Alba
A Cognitive Assistant for Network Administration

Francisco J. Martin†*and Enric Plaza‡
† *School of Electrical Engineering and Computer Science*
Oregon State University, Corvallis, 97331 OR, USA
‡IIIA - Artificial Intelligence Research Institute
CSIC - Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia, Spain
fmartin@cs.orst.edu; enric@iiia.csic.es

Abstract

We are developing a first prototype of an agent-aided intrusion detection tool called Alba (ALert BArrage) that assists a network administrator's decision making, reducing the burdensome output produced by current intrusion detection systems (IDSes). This work describes the cognitive machinery that allows Alba to reason about computer security incidents and learn the salient features of an incident so that they can be later employed to recognize similar situations and predict the likely effects of a new attack.

1 Introduction

Network administrators' responsibilities include, among other perimeter defense tasks, prevention, detection and response to computer security intrusions[1]. Nowadays, Intrusion Detection Systems (IDSes) have become common tools (eTrust, eSafe, IntruShield, RealSecure, etc) deployed by network administrators to combat unauthorized use of computer systems. An IDS aims at discovering intrusion attempts and whenever an intrusion (or intrusive behavior) is detected the IDS notifies the network administrator by means of *alerts*. Commonly, alerts take the form of emails, database or log entries, etc and their format and content vary according to the particular IDS (Fig. 3 shows an alert signaled by Snort [16]). There are intrusions that require only a single action (hit-and-run intrusions) to cause a malign effect (i.e. the *Ping of Death* attack[2]). Other intrusions, on the contrary, are sophisticated multi-stage attacks where each intruder's action is a step intended to result in a change of state of the target computer system that prepares it to accept the next intruder's action (i.e. the *Mitnick* attack [15]). Normally, an isolated alert (per se) cannot be correctly classified as malicious or innocuous. For example, imagine an alert corresponding to a `scan` action coming from an unknown `IP`. This alert can correspond to a malefactor performing a reconnaissance or on the contrary it could really correspond to one of our engineers checking at our customer's

*On sabbatical leave from iSOCO - Intelligent Software Components, S. A.

[1]The intentional or unintentional access to sensitive information or unauthorized use of a computer system.

[2]http://www.pp.asu.edu/support/ping-o-death.html

office that one of our web services is up. Thus, more evidence is needed before one can go further and initiate a considered response (e.g. creating a new rule in a firewall).

Moreover, the current generation of IDSes generates an unmanageable number of false positive alerts[3] that in turn increases the difficulties for the proper identification of real and malicious attacks. Network administrators are so overwhelmed that they frequently disable the alert device due to the consistent assumption that nothing is wrong reinforced by the fact that the alert device "cried wolf" too often. There are those who even postulate that traditional IDSes not only have failed to provide an additional layer of security but have also added complexity to the security administration task. Therefore, there is a compelling need for developing a new generation of tools that help to automate security administration tasks such as the interpretation and correct diagnosis of IDSes output. The fact of the matter is that as long as the number of networked organizations proliferates and the number of computer security threats increases this need accentuates.

To make the aforementioned tasks more manageable we envisage a new generation of intrusion detection tools under the heading of *agent-aided intrusion detection*. Some recent works can be seen as the prelude of this tendency [4, 8, 12, 18, 19]. This work describes a first prototype of an agent-aided intrusion detection tool called Alba (ALert BArrage) that mediates between an IDS and the corresponding network administrator. Alba rapidly produces an alert triage (i.e. an approximate priorization for subsequent action) on behalf of its network administrator. Alba employs a case-based strategy to analyze the sequence of alerts provided by a conventional IDS, identifying false positives, deeming alerts due to innocuous attacks, and predicting new alerts corresponding to malicious attacks that are still undergoing. We will see how stored cases, in the form of *alert trees* and learnt through frequent episode algorithms, allow Alba to identify subsequences of alerts in the alert stream that are similar to past situations where a computer security incident occurred. Once identified a past situation Alba uses it to predict the likely intentions of an attacker so that the corresponding alerts can be priorized conveniently and the suitable response can be initiated as soon as possible.

This work describes the cognitive machinery that underpins Alba and proceeds as follows. Next Section describes the task that confronts a human network administrator who must detect a human or artificial intruder rapidly and respond appropriately to minimize damage. Section 3 briefly overviews Alba's cognitive architecture. Then, we concentrate on four key components. Firstly, Sec. 4 describes SOID a simple ontology for intrusion detection that allows Alba to reason at a higher level of abstraction than most IDSes. Secondly, Sec. 5 introduces alert trees as the knowledge structure that supports sequence recognition. Thirdly, Sec. 6 explains how an IDS alert stream is analyzed to recognize past episodes. Fourthly, the frequent episode discovery methods deployed by Alba are described in Sec. 7. Finally, Sec. 8 summarizes some approaches addressing the number of alerts that a human network administrator has to handle and Sec. 9 presents some concluding remarks.

2 Network Administration

Responding to intruders (human, artificial or a combination of both) and keeping networks and applications safe encompasses a collection of tasks that are best explained depending on the time at which they are performed by a network administrator: before, during or after the occurrence of an intrusion. See the temporal model depicted by Figure 1.

[3] Alerts signaled when there is a manifest absence of intrusive behavior.

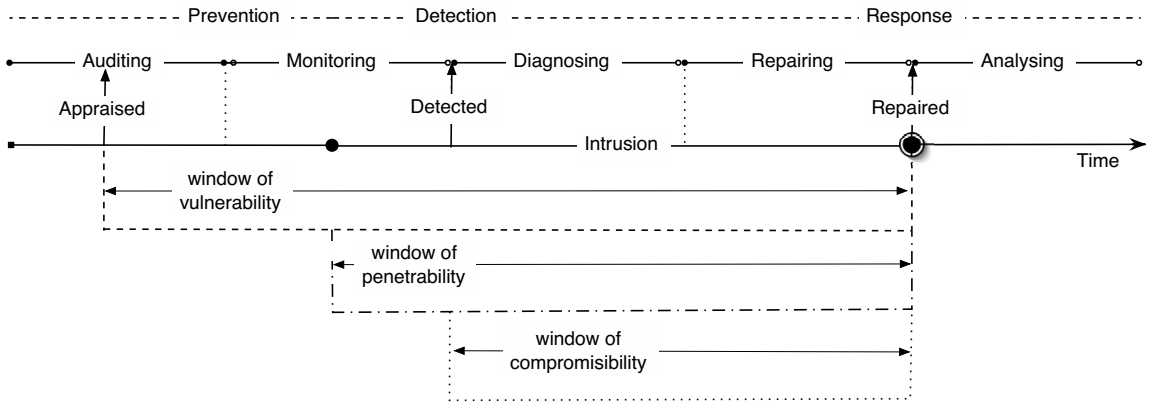

Figure 1: Time axis model of incident prevention, detection, and response tasks.

Prevention Tasks Network administrators according to the security policy prepare beforehand a collection of procedures to effectively handle future intrusions and to enable rapid reaction. They try to minimize the likeliness of future intrusions by constantly auditing the system and eliminating the greatest number of threats. A network administrator proactively performs security audits testing the computer systems for weaknesses —vulnerabilities or exposures. However scan tools (i.e. Nessus, Satan, Oval, etc) used for penetration or vulnerability testing only recognize a limited number of vulnerabilities given the ever increasing frequency of newly detected possibilities for breaking into a computer system or disturbing its normal operation. Thus, network administrators continuously update scan tools with new plug-ins that permit to perceive new vulnerabilities. Once the existence of a vulnerability or exposure is appraised, network administrators assess the convenience of discontinuing the service or application affected until the corresponding patch or intrusion detection signature is available. A tradeoff between risk level and service level is made in every assessment. Network administrators aim at shrinking the *window of vulnerability*, the time gap between when a new vulnerability or exposure is appraised and a preventing solution (patch, new configuration, etc) is provided, as much as possible. A basic strategy to accomplish that objective is based on two conservative tasks: first, minimizing the number of exposures (i.e. disabling unnecessary or optional services configuring firewalls to allow only the use of ports that are necessary for the site to function) and, second, increasing awareness of new vulnerabilities and exposures. Finally, network administrators continuously monitor the system so that pre intrusion behavioral patterns can be understood and used for further reference when an intrusion occurs. *Monitoring* is a preventive and ongoing task that normally conveys to detect an intrusion.

Detection Tasks The sooner an intrusion is detected, the more chances there are for impeding an unauthorized use or misuse of the computer system. Network administrators monitor computer activities at different level of detail: system calls traces, operating system logs, audit trail records, resources usage, network connections, etc. Normally, they constantly try to fusion and correlate real-time reports and alerts stemming from different security devices (i.e. firewalls, intrusion detection systems, etc) to stop suspicious activities before they have a negative impact (i.e. degrading or disrupting operations). Different sources of evidence are valuable given the evolving capabilities of intruders to elude security devices. The degree

of suspicion and malignancy associated to each report or alert still requires continuous human oversight. Thereby, network administrators are continuously overwhelmed with a vast amount of log information and bombarded with countless alerts. Thereabout, network administrators tune security devices to provide an admissible number of false alerts at risk of not detecting real intrusions. The time at which an intrusion is detected directly affects to the damage that an intrusion causes. An objective of network administrators is to reduce the *window of penetrability*, the time span that initiates when a computer system has been broken into and extends until the damage has been completely repaired. The correct diagnosis of an intrusion allows a network administrator to initiate the most convenient response. However, a tradeoff between quality and rapidness is made in every diagnostic. *Diagnosing* is a detection task that conveys to respond an intrusion.

Response and Recovery Tasks As soon as a diagnostic on an intrusion is available network administrators initiate a considered response. This response tries to minimize the impact on the operations (i.e. do not close all ports in a firewall if blocking a unique IP is enough). Network administrators try to narrow the *window of compromisibility* of each intrusion —the time gap that starts when an intrusion has been detected and ends when the proper response has taken effect— employing a collection of ad-hoc operating procedures that indicate how to respond and recover from a type of intrusion. The responses to an attack range from terminating a user job or suspending a session to blocking an IP or disconnecting from the network to disable the compromised service or host. *Repairing* or damage recovery entails to restore the control, resources, and services of the compromised network entities. Network managers use a disaster recovery process that depending on the severity of the intrusion will even require to regenerate the complete system from scratch. Repairing usually entails keeping the level of service while the system is being repaired what hardens automation. Often, it is impossible to keep the level of service (i.e. when a system reboot is required after a DoS). Once the system in completely recovered from an intrusion, network managers collect all possible data to thoroughly analyze the intrusion, traceback what happened, and evaluate the damages. Thus, system logs are continuously backed up. The goal of *post-mortem analysis* is threefold. First, to gather forensic evidence (contemplating different legal requirements) that will support legal investigations and prosecution. Second, to compile experience and provide or improve documentation and procedures that will facilitate the recognition and rapid repelling of similar intrusions in the future, and, third, to validate the current security policy. *Post-mortem analysis* is a response task that conveys to prevent intrusions.

3 Alba Architecture

Based on Brachman's definition of a cognitive system [3], we have defined a cognitive assistant for network administration as a computer system that is capable of:

- *reasoning* in terms of large amounts of knowledge represented at a useful level of description.
- *learning* from past experiences and continuously improving its workings and results over time.
- *explaining* itself in terms that are meaningful for a network administrator.

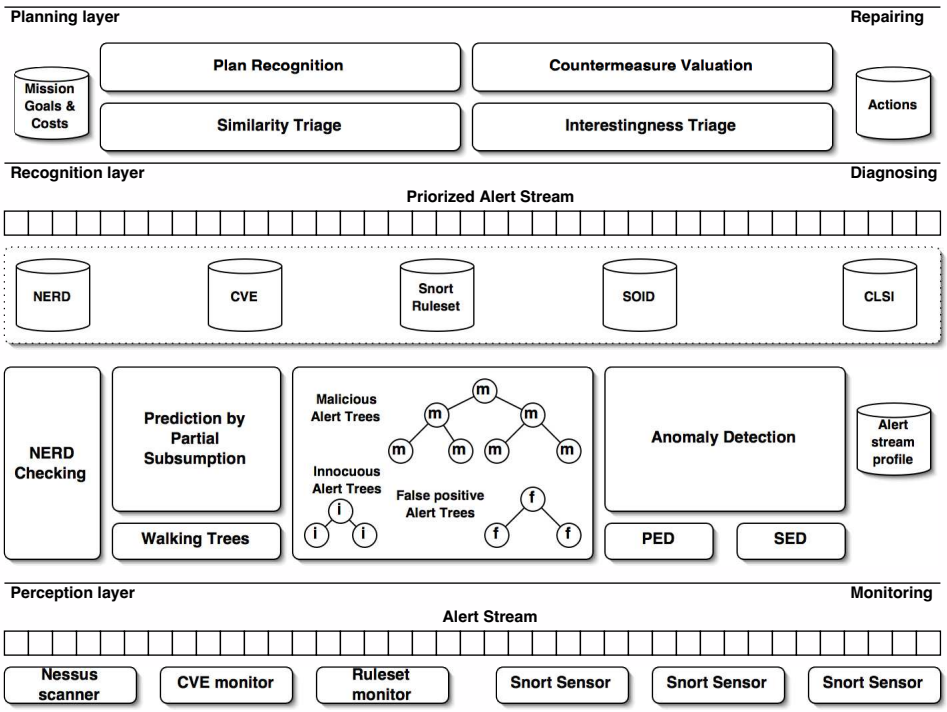

Figure 2: Alba Architecture

- *accepting* direction from a network administrator that tells it what to do.
- *being aware* of its behavior and capabilities (self-aware).
- *responding* in a robust manner and proactively to surprises (survivable).
- *collaborating* with other network administrator's assistants to complete tasks.

In this work, for the sake of brevity, we will only address the first four capabilities. Figure 2 sketches the architecture that provides the primitive resources that allows Alba to reason, learn, accept direction and explain itself meaningfully. Next, we provide a brief description of the complete architecture.

Perception Layer The first layer provides, on the one hand, a collection of sensors strategically placed to continuously monitor and analyze every packet on the protected network, and, on the other hand, a number of scanners and monitors that allows several sources of knowledge (that we will introduce later on) to be constantly updated. This layer allows Alba to perform preventive tasks such as pinpointing security weaknesses for correction.

Recognition Layer The second layer provides Alba with deliberative capabilities. First, a collection of models —expressed on top of the concepts defined by SOID, a simple ontology for intrusion detection— allows Alba to reason about security incidents, vulnerabilities, alerts, and the protected network. Second, alert trees permit Alba to represent sequential patterns of malicious and innocuous activity and store past experiences. Third, Alba uses a collection of methods (Checking NERD and Walking Trees) for continuously overseeing the alert stream looking for an explanation for each isolated alert or group of alerts so that they

can be conveniently priorized. Forth, the *prediction by partial subsumption* method allows Alba to predict new alerts stemming from attacks that are still underway. Fifth, a collection of methods allows Alba to keep an updated profile of the alert stream and constantly learn new alert trees.

Planning Layer The third layer provides Alba with reflective capabilities. Firstly, a model of the network mission and costs allows Alba to make savvy judgements on the priorization of certain malicious alerts as well as to keep the number of false positives under control. Secondly, a plan recognition model uses priorized alerts and predicted alerts to properly anticipate the plans of a malefactor and initiate the corresponding plan of countermeasures using a collection of prespecified actions. Thirdly, Alba accepts direction from a network administrator who can provide it with different criteria to estimate the similarity of two sequence of alerts and establishes different measures of interest to properly prune the discovery of new alert trees.

Alba has been coded using Noos, an object-centered knowledge representation language useful for developing knowledge systems that integrate problem solving and learning [1]. Noos also provides agent-programming constructs . The three basic concepts that underpin the Noos knowledge representation language are: *sorts*, *feature terms*, and *subsumption*. A sort is defined as a symbol that denotes a set of the individuals of a domain. Sorts form a collection of partially ordered symbols. Noos is formalized using *feature terms*. Feature terms are a generalization of first order terms and lambda terms. Feature terms constitute the Noos basic data structure and can be seen as extendable records organized in a *subsumption* hierarchy [1]. Feature terms are represented graphically by means of labeled directed graphs (see Fig. 4 and Fig. 5). In Noos subsumption is defined as an informational ordering among feature terms. A feature term Ψ is subsumed by another feature term Ψ' when all information provided by Ψ' is also provided by Ψ. Subsumption is crucial in our approach since it is at the core of the algorithms that Alba uses to compare sequences of alerts.

In the following, we describe in further detail four different components of the recognition layer aimed at performing an effective alert triage.

4 SOID

SOID aims at providing a domain-specific representation language for alert triage in intrusion detection. At a quick glance, in order to automate the alert triage task we have identified four key sources of knowledge to be conceptualized (networks, incidents, vulnerabilities, and alerts) and built a separate ontology for each of them using the knowledge representation language Noos [1]. Finally, we have merged these partial ontologies in a more global ontology that we have called SOID —a Simple Ontology for Intrusion Detection.

Networks A network is the computer system to be protected. We have defined a set of concepts and relationships to model a network based on the Network Entity Relationship Database (NERD) proposed in [7]. Properly modelling the network allows the importance of each alert to be correctly assessed. For instance, determining whether a given alert corresponds to an innocuous attack or not. That is the objective of the NERD checking method. Network models based on SOID can easily be coded into Noos and automatically updated translating the reports provided by Nessus (an open source network scanner).

Incidents An incident is a unauthorized use or abuse of the protected system. We have followed CLCSI [10] that defines an incident taxonomy based on three key concepts: *events*, *attacks* and *incidents*. An event is an *action* directed at a *target* which is intended to result in a change of state of the *target*. An attack is defined as a sequence of actions directed at a target taken by an *attacker* making use of some tool exploiting a computer or network vulnerability. Finally, an incident is defined as a set of attacks carried out by one or more attackers with one or more goals.

Vulnerabilities A vulnerability is a flaw in a target that could allow an unauthorized result. Knowing the vulnerabilities in our network is the main source of knowledge to automatically decide if a given alert corresponds to an innocuous attack or not. We have incorporated common vulnerabilities and exposures (CVE) dictionary provided by the MITRE corporation into our ontology. A monitor advertises Alba of new published vulnerabilities. Alba contrasts new vulnerabilities against the network model (NERD) and pintpoints new security weaknesses for correction.

Alerts We have conceptualized alerts according to the Snort ruleset. Snort is a network IDS where alerts are triggered by a collection of rules [16]. Each Snort rule is composed of a Snort identification number (SID), a message that is included in the alert when the rule is triggered, an attack signature, and references to sources of information about the attack. Each alert is provided with an identifier, time and date, sensor identifier, triggered signature, IP and TCP headers and payload. In Fig. 4 an alert corresponding to an attempt of propagation of the CodeRed worm is shown.

5 Alert Trees

An alert tree is a knowledge structure to describe past security incidents. An alert tree represents the serial structure of a group of alerts that occurred together within a specified window of time at the end of which a particular situation took place (i.e. an concrete attack). The root node represents the goal and target of an attack and the leaf nodes the alerts corresponding to the sequence of actions needed to achieve that goal [17]. An alert tree establishes a partial order among the leaf nodes. An alert tree has two types of intermediate nodes: *a-nodes* (parallel nodes) and *s-nodes* (serial nodes). An a-node indicates that to occur all its subnodes have to occur before (indepently of the order) and therefore establishes a partial order among them

```
   #(1 - 12064) [2002-11-29 18:47:22]  WEB-IIS CodeRed v2 root.exe access
   IPv4: 80.34.49.201 -> 172.26.0.4
hlen=5 TOS=0 dlen=112 ID=16914 flags=0 offset=0 TTL=118 chksum=37996
   TCP:  port=3421 -> dport: 80  flags=***AP*** seq=1955827854
ack=1657159142 off=5 res=0 win=17520 urp=0 chksum=44219
   Payload:  length = 72

   000 : 47 45 54 20 2F 73 63 72 69 70 74 73 2F 72 6F 6F   GET /scripts/roo
   010 : 74 2E 65 78 65 3F 2F 63 2B 64 69 72 20 48 54 54   t.exe?/c+dir HTT
   020 : 50 2F 31 2E 30 0D 0A 48 6F 73 74 3A 20 77 77 77   P/1.0..Host: www
   030 : 0D 0A 43 6F 6E 6E 6E 65 63 74 69 6F 6E 3A 20 63   ..Connnection: c
   040 : 6C 6F 73 65 0D 0A 0D 0A                           lose....
```

Figure 3: CodeRed Worm propagation attempt.

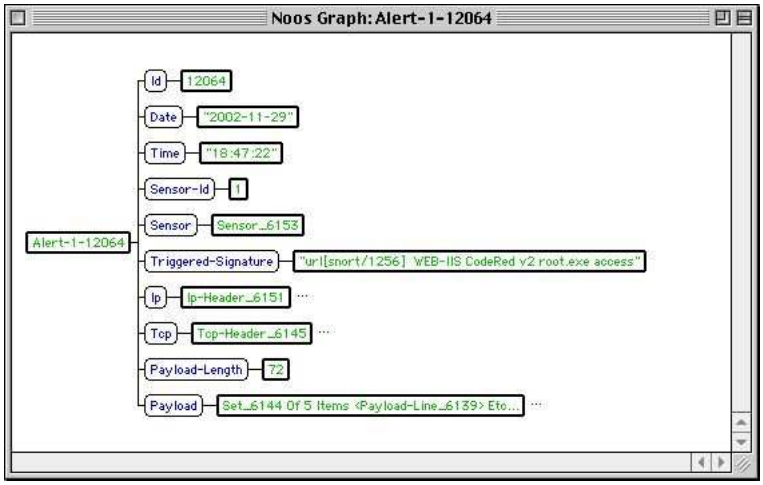

Figure 4: CodeRed alert in Noos

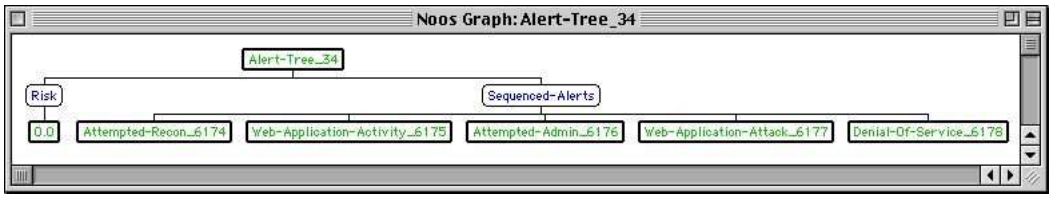

Figure 5: Innocuous Alert Tree.

whereas a s-node indicates that its subnodes are ordered following a total (lexicographic) order. Therefore, an alert tree induces a partial order among all the leaf nodes. We call *episode* to each one of the induced sequences of leaf nodes. The nodes of an alert tree also have other features such as *risk* to indicate the possibilities of suffering the attack at the root node once all its subnodes have occurred and *cost* that indicates the impact of such attack on a give mission. Fig. 5 shows an alert-tree to represent a multi-stage attack as a sequence of Snort alert classes with a null risk factor that indicates that the attack is innocuous.

6 PPS

Alba constantly searches the set of alert trees that best explain the current alert stream. Alba uses a walking tree method —inspired by a family of heuristics to align biologically reasonable strings [5]— to efficiently and incrementally computing the similarity between an alert tree and the alert stream (see Fig. 6). Computing the similarity between two sequences of alerts can be interpreted as the search of evidence that some alerts in the alert stream and a past episode are derived from a common attack pattern perhaps altered using new and undetectable vulnerabilities. The Prediction by Partial Subsumption (PPS) method at any time considers a window ω composed of the last N alerts in the alert stream and looks for partial occurrences of episodes. We define the partial occurrence of an episode and its confidence threshold as follows:

Partial Occurrence A window $\omega = \langle a_{t_{n-N+1}}, \ldots, a_{t_{n-1}} a_{t_n} \rangle$ of length N on an alert stream $\vec{A} = \langle a_{t_1}, .., a_{t_n} \rangle$ is a partial occurrence of episode ε iff the optimal alignment of ε to ω has score at least θ.

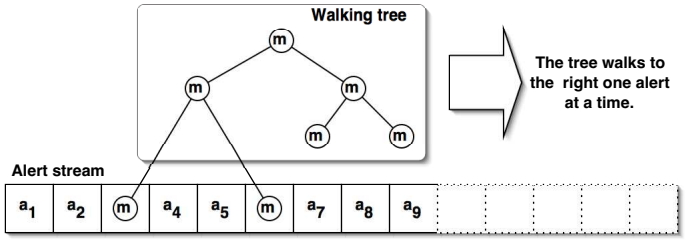

Figure 6: Walking Tree.

Confidence Threshold A window $\omega = \langle a_{t_{n-N+1}}, \ldots, a_{t_{n-1}}, a_{t_n} \rangle$ of length N on an alert stream $\vec{A} = \langle a_{t_1}, .., a_{t_n} \rangle$ such that ω is a partial occurrence of episode ε then we can predict that the whole episode will occur with confidence θ' given by the odds ratio $\prod_i \frac{p_{\omega[i]\varepsilon[i]}}{q_{\omega[i]}q_{\varepsilon[i]}}$. Where $\prod_i q_{\omega[i]} \prod_i q_{\varepsilon[i]}$ represents the probability that both sequences of alerts ω and ε were unrelated or appeared randomly. And $\prod_i p_{\omega[i]\varepsilon[i]}$ can be thought as the probability that the alerts on the window ω of the alert stream have been caused by an incident similar to the episode ε [6].

For each episode ε induced by an alert tree, PPS searches the optimal alignment between a suffix of the window ω and a prefix of the episode ε. When both the decision threshold θ and the confidence threshold θ' are reached or exceeded then Alba is capable of foreseeing that the corresponding suffix of such episode will occur (at least with a level of confidence θ') and therefore preventing the incident $\mathcal{I}'_\varepsilon$. On the other hand, the alerts on which Alba bases this prediction are priorized for further investigation. Notice that once a window ω and a episode ε are deemed to be similar Alba will apply other methods that will look for more evidence before it can conclude that an attack is undergoing (i.e. verifying that the source IP of the different alerts involved are the same). Other issue that deserves attention is that a predictive model is only as good as the trust its user (a network administrator in our case) puts in it. Thus it is fundamental to minimize the number of false positives ("do not cry wolf too many times") while keeping the true positive fraction as high as possible. Alba computes an optimal decision threshold θ with such criteria using ROC analysis. Figure 7 shows the ROC curve generated in a set of preliminary experiments where we employed an alert stream composed of 84168 alerts coming from 8848 different IPs that was generated after four months of real surveillance in a networked organization using 3 Snort sensors, 18 episodes corresponding to well-known attack patterns, an error type weighting of 1:500 (i.e. a cost of 1 for each false positive and a cost of 500 for each false negative), and 12 variants of 3 different multi-stage attacks. The optimal decision threshold corresponded to the iso-performance line with slope equal to 2.2 (see Fig. 7).

7 Frequent Episode Discovery

Alert trees can be provided by a network administrator or learn by means of anomaly detection algorithms[4]. There exists a number of algorithms for frequent episode discovery in sequence of events. Winepi [11], Minepi [11], Seq-Ready&Go [2], etc that are valid for our purpose. These three-phase algorithms exploit the notion that if a given episode is frequent

[4]An anomaly detection algorithm is a generalized inductive learning based on past cases that recognizes unusual features of data, i.e. recurrent combinations that occur with greater or lesser frequency than originally might be expected [9].

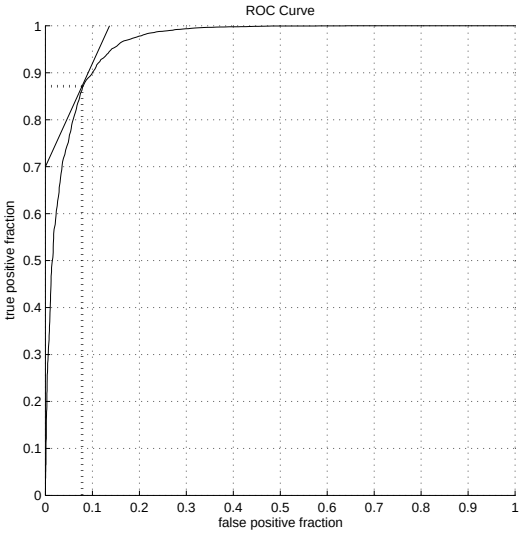

Figure 7: ROC curve and optimal decision threshold.

then all its subepisodes are also frequent. In the first phase, these algorithms generate candidates while in the second phase evaluate the support (the number of times that the candidate occurs) and prune those candidates that are under a preselected *support*. The first two phases are looped until no new candidates can be generated. Then, the third phase generates association rules that are over a given *confidence* threshold. A drawback of these algorithms is the unmanageable number of rules that can generate. For instance, using Winepi to mine an alert stream composed of 31482 alerts corresponding to 3 months of activity of a Snort sensor. After four hours of computation, we found 171566 rules with a minimal confidence of 15% and a minimal frequency of 10% using a window of 30 seconds. Thus, Alba uses a new method based on the Winepi [11] algorithm that is able to support noise data [20] and generalization over a hierarchical taxonomy of sorts [2]. A novelty of this method is that it is also able to use different interestingness measures provided by the network administrator to prune new candidates. We are now experimenting with different interestingness measures such as *general impressions* or *surprisingness* [9]. We have developed two variants one for parallel episode discovery (PED) and other for serial episode discovery (SED) —a-nodes and s-nodes respectively. On the other hand, Alba continually revises models of correct behavior (alert stream profiles i.e. number of alert per minute, frequency of alert sorts, α-rarity, IP sources, etc) and evaluates statical deviation from past history in order to advert meaningful differences.

8 Related Work

The correct interpretation of an IDS alert stream is an active area of research in the intrusion detection community [7, 13]. As our approach M2D2 [13] reuses models proposed by others and integrates multiple interesting concepts into a unified framework. The most significative difference between both approaches is that M2D2 uses the B formal method to model the different sources of information for the alert management task whereas we are using a description logic like language like [7]. Crosbie and Spafford were the first to propose autonomous agents in the context of intrusion detection. Their initial proposal evolved to become AAFID [18]. Other works such as Cooperating Security Managers have proposed a multi-agent sys-

tem to handle intrusions instead of only detecting them [19]. However, in these works agents lack reasoning capabilities and are used for mere monitoring. More sophisticated agents with richer functionality were provided by [8] where an ontology centered on computer attacks was introduced. That ontology provides a hierarchy of notions specifying a set of harmful actions in different levels of granularity —from high level intentions to low level actions. It has been argued in [4] that a collection of heterogenous software agents can reduce risks during the window of vulnerability introduced between when an intrusion is detected and the security manager can take an active role in the defense of the computer system. Some works have also proposed to deal with intrusion detection at higher level of abstraction. The benefit of dealing with intrusions at higher level of abstraction is twofold: it allows irrelevant details to be removed and the differences between heterogenous systems to be hidden [14].

9 Conclusions

Ideally, the ultimate goal of secure network administration is to make the three windows (vulnerability, penetrability and compromisibility) of each possible intrusion converge into a single point in time. Pursuing that objective is a manpower intensive process. Moreover, the astounding growth of networks and the speed at which Internet software has been developed and released inevitably has led to an exponential growth in the number of current vulnerabilities and exposures and therefore in the complexity of network administration. Only the smart automation of network administration tasks will alleviate the ever increasing manpower needed for secure network administration. This work provides a brief overview of a cognitive assistant, Alba, that reduces the burdensome output produced by current IDSes and contributes to minimize the number of false positives due to innocuous attacks and to increase the predictive power for malicious multi-stage attacks. This work forms part of a more ambitious effort where we are involved in developing CBR techniques for the assessment of dynamic processes in imprecise and adversarial environments.

Acknowledgments

Part of this work has been performed in the context of the MCYT-FEDER project SAMAP (TIC2002-04146-C05-01) and the SWWS EU-funded project under contract number IST-2001-37134.

References

[1] J. L. Arcos and E. Plaza. Inference and reflection in the object-centered representation language Noos. *Journal of Future Generation Computer Systems*, 12:173–188, 1996.

[2] J. Baixeries, G. Casas, and J. L. Balcázar. Frequent sets, sequences, and taxonomies: new, efficient algorithmic proposals. Technical Report LSI-00-78-R, UPC, 2000.

[3] R. Brachman. Developing cognitive systems. a convergence of thinking. Technical report, Defense Advanced Research Projects Agency, 2002.

[4] C. A. Carver, J. M. Hill, J. R. Surdu, and U. W. Pooch. A methodology for using intelligent agents to provide automated intrusion response. In *Proc. of the IEEE WIAS*, pages 110–116, 2000.

[5] P. Cull and T. Hsu. Improved parallel and sequential walking tree methods for biological string alignments. In *Proc. of the ACM/IEEE conference on Supercomputing*, 1999.

[6] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998.

[7] R. P. Goldman, W. Heimerdinger, S. A. Harp, C. W. Geib, V. Thomas, and R. L. Carter. Information modeling for intrusion report aggregation. In *DICEX*. IEEE Computer Society, 2001.

[8] V. I. Gorodetski, L. J. Popyack, I. V. Kotenko, and V. A. Skormin. Ontology-based multi-agent model of information security system. In *7th RSFDGrC*, number 1711 in LNAI, pages 528–532. Springer, 1999.

[9] R. J. Hilderman and H. J. Hamilton. *Knowldege Discovery and Measures of Interest*. Kluwer Academic Publishers, 2001.

[10] J. Howard and T. Longstaff. A common language for computer security incidents. Technical Report SAND98-8667, SNL, 1998.

[11] H. Mannila, H. Toivonen, and A. I. Verkano. Discovery of frequent episodes in event sequences. Technical report, University of Helsinki, 1997.

[12] F. J. Martin and E. Plaza. SOID: an ontology for agent-aided intrusion detection. In *7th International Conference on Knowledge-based Intelligent Information & Engineering Systems*, 2003.

[13] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2d2: A formal data model for ids alert correlation. In *Proc. of the RAID 2002*, 2002.

[14] P. Ning, S. Jajodia, and X. Wang. Abstraction-based intrusion detection in distributed environments. *ACM Transactions on Information and System Security*, 4(4):407–452, 2001.

[15] S. Northcutt. *Network Intrusion Detection. An Analyst's Handbook*. New Riders, 1999.

[16] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of LISA '99: 13th Systems Administration Conference Seattle, Washington, USA*, November 1999.

[17] B. Schneier. Modeling security threats. *Dr. Dobb's Journal*, 1999.

[18] E. H. Spafford and D. Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, 34:547–570, 2000.

[19] M. White, E. Fisch, and U. Pooch. Cooperating security managers: A peer-based intrusion detection system. *IEEE Network*, 10:20–23, 1996.

[20] Q. Zheng, K. Xu, W. Lv, and S. Ma. Intelligent search of correlated alarms from database containing noise data, 2002.