Using Context-Based Correlation in Network Operations and Management

Louis Perrochon, Stanford University

perrochon@pavg.stanford.edu

Abstract

Network operation consists to a large degree of reaction to activities happening in the network. Better knowledge of the network at any time allows more appropriate reactions. On the example of intrusion detection, we show how context-based correlation of such activities can provide a more detailed view of the network in shorter time. We first present how we model *context* and then describe the architecture of the Stanford University CEP context-based correlator. Correlation is specified as event patterns in a declarative language that allows us to specify what needs to be detected, instead of specifying how it should be detected. CEP introduces the concept of causal context to intrusion detection. The correlator is able to process events on-line, as they are generated and it can be reconfigured at dynamically. We then show how it increases detection rate, reduce false alarms, and detect large-scale attack pattern at an early stage.

Keywords: Network operations and management, correlation, root cause analysis, intrusion detection.

1 Introduction

As organizations build ever-larger computer networks, security has become an increasingly important issue. The growth of the Internet has resulted in an increase in the size of individual networks as well as an increase in the volume of traffic flowing through the network. As system log files have grown to contain gigabits of data, it has become almost impossible to manually trace through log files to detect attempts at intrusion activity or other security violations. This has given rise to a need for software tools that automate network monitoring for security intrusions [1, 2][3-7].

Typically, these tools gain knowledge of the target system based on individual *events* that denote the target system's activities. When multiple events are considered at the same time we call the process *correlation*. Correlation allows us to deduce knowledge beyond what is in the individual events. *Context-based correlation* considers the context of events, including their temporal and causal relations to each other. The Stanford University CEP correlator models relations between activities as partially ordered sets of events. Partial orders are a natural model of activity in asynchronous distributed systems that may not have a central clock or a single thread of causality. In this article we describe the CEP correlator and its application to network management in general and intrusion detection in particular.

Some of the results we achieved by using context-based correlation are:

- *False Alarm Reduction:* Alarms that are not subsequently confirmed by another alarm may be false or irrelevant. As an example, denial of service attacks may be used to create a diversion by flooding administrators with low priority alarms to mask penetration attacks. Compared to the ongoing penetration, the denial of service may be irrelevant.
- *Increased Detection Rate:* Additional intrusions may be detected by deduction from observed attacks. As an example, an observed action to gain root privilege indicates that the host has been compromised before. If no penetration has been detected so far an undetected penetration is likely.
- *Early Large-scale Attack Detection:* A typical attack pattern goes from network-probing to

creation of diversion by denial of service attacks, to penetration of a critical host, and then to gaining root-access. Correlating the probing alerts with penetration alerts may detect ongoing attack patterns in early stages.

CEP has applications beyond intrusion detection. More generally, CEP provides on on-line overview of the state of the network. Patterns of events that may lead to a failure (e.g. a DNS server having slower and slower response time) are detected as they occur. CEP provides immediate notification of such conditions. Because the context of events is maintained, user driven drill-down from a notification message back to the root cause is possible. Automatic prioritizing of alerts and quick root cause analysis leads to reduced response time, higher up time and allows system managers to quickly respond to critical situations. CEP also allows automatic response based on predefined rules.

Section 2 describes complex correlation and our correlation engine. Section 3 and 4 describe recent experiment in intrusion detection and network management in general. Sections 5 and 6 discuss related work, and future directions.

2 The Stanford University Event Correlator

This section starts out with our model of context-based event correlation and describes our approach to allow flexible configuration of the correlation. It then describes the architecture of our correlation engine.

2.1. Context-based Correlation

In order to understand complex systems and efficiently deal with complex patterns simple filters solely based on event content are not enough. CEP correlation algorithms are based on earlier work on CEP theory (e.g. [8-11]). The following aspects of context are part of CEP technology and are implemented in the Stanford University CEP engine:

1. *Event Context:* Events are stored as complex objects together with their relationships to each other. CEP supports event relationships beyond *time*, e.g. *causality*: one event *causes* another. In today's networked environments, events come from multiple independent sources and not all events are ordered in respect to each other. If the original implicit partial order of events is reduced to a total order then information is lost and non-determinism is introduced [8]. This is typically the case when activity is logged to a log file. Thus we use partial orders to maintain the original relations.

2. *Abstraction Hierarchies:* These are useful for understanding a system as they visualize levels of abstractions often used by humans when thinking about complex systems. Hierarchies are implemented by sequence of aggregation steps (called *maps*) that create more and more abstract events, all deduced from a basic source of events. The same mechanism can be used to provide different views of the same system to different users. The relation of events is maintained between different views.

3. *Architecture Support:* Complex systems have an architecture of components and connections between these components. This architecture may change while the target system is executing [12]. Often the architectural context of events is relevant: A particular intrusion may only be detected because an attacker replaced a specific secure connection with an insecure connection. Or a previously undetected penetration can be inferred because the only connection to a newly compromised component comes from a component that we thought was safe.

2.2. Dynamic Configuration of Correlation

CEP queries are flexible and configurable at run time so the user can adjust them at any time. This includes starting a new query against an ongoing event stream, that either considers only new events, only old events, or both. A formal pattern language [13] supports the construction of *filters* and *maps* [14]. Filters and maps consist of rules of the form `if` *condition* `then` *statement*. As a simple example, consider the following rule:

```
PenetrationAttempt(targetIP is ?h) ~ RootActivity(hostIP is ?h)
  => Escalation(target is ?h)
```

The pattern specifies that two events `PenetrationAttempt` and `RootActivity` together trigger (=>) the creation of an `Escalation` event. The value of the event parameters `targetIP` and `hostIP` is bound to the variable `?host`, and has to be the same in both triggering events. This value is also assigned to the `target` parameter of the new `Escalation` event.

Having a high level event pattern language allows for rapid configuration of filters and maps. Patterns represent a declarative way of specifying what needs to be detected, instead of specifying how it should be detected. All filters and maps can be added, modified and removed at run time. Easy configuration is especially important for intrusion detection as policies may be site specific. Any intrusion detection system must be tuned for its environment.

2.3. Event Processing Networks

CEP is component-based to allow a variety of different components to process a broad spectrum of events. A component architecture also allows reuse of components across different environments. We build *Event-Processing Networks* (EPNs) consisting of any number of *Event-Processing Agents* (EPAs), namely event *suppliers*, event *processors,* and event *consumers*. EPNs are networks of communicating agents, where the communication topology of the network can vary at run time (i.e., is dynamic). Figure 1 shows an overview over the three categories, with broad arrows indicating the (logical) flow of events from suppliers through processors to consumers.

Event sources in our applications are typically middleware *sniffers*. The target system middleware can be pure TCP/IP, CIDF (see experiment below), or any middleware layer, etc.

Typical examples for event processors are *filters* and *maps*. Filters pass on only a subset of their input, maps aggregate multiple events in the input to output events thus generating events on a higher level of abstraction. Any third party event processor can be inserted into an EPN allowing for the integration with other approaches. Typical consumers are viewers like a graphical viewer for partially ordered sets of events, or a tabular viewer of event frequency. Another important consumer in the CIDF context is a responder (figure 1). Responders generate activities in the target system based on events.

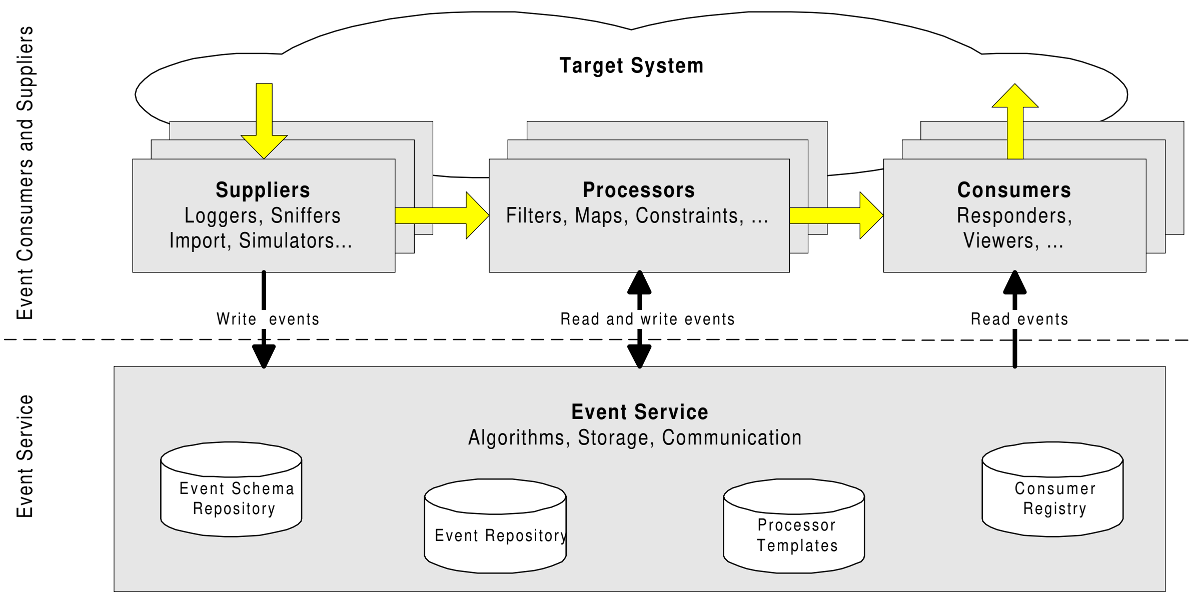

Figure 1. Complex Event Processing (CEP)

EPNs are *dynamic* in that all EPAs can be added and removed at run time. Newly added agents can ignore all previous events and just start with the current event at the time they are added, or they can try to catch up all events from the beginning. As EPNs are distributed, EPAs can reside on machines distributed across a network.

2.4. Event Processing Infrastructure

The EPN is supported by an event service. The event service provides repositories for events and event schemas, templates for processors, and a consumer registry. Event Schemas define the format of events forwarded by the event service. The event repository stores events. Processor templates are configurable templates that can be activated to filter, aggregate, or otherwise process events. The consumer registry keeps track of which consumer is interested in which events.

Data needs to be stored persistently because agents may want to access past events, even long after they have happened. Also the number of objects currently under consideration may easily exceed the size of the available main memory, thus CEP requires some way of storing objects temporarily to disk. CEP includes an event repository that keeps track of all the objects. New objects are written into the data store from where agents and viewers read them. A communication service notifies other EPAs when new objects are added.

Events are processed on-line and are displayed to users as soon as they are created, limited only by the speed of the underlying infrastructure. Network sniffers produce relevant events at rates of thousands/minute or more. CEP supports these rates through efficient pattern matching algorithms, distribution and concurrency, and distributed query optimization. Processed events are displayed in viewers shortly after the underlying events have been created by the event source.

One of the future areas of CEP development is event mining. In the context of intrusion detection, event mining will search stream of GIDOs for previously unknown attack patterns to improve the added-value of correlation [15].

3 The Intrusion Detection Experiment

Our experiment brings us to the world of information warfare. We are part of a project that tries to detect large-scale attacks to the national information infrastructure.

An intrusion correlator typically is one component in a larger intrusion detection system. The Common Intrusion Detection Framework (CIDF) [16] defines an integration platform for software applications defending in information warfare. Among the intrusion detectors that currently support CIDF are Network Associate Inc's Cybercop Scanner [1], ISS RealSecure [2], or SRI's Emerald [7]. A CIDF system consists of any number of distributed, heterogeneous *detectors*, *correlators*, and *responders*. They all communicate using the Common Intrusion Specification Language (CISL). CISL defines General Intrusion Detection Objects (GIDOs) that contain information about potential attacks and counteractions. Some of the ideas involved in CIDF have encouraged the creation of an Internet Engineering Task Force (IETF) working group, named the Intrusion Detection Working Group [17].

The CEP intrusion correlator receives attack notification from detectors using CIDF and performs further correlation to create added value by sharing the results of multiple components. Figure 2 shows how CEP is integrated into the Common Intrusion Detection Framework (CIDF). The framework includes middleware that provides secure transport of General Intrusion Detection Objects (GIDOs). These GIDOs are formulated in the Common Intrusion Specification Language (CISL). The middleware and the language provide a mechanism for components to communicate (shown as thick arrows). The main components in the framework are detectors and correlators. Detectors monitor a part of the information infrastructure (network in figure 2), such as a (sub)network, a host, or the middleware. Correlators investigate the output for several purposes like reduction of false alarm or detection of large-scale attack pattern that involve simultaneous attacks to multiple parts of the target system. In the following *attack* is used for a single action detected by a detector and represented as a GIDO, *attack pattern* is used for a group of correlated attacks.

CEP/CIDF integration starts with a `Log` agent that listens to CIDF traffic and intercepts relevant

GIDOs. Each GIDOs is then translated into an event in the CEP internal format (`Level 0` events). In figure 2, these level 0 events are stored to disk for later analysis. They are also displayed to the user in a *viewer*, in this case the `CIDF Traffic Viewer`. The level 0 events are also the basis for a first *event-processing agent,* called `Extract`. Because CEP has been optimized for strongly structured events, and CIDF events have a very flexible structure, we extract relevant information from the level 0 events and create another layer of events with a rigid structure (`level 1`). These events are then used as input for several other event-processing agents that may be built on top of each other as shown in figure 2 (`Correlate, Prioritize`).

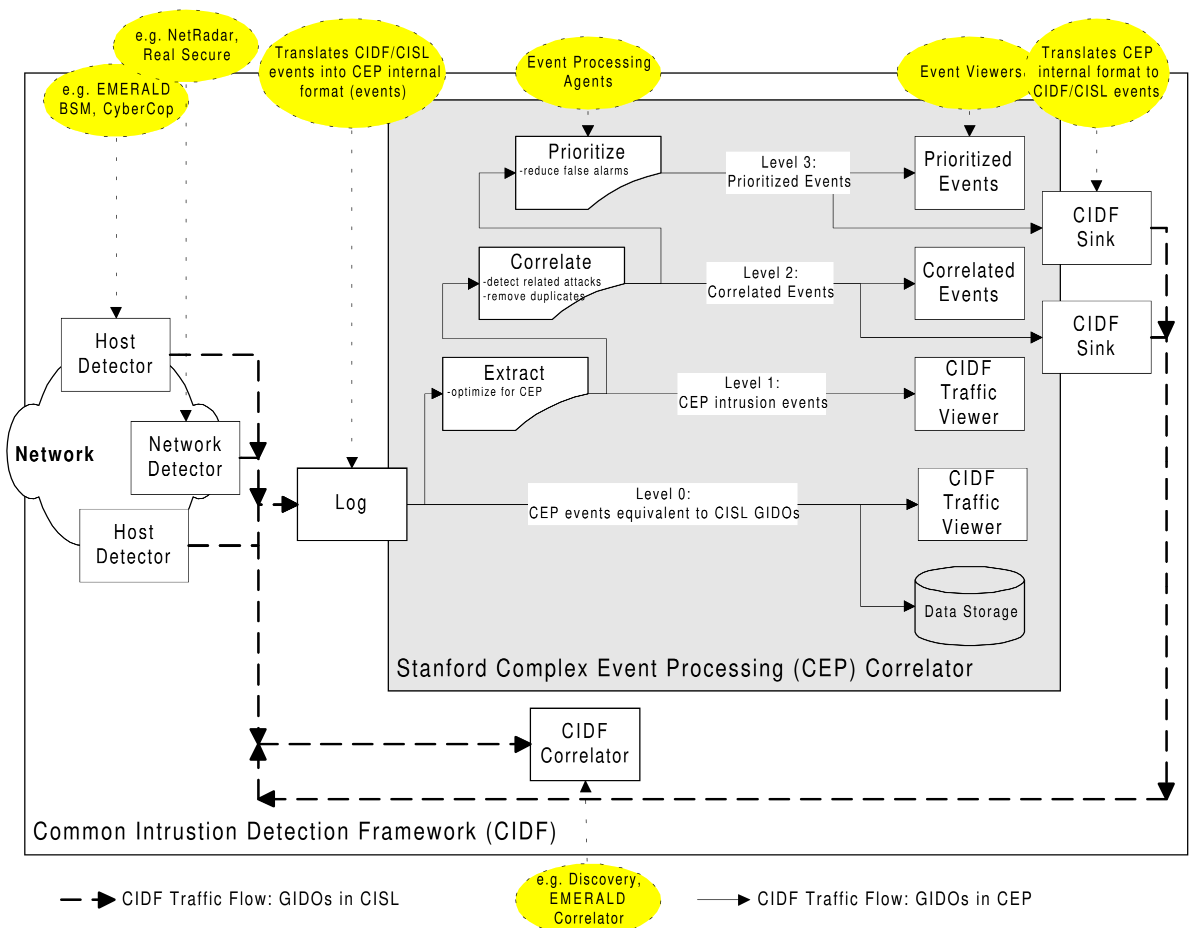

Figure 2. CIDF-CEP Integration

The logger, the event-processing agents, the viewers, and their connections as displayed in figure 2 build a small event-processing network (EPN).

In the following we look at a typical attack patterns going from network-probe to penetration of a host to gaining root-access. We set up an EPN that should alert us of such attack patterns as they happen. Figure 3 shows a screenshot of our management application `RapNet` displaying the EPN used for the following on the top right. `RapNet` lets you configure individual agents and networks of agents. Networks can be saved, edited, loaded, started, and stopped from within `RapNet`. The `Operators` panel displays the library of available agent templates as a hierarchy, grouped into `Inputs` (producers), `Maps` and `Filters` (processors), and `Outputs` (consumers). The maps group is currently expanded, and shows the five map agents available in the library. Two more entries contain `Connections` and drawing elements (`Decoration`) for documentation purposes. The agents in the library can be instantiated by dragging them over to the drawing panel at the right. The drawing panel currently shows the EPN consisting of a logger, four maps (their graphical representation is a box with a missing corner), and four viewers. The output of the logger is fed into a first map (`Extract`) the output of which is fed into three more maps and one viewer. The output of the three second level maps is fed into a viewer each.

The remaining four windows in figure 3 are all tabular viewers. They correspond to the four boxes on the far right of the RapNet window. The bottom right window shows the GIDOs from the extract agent. It corresponds to the lowest viewer element in the EPN displayed above. This viewer shows GIDOs in our internal format (some false positives have already been filtered out inside the extract agent). The top left window displays the output of the top map in the EPN. It shows ongoing attack patterns in different stages of escalation. In this window, each ongoing attack pattern is displayed as a separate line. The second line shows an alert about unusual activity on one host. CEP assumes this attack is part of a bigger attack pattern, however, no penetration attacks to this host have been observed so far. Thus CEP suspects undetected penetration attacks. The two viewers on the bottom show attacks grouped by attack ID or the source IP number of the attack.

Figure 4 shows a tool that lets you review the history of correlated data. The window on the top right is again a control window that manages the other windows. The other two windows are tabular viewers, the upper one displaying the escalation events and the lower one the GIDOs from the extract agent. They display the same data as the top left and bottom right viewers in figure 3. However, in figure 3 each line represented an ongoing attack pattern and was updated on every new attack observed. The viewers in figure 4 display one attack per line, thus an attack pattern is visible over multiple lines.

We now want to investigate the first escalation that is found by our escalation map. It is an attack pattern coming from a host with IP number 134.52.160.125. This escalation corresponds to the fourth line in the top left window of figure 3. In figure 4, it is event 11 in the upper viewer. In figure 3, the attack pattern has proceeded further in time, thus the line reads `Repeated penetration attack`. At the time displayed in figure 4, the message tells us that a first penetration attack from IP number 134.52.160.125 was just observed. The message contains `ESCALATION`, because so far we only observed network probes which are considered less harmful. All GIDOs that are correlated to the escalation event selected in the top viewer are automatically selected in the bottom viewer. We can see that before the penetration there was a `SATAN scan` from this machine, and then two `IMAP buffer` attacks. This mechanism allows us to quickly view an attack pattern over several levels of abstraction hierarchies.

A few more things to note: The second column in the lower window displays the detector that observed an attack. In our example, two detectors were involved (IP numbers ending with `114` and `163`). In this example CEP does not know the names of these detectors and displays `<unknown>`. For `StackGuard` it knows the name, and also displays it (event 9 in the bottom viewer). The escalation viewer displays the highest state of attack patterns in the 5th column. In the example above, this is only `Penetration`. More serious are the ones labelled `Root`, as these attackers successfully penetrated a machine and are working on getting root access.

4 Performance Experiments

The event rate in the CIDF experiment was only a few events a second. In another experiment we have to deal with much higher data rates. In one particular instance, we used a 31957-record log file produced by a CISCO NetFlow FlowCollector running on the Stanford Network (SUNet) gateway. The FlowCollector logged traffic over a ten minute time interval and aggregated it into a log file In this experiment our correlator watches network traffic and correlation is used to build a higher level view of the network traffic and also to detect suspicious activity like the detectors in the CIDF experiment. This log file was processed as follows:

- A logger produced imported the file with 31957 events within 1 minute of wall-clock time. It consumed log file records at an average rate of 500 records/second and generated events at the rate of 500 events/second.
- A filter applied to the output of the logger to filter out mail, web, DNS, and Auth traffic and passed 16888 events. This application took 2 minutes, 45 seconds of wall-clock time to run. It processed events at an average rate of 200 events/second and passed on events at an average rate of 100 events/second (as on average only every second event got passed through).
- A map applied to the output of the filter produced an output containing 3288 events. This application took 6 minutes, 15 seconds to run and processed events at a rate of 45 events/minute and generated events at an average rate of 9 events/second. In other words, this map reduced the input stream by a factor of about 20%.

The total time taken for all processing steps to be completed was 10 minutes. Thus a log file containing 10 minutes worth of traffic data was processed in 10 minutes. Since the various processing agents can be run concurrently within one EPN, the total processing time can be reduced to the total time taken by the map. We are constantly improving the infrastructure to increase throughput. Also, in all our experiments so far performance is CPU bound. This indicates that faster (or multiple) processors will increase our performance.

In other, scale related experiments, we run EPNs with up to millions of events with no decline in throughput. We are also actively looking for industry partners for further experiments.

5 Related Work

CEP is not intended as a replacement to other intrusion detection tools but rather as a complementary correlator that combines the output of multiple tools to produce better results. Tools like EMERALD [7] already have slots available to plug in future and third party analysis engines. [18]. EMERALD and its predecessor at SRI (IDES, NIDES) date back to 1992 [19]. They mostly perform statistical analyses of events. Signature analysis and the integration of an expert system added additional correlation capabilities. This makes EMERALD more advanced than commercial systems that basically support only simple signature analysis, but the full context of events is not yet a part of the correlation. In fact, the overall architecture of EMERALD is very similar to our own architecture as it supports distributed components that work together to perform correlation. Using CIDF as described in the example above EMERALD and CEP components can easily exchange events.

The Purdue University COAST project approached intrusion detection with pattern matching based on colored Petri Nets [20]. Apparently, only a small prototype has been implemented, and performance results are missing [21]. Kumar and Spafford recognized that “the sheer rate at which data is generated” is the most dominant problem, the need for partial orders to model concurrency, online processing, and dynamic patterns (i.e. patterns can be changed at run time). COAST intrusion signatures (i. e. event patterns) are represented as Petri Nets. Transitions are associated with the type of events in the pattern. The incoming events trigger tokens to flow through the net. Events in the middle of the net represent partial matches of patterns. The main problem of this approach is that partial matches are kept around forever. Also, some patterns representing repetition of events are hard to implement in Petri Nets. Last but not least, COAST Petri Nets are based on temporal relations between events only, they do not model causality.

P-BEST [22], Wisdom and Sense [23] and NADIR [24] are examples of intrusion detection systems that employ rule-based analysis.

A number of commercial products are also available. Due to the proprietary nature of these systems, it is difficult to understand how their correlation engines work. Kane Security Monotor (KSM) checks the security log files on servers and workstations being monitored for certain activities and patterns. KSM analyses these using its SHADOWARE artificial intelligence engine, and determines which events are normal, and which are attempts to breach the systems security. As with all commercial products, information about the capabilities of the correlators is hard to find. Event Orchestrar [25] is the current Network Associates, Inc. correlator. However, as for most products at this stage, the focus is on interoperation with all their tools.

The Open View correlation service as described in [26] is limited to simple filtering and aggregation. SMART correlation books is restricted to simple event correlation [27]. The correlation is not done by an EPN, but rather by single process based on a pre-compiled table of correlations. None of them support context-based correlation. Also, both [26] and [27] are focused on network management events, while CEP is applicable to any domain.

Intrusion Detectors like RealSecure [2], NetRanger [6], or [4] are strongly focused on intrusion detection on the packet level. They do not perform complex correlation and produce large amount of alarms. However, their output makes a great input for CEP.

6 Future Directions

Future directions are split into two directions: Implementation and theory. We are constantly improving throughput using engineering methods. Currently we are working on better support for concurrency so EPNs can be more efficiently distributed.

Regarding correlation theory, we are developping new pattern languages tailored to specific domain. For now, we used an existing pattern language for the intrusion domain. Our current language does not support statistical analysis very well. We plan to introduce support for simple statistics, like counting events that fit a certain signature.

We are also planning to make EPNs self-modifying by creating special agents that can modify other agents and create new agents to improve correlation. As an example, complex and time consuming agents may be turned on and off by other agents only when needed. We need to improve our theoretical foundations to be able to model such self-modification. Also, event patters are often not a priori known. Event mining allows automatic detection of new patterns, irregularities, divergences, and discontinuities on-line [15]. Event mining distinguishes itself from data mining (e.g. [28]) by the complexity of the underlying data (posets, vs relational tables as often the case in data mining) and by the time constraints (constant on average per new event).

7 References

1. Network Associates Inc., *CyberCop Scanner.* 1999. <http://www.nai.com/>.

2. Internet Security Systems, *RealSecure.* 1999. <http://www.iss.net/>.

3. Security Dynamics Inc., *Kane Security Monitor.* 1999.

<http://www.securitydynamics.com/>.

4. AXENT, *NetProwler.* 1999. <http://www.axent.com/>.

5. Net Flight Recorder Inc., *Net Flight Recorder.* 1999. <http://www.nfr.net/>.

6. CISCO, *NetRanger.* 1999.

7. SRI, *Emerald.* 1999. <http://www.sdl.sri.com/emerald/>.

8. Pratt, V.R., *Modeling concurrency with partial orders.* Int. J. of Parallel Programming, 1986. 15(1): p. 33-71.

9. Fidge, C.J., *Timestamps in message-passing systems that preserve the partial ordering.* Australian Computer Science Communications, 1988. 10(1): p. 55-66.

10. Mattern, F. *Virtual Time and Global States of Distributed Systems*. in *Parallel and Distributed Algorithms*. 1988: Elsevier Science Publishers.

11. Meldal, S., S. Sankar, and J. Vera. *Exploiting Locality in Maintaining Potential Causality*. in *10th ACM Symposium on the Principles of Distributed Computing*. 1991. New York, New York: ACM Press. p. 231-239.

12. Vera, J., L. Perrochon, and D.C. Luckham. *Event Based Execution Architectures for Dynamic Software Systems*. in *TC2 First Working IFIP Conference on Software Architecture (WICSA1)*. 1999. San Antonio, Texas, USA: Kluwer. p. 303-317.

13. RAPIDE, *Rapide 1.0 Pattern Language Reference Manual.* . Stanford University. 1997. <http://pavg.stanford.edu/>.

14. Luckham, D.C. and B. Frasca, *Complex Event Processing in Distributed Systems.* Computer Systems Laboratory Technical Report CSL-TR-98-754. Stanford University. 1998. <http://pavg.stanford.edu/cep/>.

15. Perrochon, L., *et al. Event Mining with Event Processing Networks*. in *The Third Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-99)*. 1999. Beijing, China: Springer. p. 474-478. <http://www.perrochon.com/>.

16. Kahn, C., *et al.*, *A Common Intrusion Detection Framework (CIDF).* . . 1999.

17. IDWG, *IETF Intrusion Detection Working Group*, . 1999, Internet Engineering Task Force (IETF). <http://www.ietf.org/html.charters/idwg-charter.html>.

18. Porras, P.A. and P.G. Neumann. *EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances*. in *Nineteenth National Computer Security Conference*. 1997. Baltimore, Maryland. p. 353-365.

19. Lunt, T.F., *et al.*, *A Real-Time Intrusion-Detection Expert System (IDES).* Technical Report . Computer Science Laboratory SRI International. 1992.

20. Kumar, S. and E.H. Spafford, *An Application of Pattern Matching in Intrusion Detection.* Technical Report CSD-TR-94-013. Departement of Computer Science, Purdue University. 1994.

21. Kumar, S. and E.H. Spafford, *A Software Architecture to support Misuse Intrusion Detection.* Technical Report CSD-TR-95-009. Departement of Computer Science, Purdue University. 1995.

22. Lindqvist, U. and P.A. Porras. *Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)*. in *1999 IEEE Symposium on Security and Privacy*. 1999. Oakland, California: IEEE Computer Society Press.

23. Vaccaro, H.S. and G.E. Liepins. *Detection of anomalous computer session activity*. in *1989 IEEE Symposium on Security and Privacy*. 1989. Oakland, California: IEEE Computer Society Press. p. 280-289.

24. Jackson, K.A., D.H. DuBois, and C.A. Stallings. *An Expert System Application for Network Intrusion Detection*. in *14th National Computer Security Conference*. 1991. Washington, DC: National Institute of Standards and Technology/National Computer Security Center. p. 215-225.

25. Network Associates Inc., *Event Orchestrator.* 1999. <http://www.nai.com/>.

26. Sheers, K.R., *HP OpenView Event Correlation Services.* Hewlett-Packard Journal, 1996(October).

27. Yemini, S., *et al.*, *High Speed & Robust Event Correlation.* Technical Report . System Management Arts (SMARTS). 1996.

28. Piatetsky-Shapiro, G. and W.J. Frawley, eds. *Knowledge Discovery in Databases*. . 1991, AAAI Press / The MIT Press.

Note: The next two pages should print figures 3 and 4. They may take some time to print as they contain raster data.

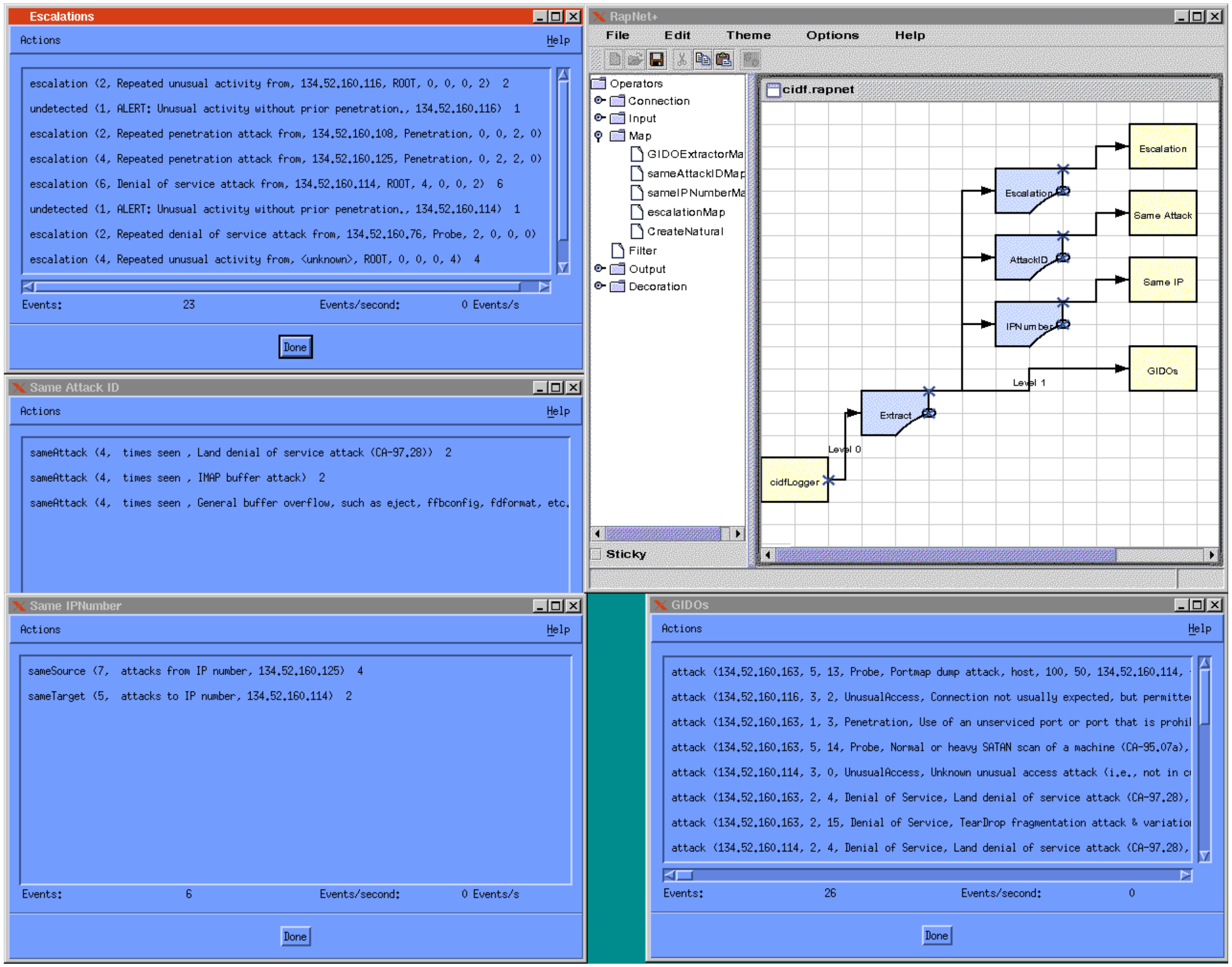

Figure 3. Screenshot of CEP running on CIDF data.

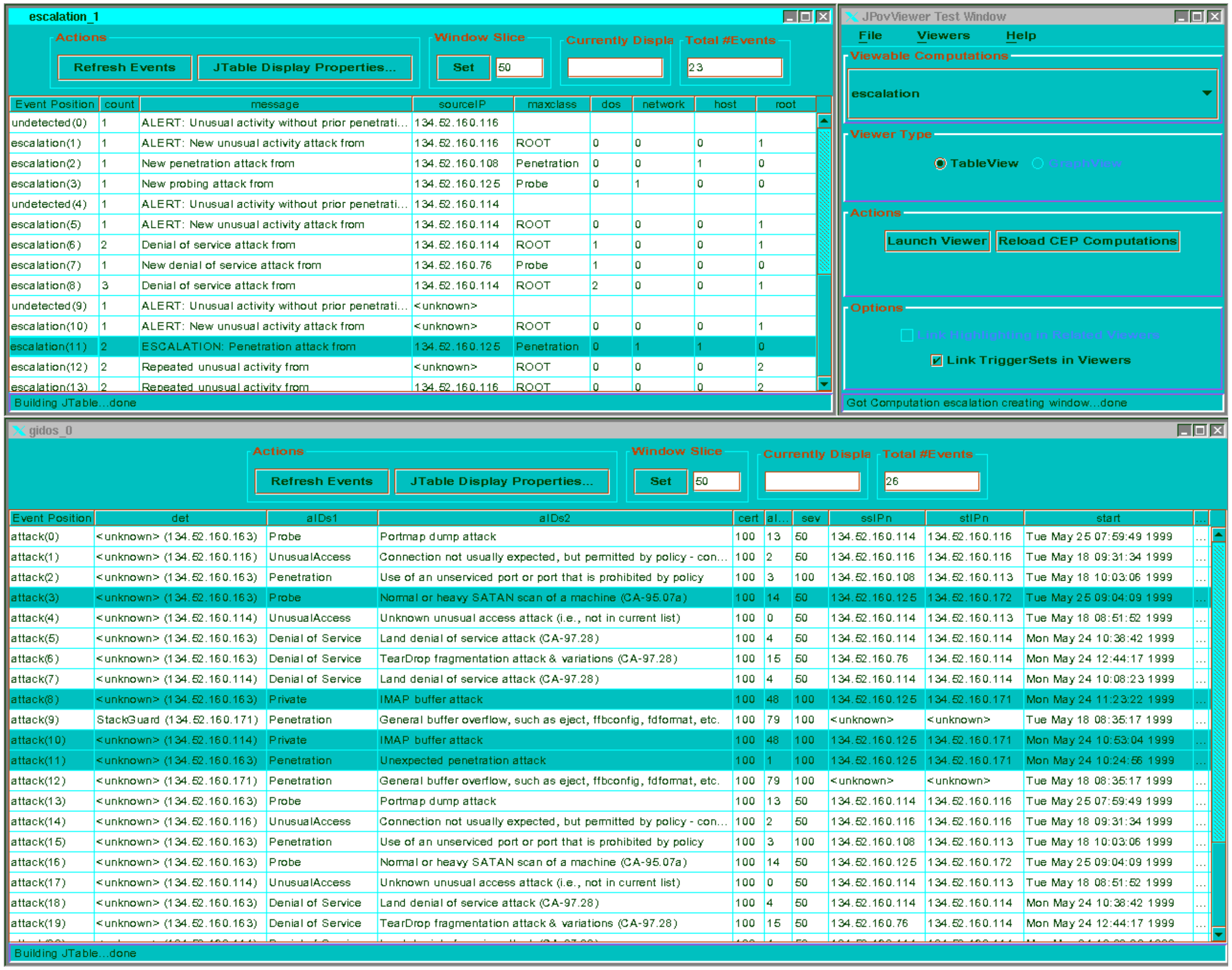

Figure 4. Screenshot of CIDF data analysis with CEP