Aggregation and Correlation of Intrusion-Detection Alerts

Hervé Debar[1,⋆] and Andreas Wespi[2,⋆⋆]

[1] France Télécom R&D
42 Rue des Coutures, F–14000 Caen, France
herve.debar@francetelecom.com
[2] IBM Research
Zurich Research Laboratory
Säumerstrasse 4, CH–8803 Rüschlikon, Switzerland
anw@zurich.ibm.com

Abstract. This paper describes an aggregation and correlation algorithm used in the design and implementation of an intrusion-detection console built on top of the Tivoli Enterprise Console (TEC). The aggregation and correlation algorithm aims at acquiring intrusion-detection alerts and relating them together to expose a more condensed view of the security issues raised by intrusion-detection systems.

Keywords: Intrusion detection, alert aggregation, alert correlation, alert data model.

1 Introduction

Intrusion-detection products have become widely available in recent years, and are beginning to gain acceptance in enterprises as a worthwhile improvement on security. They monitor accesses and data flows in information systems to determine whether malicious behavior is taking place, either from outside or inside, and make this information available to the operators of the information system. In addition, they can also react to malicious behavior and take some countermeasures.

The purpose of this work is to address the following areas of intrusion detection, which are known to present weaknesses:

1. *Flooding*. Intrusion-detection systems are prone to alert flooding, i.e., they provide a large number of alerts to the operator, who then has difficulties coping with the load. This problem has been recently highlighted by the release of *stick*[1] and *IDSwakeup*[2], two tools that flood an intrusion-detection

[⋆] This work was performed while employed by the IBM Zurich Research Laboratory.

[⋆⋆] This work was partially supported by the European IST Project MAFTIA (IST-1999-11583). However, it represents the view of the author. The MAFTIA project is partially funded by the European Commission and the Swiss Department for Education and Science.

[1] http://www.eurocompton.net/stick/projects8.html

[2] http://www.hsc.fr/ressources/outils/idswakeup/

system with unrelated alerts, carrying an effective denial-of-service attack against the operator if the intrusion-detection system manages to cope with the flux of anomalous events.
2. *Context.* Attacks are likely to generate multiple related alerts. Current intrusion-detection systems do not make it easy for operators to logically group related alerts.
3. *False alerts.* Existing intrusion-detection systems are likely to generate false alerts, be it false positives or false negatives, for various reasons. For example, attack descriptions (a.k.a. signatures) are often imprecise and also cover acceptable behaviors of the monitored information system, or the implementation does not conform to the specification and known attacks go undetected.
4. *Scalability.* Current intrusion-detection system architectures make it difficult to achieve large-scale deployment.

The outline of this paper is as follows. Section 2 describes a generic and scalable intrusion-detection architecture and introduces the concept of an *aggregation and correlation component* (ACC) that can analyze and correlate the alerts generated by intrusion-detection probes attached to it. In Section 3, we list the conceptual and operational requirements this ACC has to fulfill. Section 4 describes our architecture for alert processing as well as the unified alert data model we use. In Section 5, we discuss the aggregation and correlation algorithm that we have implemented in our prototype system, and in Section 6 we give a usage example. Section 7 contains the conclusions and offers ideas for future work.

2 A Generic, Scalable Intrusion-Detection Architecture

In the current state of intrusion-detection technology, we wish to distinguish between two kinds of components, the *probes* and the *aggregation and correlation components (ACCs).* What we call intrusion-detection probes in this context are usually referred to as intrusion-detection systems available either as commercial products or in the public domain. The purpose of the ACC is to correlate the output of several probes and give the operator a condensed view of the reported security issues. In the past few years, intrusion-detection research focussed on developing new probes or improving the technology behind existing ones. However, when a set of probes is deployed in large environments, the correlation problem becomes apparent. Thus far, aggregation and correlation have not been addressed by intrusion-detection product developers and are only now gaining momentum in the research community.

The relationship between probes and ACCs is shown in Figure 1. As one can see, the architecture is a distributed set of components, hierarchically layered to enable scalability of the whole system.

We have implemented a prototype of an ACC that is based on the Tivoli Enterprise Console (TEC) [4, 7]. The TEC is the event-handling product of

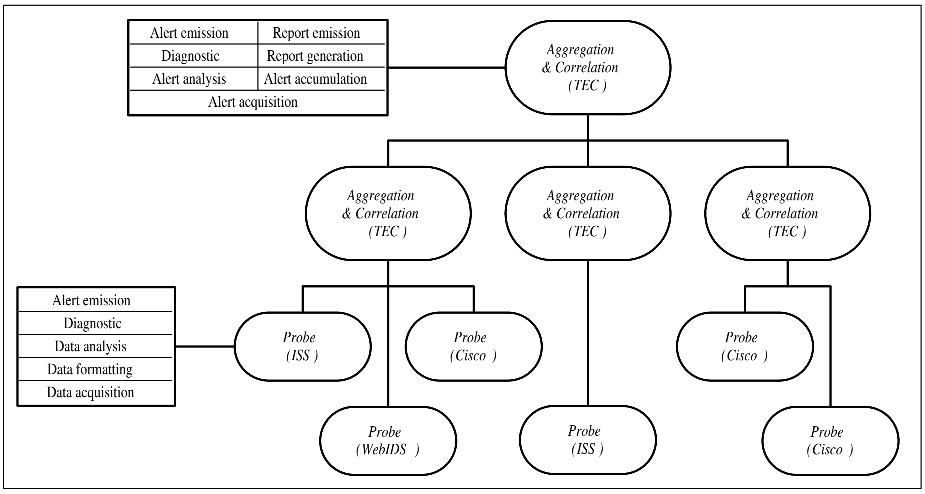


Fig. 1. Overall intrusion-detection architecture.

IBM/Tivoli Systems. Based on our technology, there is also a product available now, the *Tivoli SecureWay Risk Manager* [5, 6].

2.1 Probes

A state-of-the-art review of intrusion-detection systems oriented towards a taxonomy [3] lists many different probes, both host-based (e.g. using data sources provided on hosts) and network-based (e.g. tapping into the network traffic to retrieve data).

In our implementation and experiments, we used such probes as Internet Security Systems' *RealSecure Network Engine*[3] and Cisco Systems' *Cisco Secure IDS*,[4] as well as our own development prototype *Web IDS* [1] and simple, freely available tools such as *TCP Wrapper* [8] and *Klaxon*. We believe that these systems, although extremely useful, face a number of issues as far as large-scale deployment and operation are concerned, and that they should be considered as part of a larger ensemble rather than as individual solutions to the problem of detecting malicious activities against information systems.

As shown in Figure 1, probes are responsible for acquiring data from external sources (e.g. audit logs, accounting files, network packets), formatting this data to extract the information that is of interest for the analysis mechanism and preprocessing it to allow proper analysis. This analysis may lead the probe to generate *alerts* which are transmitted higher up in the architecture. Probes are

[3] http://www.iss.net/

[4] http://www.cisco.com/

expected to adhere to the evolving IDWG[5] standards for an intrusion-detection message exchange format and an intrusion alert protocol.

Probe implementors need to allocate resources to all functions of the probe. In particular, the data formatting, e.g. reassembling network packets or reconstructing process activity, can turn out to be very resource intensive. The time spent on data formatting is lost for analysis. Therefore, data analysis has to be kept simple, which is one reason why probes tend to generate false alerts.

2.2 Aggregation and Correlation Components (ACCs)

Each ACC receives alerts from probes and other ACCs. Alerts are sent in a standard format and do not need to be altered. Once an alert has been received, two sets of tasks are carried out, one to analyze the received alert in the context of the ACC (comprising the alerts received earlier and configuration information) and one to provide output to the local operator, if any.

The first set of tasks is covered by the aggregation and correlation algorithm described in Section 5. The second set of tasks allows an operator to interact with the ACC regardless of where this component is located in the tree. This feature accommodates the fact that organizations have multiple reporting levels; in our architecture, each department deploying the probes would have at least one ACC as well, which would be used for local reporting and for feeding higher-level ACCs corresponding to higher levels of the hierarchy.

3 Requirements

When we started this project, we had two groups of requirements. Conceptual requirements focus on the service that the ACC provides to the operator, independently of any implementation considerations. Operational requirements describe additional issues that are important but for which we are not yet able to provide a generic solution, or issues that are handled on an ad hoc basis.

3.1 Analysis of the Issues

This subsection provides a deeper analysis of the issues listed in the introduction, and describes the contribution of our ACC towards solving each issue.

Flooding One of the most apparent characteristics of intrusion-detection systems is that they tend to generate numerous alerts. An operator can be flooded with alerts very easily, and the usual reaction is to reduce the flux by restricting or even turning off a large part of the signatures that can be searched. This behavior is undesirable for two reasons:

1. The attack may still take place, and disabling signatures may reduce the number of false positives but may also increase the number of false negatives. Therefore, this approach does not solve the problem.

[5] http://www.ietf.org/html.charters/idwg-charter.html

2. For reporting, many alerts provide useful information, even though an operator may not wish to see them in real time.

Our ACC intelligently groups alerts together to provide the operator with sets of alerts. It offers multi-level views on sets of alerts, and by displaying only the most important views, the operator can concentrate on activities that could actually lead to successful compromise or denial-of-service.

Context An attack is very likely to manifest itself in multiple alerts spread over a period of time. A skilled attacker will first probe to evaluate its target and then start penetrating, the entire activity being sufficiently widely spread to go barely noticed.

Our ACC groups alerts together to provide an analysis of the entire context, not alert-per-alert. This allows the operator to carefully evaluate the progress of the attacker and the knowledge he has gained, and to ensure that the countermeasures taken are appropriate.

False alerts As experience shows, probes tend to generate false alerts that are directly related to the particularities of the information system in which they are deployed. Given that developers of intrusion-detection systems – by trying to avoid false negatives – accept a certain number of false positives, this is a frequent issue in today's systems. Inaccuracy of the alerts in the correlation component is considered in two forms:

1. *Intrinsic* inaccuracy of the alert owing to the probe's detection code being poorly written such that it does not discriminate well between normal and malicious activity for this particular attack.
2. *Relative* inaccuracy of the alert owing to the information system being monitored exhibiting characteristics similar to those of malicious activities.

Both aspects are taken into account in the ACC by associating each alert with a confidence value. This value is determined independently of the product vendor. Default values take into account the intrinsic inaccuracy of the alert and should be modified by the operator to consider relative inaccuracy, which results in reducing the confidence value.

Further, our ACC can be configured such that it knows under which circumstances different probes should report the same attack. If an expected alert does not arrive at the correlation component, it can be deduced that a probe is no longer working properly or does not adhere to its specification.

Scalability Given the number of alerts generated by current intrusion-detection probes, scalability rapidly becomes an issue. When deploying a larger number of probes, operators are forced to reduce the number of vulnerabilities monitored to limit the number of alerts to a reasonable level.

Our ACC distributes the load of handling alerts to the appropriate level in the hierarchy of any organization.

Of course, some of these issues could be solved at the probe level. For example, by writing better signatures many false alerts could be avoided. Although we expect to see better probes in the future, there will be issues that are site-specific or are not addressed perfectly by the probes. Therefore, we think that the ACC has to address all the four issues listed above.

3.2 Conceptual Requirements

The conceptual requirements focus on the *semantic* of the information that is provided to operators, on the *scalability* of the entire intrusion-detection system, on the *reactivity*, and on the *proactivity* that must be achieved to make such a system usable.

Semantics One of the main issues that prevents large-scale deployment of intrusion-detection probes is the level of information they generate. Intrusion-detection alerts are very low-level bits of information from which it is difficult to obtain a global picture. The requirement on our ACC is to present one alert per attack, even if this attack has generated many alerts. It must thus present the most concise picture for each attack, while providing as much information as possible. This includes aggregating and correlating from as many sources as possible to ensure that no false alert is raised to the operator. To satisfy this requirement, we have developed the concept of *situations*, described in Section 5.3.

Scalability Trade-offs have to be made between information collection and system performance. In environments with a large number of probes, a single ACC is easily overflooded.

In our architecture, multiple ACCs can be deployed, each of them handling a set of probes on a topological or logical basis. These correlation components in turn report to a master in a tree structure. As many layers of ACCs as needed can be introduced to handle the required number of probes. Each ACC provides an interface for local reporting, and sends data to the upper layer.

Reactivity The ACC must at least allow and better manage the reaction to an intrusion by:

- automatically gathering more information (e.g. raw logs, audit session, nslookup host, finger user). Being integrated with a network management platform allows the ACC to make use of the network topology or available discovery services;
- automatically modifying the setup of the intrusion-detection probes. This includes modifying the configuration of downstream reasoning engines in a distributed correlation setting;
- automatically escalating and warning the appropriate person;
- automatically applying appropriate countermeasures.

Our ACC has the potential and the mechanisms to enable automatic countermeasures. However, this part is still work in progress, as automatic countermeasures introduce a high risk of self-inflicted denial-of-service attacks.

Proactivity The ACC must be able to proactively expect intrusion-detection alerts according to the current flow of alerts or according to the time of day. For example, routine vulnerability assessment scans are usually scheduled periodically.

Our ACC can be configured to expect such scans and provide an alert if the scan does not take place. The same mechanism applies time constraints to series of alerts; frequently, sequences of alerts can be shown to correspond to automated, information-system specific behavior. Once these sequences have been identified, they can be canceled out by the ACC, and only anomalies in the sequence chains are reported.

3.3 Operational Requirements

These requirements focus on the interaction of the entire intrusion-detection system (probes and ACCs) with the management platform. As such, the intrusion-detection system must *integrate* with the management platform, and ensure an easy *configuration* and a certain level of *performance.*

Integration We believe that in its current state, intrusion-detection systems impose an additional burden on operators to deploy, configure and maintain. Integration into an existing framework will give operators a familiar look-and-feel, and allow them to use intrusion detection like any other application in their realm.
Our prototype is implemented on top of the Tivoli Enterprise Console (TEC), a commercial event-handling product. It takes advantage of the existing facilities for event acquisition, storage in a database and display. The ACC is implemented as a set of prolog rules for the prolog engine that sits inside the TEC and processes events.

Configurability The ACC must allow an easy reconfiguration of the reasoning parameters (i.e. severity, confidence, specific known-bad or known-good hosts). This should alleviate but not solve the false-alert problem, as operators observing false alerts could reduce the trust or level of the corresponding alerts, so that their weight in the correlation process is lowered accordingly. In our prototype, configuration remains manual, by means of configuration files.

Performance Network management environments are often capable of handling hundreds of alerts per second. However, given the complexity of the processing required to achieve meaningful correlation and the diversity of information contained in the alerts, it is difficult for an ACC to achieve the same performance. Therefore, ACCs have to be carefully designed and implemented so that they offer optimal performance [2].
Our ACC is developed with the target of handling one alert per second. This corresponds to the maximum alert rate we observe in the environment for which our prototype was built.

4 Representation of Alerts

The first task we faced when developing the ACC was to develop a unified framework for intrusion-detection alerts that would allow us to process them regardless of their origin. This led to the definition of a system architecture to gather alerts and the definition of a data model in the form of a class hierarchy.

4.1 System Architecture

The ACC acquires alerts from the intrusion-detection probes as shown in Figure 2. We distinguish between two kinds of probes, Tivoli-aware and not.

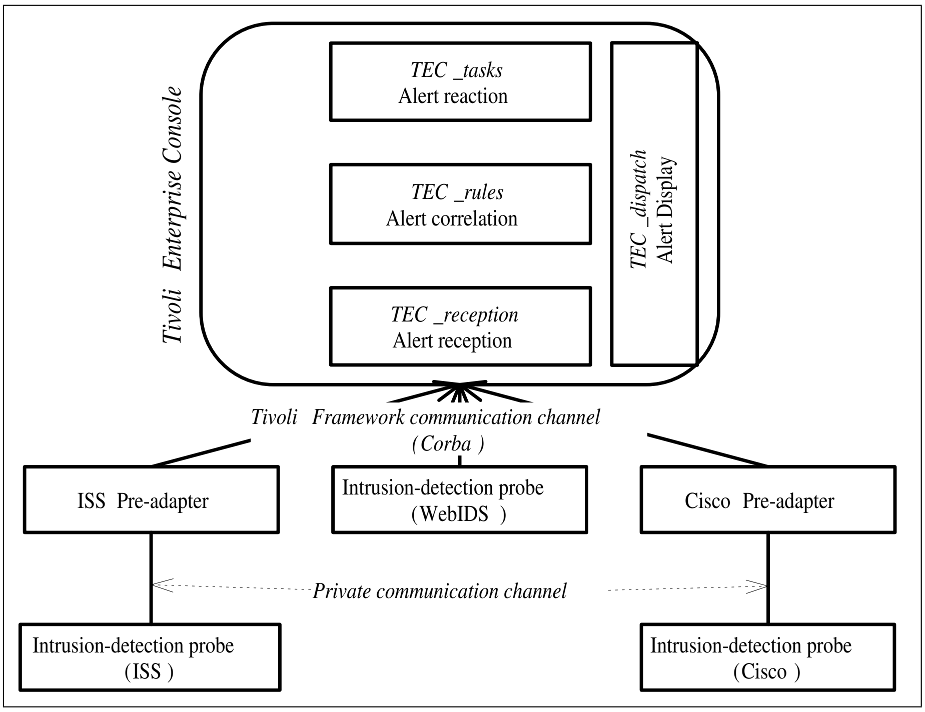


Fig. 2. System architecture for the ACC.

A Tivoli-aware probe can send events directly to the Tivoli Enterprise Console (TEC); our architecture in this case assumes that the probe also knows the data model and the appropriate format for events. An example of such a probe is Web IDS [6].

A non-Tivoli-aware probe requires an additional interface component to realize the transformation of native events into our data model. This probe-specific interface component is called a pre-adapter. It runs on the probe and knows how to read alert information in real time (or as close to it as possible). It formats the alert according to the alert class hierarchy described in Section 4.2. It then ships this formatted alert using the Tivoli Framework communication facility to the TEC where the ACC is implemented. Currently, the pre-adapter has no additional function. However, if performance becomes an issue, we envisage giving it more functionality, such as providing alert counts, to reduce the workload of the ACC.

Note that alerts are defined in both places: in the probe or the pre-adapter and in the TEC. If the two do not match, the alerts will not be processed by the correlation rules. Also note that this architecture does not prevent us from using the standard event sources (e.g. syslog) already found in information systems. Additional adaptation can be performed by the ACC, with some performance cost (see Section 5.1).

4.2 Alert Class Hierarchy

One of the main challenges of this work was the creation of a unified data model for acquiring intrusion-detection alerts. This data model is vital to the project for two reasons: ensuring independence between the correlation algorithms and the actual alerts generated by the probes, and ensuring that any probe can easily be integrated into the system and benefit from the generic correlation rules. The data model deployed in our prototype was also used as the basis of the IDWG work on a standard message-exchange format.

There are two kinds of classes. *Abstract* classes represent generic alert concepts that can be used by all intrusion-detection probes. *Implementation* classes inherit from abstract classes and carry specific alerts generated by a specific probe. The design of the abstract classes is partially shown using UML notation in Figure 3.

The abstract classes hierarchy heavily uses the inheritance relationship provided by object-oriented design methods. This inheritance mechanism provides a very strong structure to the model and prevents ambiguities when deciding where an implementation class for an alert belongs in the hierarchy.

The model is data-driven, which means that specialized classes must provide more information than the more general classes from which they are derived; in practice this translates into adding attributes to the derived classes.

The data model is roughly organized in layers to facilitate understanding, illustrated as shaded boxes in Figure 3. They are described from top to bottom.

Probe layer The probe layer carries information about the probe itself (primarily error or configuration messages) as well as very generic information about the alert, e.g., a short alert description or the time stamp of the alert.

Target layer The target layer ensures that all alerts instantiated by a class inheriting from this level include target information, i.e. the identification of one or multiple hosts that are the possible victims of an attack. When the alert reports that multiple hosts are targets of the attack, at least a headcount is provided.

Source layer The source layer ensures that all alerts instantiated by a class inheriting from this level include source information, i.e. some identification of the host from which the attack appears to be launched. We distinguish between two kinds of sources, *real* and *spoofed* sources. *Real* unfortunately does not mean that the source of the attack is identified reliably; it simply means that there are no indications of obvious spoofing. *Spoofed* on the other hand is used only when the attack *by construction* requires IP spoofing.

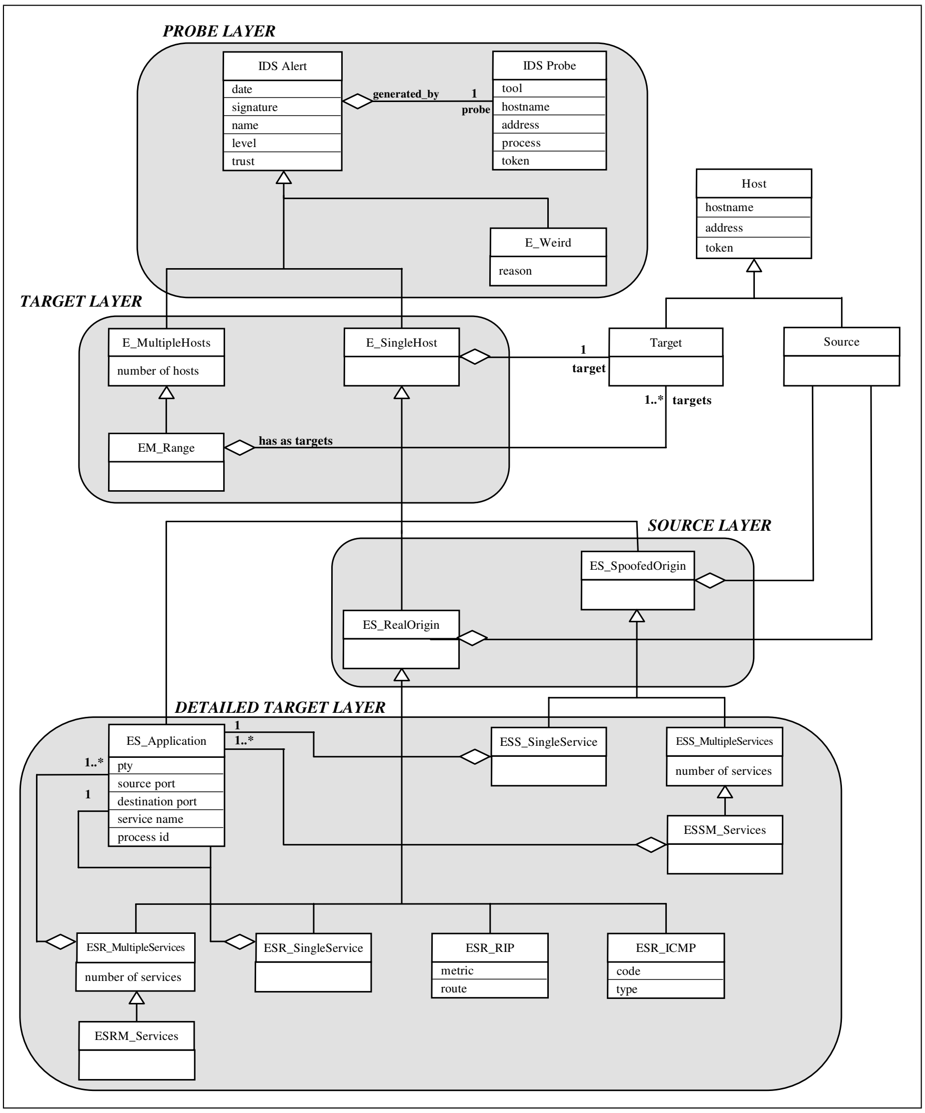


Fig. 3. Description of the alert class hierarchy.

We expect most alerts to provide at least this level of detail. In fact, 90% of the alerts we generate inherit from children of the *ES_RealOrigin* class.

Detailed target layer At this layer, detailed information is provided about the target. In particular, additional information such as email addresses, URLs, and ICMP information is provided by classes defined at this layer.

To give a better idea of the complexity of the data model, there are about 45 abstract classes, 175 implementation classes for the RealSecure preadapter and 80 for the Cisco Secure IDS pre-adapter.

5 The Aggregation and Correlation (AC) Algorithm

The goal of the aggregation and correlation algorithm is to form groups of alerts by creating a small number of relationships that are exposed to the operator, instead of exposing the raw alerts. The ACC uses two different kind of relationships:

Correlation relationship The algorithm creates correlation relationships between related events according to explicit rules programmed into the ACC or derived by the ACC from configuration information. Once events are in a relationship, they are considered part of the same trend of attacks and as such they are processed together. There are currently two kinds of correlation relationships between events: *duplicates* and *consequences*.

Aggregation relationship The algorithm groups events together according to certain criteria (similar to database views) to compute an aggregated severity level. Whereas the individual severity level of each event might not be sufficient to warrant specific analysis, the grouping may reveal trends and associations that clarify the intentions of the attacker.

The AC algorithm is applied in steps. It processes first the incoming alert by extracting common information, such as source and target host, and tries to find a match for these attributes in previous observations. It then creates correlation relationships by checking for duplicates and consequences, and finally creates the aggregation relationships through the *situations*. Once the situations have been updated, the appropriate alarms are generated, if necessary, and the alert is stored in a database. In Sections 5.1 - 5.3 each step is described in more detail.

5.1 Preprocessing of Alerts

Preprocessing is needed to unify the information available in an alert and provides some error checking to verify that the alert does not contain information that is obviously wrong, e.g. an invalid time stamp, and would prevent the correlation algorithm from working correctly.

Probes identify the source and target of an attack in different ways. In particular, network-based systems are very likely to provide IP addresses, whereas host-based systems will provide host names. To ensure unique identification of these actors, each is assigned a *token*. The preprocessing translates the information available in the alert into three tokens: PROBE, SOURCE and TARGET. This translation process does not use external information acquisition, such as DNS queries, because of the latency involved in retrieving the information, but relies on information available in configuration files.

Each of the three token definitions is separate. This means that if a host acts as probe, source and target, it has to be configured three times. If a host is not configured, the AC algorithm generates the appropriate token and stores whatever information is available in the alert in this token. The original tokens were integers assigned via a monotonically increasing function. However, this scheme poses the problem of synchronizing tokens between multiple ACCs in a distributed environment. The current token scheme uses the IP address of the host as a token. When dealing with multi-homed hosts or name aliasing, one IP address needs to be selected as the primary identification token of the host. Then, many associations of this token IP address with host names or additional alternate IP addresses can be configured to identify the host as a single entity.

The AC algorithm needs a unified time reference for alerts. This translates into the requirement that probes be synchronized. For example, in the current implementation we require that the time of the probes does not differ by more than two seconds. This does not mean that alerts should arrive in a time-ordered way at the ACC, only that their time stamps were generated in the same time frame. The AC algorithm keeps the time of the most recent event received and updates this time as events arrive. When a large difference between this "most recent" time and the time of the current event is observed, the AC algorithm can underevaluate the severity of events. This is abnormal and the ACC generates a warning message. In this case, the "most recent" date is reset to the time of the last alert received.

In addition, the normalization process provides mechanisms for translating well-known service names into port numbers and vice versa.

5.2 The Correlation Relationship: Duplicates and Consequences

This part of the AC algorithm deals with alerts that are logically linked with each other. The reasoning is separated into two parts. First, a backward-looking part determines whether an alert represents a cause that has already been taken into account by the AC algorithm, e.g. is a *duplicate* alert. Then, a forward-looking part determines whether the current alert must be followed by others, and in which condition this linkage occurs, e.g. whether *consequence* alerts have to be expected.

Duplicates The detection of duplicates relies on the provision of common information by different intrusion-detection sensors, or the provision of mechanisms to bridge the different kinds of information. Provision of common information means, for example, that two alerts generated by two network-based intrusion-detection systems contain an identical quadruple (source address, source port, target address, target port) and times that are close to each other. Provision of mechanisms to bridge different kinds of information means, for example, that there are facts and rules to indicate that two kinds of information are the same (for example port number 80 and port name www; two events, one generated by a host-based probe as (source_hostname, target_hostname, service) and the other by a network-based probe (source_ip, source_port, target_ip, target_port), will be considered the same if the service can be related to the port target_port, and if the addresses and host names match).

In a configuration file, it has to be specified which alerts are considered duplicates. The *duplicate definition* consists of four associated terms:

Initial alert class The first term is the class of the alert received first.

Duplicate alert class The second term is the class of the alert that is under evaluation to be a duplicate of the first one.

List of attributes The third term is a list of alert attributes. These attributes must be equal for the two events to be considered duplicates.

Severity level The fourth term is a new severity level for the duplicate alert. When a duplicate is found, the AC algorithm uses this severity level for further processing instead of the original severity level.

When an alert is received, the AC algorithm searches for duplicate definitions that have as their second term the class of the current alert. It then searches the base of previously received alerts for one that matches the class of the first term and conforms to the attribute conditions. If this original alert is found, the current one is linked to the original one and the severity level of the duplicate definition is assigned to the duplicate and used for further processing.

Note that the duplicate notion implies an ordered sequence. If there is no notion of order, then two duplicate definitions should be configured by reversing the order of the classes.

Also, the severity level is a value that can be both positive (the situation is more dangerous if the alerts happen one after another) or negative (the situation is less dangerous). A specific processing has been applied to the null value. When the severity level is 0, the duplicate is simply ignored.

Actually, the AC algorithm does more in this case: if the duplicate definition triggers more than Y times (i.e. the counter on the original event reaches a given threshold), then a new signature is generated (`Event of class X repeated Y times`) and the contribution of the original event severity times Y is added to the severity of the situation. This mechanism has been designed to handle alert floods, such as repeated *Ping of Death* or *Teardrop* attacks. Such alerts usually arrive in batches of 50 alerts or more. The Y threshold is actually a list of thresholds (currently 10, 50, 100, 200 and 500), and such alerts will only be processed when the count on the original alert reaches one of these values.

Consequences A consequence chain is a set of alerts linked in a given order, where the link **must** occur within a given time interval. If the link does not occur, an internal alert is generated with the severity given by the consequence definition.

As it holds for the duplicates, consequences have to be defined in a configuration file. The *consequence definition* consists of six associated terms:

Initial alert class The first term is the class of the original alert.

Initial probe token The second term indicates the probe from which the alert comes. This can be a variable (in which case it should be the same variable as in the fourth term, to indicate that the only constraint is that the two probes are one and the same) or a wildcard.

Consequence alert class The third term is the class of the alert considered a consequence of the first one.

Consequence probe token The fourth term indicates the probe from which the alert comes. This can be a variable (in which case it should be the same variable as in the second term, to indicate that the only constraint is that the two probes are one and the same) or a wildcard.

Severity level The fifth term is a severity level for the signature simulating the missed alert.

Wait period The sixth term indicates the time to wait for the consequence alert.

When an event occurs, the consequence mechanism is triggered. As part of this process, the consequence mechanism verifies that the current alert is a consequence of some previously received alert. If this is the case, the consequence link is marked as having taken place. When the consequence timer expires on the original event and all consequences have been satisfied, the processing of the original event is terminated.

The consequence mechanism requires the events to occur in a fixed sequence. If the relationship is true regardless of the order, then two consequence definitions should be used. The consequence mechanism does not provide a way to indicate that a consequence occurred after the fixed time limit. Once the timer has expired, all linkage is removed.

Example of Duplicates and Consequences Let us look at the example of a network-based probe able to detect attacks against Web servers in the network packets and the Web IDS [1] probe that looks for signs of attacks in the Web server log files. When the network-based probe recognizes a malicious CGI script request, it does not know whether the request actually succeeded because it only analyzes the request to the Web server but not the corresponding answer from the Web server. Since alerts that are related to suspicious CGI scripts occur quite frequently, the confidence and severity of this type of alert are low. Web IDS on the other hand knows whether the attack succeeded because the Web server log entry shows the status code in addition to the URL.

When the Web IDS alert is received, it is used to upgrade the base event and redo the analysis in that light, to lower the severity if the request was unsuccessful, or to increase it if the request was successful.

An example of what the AC algorithm achieves with the detection of duplicates is shown in Figure 4, which presents an extract of log messages caused by two PHF attacks against our Web server. The first PHF attack retrieves the `/etc/group` file, and the second one retrieves the `/etc/passwd` file. Both attacks are successful in the sense that the Web server returned status code 200 (but there is no way to tell whether the requested file was actually shipped).

```
WebIDS: 934190025 pattern(cgi) phf 10.10.10.62
        "GET /cgi-bin/phf?/etc/group HTTP/1.0" 200 442
WebIDS: 934190025 pattern(UrlSuccess) ^2 10.10.10.62
        "GET /cgi-bin/phf?/etc/group HTTP/1.0" 200 442
WebIDS: 934190025 decision(followup) warnings 10.10.10.62
        "GET /cgi-bin/phf?/etc/group HTTP/1.0" 200 442
WebIDS: 934190027 pattern(suspiciousCgi) passwd 10.10.10.62
        "GET /cgi-bin/phf?/etc/passwd HTTP/1.0" 200 444
WebIDS: 934190027 pattern(cgi) phf 10.10.10.62
        "GET /cgi-bin/phf?/etc/passwd HTTP/1.0" 200 444
WebIDS: 934190027 pattern(UrlSuccess) ^2 10.10.10.62
        "GET /cgi-bin/phf?/etc/passwd HTTP/1.0" 200 444
WebIDS: 934190027 decision(followup) warnings 10.10.10.62
        "GET /cgi-bin/phf?/etc/passwd HTTP/1.0" 200 444
RealSecure: 934190025 1 HTTP\_PHF 10.10.10.62:9285 10.10.10.61:80
            URL="/cgi-bin/phf?/etc/group"
RealSecure: 934190027 1 HTTP\_PHF 10.10.10.62:9304 10.10.10.61:80
            URL="/cgi-bin/phf?/etc/passwd"
RealSecure: 934190027 1 HTTP\_Unix\_Passwords 10.10.10.62:9304
            10.10.10.61:80 URL="/cgi-bin/phf?/etc/passwd"
```

Fig. 4. Web IDS and RealSecure alerts.

Web IDS is quicker to report the two attacks, so the Web IDS messages are packed together first, and then the RealSecure messages arrive. The first three Web IDS messages indicate that the PHF script has been requested, that a script has been found, and that a warning should be generated as a consequence of these two actions. For the same URL, RealSecure gives only the first warning, the PHF request. Analysis is similar for the additional four Web IDS messages and two RealSecure messages concerning the PHF request for `/etc/passwd`. If we analyze in which ways these messages are duplicates, we find that they have the same source and destination, the same URL, and the same time stamp. In addition, the RealSecure messages have the same port information. We conclude that for the same probe it is quite easy to identify duplicate alerts, whereas finding duplicates for different probes is more difficult, especially if we cannot rely on time accuracy.

An additional principle to take into account is the localization of the detector, and the possibility to trace the path of an attack. In particular, placing two network-based sensors on both interfaces of a firewall with network address translation (NAT) is certainly interesting but requires much more work to actually relate the alerts to the same "logical" packet.

5.3 The Aggregation Relationship: Situations

In many cases, isolated events are not considered significant. Therefore, intrusion-detection alerts are aggregated into so-called "situations". A situation is a set of alerts that have certain characteristics in common.

Each intrusion-detection event contains information that can act as an aggregation axis. Our current prototype recognizes three aggregation axes, namely the source, the target, and the class of the attack.

The *situation definition* consists of the following terms:

Alert class The first term is the class of the alert or a wildcard. A wildcard indicates that the alert class is not used as an aggregation axis.

Source token The second term is the attack source or a wildcard.

Target token The third term is the target of the attack or a wildcard.

Severity level The fourth term is a threshold. If the aggregated severity of the situation alerts exceeds this threshold, an alarm is generated.

Since wildcards are allowed, different situation types are possible. By systematically evaluating all combinations of aggregation axes and leaving out the case where a wildcard is given for source, target, and alert class, we end up with seven different situations as follows:

Situation 1 Alerts with the same source, the same target and belonging to the same alert class are aggregated in Situation 1. This allows one to detect, for example, an attacker who is launching a series of Web server attacks against a single Web server.

Situation 2-1 Alerts with the same source and destination are aggregated. This situation is intended to detect, for example, an attacker who runs a series of attacks against the various services available on the target machine.

Situation 2-2 Alerts with the same target and belonging to the same alert class are aggregated. This situation can be used to detect a distributed attack against a single target.

Situation 2-3 Alerts with the same source and belonging to the same alert class are aggregated. This situation allows one, for example, to find an attacker who is running a series of name server attacks against a set of name servers.

Situation 3-1 Alerts with the same source are aggregated. The goal is to detect a single attacker who runs various attacks against different targets.

Situation 3-2 Alerts with the same target are aggregated. This allows one to detect distributed attacks.

Situation 3-3 Alerts belonging to the same attack class are aggregated. This situation is triggered if, for example, a large number of people are trying out a new attack that was recently posted to a hacker mailing list.

More specific situation alarms have precedence over less specific situation alarms. For example, if we assume that the thresholds for all situations are set to the same value and situation 1 is triggered, only an alarm for situation 1 but not for situations 2 and 3 is generated.

For situation 2, a list of the values specified in the wildcard field is maintained. We refer to it as an alert property list. For example, for situation 2-1 the alert property list is the list of all attacked hosts. In the case of situation 3, two alert property lists are maintained.

Determining Authorized Scans Our ACC can detect authorized security scans. The assumption made is that the authorized scan is always executed from the same machine. The operator of the ACC knows the address of the source machine and can configure the ACC accordingly.

When messages related to the preconfigured source address appear, within (or without) a specific time interval, they are processed by a specific set of situations based on situation 3-1. However, in addition to accumulating scan data on that particular class, a comparison of the signatures of the previous scan in terms of hosts touched and signatures tried is performed to see whether target machines or signatures have been added to or removed from the test. This helps pinpoint scan anomalies and also measures the effectiveness of the probes.

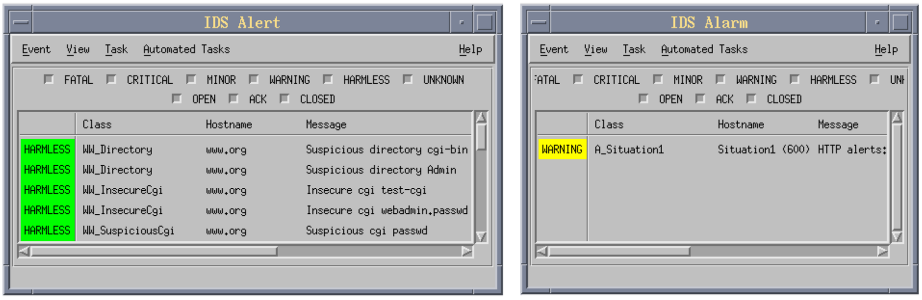
Changing Situations According to Sets of Events A trigger can be associated with situations. There are two types of triggers, one for *situation reevaluation* and one for *multi-situation assessment.*

Situation reevaluation addresses the case when the operator considers the severity of the combination of several alerts not being appropriately reflected by the sum of the severities of the individual alerts. For example, the operator may believe that a single unsuccessful login request is a mistake and therefore has a low severity. However, if somebody tries to unsuccessfully login to ten default accounts, then the severity should be much higher than ten times the severity of a single unsuccessful login request. The situation reevaluation operates on the lists of aggregated alert properties to verify whether the list matches a given criterion. For the moment, two criteria have been identified, the length of the list, and the fact that the list contains a given sublist. This works for situations 2-1, 2-2 and 2-3. For situations 3-1, 3-2 and 3-3, the operator has to indicate if the reevaluation applies to one property list only, to both of them jointly, or to either one of them. The reevaluation takes the form of a modification of the severity value, either by a constant, or by a multiplication factor.

Multi-situation assessment is mostly of use for situation 1, and consists of modifying the severity value if another, related situation exists. For example, if a host has probed the primary DNS server and now also probes the secondary DNS server, the severity of both situations should be increased.

6 Usage Example

We give a simple example to demonstrate what our intrusion-detection console looks like and how it is intended to be used. The TEC allows one to define different views of the alerts that were sent to or generated by the TEC server. We have introduced basically two new views, the *IDS Alert* view where all intrusion-detection related alerts are accessible, and the *IDS Alarm* view which provides an interface to those alerts the operator has to take care of. As a simple example, Figure 5 shows part of the events the Web IDS probe has generated as a result of a detected Web-scan attack.



	Class	Hostname	Message
HARMLESS	WW_Directory	www.org	Suspicious directory cgi-bin
HARMLESS	WW_Directory	www.org	Suspicious directory Admin
HARMLESS	WW_InsecureCgi	www.org	Insecure cgi test-cgi
HARMLESS	WW_InsecureCgi	www.org	Insecure cgi webadmin.passwd
HARMLESS	WW_SuspiciousCgi	www.org	Suspicious cgi passwd

	Class	Hostname	Message
WARNING	A_Situation1	Situation1 (600)	HTTP alerts:

Fig. 5. IDS Alert and IDS Alarm views

In the *IDS Alert* window shown on the left, all the different CGI attacks that were executed show up. In the *IDS Alarm* window on the right, only one message pops up, a situation 1 alarm that summarizes all the individual Web attacks.

The idea is that the operator has only to look at the messages in the *IDS Alarm* window. All the security-relevant alarms are displayed in a condensed manner in this window. However, access to single alert messages is still possible via the *IDS Alert* view.

7 Conclusions and Future Work

In this paper, we have shown the need for an aggregation and correlation component (ACC) that can handle alerts generated by intrusion-detection probes. We have discussed the requirements such an ACC has to fulfill and have derived a system architecture to gather and process alerts in a central place.

The ACC comprises two main parts: a unified data model for intrusion-detection alerts and a set of rules to process the alerts. The AC algorithm can detect duplicates, i.e., alerts that are reported by different probes but are related to the same attack, as well as consequences, i.e., alerts that are related and should occur together. We have introduced the concept of situations that allows us to aggregate similar alerts and thus provide the operator with a more condensed view of the security issues to be addressed. Alerts are aggregated into situations based on any combination of the three attributes source, target and alert class.

Our ACC is built on the Tivoli Enterprise Console. The integration into an existing event management framework had the advantage that we could concentrate on the correlation aspects of our work. Furthermore, the operator is provided with a familiar user interface.

As for future work, we plan to enhance the generic rules of the AC algorithm with more specific ones. To improve performance, we will investigate the possibilities of doing the normalization of alerts as a front-end to the TEC. Currently, the ACC does not control the probes but only processes the alerts they generate. Therefore, a further enhancement will be to better integrate probes and ACCs.

Our framework will benefit from having a wide variety of probes integrated with it. Hence, we would like to collaborate with partners who are interested in seeing their probes being integrated in our framework.

References

[1] Magnus Almgren, Hervé Debar, and Marc Dacier. A lightweight tool for detecting web server attacks. In *Symposium on Network and Distributed Systems Security (NDSS '00)*, pages 157–170, San Diego, CA, February 2000. Internet Society.

[2] Taurus Bey. Tec rules: Planning for efficiency. *The Managed View*, 3(2):5–20, Spring 1999.

[3] Hervé Debar, Marc Dacier, and Andreas Wespi. A revised taxonomy for intrusion-detection systems. *Annales des télécommunications*, 55(7–8):361–378, July–August 2000.

[4] IBM International Technical Support Organization. *Early Experiences with Tivoli Enterprise Console 3.7*, November 2000. IBM Redbook SG24-6015-00.

[5] IBM International Technical Support Organization. *Tivoli SecureWay Risk Manager: Correlating Enterprise Risk Management*, November 2000. IBM Redbook SG24-6021-00.

[6] Tivoli Systems. *Tivoli SecureWay Risk Manager, User's Guide, Version 3.7*, December 2000.

[7] Tivoli Systems. *TME 10 Enterprise Console, User's Guide, Version 3.7*, November 2000.

[8] Wietse Venema. TCP WRAPPER: Network monitoring, access control and booby traps. In *UNIX Security III Symposium*, pages 85–92, Baltmore, MD, September 1992. Usenix.