computers in that partition, and partitions with lower security classification levels.

The system contains ICN service nodes that administrate the system. These service nodes store files, authenticate users, enforce access controls, schedule jobs, move files between partitions, provide hard copy output etc. These nodes also enforce the network partitioning, by, for instance, blocking access to classified files by unclassified users etc.

The service nodes can be divided into three classes:

Network security controller (NSM) This service node provides authentication and access control service to the ICN.

Common file system (CFS) Stores data that is to be made available to the ICN. It stores data from different partitions separately, and prevent access from lower-partition machines, to higher-partition data.

[8] It is difficult to over-stress the importance of this statement.

Security assurance machine (SAM) The SAM is responsible for all down classing of files in the ICN. It authenticates and records all attempts to perform this operation.

2.7.3 NADIR System organisation

NADIR is implemented on a Sun SPARCstation II, using the Sybase relational database management system. NADIR collects audit information from the three different kinds of service nodes discussed above. The audit data is collected and subjected to extensive processing before being entered into the relational database as audit information. The audit data consists of audit data pertaining to the different kinds of service nodes, and the network traffic that they generate and receive.

Each audit record entered into NADIR pertains to a specific event. The information for the event is logged; whether the corresponding action succeeded or not, and contains a unique ID for the ICN user, the date and time, an accounting parameter, and an error code. The rest of the record describes the event itself. This description varies depending on what kind of service node it originates from.

The data that is kept in the database of NADIR is treated as sensitive by the operators of the system and NADIR itself is part of the security hierarchy that Los Alamos operates under, to ascertain that NADIR itself does not become a security liability, instead of a security asset.

NADIR calculates an individual user profile on a weekly basis, where each record summarises the user's behaviour. The profile contains static information about the user, historic information about the user, such as; the number and a list of the different source machines from which the user has attempted to login to the ICN; blacklist—the number of times and the date upon which a user was last blacklisted.[9], and so on. The user activity field contains account statistics for different types of activity during the week for the three classes of service nodes, such as; source—eight counters that tally all attempted logins from source machines in each of the four partitions etc.

Furthermore, a composite user profile is constructed. This profile describes the system as a whole; the number of valid and invalid logins onto the NSC for each hour, for example.

These profiles are then compared against a set of expert system rules. These rules were derived from a number of different sources. First, and foremost, security experts where interviewed, and the security policy of the laboratory was encoded as a set of expert system rules. Second, statistical analysis of the audit records from the system was performed, and the results from this analysis was hard coded into the system as rules in the expert system.

NADIR generates weekly hardcopy reports in the form of activity summaries for each node, and various graphs plotted for user activity for that node. These graphs enable the SSO, or rather the SSO team in this case, to quickly grasp any abnormalities in the profiles. When the SSO decides to further investigate any of these reports, he can generate other reports on the spot, from either historical, or near real-time data. This helps in the investigation of both past intrusion

[9] Blacklisted individuals lose their ICN privileges under certain circumstances of unauthorised behaviour.

attempts, and ongoing suspicious activity. The raw audit data is also made available to authorised personnel for other statistical analysis, for example.

2.7.4 Results

The authors found that users responded very positively to the application of NADIR. The authors attribute this to three identified benefits beyond increased security:

Error detection The application of NADIR uncovered errors both in the auditing mechanisms, and in the audited systems, thus helping system managers to improve their systems.

Systems management The profiles produced by NADIR helped understanding how the system operated. In several instances the authors identified operation of the system that was not what was expected, or even specified.

User education The authors often identified non-malicious but undesirable[10] user behaviour, such as severe programming errors. In such cases the authors worked with the users to help them avoid such errors in the future. This the authors identify as having helped increase user support for NADIR.

The authors recognise that the evaluation of intrusion detection systems is difficult, because the frequency of positives (i.e. the actual number of intrusion attempts) is unknown. The number of false positives, i.e. legitimate use that is misclassified as intrusive is more straightforward to handle. NADIR has quite a high number of false positives according to its authors. However, since they envisioned NADIR as a highly interactive tool, they do not see this as major problem. Furthermore, they note that since the list of false positives is short enough to permit quick review, that also makes the problem tolerable.

The authors state that NADIR has not failed to detect an intrusion attempt that was subsequently discovered by other means. Also, NADIR has managed to detect intrusion attempts that were staged by security officers, as well as many real intrusion attempts such as: automated logins, misuse of special-use user numbers, attempted (unsuccessful) logins using another person's user number, attempted logins from terminals in partitions to which the user had no access, and attempted logins to computers in partitions to which the user did not have access.

For the future, the authors envision a system that would operate under near real-time constraints, the requirement mentioned is detection in under 30 seconds from the intrusion attempt, and to complement NADIR with a true anomaly detection component that learns the behaviour of the users of the system.

2.7.5 Survey conclusions

The work presented is interesting in that it is based on solid experience in the handling and nature of security incidents, in an organisation that takes these incidents seriously enough to have instigated manual computer security audit

[10] "Never attribute to malice that which can be adequately explained by stupidity"—Unknown.

review. This enables the authors to discuss issues relating to how the system can aid a site security officer in his task of monitoring the system, utilising audit data visualisation, and how an intrusion detection system can be fielded in such a manner as to gain the support of the users of the system.

This close ties with the organisation that NADIR operates in is perhaps a source of weakness as well. It is difficult to determine whether the experiences from the system would transfer to another environment, NADIR is perhaps too closely tied to the ICN at Los Alamos National Laboratories.

Furthermore, the authors clearly discuss the nature of the security violations they wish to detect, and to what degree they were able to do so.

2.8 Hyperview—A neural network component for intrusion detection

2.8.1 Introduction

Hyperview [7] 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 behaviour of a user and raises an alarm when the audit trail deviates from this already "learned" behaviour.

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—"raw" if you will—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.

2.8.2 Underlying hypotheses about user behaviour 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 the 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 acquires preferences vis-a-vis which way the task should be performed. Between 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 behaviour 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 a set 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—almost 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.

2.8.3 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, easiiy trained. However, there would be a number of problems with this approach:

- 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 that the correlations between input patterns are not taken into account with this model, since these type of networks learn to recognise 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 as 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 non-recurring ones.

2.8.4 System implementation

The design of the system as a whole is a complex and interesting one. The authors choose to connect the artificial neural network to two expert systems. One monitors the operation, the training of the network—to prevent the network from learning anomalous behaviour for instance—and evaluates the output of it. 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 the 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 2.2.

It becomes clear from the system graph, that the artificial neural network component of the system could be viewed as a filter that filters 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 labour 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.

2.8.5 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, of 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

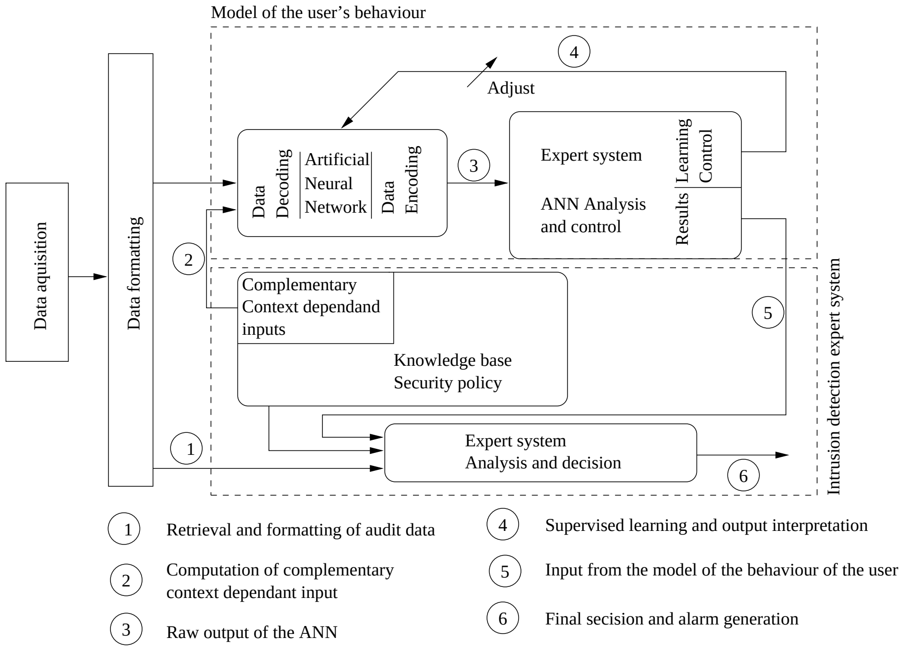

Figure 2.2: Block diagram of the Hyperview system (from [7])

sequentially, and asked to predict the next command in the sequence. When the next one was presented the network was retrained to reflect the new discovery. The commands, of which there where 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 59 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 user behaviour.

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.

The researchers 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 characterised 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 a `csh` or 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 of a more benign kind.

2.8.6 Conclusions

There were at the time publication of the system several problems with the use of artificial neural networks. There were, and to a certain extent, still are, few theoretical results on recurrent neural networks. The authors found it difficult to determine the correct size for the network, and the parameters would often have to be determined by trial and error thus leading to a time consuming design process.

Furthermore, since recurrent artificial neural networks are an example of systems with feedback there would be stability concerns. The researchers saw unstable configurations, that they could not, at the time, fully understand.

2.8.7 Survey conclusions

The paper clearly demonstrates that artificial neural networks could have a place in the detection of anomalous computer system activity. The present author feels that current interest in ANNs probably lie elsewhere, further research in intrusion detection has not employed ANNs to any significant degree. One could of course envision the use of ANNs for policy based detection as well, but the present author knows of no such approach.

The work presented is furthermore valuable in that it discusses the effectiveness of the approach, when subjected to test data.

2.9 DIDS—Distributed intrusion detection prototype

DIDS [52], is a distributed intrusion detection system, that incorporates Haystack (see section 2.1, on page 17), and NSM (see section 2.6, on page 31), in its framework.

2.9.1 Introduction

DIDS tries to correlate information about the individual monitored users, via a NID (Network Identifier) concept, where each user is tracked as he "moves" across the network, and in doing so, assuming different identities. Another strong point of DIDS is that it attempts to solve the problem of how to handle hosts on the network which do not participate in the host logging mechanism. DIDS attempts to keep track of actions performed by these hosts via the LAN manager, since each action such a host takes, eventually will manifest itself on the network level, if that host is to communicate via that network with the outside world. DIDS is specifically designed to deal with C2 compliant hosts in a heterogenous environment. In the prototype implementation the hosts typically run SunOS 4.1.1, with the BSM (Basic Security Module) installed, although the authors report on developing parts of DIDS to run on VMS.

DIDS is made of up of three major components. On each host, a *host monitor*, performs local intrusion detection, and summarises results, and parts of the audit trail for communication with the DIDS director. Furthermore each (broadcast) network segment houses its own *LAN monitor*, that monitors traffic on the LAN, and reports on it to the DIDS director. Finally, the centralised *DIDS director*, analyses material from the host monitors and the LAN monitors, that report to it, and communicate the results to the SSO.

2.9.2 Host monitor

The host monitor performs the local intrusion detection and reporting. It is made up of five major components, three of which are responsible for analysis of the audit data. The audit data is analysed in parallel. The five components are:

Preprocessor The preprocessor converts the raw audit trail into a canonical format suitable for further processing. It also filters the audit trail for superfluous audit data, and passes it along to the three processing elements.

Signature analysis The host monitor performs signature analysis, whereby patterns of known violations of security policies are scanned for.

Notable events The notable events processing analyses the canonical audit trail event by event (record by record) to determine if an event in and of itself, is sufficiently interesting to be forwarded to the DIDS director's expert system directly. This is always done for such events as logins, remote logins, etc. to help in the collection of the NID data, but also to feed the centralised expert system data that it is a priori interested in.

Haystack Each instance of the host monitor runs a copy of Haystack, to build session profiles of user and system behaviour, and to look for statistical anomalies in light of these profiles. Each such anomaly is transmitted to the DIDS director for further analysis.

Host agent The host agent is responsible for correlating the information produced by the three analysis components. In order to do so, it consults tables that list the higher level events that the three analysis components can generate, and which of these to send to the DIDS director's expert system for further analysis. Some of these events that are important for the system's construction of the NID:s are always forwarded to the DIDS director.

2.9.3 The LAN monitor

The LAN monitor is a subset of the NSM, the Network Security Monitor, developed at UC Davis, California, USA. The LAN monitor analyses every packet on the network to form a view of significant events on that network segment. This simple analysis identifies certain types of network behaviour; the use of certain protocols—`telnet`, `rlogin`—traffic that emanates from an unmonitored host, and therefore is interesting, etc., etc. among other things to identify users as they move across the network. Furthermore the LAN monitor constructs profiles of host behaviour; which hosts are likely to communicate with which hosts, utilising which protocols etc. The LAN monitor uses simple heuristics to try and ascertain whether a particular connection represents intrusive behaviour or not.

2.9.4 The DIDS director

The director is the brain of the DIDS intrusion detection system. The director contains the user interface by which the SSO can configure the system, and which he uses to attain knowledge about presumed intrusions etc.

The director consist of two main parts: the communications manager, and the expert system. The communications manager is responsible for collecting the data sent to it from the host managers, and LAN managers, respectively, and communicate this data to the expert system for further processing.

The expert system makes inferences about the security state of the system, and each individual host, and aggregates the information for presentation to the SSO. The expert system is an ordinary rule-based (or *production* based) expert system. It is implemented in CLIPS, a C language expert system implementation from NASA.

The low level events reported by the host and LAN monitors are asserted as facts in the expert system database. The reported facts are independent of the system of platform, from which they originated. The expert system then tries to assign a single identifier to each user of the system as a whole, the NID concept, and each user's activities are attributed to this NID. Events are then placed in context. Two major types of contexts exist; spatial and temporal. For instance, the authors give the example of some behaviour that would be perfectly innocuous when performed during business hours, but highly suspect when performed in the middle of the night, as an example of temporal behaviour. The expert system uses time windows to to correlate events that occur in temporal proximity.

Spatial events take into account the source of the event, certain events from one user may be more indicative of intrusive behaviour, than the same event originating from another user.

The NID-instance of a user, is represented using a four tuple {*session_start, user-id, host-id, timestamp*}, where each login to the system creates a new instance of a NID. DIDS correlate different users identity when they traverse through an unmonitored host, or where several connections from the outside world exist, to try to ascertain if indeed any of these connections could be another instance of a user already connected to the system. The authors state that even though they are well on their way to solve the problems with building the NIDs, there are some areas that remain yet to be addressed.

2.9.5 Results and future development

Preliminary trials with the prototype indicated that the system performed as expected, it managed to track users as they moved across the network, and correctly classified simulated intrusions as they occurred, however, no performance figures are available. Furthermore, the authors planed to develop host monitors that would monitor specific hosts, such as file servers, and network servers, in addition to ordinary user workstations.

2.9.6 Survey conclusions

DIDS incorporates two other systems in its design, Haystack, and NSM. DIDS addresses the question of how to handle distributed, heterogenous systems. There is precious little work in the field of how to handle heterogenous systems, some of which may not be willing to participate in intrusion detection. DIDS itself is not fully distributed, but relies on both distributed and centralised resources to detect intrusions. It is difficult to determine whether DIDS manages to make the optimal division of labour without any performance figures neither pertaining to the effectiveness, nor the efficiency of the system.

2.10 ASAX—Architecture and rule-based language for universal audit trail analysis

The paper describing ASAX [18] only describes a (proposed) prototype of the system, and hence, it cannot be fully surveyed.

2.10.1 Introduction

ASAX is a rule-based intrusion detection system, with a specialised, efficient language (RUSSEL) for describing the rules. In the views of the authors', there are four major problems with the automatic analysis of audit trails:

Disparity of security breeches There are several different types of security intrusions, and each of these require different methods to detect the intrusions. The authors state that the two main principles are; statistical modelling of normal behaviour, and modelling of experts' knowledge about known intrusive behaviour. The authors then go on to claim that: "The former approaches are appropriate to detect known penetration scenarios and the latter ones are appropriate to detect unknown intrusions." The present author does not know if this statement originates in poor proofreading, or if there is something more substantial underlying the reasoning. The former seems more likely...

Amount of audit data The operating systems, in and of themselves, provide a huge amount of data to be processed, and for storage and efficiency reasons this amount of data has to be culled. The authors differentiate between *preselection* where by the SSO determines what audit data the system should collect—being careful to collect enough data to be able to determine if an intrusion has taken place—and *postselection*, where the data is reduced in the later analysis stage. The authors state that it is probably wise to employ some sort of simple preselection in all cases, to lessen the burden on the analysis algorithm, freeing it from irrelevant data.

Reusability of the intrusion detection system The authors state that the intrusion detection system should be reusable, and then go on to define two different classes of reusable systems; *generic*, and *universal*. Characteristic of the generic system is that it can be instantiated for different types of audit trails, while the universal intrusion detection system is applicable to any audit trail, provided this has first been converted into a generic format. The authors put ASAX into the latter class.

User interface The authors state that: "An auditing system should have a suitable user interface allowing security officers to converse easily with the system and to take advantage of all its features." but then the scope narrows considerably when they go on to write that: "Practically, the purpose is to make a compromise between a powerful language allowing to express complex queries and to update the system knowledge, and an easy but less powerful language which does not require tedious training." While the latter certainly may be true, the present author feels that there is much more to the issue of SSO interaction, then the authors would have us believe.

2.10.2 ASAX architecture and operation

ASAX first converts the underlying (UNIX-like) operating system's audit trail to a canonical format— named NADF by the authors—and then processes the resulting audit trail in one pass, by evaluating rules in the RUSSEL language.

The audit trail conversion is aided by the fact that, in the authors words, NADF is simple and flexible enough to allow all existing audit trails to be translated in a straightforward way. Furthermore, the system saves information from the format translator in external files, to preserve the connection between the raw audit trail, and the translated audit trail, thus enabling later analysis queries to be stated with reference to the external format.

The RUSSEL language, is a declarative rule-based language, that is specifically tailored to audit trail analysis. The authors state that: "a general purpose rule-based language should not necessarily allow encoding of any kind of declarative knowledge or making a general reasoning about that knowledge." This in contrast with more general expert systems, such a P-BEST (see sections 2.2, 2.3, and 2.14), that the authors state is more cumbersome for the SSO to use. Rules in the RUSSEL language are applied to each audit record sequentially, they encapsulate all the relevant knowledge about past results of the analysis in the form of new rules, and they are active only once, requiring explicit re-instantiation when they have fired.

The authors claim that this language may still be too opaque for the average SSO, and hence a higher level language (RUSSEL2), that will be converted to RUSSEL was suggested as a further development. By nature of its simplicity, and straightforwardness, the authors envision an efficient implementation—more so than for example that of P-BEST—where the expressions in RUSSEL are converted to an internal code that can be efficiently executed on the target machine. They also suggest some (trivial) optimisation techniques, that could be applied.

2.10.3 Survey conclusions

The paper presents work that was somewhat immature, from a systems perspective, at the time of its publication. The introductory analysis has merit, and the system probably would also, had it been available for evaluation, by the authors at least.

The criticisms of other approaches (notably the P-BEST system) are not well founded in argument, and in the present author's opinion lack merit. One cannot make specific claims about differences in efficiency between an existing system, and a proposed one, without figures, based on the kind of loose argumentation presented here.

2.11 USTAT—State transition analysis

2.11.1 Introduction

USTAT [23, 24] is a mature prototype implementation of the state transition analysis approach to intrusion detection. State transition analysis takes the view that the computer is initially in some secure state, and via a number of penetrations, modeled as state transitions, the computer ends up in a compromised target state. (U)STAT 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.

2.11.2 More about state transitions

Table 2.1 depicts a UNIX intrusion scenario in which an attacker gains administrative privileges by exploiting a flaw present in the 4.2 BSD UNIX distribution.

Table 2.1: STAT: Penetration scenario (from [24])

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#`	

The specific flaw in *mail* is that it does not reset the setuid-bit when changing owner of the mail-file, that it has just appended the newly delivered mail to. The attacker can exploit this by copying a setuid command interpreter to the mail directory, have *mail* append essentially nothing to it, and at the same time have it change owner of the shell to the user *root*. The attacker has thus gained administrative (or root/super-user) privileges.

To model this scenario as a number of state transitions, we first identify the start and goal states. In order to execute the first step above, *root* cannot have a mail file, that is the first assumption that must hold. As we progress through the steps in the example, we find that also; the intruder must have write permission to the mail delivery directory, he must be able to execute *cp*, *chmod*, *touch* (or a variation thereof) and *mail*. The authors make the observation that on an ordinary UNIX system the first two assertions almost always hold true, and they can thus be ignored. 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!

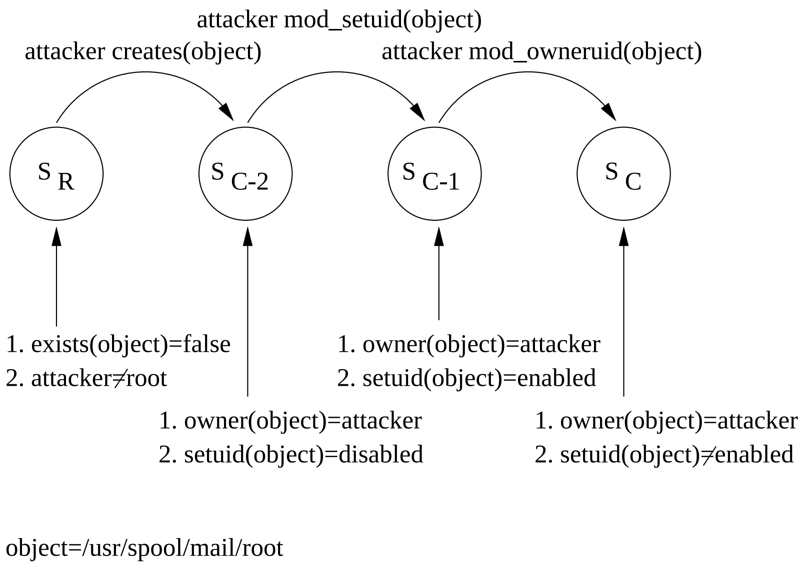

Figure 2.3: USTAT: State transition diagram (from [24])

The intrusion described above leads to the state transition diagram in figure 2.3. 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 abstracted 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.

2.11.3 The prototype system

In order to apply the state transition diagrams presented earlier the authors make two provisions:

1. The intrusion must have a visible effect on the system state and,
2. That visible effect must be recognisable without knowledge external to the system, i.e. the attacker's true identity for example.

Some types of intrusive behaviour, does not fall in the category described above. For instance, the passive monitoring of broadcast network traffic could be difficult to detect from outside of the resource employed to perform the monitoring. Another problematic intruder that is difficult to detect is the masquerader. However, if that masquerader then goes on to perform any of a number of intrusion-attempts, to gain greater privileges, state transition will have the opportunity of catching him. The C2 audit trail produced by the computer is used as the source of information about the system's state transitions.

The USTAT prototype is intended as a real-time expert system for detecting intrusions in real-time. The prototype runs on SunOS 4.1.1, with the SunOS BSM (Basic security module) installed. This module provides USTAT with a "C2" compliant audit trail. USTAT's design can be divided into four major modules:

Audit collection/preprocessing The purpose of this module is to collect audit data, and to store that data for future reference. In the prototype this module reads, filters, and passes the BSM audit records to the inference engine. The USTAT canonical format identifies the audit record according to the triple: {*Subject, Action, Object*}, where the normal BSM/UNIX audit record contents are mapped onto these fields.

Knowledge Base The knowledge base consists of two components, the rule-base, and the fact-base. The fact-base contains information about the objects in the system, i.e. groups of files or directories (called filesets by the authors) that share certain characteristics that make them vulnerable to certain types of attacks. 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 store the state assertions, depicted below the circles in figure 2.3, and the signature action table stores the signature actions, placed above the arcs in figure 2.3.

Inference engine The inference engine then evaluates the new audit records, using information from the rule-base, and the fact-base, and updating the fact-base with state information. The evaluation is done in a forward chaining fashion, i.e. new facts (audit records) lead to the evaluation of all rules that could depend on the newly asserted fact, and the fact-base is updated accordingly, and/or a possible intrusion is reported. The evaluation of the intrusion scenarios invariably lead to a lot of partial matching, the state of which has to be stored in the fact-base for possible future matching against new audit records, that could complete the intrusion scenario. These facts are stored in a table maintained by the inference engine.

Decision engine The decision engine informs the SSO that the inference engine has detected a possible intrusion. In the prototype, the decision engine reports the detected intrusion to the SSO, informs him whenever a state of any instance of the scenario has been satisfied, and suggests possible actions to the SSO to preempt a state transition that can lead to a compromised state. The authors suggest that a fourth mechanism could be added to make the decision engine respond actively to thwart the attack, as it progresses.

2.11.4 Results

The authors put the prototype to two kinds of tests, function as well as performance was evaluated. The prototype was put against a number of possible intrusion scenarios, and variations 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.

2.11.5 Survey conclusions

It is interesting to note that the idea behind the system presented started with the research into the question of how to represent intrusion scenarios. One problem the authors mention is that of representing possibly parallel prerequisite actions to prepare for the intrusion in the scenario. It is interesting to note that later work (presented in section 2.13 on page 53) has expanded on the model presented, while incorporating it in the mathematical framework of Petri nets. (It is not known to the present author whether the later research was specifically inspired by the approach taken here.) Despite this objection, the approach appears to have merit, especially since it lends itself to efficient execution.

Otherwise, the work is presented with unusual thoroughness, with performance tests and figures.

2.12 DPEM—Execution monitoring

2.12.1 Introduction

The author(s) make the observation that past efforts in the field of the detection of the exploitation of previously known intrusions have focused on the patterns of use that arises from these exploitations [29–31]. Instead the authors proposes that the opposite approach could be taken, i.e. that the intrusion detection system focus on the correct security behaviour of the system, specifically a security privileged application that runs on the system, as specified. The authors have designed a prototype, DPEM, that reads security specifications of acceptable behaviour of privileged UNIX programs, and checks audit trails for violations of these security specifications.

There are many different security relevant aspects of program behaviour. For instance:

Access of system objects The set of objects, typically files in a UNIX environment, that a program accesses as it runs. This is a simple, yet important measure. Many potential attacks can be detected if stringent demands is made on a program with respect to the files is can access.

Sequencing In some instances, it is not only the access to objects that matters, but also the sequence in which these objects are accessed. For instance, the *`login`* program should read the user authentication database file, i.e. `/etc/passwd` before executing the command interpreter for that user, failure to do so would be a security concern.

Synchronisation In a distributed system, security failures often result from improper synchronisation of programs. If, for example, a user changes his password, while the system administrator is updating the password file, the file may be left in an inconsistent state.

Race conditions This is a special case of the synchronisation problem. If a program has a race-condition flaw, an attacker can affect the operation of the program by performing certain operations during the execution of the program. This is difficult to monitor.

2.12.2 The specification language

In order to be able to specify these different requirements on the execution of privileged (UNIX set-UID) programs the authors specified a language, based on predicate logic, and a method for parsing this language, in which to specify the correct security benign operation of a program. More formally, the authors state that a *trace policy*, that captures the intended behaviour of a program, is specified by means of a grammar. This grammar defines a formal language (a set of sentences of that language) whose alphabet consists of program operations. Monitoring a program amounts to syntax driven parsing of the sequence of program operations executed by the subject. This sequence of operations (the trace from the execution of the program) is obtained from audit trails in real time. An unsuccessful parsing attempt indicates a violation of the trace policy and triggers remedial responses.

The authors have developed the reasoning about the specification language, and its grammar substantially during the period from the earliest publication of their results, to the later ones. An example of a specification for the *`finger daemon`* is in figure 2.4.

The specification in figure 2.4 first states the execution of the *`fingerd`* daemon, as user *`U`*. The allowed sequence continues with a rule that states that it is allowed to read file `X`, if, and only if, file `X` is world readable, i.e. there are no read-access restrictions, on it. *`Fingerd`* is then allowed to open port 79 for the reply, write to its log file, and execute the *`finger`* program to provide the remote user with the same output as he would have received, had he run the *`finger`* program locally. One would then have to specify the security policy

```
PROGRAM fingerd(U)
    read(X) :- worldreadable(X);
    bind(79);
    write("/etc/log");
    exec("/usr/ucb/finger");
END
```

Figure 2.4: ***Finger*** daemon example (from [30])

for the invocation of the ***finger*** program, in order to have a more complete example.

2.12.3 Design and implementation

The ideas presented have been implemented in a prototype named DPEM—the Distributed Program Execution Monitor. DPEM, as its name suggests, monitors programs executed in a distributed system. This is accomplished by collecting execution traces from the various hosts, and (possibly) distribute them across the network for processing. DPEM consists of a *director*, a *specification manager*, *trace dispatchers*, *trace collectors*, and *analysers*, that are spread across the hosts of the network.

More specifically; traces from the various hosts are sent on demand to a central location where the trace dispatcher combines the various traces to form one system wide trace, as requested by the then active analysers. Meanwhile the specification manager consults its database to se if any of the processes recently started by any and all particular users should be monitored. If this is found to be the case, the specification manager distributes the process started by the particular user to be analysed by an analyser. At no time will any subject/process pair be monitored by more than one analyser. The analyser then applies the specific trace policy to the trace obtained from executing the process, and decides if a violation has taken place. If so, the analyser reports a violation to the site security officer, by configurable means.

2.12.4 Performance of the prototype

The prototype was implemented in C, on top of the Solaris 2.4 operating system, using the SunBSM audit subsystem, to collect audit data. When run on a Sun SPARCstation 5 with 32 MB of memory, and activating it with well known vulnerabilities in ***rdist***, ***sendmail***, and ***binmail***, the system responded quite quickly, and reported policy violations in under 0.1 seconds in all cases.

2.12.5 Survey conclusions

This is the first, and perhaps only, example of a system that utilises policy based detection, that is a policy with a default deny stance. Furthermore, the presented work discusses the nature of the intrusive behaviour that the method could detect. The discussion on scalability, and how the system distributes is also thorough. The test cases could be more thorough, but given the current state of affairs, one must of course be satisfied with the fact that the authors make any claims of the effectiveness, and efficiency of the system at all.

2.13 IDIOT—An application of petri-nets to intrusion detection

2.13.1 Introduction

IDIOT [6, 32–35], is a system developed at COAST, University of Purdue, IN, USA. The basic idea behind IDIOT is to employ coloured Petri nets for signature based intrusion detection. The authors suggest that a layered approach be taken when applying signature (pattern matching) based techniques to the problem of intrusion detection.

2.13.2 Model

The authors suggest 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 behaviour in a system independent fashion, by the use of a virtual machine model.

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

The proposed model has as it's 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 an "immediately follows" semantics, in terms of the events that the matcher would see. The authors have previously identified that UNIX attacks could be classified, from a signature perspective into the following classes [32]:

Existence The mere fact that something ever existed is evidence of an intrusion in some instances. For example, searching for the presence of a particular file, with particular permissions could provide enough evidence.

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

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

Duration Something happened for not more than, or less than, x seconds.

Interval Events took place an exact (plus or minus some delta) interval apart. Thus, the specification says that event y took place not, more than t_1 or less than t_2 time after event x. The exploitation of a race condition typically gives rise to such specifications.

The authors believe that the majority of known intrusions in UNIX systems fall in the first and second category above.

2.13.3 Applying Petri nets to the proposed IDS model

The authors argue that of the many available techniques of pattern matching, coloured Petri nets, or CP-nets for short, would be the best technique to apply, since it does not suffer from a number of shortcomings common to other techniques. These latter techniques do not allow conditional matching of patterns, do not lend themselves to a graphical representation etc.

The proposed Coloured Petri Automaton (CPA henceforth) differs from "regular" CPA:s in a number of respects, they lack concurrency for example. They retain all the features necessary for use in intrusion detection however. For an introduction to CPAs the authors recommend [26].

For an example of a specification of an intrusion signature using the proposed Petri nets, see figure 2.5.

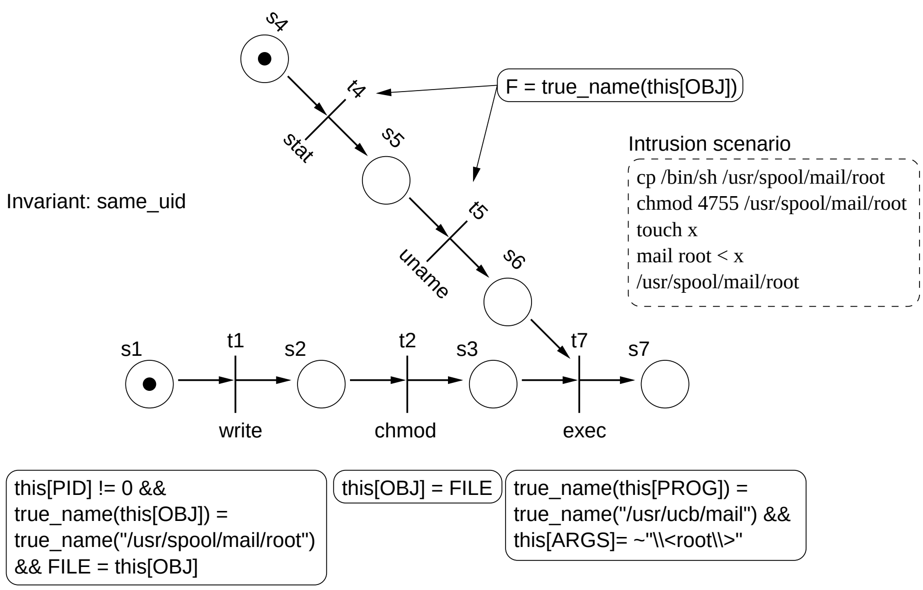

Figure 2.5: IDIOT: A Petri-net intrusion signature (from [34])

In 2.5 the start states are $s1$ and $s4$, with $s7$ the final state. The evaluation begins with the system placing a token in each start state. These tokens[11] then "flow" through the Petri net making the transitions specified by the vertical bars in the picture, when the event marking the transition occurs. 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 preconditions, postconditions, and invariants. These are similar to guards that must be *true* to be successful. Patterns

[11] Each token also has a set of variable associated with it, it is coloured in CP-net terminology.

that have no transitions can be specified using pre-conditions to an empty pattern. The authors state that post-conditions are provided for symmetry, and to allow recursive invocation of the same pattern. Invariants are provided to allow the user to specify some condition that should not be true while the matcher is busy with another pattern. The authors felt that it would unnecessarily clutter the Petri net to introduce these negative conditions directly.

In order to instantiate this generic model to a specific platform, say UNIX the user would have to define the primitives supported by the guard expressions, coding file test operations, system interaction hooks etc., etc.

2.13.4 System overview

IDIOT consists of four major components:

Audit trail Of course technically the audit trail is not a part of IDIOT, even though 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[12] 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 labour is intended to ease porting of IDIOT to other platforms, with other forms of audit trails.[13] *showaudit.pl* can be run either in batch mode, to convert an already existing audit 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 an audit record from *showaudit.pl*, steps through the different intrusion detection patterns (implemented by a pattern matching engine 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.

Of the parts described above, only the audit trail, and the *showaudit.pl* script needs to be ported when moving IDIOT to a new platform, the C2_engine, and (where applicable) the patterns can be moved unchanged to the new platform, they are intended to be platform independent.

The patterns play a major role in IDIOT, they are written in an ordinary textual language, and parsed, resulting in a new pattern matching engine. As previously mentioned this engine can then be dynamically added to an already running IDIOT instance, via the user interface. Furthermore, the user can extend IDIOT, to recognise new audit events, for example.

[12] This audit trail generation mechanism is designed to provide an audit trail that meets the "Orange book" C2 criteria. [13] This is the same approach as taken in NIDES, see section 2.14 on page 56.

2.13.5 Survey conclusions

The work presented is thorough on the nature of the intrusions that the system is supposed to detect. It is in fact by far the most thorough presentation in all the work surveyed. The description of the patterns that describe the intrusions is based on theoretical foundations, and thus not ad-hoc in nature. The authors furthermore stress the necessity of testing the effectiveness of intrusion detection, by building a set of standard test cases. Although the work presented is a few years old, this has not yet been performed, although at the time of writing such an effort appears to be underway.

2.14 NIDES—Next generation intrusion detection system

NIDES [1, 2] is the direct continuation of the IDES project (see section 2.3). Following the tradition of its predecessor it is very well documented, there are many more references available than the two given here.

2.14.1 Introduction

It is difficult to speak of one NIDES system—a trait it has in common with its ancestor—there are really four different systems, each built on top of the previous system. NIDES follows the same general principles as the later versions of IDES, i.e. it has a strong anomaly detection foundation, complemented with a signature based expert system component. The latter component is implemented using the P-BEST expert system shell. This is a later version of P-BEST than that presented in the survey of MIDAS (see section 2.2), implemented in C, and generating C as output.

2.14.2 The major versions

NIDES development resulted in four major versions of the software, each with refinements, based on input both from further research at CSL-SRI, and user input. The four major versions are presented in the following paragraphs.

NIDES Alpha—Feb 1993

This release was really a version of IDES, the same functionality was there, but the architecture was changed. NIDES is more architecturally sound, it is modular and built on a client-server architecture.

Furthermore, while the rule-based intrusion detection system remained the same, the anomaly detection functionality was changed. This change came about to enable NIDES to deal not only with simpler parameterised distributions, but also with multi modal distributions, such as could arise from a user that performs two completely different tasks; he is either developing software, or he is writing documentation for that software, and on Friday mornings he is busy using the time reporting software. Usage patterns such as this one requires a statistical model that takes the different *modes* the user is in, into account.

NIDES Alpha patch—Oct 1993

This version was a result of user feedback as to the performance of NIDES. In order to speed up development, no changes were made to the user interface proper, all features introduced where to be controlled by setting environment variables, or writing/changing configuration files.

Three changes were made in an attempt to alleviate what was being experienced as poor performance:

1. The statistical analysis component stores information about file and directory accesses as lists. It was found that these lists could grow quite large—thousands of entries—in some circumstances. In order to alleviate this problem, the authors redesigned the analysis algorithm so that it need not traverse the entire list at the time of audit record processing, this processing was moved to profile generation stage instead.

   Furthermore, it was found that many of the files that where considered in the previous paragraph was of a temporary nature, and that they would not be included in the final profile of that user anyway. NIDES was thus extended to be able to ignore those files, by naming directories such as `/usr/tmp`, `/tmp`, etc. to be excluded from further processing.

2. A feature was added to give the user the choice of having the real-time NIDES update profiles based on the audit record timestamps instead of the real time clock, exclusively.

3. A user configurable subject profile cache was added, to speed up processing in the anomaly detection module.

NIDES Beta—May 1994

This represented a major overhaul of the NIDES system. Several new features were added along with a new user manual. The features were (the list headings verbatim from [1]):

- Optimised profile storage structure. NIDES generates two files per subject for the storage of short term, and long term profiles. Users of NIDES with many subjects expressed concern that the profile storage consumed too much space, and by judicious culling of the stored data, as well as some format changes, the authors were able to reduce the baseline storage requirements of the two files by as much as 62%.

- Real-time configuration of NIDES analysis, both for real-time detection, and batch mode detection. The beta release of NIDES introduced extensive possibilities for the SSO to configure almost any aspect of the analysis components, from how detected intrusions are reported, to the various parameters that govern the anomaly detection component of NIDES. This (re)configuration can be performed in real-time, when NIDES is running.

- Expanded status reporting. The status reporting from the running NIDES was improved in three major areas:

  1. NIDES reports extensively on various measures of throughput, and state, for analysis and data storage functions.

2. NIDES reports the status and configuration of each monitored host.
3. When NIDES is analysing audit data in batch mode, the status, and summaries of alerts are reported periodically.

- Data management facility. This facility enables the SSO to archive and retrieve audit data, and result data from processing.
- Expanded rule-base. The policy based detection module had its rule-base expanded from 21, to 39 rules.

NIDES Beta-update—Nov 1994

The is considered the final release by the authors, and it consisted of bug-fixes, performance improvements and added features.

The main performance improvement improved on the file access statistics modifications introduced in the NIDES alpha patch release. It was noted that some users would access on the order of 100,000–300,000 files in a four day period, even though temporary files were already not being considered. However, most of these files would not be included in the user profile for that subject anyway, due to them being deemed insignificant by the statistical profile generator. NIDES was thus enhanced to be able to remove these files from the outset, and thus they would not later come to burden the statistical profile generator. This enabled NIDES to process these difficult cases in a matter of hours, instead of aborting processing altogether.

The major new features were:

- Introduction of the Perl script `agen`. Previously the converter from the host specific audit trail to the NIDES canonical format was written i C, somewhat limited, and hard to port. To alleviate these problems a Perl version was constructed, since Perl is powerful, and available for a number of different platforms. A number of sample Perl scripts to interface the auditing mechanisms to NIDES where also provided.
- An `agen` monitor for promiscuous Ethernet interfaces. To be enable the detection of the use of "sniffers."
- Expanded audit record fact template. This was performed to enable the rule based detection part of NIDES to consider all the available fields in the audit record for detection. Previously the expert system was only aware of a smaller subset of the possible fields.

2.14.3 System organisation

The NIDES system is highly modularised, with well defined interfaces between components, and built on a client-server architecture. The system is centralized in that the analysis runs on a specific host, named the *NIDES host* by the authors, and collects data from various hosts via a computer network. The *target hosts* collect audit data, from various host-based logs—there is a provision to utilise TCP WRAPPER [55] i.e. host-based network traffic logs—converts them to the canonical NIDES format, and transmits it to the NIDES host. The SSO interacts with the system through the NIDES host.

The key components of the NIDES system are:

Persistent storage This component provides the rest of NIDES with storage management functions, for audit data etc.

Agend The *agend* process runs on all NIDES target hosts, and is responsible for starting and stopping the *agen* audit data converter, when instructed to do so by the NIDES user interface. It is implanted using the RPC protocol.

Agen This is the audit data converter, that converts audit data to the NIDES canonical format. The converted audit records are then handed to the *arpool* process.

Arpool This process collects the audit data from *agen* and provides it to the statistical, and rule-based analysis components on demand. *Arpool* runs on the NIDES host.

Statistical analysis This module performs the statistical intrusion detection. In real-time or in batch mode, i.e. non-real time.

Rule-based analysis This module performs the signature based intrusion detection, also in real-time or in batch mode fashion. Both these modules report their findings to the resolver.

Resolver This component is responsible for evaluating, and acting on the data received from the statistical and rule-based analysis modules. The authors state that a user action could well result in tens or hundreds of different alarms, in order not to drown the SSO in irrelevant alarms, the *resolver* aggregates them and makes a compound decision. The SSO also has the ability to turn off reporting altogether for specified subjects, that for instance, is performing some known new task etc. that upsets the anomaly detection.

Archiver The *archiver* is responsible for storing audit records, analysis results, and alerts.

Batch analysis The batch analysis component allows the SSO to experiment with new configurations on old, known audit data, in parallel with running the production NIDES system.

User interface The user interface is responsible for communicating with the SSO. This is the place from where the SSO controls all of NIDES, and NIDES reports suspected violations of security to the SSO, as well as a wealth of general processing status. Only one instance of the user interface can be active at one time, and it always runs on the NIDES host. It is implemented using the MOTIF toolkit under X-Windows.

2.14.4 Experimental results

The authors have made extensive experiments with a version of NIDES modified to study application behaviour, instead of (typically) user behaviour. For a detailed account of these experiments the reader is referred to [2]. A short summary of the findings is that NIDES is indeed capable of detecting these types of anomalies, and both the false positive rate, and the false negative rate can be kept at reasonable levels.

2.14.5 Future directions

The authors list several possible future enhancements to NIDES, the construction of a test bed to enable testing of intrusion detection systems etc. However, one of the more interesting areas mentioned regard the threat against NIDES itself, and the increased risk to the computer installation that the employment of an intrusion detection system could result in.

The authors state that they perceive two major areas of risk when it comes to NIDES; tampering, and reverse engineering. Tampering would seek to render NIDES ineffective directly, by shutting it down, for example. By reverse engineering the attacker would attempt to learn of NIDES's rule base, for instance, and armed with this knowledge he would attempt to devise an attack that would go undetected by NIDES.[14]

The authors state seven security goals for NIDES:

1. Target system integrity. NIDES should not have an adverse effect on the target systems, and should authenticate all interactions, as well as keep track of the status of the target systems, to monitor any unscheduled shutdowns.
2. Audit data security. Since the audit data itself could be very interesting to an attacker, NIDES must protect the confidentiality, integrity, and availability of this audit data.
3. NIDES system integrity. NIDES itself must be protected from undue outside influences.
4. Availability. NIDES must remain available, and not fall to denial-of-service attacks.
5. Rule-base protection. The major dangers with the rule-base is that of reverse engineering, and of course undue modification. NIDES must resist these types of attack.
6. User access. Due to the sensitive nature of the data that NIDES processes, access to NIDES itself must be restricted to authorised personnel only.
7. User accountability. The authorisation of personnel using NIDES is not enough in itself, NIDES must itself be monitored for management to be able to hold users of NIDES accountable, and to detect attempts at misuse, and intrusions by unauthorised personnel.

The authors further state that with the modular architecture of NIDES, improvements to make it more tamper-resistant, and able to withstand reverse engineering could include: separation of roles, interprocess and server authentication, detection of—and protection against—denials-of-service, improved target system integrity/availability, `arpool` target host authentication, smokescreen detection, logging of NIDES user actions, improvement of NIDES processes integrity/security, improvement of alert reporting integrity, authentication of NIDES users, restrictions on TCP/IP services, protection of the rule-base and software from reverse engineering etc.

[14] "Flying under the radar" so to speak.

2.14.6 Survey conclusions

It is interesting to note the position of NIDES between IDES, and EMERALD (see section 2.19 on page 72). Many of the components of EMERALD clearly was incepted in the NIDES project, generalised, and carried over to the EMERALD project.

The "future directions" section in [1] is interesting in that it is one of the first to recognise the threat against the intrusion detection system itself, as well as the fact that the employment of an intrusion detection system in itself could result in an increased risk to the computer installation.

NIDES in itself represents a major research effort, strong in; theory, discussion of effectiveness, implementation, and last but not least, documentation. The researchers at the three CSL/SRI projects referred in this survey (IDES, NIDES, EMERALD) have made available more documentation[15] than the other systems surveyed put together.[16] This of course, of great value to the research community.

2.15 GrIDS—A graph based intrusion detection system for large networks

2.15.1 Introduction

The authors suggest a method for constructing graphs of the network activity in large networks, to aid in intrusion detection [53]. The graphs typically codify hosts on the networks as nodes, and connections between hosts as edges between these nodes. Which traffic is chosen to represent activity in the form of edges is decided on the basis of user supplied rule sets. The graph globally, and the edges locally, have attributes, such as time of connection etc., that are computed by the user supplied rule sets. The authors suggest that these graphs present network events in a graphical fashion that enables the viewer to ascertain if suspicious network activity is taking place.

2.15.2 Design goals

The authors identify some large scale network attacks:

Sweep A sweep occurs when a single host systematically contacts many other hosts i rapid succession.

Coordinated attacks These attacks are multi-step exploitations using parallel sessions where the distribution of steps between sessions is designed to obscure the unified nature of the attack, or to allow the attack to proceed more quickly.

Worms A worm is; "a program that propagates itself across a network using resources on one machine to attack other machines."

[15] Most of this documentation is available on the web at the time of writing, at: http://www.csl.sri.com. [16] The IDIOT project, see section 2.13 on page 53, is also strong in this respect. See ftp://coast.cs.purdue.edu for more information.

2.15.3 Paradigm

The construction of the network activity graph is based on the organisational paradigm of a hierarchy of departments. A department consists of several hosts, and the department centrally collects audit data and combine it into department graphs according to the specified rule sets. If network events in the department involve entities (hosts) outside the department, then the network graph of the respective departments can be combined, according to rules specified in a rule set. The new graph consists of nodes that are the two departments, and edges that represent the network traffic between them. This recombination is done at the next higher level of departments, that include the two original departments. This process can be repeated, and a hierarchy of departments is formed.

2.15.4 Graph building

Reporting all network activity in one single graph would be unwieldy. Therefore, the system allows several rule sets that define one graph each. All the collected data are considered for inclusion in all the rule sets, and thus two different rule sets could render the same audit data into two different graphs.

2.15.5 Rule sets

The rule sets serve several purposes, to decide whether to combine graphs into higher level graph, to control how this combination should take place, how to compute the attributes of the graphs—both originally, and when they are combined—and to decide what *actions* to take, if any, when graphs are constructed, or combined. The last point is interesting, because it is in this activity that the automatic intrusion detection capability of the system lies.

The rule sets can be quite complicated to specify, and especially to specify correctly. Because of this, GrIDS contains a policy language, with which to specify policies of acceptable, and unacceptable network behaviour. The prototype implementation allows the user to specify whether a connection, represented as an edge in the graph, is allowed or not. The specification is in the form of a tuple, {*action, time, source, destination, protocol, stage, status, ...*}, where action is *allow*, or *deny*, *time* qualifies the rule with respect to a clock or time interval, *source*, and *destination* describe the connection endpoints, and *protocol* describes the connection type. As a connection progresses though its stages, i.e. start, login, authentication, etc., the stage, and status attributes further characterise the connection.

2.15.6 Implementation

The GrIDS system consists of a *software manager*, a *graph building engine*, a *module controller*, and *data sources*.

All software in the GrIDS system consists of configurable modules with a standardised interface. The specialised *software manager* module manages the state of the hierarchy, and the distributed modules. The *data sources* are modules that monitor host, or network, activity and reports this to the rest of GrIDS. The *graph building engine*, receives data from the data sources, applies the different rule sets to build the graphs, and reports summaries of graphs to higher departments.

2.15.7 Survey conclusions

The GrIDS system purports to be "graphical" but it is difficult to ascertain just what the authors intend by that statement. Certainly, it is not meant to display network information to the SSO in a way that enables him to more easily detect anomalous behaviour. The work presented is furthermore weak in the areas of effectiveness, and efficiency. The authors discuss scalability in a convincing fashion however, and the system probably has merit in this respect.

2.16 CMS—Cooperating security managers

2.16.1 Introduction

The authors of cooperating security managers [58] make the observation that as networks grow larger, centralised intrusion detection will not scale well with them. In order to alleviate this problem they suggest that several intrusion detection agents, one at each computer connected to the network, cooperate in a distributed fashion, where the computer from which a user first entered the system is made responsible for all that user's subsequent actions. This, the authors claim, result in the load being evenly distributed among the cooperating entities.

Three main ideas stand behind cooperating security managers:

1. Each computer on the network run a copy of the security manager. This manager is responsible for detecting intrusions—anomaly and signature based—on the local system, as well as intrusive behaviour originating from an original user of the machine. In order to accomplish the latter, the manager collects information from other computers the user under interest may be connected to. This results in an assurance that one computer, the one the user first connects to, will have a complete record of that user's activity on the networked computer system.

2. When a user accesses a host from another host, the managers in question connect to each other, and communicate information about the user, and the nature of the connection. This helps in tracking the user, but more importantly, the authors claim, helps in preventing spoofing attacks.

3. Finally, the SSO has the ability to initiate a trace request, where by a user's connections across the network are traced. If this results in the user appearing to connect from two distinct locations, the alarm is raised, since ordinarily, users are never in more than once place at any one time.

With these features, the managers can cooperate to detect intrusive behaviour through out the network, without having to rely on some centralised resource. The authors does not describe a fully developed system, but a fairly full featured prototype version.

2.16.2 System overview

The cooperating security managers system is comprised of five separate components:

Local intrusion detection system This is the system that is responsible for detecting intrusive behaviour local to the computer on which it is running. The local intrusion detection system uses command based audit logs as its source of audit data. The authors envision that future enhancements will include GUI-based command interception, and the ability to include monitoring of protocols such as `http`. The system maintains a suspicion level for each user that it monitors, and one system wide suspicion level. When either of these suspicion levels exceed a SSO-defined threshold, the system generates an alarm to that effect.

Distributed intrusion detection system Is responsible for detecting intrusive behaviour that originates from this host, by a user who has connected to another host through the network. This task is accomplished by communicating with both the local intrusion detection system, in order to learn about the user's activities, and to communicate with the remote system that the user has entered.

User tracking system Tracks a user as he moves across the network. The user tracking is integrated with the distributed intrusion detection above, since in order to be able to receive audit data on the user from the remote computer, the system has to keep track of him. Information about the user's path through the system is also needed for intrusion alarms, since cooperating security managers report any suspected intrusive behaviour to all systems that the user has traversed.

Intruder handling Decides what action to take, if any, when possibly intrusive behaviour has been detected. The module is able to terminate the suspect connection if so configured, i.e. this module has active response capabilities. This termination can be the result of SSO input via the user interface, but also the result of preset suspicion thresholds being reached, whereby the intruder handling module can be configured to terminate the connection automatically.

User interface The user interface component has the responsibility to communicate with the SSO, to configure the system, report alarms, to track users as they move through the network, and to terminate a suspicious connection.

2.16.3 Prototype test results

The prototype was evaluated by running a variety of prepacked exploit scripts. These were of two types. The first was designed to test the system's ability to detect intrusions being performed locally, and the second type was designed to test the ability to detect intrusive behaviour on other hosts.

A small number of these simulated intrusions were performed, and the authors report favourable results. Cooperating security managers operated as specified, the SSO was able to sever connections, and trace users across the network, even when the paths of two such users crossed. Furthermore, each instance of intrusive behaviour was correctly identified as such, and when thresholds were exceeded appropriate alarms were generated.

2.16.4 Survey conclusions

The work presented is (yet) another approach to the solution of the problem of how to make intrusion detection system scale, when introduced into a network environment. It fails to address the problem of how to handle scalability of the SSO in such a situation. The present author knows of no research that has continued along the line of reasoning presented.

2.17 Janus—A secure environment for untrusted helper applications

Janus [17] is a security tool inspired by the reference monitor concept, and Janus was developed at the University of California, Berkeley. While Janus isn't an intrusion detection system per se. It has many interesting similarities with specification based intrusion detection, and it's high degree of active influence over the running application makes it an interesting case-in-point, when studying active response.

2.17.1 Introduction

Janus is a user-space, per application reference monitor that is intended to supervise the running of potentially harmful web browsing helper applications. It does this by enclosing the application in a restricted environment, as so called "sand box." In the words of its authors, the main idea behind Janus is:

> An application can do little harm if its access to the underlying operating system is appropriately restricted.

From this statement the authors derive the corollary that the application is thus allowed to perform any action as long as that does not entail the invocation of a system call to the underlying operating system.

Janus is a policy based tool, where the user specifies a (restrictive) "default deny" policy on what type of actions the supervised program may perform. If an action is not explicitly allowed, Janus by default will not permit it.

The authors discuss several other possible solutions to the problem of hindering applications from making malicious system calls—such as modifying the operating system kernel—and reject the proposals one-by-one, arriving at the present solution.

2.17.2 Architecture

Janus is implemented as a set of security modules, connected through a framework. Janus utilises the `/proc` interface of the Solaris 2.4 operating system—originally intended as a system call tracing, and debugging facility—to watch over the monitored application. Since a rogue application has complete control over its own address space, Janus is implemented in a separate address space, that communicates with the supervised program via the `/proc` facility. See [28] for an introduction to the `/proc` "processes as files" interface.

The Janus framework first reads a configuration file that defines the security policy, and consequently which modules should be loaded. These modules then register interest in certain system calls with the operating system, and the supervised application is started. During the running of the application the supervised system calls are checked against the current policy in a top down fashion, where the most general rules are checked first. Latter, more specific rules may override these.

The security modules (also called policy modules) may also contain special code that is run just prior to the operating system receiving the system call parameters, in order to check these parameters for illegal activity.

2.17.3 Security modules

Several policy modules are implemented, each responsible for supervising one part of the policy. Examples of such modules are:

basic This module supplies defaults for the simple system calls that for instance are always denied, such as *`setuid`*, *`mount`*, *`chdir`* etc. These calls are always denied because they would either not be allowed to run by an unprivileged process in the first place, or they violate the basic "sand box" that Janus tries to enforce.

putenv The *`putenv`* module sanitizes the running application's environment, so that dangerous variables such as *IFS* etc. are removed.

tcpconnect A special module exists that supervises the application's network activity. The user may as a matter of policy restrict with whom the application may communicate, using which ports etc.

path The *`path`* module is the most complex module. It supervises all *`open`* requests, and checks the requested pathnames, only allowing access to the sand box environment, and a small set of carefully chosen system files, such as shared libraries necessary for the running of the application, and system wide configuration files. The *`path`* module differentiates between read and write access. While it would for instance allow read access to the `.rhosts` file, it would universally disallow write access.

When the framework detects that a policy module would disallow a certain system call, it aborts the system call with an *EINTR* error, before it has been executed. To the supervised program this is indistinguishable from an interrupted system call, and some programs are designed to retry the system call when this condition becomes true. Janus detects this situation when 100 invocations of the same system call has been denied, and then opts to kill the application completely. The authors note that they would like to abort the system call with a more appropriate error code, perhaps *EPERM*, signalling that the process had insufficient privileges for the operations. However the Solaris *`/proc`* interface lacks this facility.

2.17.4 Results

The authors note that since the user community general is more interested in performance than security, the performance aspects of their implementation must be studied. In addition, they study the applicability, ease of use, and security of their prototype.

When Janus is run on two typical, performance critical applications, *ghostview*, and *mpeg_play*, no significant slowdown is encountered. This the authors attribute to the fact that since only "important" system calls are supervised[17], and that system calls are already expensive, the extra overhead of Janus is not noticeable. The application is free to go about it's business in the common case.

2.17.5 Conclusions

The authors stress that there are several other possible routes that could be explored towards the solution of the stated goal of achieving security in the face of complex helper applications that receive their input from suspect sources. For example security logging could easily be added to provide after the fact capabilities. However, they feel confident in having proved their concept a workable solution to the problems at hand.

The authors list the URL: http://www.cs.berkeley.edu/aw/janus/ as a reference for more information about the Janus system, and its availability.

2.17.6 Survey conclusions

Janus is an interesting and fresh approach to the problem of intrusion avoidance. While the authors don't present it as such, it could be argued that their implementation is an intrusion detection system, with a default deny policy specification, and strong active response. It is interesting to see that the authors have addressed the problems of ease of use and performance. The paper shares the weaknesses in the areas pertaining to *effectiveness*, so common in intrusion detection research to date.

2.18 JiNao—Scalable intrusion detection for the emerging network infrastructure

2.18.1 Introduction

The authors have developed a prototype implementation of JiNao [15], a network intrusion detection system aimed at protecting the network infrastructure itself, rather than the individual hosts on that network. The threat model assumes that some routing entities in a network can be compromised, and hence begin to mis-behave, or stop routing altogether. The prototype assumes that the routers communicate via the OSPF protocol. The project is eventually envisioned as detecting both external, and internal, intrusion attempts at the network infrastructure, in a comprehensive, and scalable, fashion, interoperating with other intrusion detection systems.

The intrusion detection in JiNao is operated using, the authors claim, three different paradigms, misuse based detection, anomaly based detection, and protocol based (misuse) detection.

[17] For example, the *read* system call is never traced in the example policies.

2.18.2 System overview

JiNao can be divided into two parts, the local intrusion detection subsystem, and the remote intrusion detection modules. The intention is that a system could be built by connecting the various local instances of JiNao running on various pieces of equipment throughout the system. While the remote element of the system had yet to be developed at the time of publication, it's the authors' intention to have a distributed system, that could be controlled centrally via the SNMP protocol.

The JiNao system is comprised of quite a few different modules, that implement different aspects of the local subsystem of the JiNao intrusion detection system.

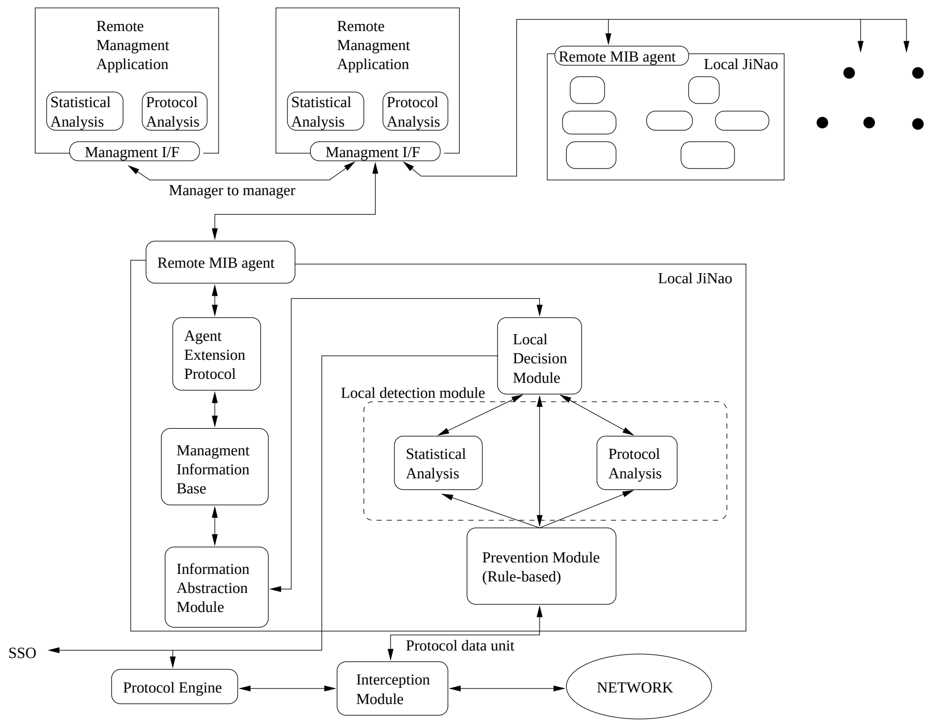

Figure 2.6: Block diagram of the JiNao system (from [15])

Figure 2.6 describes the relationships between the different modules in the local subsystem. The function of the modules are:

Interception/redirection module

This module intercepts protocol information from the network, time-stamps it, and hands it over to the prevention layer. When signaled from the prevention layer, it then releases the packet, for further study by the protocol engine.

The authors discuss the problems associated with encrypted data, something that may become more common in the future with the advent of IPSEC etc. There are three principal layers where the network traffic could be intercepted, IP/IPSEC layer, device driver layer, or in the higher layer protocols. The authors reason that we would like to intercept the traffic at the IP/IPSEC layer, but in the case where the higher level protocol is encrypted, we must intercept it when it has been decrypted. The problem with this is that information that is interesting from an intrusion detection standpoint, may well have been lost at this level, for instance, information about on which hardware network interface the information entered the system, is almost certainly lost by the time the packet has reached the higher levels of the protocol stack.

Prevention module

The prevention module acts as a first, fast acting, filter, against obvious predefined security violations.

The prevention module is further subdivided into two layers, the prevention layer, and the extraction layer. The prevention layers first task is to quickly decide whether to forward the protocol data unit, under study, to the target protocol engine. The reason the decision is made here is that the target protocol engine could observe a considerable delay in seeing the protocol data unit otherwise.

The extraction layer on the other hand has the duty to format any, and all, different network dependent data formats into a JiNao protocol data unit (packet) to simplify further processing. It may be necessary to aggregate information from several network packets, or sources, into a single JiNao packet. This would be the case for instance, when we would like to know the identity of the hardware network interface on which the information entered the local system, as well as the data itself.

Detection module

Consists of two sub modules, the statistical analysis module, and the protocol analysis module. The statistical analysis module determines if the observed behaviour is within the historically established parameters for the observed subject. The protocol analysis module analyses, via the use of state machines, the OSPF, and eventually PNNI, routing protocols, and the SNMP network management protocols, triggering whenever the protocol enters a suspect state.

Statistical analysis module The statistical analysis module draws heavily from the work done by SRI on the NIDES system (see section 2.14 on page 56). JiNao uses NIDES's statistical algorithm with some small modifications. The idea is that the subject under study should exhibit short term behaviour that is consistent with its long term behaviour, in terms of the measured quantities, certain log entries, number of packets sent, etc., etc. In general, when the short term behaviour varies sufficiently from the established long term behaviour, a warning flag is raised.

However, there will always be some variation in the short term behaviour, since this behaviour is comprised of only one activity of the subject, while the long term behaviour is comprised of all the activity performed by the subject. In order to adjust for this effect, JiNao should, according to the authors, account for the amount of deviation that it sees between short, and long term behaviour, and only flag as anomalous, short term behaviour, that is very unlikely long term behaviour, relative to the amount of deviation between these types of activities that it has seen in the past. JiNao's view of what long term behaviour is, i.e. the profile for the subject, is updated once a day.

Protocol analysis module The protocol analysis module, on the other hand, uses information about the specific protocol, and the traffic that is generated by these protocols to ascertain if some suspicious activity is taking place. The protocol analysis module maintains a number of different state machines that codify the known behaviour of the protocols that JiNao knows about. The state-machines, are not pure state machines, in the sense that JiNao state machines have been expanded with a counter feature, since counters are unwieldy to handle in normal finite state machines.

The state machines are quite specific, for example, there is one state machine for each adjacent router, codifying the correct behaviour of each and every one, despite the fact that much of the information would be the same.

The authors had two specific goals, when specifying the state machine mechanism:

1. The protocol analysis module should be reconfigurable at run time, as new intrusions become a concern, and others cease to be.

2. Adding a new state machine should not require recompilation of the protocol analysis module.

The authors identify several other means of optimising the execution of the state machines, in future versions of the software, but have not had time to apply any of them in the prototype.

Local decision module

The local decision module handles two tasks, it coordinates the information from the detection and prevention modules, and it issues commands to update, and/or activate rules in prevention module. The local decision modules also has the responsibility to report suspected intrusions to the site security officer.

Functionally the local decision module interfaces with the detection modules, the local management information base, and the protocol engine. The interface with the local detection modules is required to evaluate the possibility of an intrusion on the basis of local information, from network neighbours etc., the interface with the management information base is to gain access to remote, network wide data, to be able to take part in detection of network wide events, and to gain knowledge of network wide events that could affect the local detection system.

In the last instance, say that a part of the network has suffered a power failure, this could lead to both the detection modules to report that a router in that segment of the network was under attack, or faulty. If the local decision module where to learn about the power failure via the management information base, it could rightly conclude that the router outage was a result of the power failure, and not indicative of either router failure, or the sign of an intrusion.

The fact that the local decision module uses both local, and network wide information to reach a decision, the authors claim, leads to a scalable intrusion detection architecture.

Information abstraction module

Effects the communication between the local JiNao subsystem, and the remote JiNao modules, as well as other network management systems.

The information abstraction module, performs its task by collecting data, and intrusion indications from the local decision module, aggregating it, reducing it, converting it to management information base format, and compressing it.

JiNao management information base

Handles a collection of parameters etc. for the local intrusion detection system, that are of interest to other, remote parts of the system.

The management information base acts as a standard abstraction interface between the JiNao agent and the management applications that are interested in utilising the intrusion detection services provided by JiNao. The management applications primarily use SNMP to communicate with the various management information bases throughout the system. (It is the underlying paradigm of SNMP, to set and read various parameters in the managed systems, that probably prompts the authors to name this component the management information base.) The various services that can be performed remotely by the management information base are:

Rule/FSM configuration The rules and finite state machines used in detection can be configured, deleted, or updated.

Local detection results The result of the processing made by the local decision module can be made remotely available through the management information base.

Detection notifications Remote management applications can register interest in a certain event, and automatically receive notification when that event has taken place. This significantly decrease the time spent searching for a certain type of intrusion, when this intrusion is suspected to take place. For instance, if a remote management application suspected that a router had been compromised it could instruct a neighbour to route traffic through the suspected router, and then instruct another router, down stream from the suspected one, to trap, and immediately report the events that we would expect from a fully operational router. If these events failed to manifest themselves at the trapping router, we could conclude that the interlying, suspected, router, is indeed showing signs of suspect behaviour.

Security control Allows a system administrator to directly control the local intrusion system, instead of the indirect control afforded by the setting of parameters in the management information base proper.

Log access the prevention module logs certain interesting transactions, the management information base provides access to a search engine, that can access the logged data, the authors write that apparently it would be unrealistic to gain access to each individual log record, via the management information base.

The authors note that even though the current release of the SNMP protocol does not afford enough security measures to be realistically used in this context, versions of the protocol due in the near future, at the time of their writing the paper, does seem to provide such features.

2.18.3 Survey conclusions

It is difficult to have an opinion on the system as an intrusion detection system, since the authors make few claims in that area. As a distributed, and scalable system for collecting, and processing network information it probably has merit. The main failing of the work presented is thus, that it does not to any significant degree discuss the nature of the intrusions that it is supposed to detect, or how this detection should be performed.

2.19 EMERALD—Event monitoring enabling responses to anomalous live disturbances

2.19.1 Introduction

EMERALD [47, 48], is intended as a framework for scalable, distributed, interoperable computer and network intrusion detection. The authors begin by describing a situation in where large, organic, computing, and network resources provide critical and costly service to its operator, yet have little in the way of specific security policies, or organisational support for the specification of such policies. These resource typically contain COTS (Commercial-off-the-shelf), as well as non-COTS components, and legacy systems, integrated with current technology. These infrastructures clearly need to be protected, and yet, there is little in the way of widely available robust tools to detect, and track, intruders moving across such infrastructures. EMERALD will also contain components to enable the system to respond actively to the threats posed. The main threat that EMERALD proposes to meet is a penetrator external to the organisation, at least, external on some level. However, the proposed architecture does not preclude the detection of internal attackers.

2.19.2 Organisational model

The authors envision a distributed system (EMERALD) that operate on three different levels in an large enterprise network, made up of administratively more or less separate domains. These domains trust one and other to a greater or smaller extent—two domains could operate in a peer-to-peer relationship, while another could trust virtually no-one else, only allowing out bound connections. The enterprise network would typically be made up of thousands of entities.

EMERALD would operate on three different levels within the domain:

Service analysis level The most local level, where distributed instances of EMERALD would operate locally, on its own target.

Domain-wide level Where the locally distributed instances of EMERALD would operate in concert, sharing information to detect domain wide intrusion attempts. Picture for instance, a network file server monitor, that receives notification of some anomalous DNS event, and with that information could deduce that, indeed the requests to the file server would be highly suspicious in the light of the information received.

Enterprise-wide level Where the results of the domain-wide level analysis would propagate upwards in the organisation, and infrastructure-wide information-warfare style, attacks could be detected.

2.19.3 The EMERALD monitor

The architecture of EMERALD hinges around the local EMERALD monitor. This is the smallest complete instance of EMERALD. The service monitor is dynamically employed through out the system, to monitor points of interest. It communicates with other instances of the monitor, distributed throughout the network, via a push/pull mechanism, whereby a monitor can subscribe to notifications of interest from its peers. It's the view of the authors that this enables EMERALD to communicate efficiently the information that is needed, to where it's needed, without the overhead associated with other plausible means of communication. The interface to the module is well specified, to enable interoperation with other network intrusion detection resources, and it can receive configuration information across the same interface.

The monitor consists architecturally of the following modules:

Resource object The resource object is the heart (though not the brain) of the EMERALD monitor. The resource object handles all target specific issues, and provides interfaces to deal with these issues. Furthermore, it contains configuration parameters for the various fielded analysis engines, both for accessing and processing the local audit event format. The resource object contains the resolver's, and the resolver's decision units configuration as well, including valid response methods to detected violations, and when to invoke them. Last but not least, the resource object maintains the subscription list, for communication with its peers.

Profiler engine The profiler engine, of which there may be several, performs some anomaly based detection on the audit data. The authors have generalised the concepts from NIDES—see section 2.14—to totally separate the calculation, and analysis of the statistics for the audit event stream, from any target specific considerations. The profiler engine also subscribes to information of interest from other instances of EMERALD monitors, via the resolver.

Signature engine The signature engine provides a signature based intrusion detection capability. However, the authors point out that in many respects the signature engine departs from traditional signature analysis engines, in that it is envisioned to operate with a small set of rules, and on a reduced audit data stream, and hence with much less noise to filter out. It's the intention of the authors that this will enable efficient, and effective signature based intrusion detection.

Universal resolver The resolver is the "brain" of the EMERALD monitor, if you will. It handles correlation between the result of the local modules, decide whether an intrusion is taking place, decides whether to invoke a response, communication with the peer monitors, at higher and lower levels, with the resulting authentication etc. The resolver is at heart an expert system that receives the intrusion and suspicion reports from the profiler and signature engines, and based on these reports, and reports from other peer monitors, decides what response to invoke, and how to invoke it. As previously mentioned, it maintains state, important for the configuration of the monitor as a whole, in the resource object. One of the most critical aspects of the operation of the resolver is to handle the interface with the site security officer himself. In the view of the resolver however, he is just another EMERALD monitor.

When the EMERALD module operates at different levels in the intrusion detection framework, the various internal modules operate along different directions, say for instance, a signature analysis module will search for different signatures when it operates in a service analysis module, than when it operates in an enterprise-wide analysis module.

2.19.4 Interoperability

EMERALD specifies well defined interfaces on many levels, both internal to the EMERALD monitor, and external to it, to enable other existing intrusion detection components to interoperate with it. These components could be plugged in as an internal module in the monitor, or partake in the intrusion detection effort via the network interface. In order to resolve these two situations EMERALD defines a two layered, subscription based, message passing, communication system, and interface. The idea is that this will enable a completely implementation neutral path of communication—both internally in the EMERALD monitor, and externally—between monitors, in the distributed EMERALD system.

2.19.5 Putting it all together

The various monitors that make up the EMERALD system is envisioned to operate on the three different levels previously mentioned, by communicating intrusion detection results both between themselves, within the layer, and upward, to notify higher layers of the ongoing intrusion detection activity. This latter builds a hierarchy of EMERALD monitors, that can detect larger, and larger, scale attacks against the enterprise wide network, and to let higher level entities dynamically decide what entities to monitor, and how to perform this monitoring.

2.19.6 Survey conclusions

The papers about EMERALD of course is very general in its approach. The structure of the proposed system is discussed in detail, and appears sound. Furthermore, the proposed architecture is not intended by the authors to exist in a vacuum, the work discusses a proposed organisational structure, and the problems that structure could have, from an intrusion detection standpoint, and how the EMERALD framework would solve these problems. Substantial thought has obviously gone into how to make the system scalable, extendable, and resistant to outside influence.

It is interesting to follow the research from IDES, via NIDES, to EMERALD. In the present author's opinion the fact that the research group is on their third generation of intrusion detection system clearly shows.

2.20 Bro

2.20.1 Introduction

Bro [46] is in the words of its author "A standalone system for detecting network intruders in real-time by passively monitoring a network link over which the intruder's traffic transits." The designers envisioned that their tool would meet the following design goals and requirements (from [46]:

1. It would make high-speed, large volume monitoring of network traffic possible.
2. It would not drop packets, i.e. it would be able to process incoming packets at a sufficient rate, not to have to discard input packets before they had been processed.
3. Bro would provide the site security officer with real-time notification of ongoing, or attempted, attacks.
4. Bro would be careful to separate mechanism from policy, so that it would be simple to specify new security policies, and aid in the realisation of both simplicity and flexibility in the system.
5. The system would be extensible, foremost, it would be easy to add knowledge of new types of attack.
6. It would facilitate the user in avoiding making simple mistakes in the specification of the security policy.

The system would have to operate in an environment in which it itself would come under attack. Since this is a little studied field, how to build survivable security systems, the designers made the simplifying assumption that only one of two systems communicating would be subverted. The author note that this assumption would cost virtually nothing, since if the intruder had both systems under his control, he could then proceed to establish intricate covert channels between them.

It should be pointed out the the security environment in which Bro was designed to operate is one where security concerns aren't the highest priority. Rather the consequences of an intrusion would be mostly limited to securing the compromised machines, and perhaps a tarnished public image.

2.20.2 System overview

The Bro system is divided into three distinct layers, each layer reduces the traffic to be analysed, by making decisions on successively higher levels of abstraction, see figure 2.7. Each layer processes the traffic generated by the immediately lower one, and responds by generating traffic to the next higher level, as well as generating events to control the lower layer.

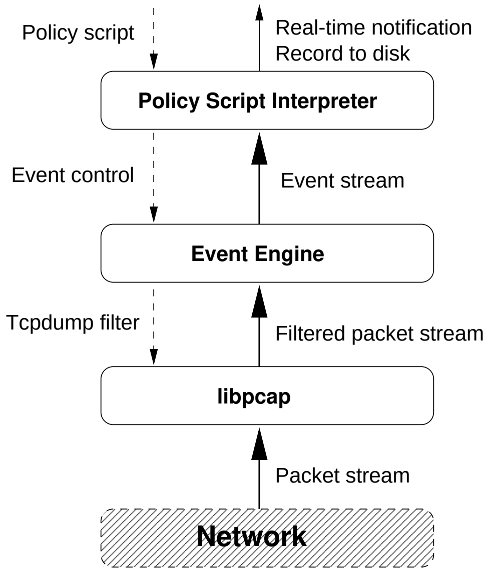

Figure 2.7: Bro: layering and dataflow (from [46])

2.20.3 libpcap

Bro uses the freely available packet capture library *libpcap*, on which the popular *tcpdump* package is built, among others. *Libpcap* has the advantage to isolate the particulars about raw network packet capture from the rest of Bro, and also the capability to download packet filters into the kernel of the operating system that Bro runs on, if the operating system is equipped with such a feature. This enables *libpcap* to discard uninteresting packets early in the processing, and thus consume much less resources in the filtering of packets.

2.20.4 Event engine

The filtered packet stream that results from *libpcap* is handed over to the event engine. The event engine first performs checks to ascertain that the header of the TCP/IP packet is well formed, if it is not, an event to that effect is generated, and the packet is discarded. Otherwise, the event engine looks up the connection state associated with the packet, or records one, if none is already recorded. The packet is then handed over to a special handler corresponding to the connection. The connection handler among other things, indicates whether Bro should log the entire packet, just the header, or nothing at all, to the master *tcpdump* log file that it keeps.

The protocol analysers for TCP, and UDP traffic, performs basic integrity checks to see that the traffic is well formed. This can in turn generate new events that are inserted into the event queue.

2.20.5 Policy script interpreter

More specialised handlers can be formulated in the Bro scripting language. The language is specifically tailored to the processing of TCP/IP traffic, with special data-types, and operators, to handle IP-addresses, lists of such etc. One interesting design feature of the language is that it lacks any operator to facilitate iteration, or recursion. This is to make it likely that the upper bound on any processing of a Bro function would be low, and simple to calculate. This is important, since there is limited amounts of CPU time to be spent in executing the parts of the system implemented in the Bro language.

In any case, after the event engine has finished processing a packet it checks to see if the processing generated any events, if this was the case, the specialised event handler written in the Bro scripting language can be invoked. This handler can generate new events, perform real time notification, record data or log notifications to disk, and record internal state for access by subsequent event handlers, or by the event engine itself. Timers can also be set, coupled with some action to be performed when they expire. This is important to provide the policy writer with the ability to remove internal state from Bro, lest Bro consumes all available memory, with state variables that are no longer deemed interesting.

That Bro is careful to distinguish between the generation of an event, and what to do in response to that event, the author claims, facilitates the separation of mechanism from policy that was one of the designs goals. Furthermore, extensibility is increased. In order to extend the system, the user typically adds new protocol analysers to the event engine, and a handler to respond to the events generated by the protocol analyser.

2.20.6 Implementation issues

At the time the reference was written Bro consisted of some 22,000 lines of C++ code, with another couple of thousand lines of Bro policy scripts. Special handlers had been written for the *finger*, *ftp*, *portmapper*, and *telnet* protocols, with more to be added. The author identified several parts of the system that could be improved, mainly in the construction of a compiler and optimiser for the Bro language, instead of the interpreter of the original system. Furthermore, issues related to check-pointing, managing of timers, especially the addition of timers to the Bro scripting language, and off-line analysis.

2.20.7 Possible attacks on the network monitor

The paper presenting Bro is interesting in that it is the first system that addresses the problem of what kinds of attacks the monitor must be capable of withstanding. Previous work in this field has not specifically addressed the resistance of the intrusion detection mechanism against malicious attacks, other than in theory.

The author classifies network attacks into three categories for the purpose of discussing how well Bro is able to withstand an attack in each category.

Overload attacks

The goal of the attack is to overload the monitor with data, to the point where it fails to keep up with the stream of data with which it has to deal. The attacks aims to first overload the monitor to the point where it will fail to later detect an attack, which it would detect under normal circumstance, had it not been overloaded. Bro averts this attack by the nature of its design; being able to handle a high load, and by logging with regular intervals, how many packets it has missed, and processed, which can alert the site security officer to the fact that the traffic volume itself is suspect.

Crash attacks

The purpose of these attacks is to make the network monitor stop working altogether. This could perhaps be done by finding programming errors in the monitor, and exploiting these, or by making the monitor consume some critical resource, to the point of exhaustion. Bro resists these types of attacks by providing a watchdog timer, that fires if Bro has been found to fail to process the packet it was processing several seconds ago. The watchdog system then stops Bro, logs this fact, and makes post mortem examination of the system possible. It then starts a straight forward *tcpdump* logging of the traffic on the net, with the intent that the network traffic be logged for later analysis—any evidence of malicious network traffic will at least be logged. There exists a window of opportunity between the time Bro stops functioning, and the time that the *tcpdump* logging is started, but at least, all is not lost, with the malfunction of the Bro monitor.

Subterfuge attacks

The attacker attempts to mislead the monitor as to the meaning of the traffic it analyses. The key principle is to find a pattern of traffic that is interpreted differently by the monitor and the receiving end point. These attacks are the most sophisticated, and most difficult to guard against. The author claims that, at each stage of the development of Bro the underlying explicit and implicit assumptions made by the system, and how violating them would enable an attack to go undetected, was carefully examined. The author lists several such attacks, and how Bro defends against them, but of course Bro makes no claim that every such attack has been identified and dealt with.

2.20.8 Conclusion

Bro has run as a part of the security system of the author's site since April 1996, and is available in source code form, an undocumented alpha release, at the time of writing. Release information is available via the world wide web from: http://www-nrg.ee.lbl.gov/bro-info.html.

2.20.9 Survey conclusions

The work presented is strong on the subject of how to make the intrusion detection system resistant to attack against itself. It is the only recent paper that discusses the nature of the attacks that the system could be subjected to, what assumptions have to be made about these attacks, and how the system counteracts them. (A similar paper [49], about the nature of network attacks against intrusion detection, was made available just after Bro was published. That treatment is more thorough than that surveyed here.)

Furthermore the presentation is strong when it presents the rather solid experiences the author have from running the system in a real-world hostile environment. The approach appears to have merit.

Bibliography

[1] D Anderson, T Frivold, and A Valdes. Next-generation intrusion-detection expert system (NIDES). Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park, CA 94025-3493, USA, May 1995.

[2] Debra Anderson, Teresa F. Lunt, Harold Javitz, Ann Tamaru, and Alfonso Valdes. Detecting unusual program behavior using the statistical component of the next-generation intrusion detection system (NIDES). Technical Report SRI-CSL-95-06, Computer Science Laboratory, SRI International, Menlo Park, CA, USA, May 1995.

[3] James P. Anderson. Computer security threat monitoring and surveillance. Technical Report Contract 79F26400, James P. Anderson Co., Box 42, Fort Washington, PA, 19034, USA, February 26, revised April 15 1980.

[4] Stefan Axelsson, Ulf Lindqvist, Ulf Gustafson, and Erland Jonsson. An approach to UNIX security logging. In *Proceedings of the 21st National Information Systems Security Conference*, pages 62–75, Crystal City, Arlington, VA, USA, October 5–8 1998. NIST, National Institute of Standards and Technology/National Computer Security Center.

[5] Jai Balasubramaniyan, Jose Omar Garcia-Fernandez, David Isacoff, E. H. Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. Technical Report Coast TR 98-05, The COAST Project, Dept. of Comp. Sciences, Purdue Univ.,West Lafayette, IN, 47907–1398, USA, 1998.

[6] Mark Crosbie, Bryn Dole, Todd Ellis, Ivan Krsul, and Eugene Spafford. *IDIOT—Users Guide*. The COAST Project, Dept. of Computer Science, Purdue University, West Lafayette, IN, USA, September 4 1996. Technical Report TR-96-050.

[7] Herve Debar, Monique Becker, and Didier Siboni. A neural network component for an intrusion detection system. In *Proceedings of the 1992 IEEE Computer Sociecty Symposium on Research in Security and Privacy*, pages 240–250, Oakland, CA, USA, May 1992. IEEE, IEEE Computer Society Press, Los Alamitos, CA, USA.

[8] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, April 1999.

[9] D. E. Denning and P. G. Neumann. Requirements and model for IDES—A real-time intrusion detection system. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, USA, 1985.

[10] Derothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, Vol. SE-13(No. 2):222–232, February 1987.

[11] Cheri Dowel and Paul Ramstedt. The computer watch data reduction tool. In *Proceedings of the 13th National Computer Security Conference*, pages 99–108, Washington DC, USA, October 1990. NIST, National Institute of Standards and Technology/National Computer Security Center.

[12] Robert Durst, Terrence Champion, Brian Witten, Eric Miller, and Luigi Spagnuolo. Testing and evaluating computer intrusion detection systems. *Communications of the ACM*, 42(7):53–61, July 1999.

[13] M Esmaili, R Safavi, Naini, and J Pieprzyk. Intrusion detection: A survey. In *Proceedings of ICCC'95. (12th International Conference on Computer Communication)*, volume xxxxii+862, pages 409–414. IOS Press, Amsterdam, Netherlands, 1995.

[14] Jeremy Frank. Artificial intelligence and intrusion detection: Current and future directions. Division of Computer Science, University of California at Davis, Davis, CA. 95619, June 9 1994.

[15] Y. Frank Jou, Fengmin Gong, Chandru Sargor, Shyhtsun Felix Wu, and Cleaveland W Rance. Architecture design of a scalable intrusion detection system for the emerging network infrastructure. Technical Report CDRL A005, Dept. of Computer Science, North Carolina State University, Releigh, N.C, USA, April 1997.

[16] Thomas D. Garvey and Teresa F. Lunt. Model-based intrusion detection. In *Proceedings of the 14:th National Computer Security Conference*, pages 372—385, Baltimore, MD, USA, October 1991. NIST, National Institute of Standards and Technology/National Computer Security Center.

[17] Ian Goldberg, David Wagner, Randi Thomans, and Eric Brewer. A secure environment for untrusted helper applications (confining the wily hacker). In *Proceedings of the Sixth USENIX UNIX Security Symposium*, San Jose, California, USA, July 1996. USENIX, USENIX Association.

[18] Jani Habra, Baudouin Le Charlier, Abdelaziz Mounji, and Isabelle Mathieu. ASAX: Software architecture and rule-based language for universal audit trail analysis. In Yves Deswarte et al., editors, *Computer Security – Proceedings of ESORICS 92*, volume 648 of *LNCS*, pages 435–450, Toulouse, France, November 23–25, 1992. Springer-Verlag.

[19] L. Halme and B. Kahn. Building a security monitor with adaptive user work profiles. In *Proceedings of the 11th National Computer Security Conference*, Washington DC, October 1988.

[20] Lawrence R. Halme and Kenneth R. Bauer. AINT misbehaving—A taxonomy of anti-intrusion techniques. In *Proceedings of the 18th National Information Systems Security Conference*, pages 163–172, Baltimore, MD, USA,

October 1995. National Institute of Standards and Technology/National Computer Security Center.

[21] Todd Heberlein, Gihan Dias, Karl Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A network security monitor. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 296–304. IEEE, IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1990.

[22] Judith Hochberg, Kathleen Jackson, Cathy Stallings, J. F. McClary, David DuBois, and Josehpine Ford. NADIR: An automated system for detecting network intrusion and misuse. *Computers & Security*, 12(3):235–248, 1993.

[23] Koral Ilgun. USTAT: A real-time intrusion detection system for UNIX. In *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, pages 16–28, Oakland, California, May 24–26, 1993. IEEE Computer Society Press.

[24] Koral Ilgun, Richard A Kemmerer, and Phillip A Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, March 1995.

[25] Kathleen A Jackson, David H DuBois, and Cathy A Stallings. An expert system application for network intrusion detection. In *Proceedings of the 14th National Computer Security Conference*, pages 215–225, Washington, D.C., October 1–4, 1991. National Institute of Standards and Technology/National Computer Security Center.

[26] Kurt Jensen. *Coloured Petri Nets—Basic Concepts I.* Springer Verlag, 1992.

[27] C. Kahn, P. Porras, S. Staniford-Chen, and B. Tung. A common intrusion detection framework. Submitted to the Journal of Computer Security, available through: http://seclab.cs.ucdavis.edu/cidf/papers/jcs-draft/cidf-paper.ps, July 1998.

[28] T J Killian. Processes as files. In *Proceedings of the USENIX Summer Conference*, pages 203–207, Salt Lake City, Utah, 1984. USENIX Association.

[29] Calvin Ko. *Execution Monitoring of Security-critical Programs in a Distributed System: A Specification-based Approach.* PhD thesis, Department of Computer Science, University of California at Davis, USA, 1996.

[30] Calvin Ko, George Fink, and Karl Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proceedings of the 10th Annual Computer Security Applications Conference*, volume xiii, pages 134–144. IEEE, IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.

[31] Calvin Ko, M. Ruschitzka, and K Levitt. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, volume ix, pages 175–187, Oakland, CA, USA, May 1997. IEEE, IEEE Computer Society Press, Los Alamitos, CA, USA. IEEE Cat. No. 97CB36097.

[32] Sandeep Kumar. *Classification and Detection of Computer Intrusions.* PhD thesis, Purdue University, West Lafayette, Indiana, August 1995.

[33] Sandeep Kumar and Eugene H. Spafford. An application of pattern matching in intrusion detection. Technical Report CSD-TR-94-013, The COAST Project, Dept. of Computer Sciences, Purdue University, West Lafayette, IN, USA, June 17 1994.

[34] Sandeep Kumar and Eugene H. Spafford. A pattern matching model for misuse intrusion detection. In *Proceedings of the 17th National Computer Security Conference*, pages 11–21, Baltimore MD, USA, 1994. NIST, National Institute of Standards and Technology/National Computer Security Center.

[35] Sandeep Kumar and Eugene H. Spafford. A software architechture to support misuse intrusion detection. Technical report, The COAST Project, Dept. of Comp. Sciences, Purdue Univ.,West Lafayette, IN, 47907–1398, USA, March 17 1995.

[36] Ulf Lindqvist. *Observations on the Nature of Computer Security Intrusions.* Licentiate thesis, School of Electrical and Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, 1996.

[37] Ulf Lindqvist and Erland Jonsson. How to systematically classify computer security intrusions. In *Proceedings of the 1997 IEEE Symposium on Security & Privacy*, pages 154–163, Oakland, CA, USA, May 4–7 1997. IEEE, IEEE Computer Society Press, Los Alamitos, CA, USA.

[38] Ulf Lindqvist, Douglas Moran, Phillip A Porras, and Mabry Tyson. Designing IDLE: The intrusion data library enterprise. Abstract presented at RAID '98 (First International Workshop on the Recent Advances in Intrusion Detection), Louvain-la-Neuve, Belgium, September 14–16, 1998.

[39] Richard P. Lippmann, Isaac Graf, S. L. Garfinkel, A. S. Gorton, K. R. Kendall, D. J. McClung, D. J. Weber, S. E. Webster, D. Wyschogrod, and M. A. Zissman. The 1998 DARPA/AFRL off-line intrusion detection evaluation. Presented to The First Intl. Workshop on Recent Advances in Intrusion Detection (RAID-98), Lovain-la-Neuve, Belgium, *No printed proceedings*, September 14–16, 1998.

[40] Teresa F Lunt. Automated audit trail analysis and intrusion detection: A survey. In *Proceedings of the 11th National Computer Security Conference*, pages 65–73, Baltimore, Maryland, October 17–20, 1988. National Institute of Standards and Technology/National Computer Security Center.

[41] Teresa F. Lunt, R. Jagannathan, Rosanna Lee, Sherry Listgarten, David L. Edwards, Peter G. Neumann, Harold S. Javitz, and Al Valdes. IDES: The enhanced prototype, A real-time intrusion detection system. Technical Report SRI Project 4185-010, SRI-CSL-88-12, CSL SRI International, Computer Science Laboratory, SRI Intl. 333 Ravenswood Ave., Menlo Park, CA 94925-3493, USA, October 1988.

[42] Teresa F. Lunt, Ann Tamaru, Fred Gilham, R. Jagannathan, Caveh Jalali, and Peter G. Neuman. A real-time intrusion-detection expert system (IDES). Technical Report Project 6784, CSL, SRI International, Computer Science Laboratory, SRI Intl. 333 Ravenswood Ave., Menlo Park, CA 94925-3493, USA, February 1992.

[43] N. McAuliffe, Wolcott. D, Schaefer. L, Kelem. N, Hubbard. B, and Haley. T. Is your computer being misused? A survey of current intrusion detection system technology. In *Proceedings of the Sixth Annual Computer Security Applications Conference*, volume xx+451, pages 260–272. IEEE, IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1990. Cat.No.90TH0351–7.

[44] Ludovic Mé. Genetic algorithms, An alternative tool for security audit trails analysis. Presented at RAID, 1:st Workshop on Recent Advances in Intrusion Detection, October 1998. Author's address: SUPÉLEC, B.P. 28, 35511 Cesson Sévigné Cedex, France.

[45] Biswanath Mukherjee, L Todd Heberlein, and Karl Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, May/June 1994.

[46] Vern Paxon. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, USA, January 1988. USENIX, USENIX Association. Corrected version, §7 overstated the traffic level on the FDDI ring by a factor of two.

[47] Philip A Porras and Peter G Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, Baltimore, Maryland, USA, October 7–10 1997. National Institute of Standards and Technology/National Computer Security Center.

[48] Philip A Porras and Alfonso Valdes. Live traffic analysis of TCP/IP gateways. In *Proceedings of the 1998 ISOC Symposium on Network and Distributed Systems Security*, San Diego, California, March 11–13 1998.

[49] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks Inc., January 1998. Available via the web:http//... at the time of writing.

[50] Michael M Sebring, Eric Shellhouse, Mary E Hanna, and R Alan Whitehurst. Expert systems in intrusion detection: A case study. In *Proceedings of the 11th National Computer Security Conference*, pages 74–81, Baltimore, Maryland, October 17–20, 1988. National Institute of Standards and Technology/National Computer Security Center.

[51] S.E. Smaha. Haystack: An intrusion detection system. In *Proceedings of the IEEE Fourth Aerospace Computer Security Applications Conference*, Orlando, FL, USA, December 1988. IEEE, IEEE Computer Society Press, Los Alamitos, CA, USA.

[52] Steven R Snapp, Stephen E Smaha, Daniel M Teal, and Tim Grance. The DIDS (distributed intrusion detection system) prototype. In *Proceedings*

of the Summer USENIX Conference, pages 227–233, San Antonio, Texas, June 8–12, 1992. USENIX Association.

[53] S. Staniford Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS—A graph based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[54] H S Vaccaro and G E Liepins. Detection of anomalous computer session activity. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 280–289, Oakland, California, May 1–3, 1989. IEEE Computer Society Press.

[55] Wietse Venema. TCP WRAPPER: Network monitoring, access control and booby traps. In *Proceedings of the 3rd USENIX UNIX Security Symposium*, pages 85–92, Baltimore, Maryland, September 14–16, 1992. USENIX Association.

[56] Greg Vert, Deborah A. Frincke, and Jesse C. McConnell. A visual mathematical model for intrusion detection. In *Proceedings of the 21st National Information Systems Security Conference*, Crystal City, Arlington, VA, USA, October 5–8 1998. NIST, National Institute of Standards and Technology/National Computer Security Center.

[57] Christina Warrender, Stephanie Forrest, and Barak Perlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, Berkeley, California, May 1999.

[58] G. White and V. Pooch. Cooperating security managers: Distributed intrusion detection systems. *Computers & Security*, Vol. 15(No. 5):441–450, 1996. Elsevier Science Ltd.