Security Operation Center Concepts & Implementation

Renaud Bidou
renaud.bidou@iv2-technologies.com

Abstract
A Security Operation Center (SOC) is made up of five distinct modules: event generators, event collectors, message database, analysis engines and reaction management software. The main problem encountered when building a SOC is the integration of all these modules, usually built as autonomous parts, while matching availability, integrity and security of data and their transmission channels. In this paper we will discuss the functional architecture needed to integrate those modules. Chapter one will introduce the concepts behind each module and briefly describe common problems encountered with each of them. In chapter two we will design the global architecture of the SOC. We will then focus on collection & analysis of data generated by sensors in chapters three and four. A short conclusion will describe further research & analysis to be performed in the field of SOC design.

1 SOC Modules

Security Operation Center is a generic term describing part or all of a platform whose purpose is to provide detection and reaction services to security incidents. According to this definition we can distinguish five operations to be performed by a SOC: security event generation, collection, storage, analysis and reaction.

For ease we will start with the revolutionary definition of "boxes" given in [1].
- E Boxes: Events generators
- D Boxes: Events databases
- R Boxes: Events reaction

We will then slightly alter the definition for A Boxes (given as "receive reports & perform analysis") to only "perform analysis", leaving the "collect operation" of data from E Boxes to specific C Boxes.
- A Boxes: Events analysis
- C Boxes: Event collection & Formatting

Another box type will be defined as we will need to manage knowledge of protected platform characteristics, as well as the vulnerability and intrusion signature database.
- K Boxes: Knowledge base

As it can be easily imagined, each box describes a functional group of "modules" performing specific operations. As an example an "E Box" may be a group of applications generating system events through the standard syslog interface of the OS on which they run. . It could also be a pool of Network IDS's. In the preceding examples, modules would be respectively the applications and Network IDS's.

From a macro point of view boxes would operate as described in Figure 1.

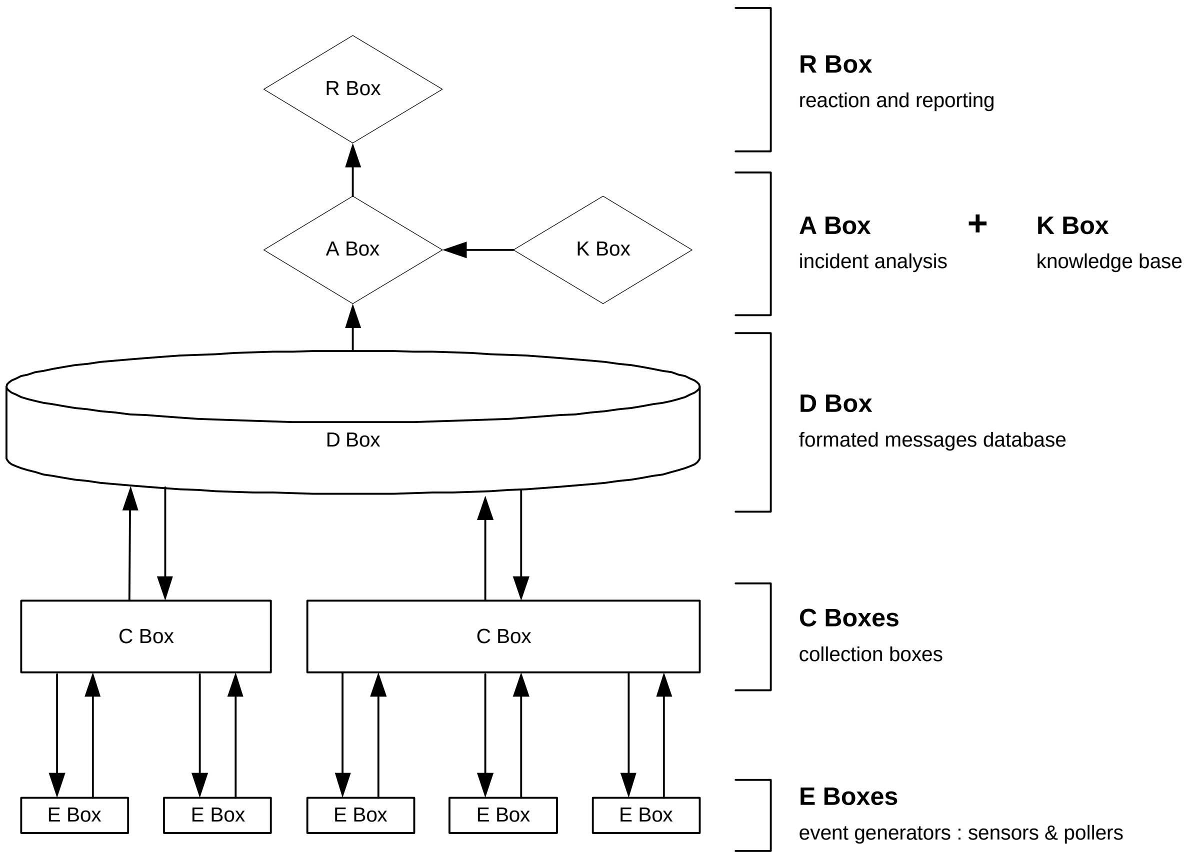

Figure 1: Boxes macro architecture

Beside the obvious problem of data interchange format between modules, each module type has its own limitations which we will describe hereafter:

E Boxes

E Boxes are responsible for event generation. We can distinguish two main families of such Boxes: event based data generators (ie. sensors), which generate events according to a specific operation performed on the OS, applications or over the network, and status based data generators (ie. Pollers), which generate an event according to the reaction to a external stimulus such as ping, data integrity checking or daemon status check.

1.1.1 Sensors

The most well known type of sensors are IDS's, be they host based or network based. We can also add to this category virtually any filtering system (network, application or user based) providing logging, ie. firewalls, routers with ACLs, switches and Wireless HUBs implementing MAC address restriction, RADIUS servers, SNMP stacks, etc. In the extreme, honeypots and network sniffers are also to be considered as sensors. In the latter case, each packet sniffed would generate an event!

Each sensor is to be considered as an autonomous agent running in a hostile environment and, matching characteristics given in [2],: run continually, be fault tolerant, resist subversion, impose a minimal overhead, be configurable & adaptable, be scalable, provides graceful degradation of service and allow dynamic reconfiguration.

However, bypassing and confusing techniques exists for each of them as described in [3] and [4].

What is more, Host Based IDS's are still at an early stage of standardization [5][6] as collection of data from multiple sources (mainly OS calls and application specific logging) and with different level of detail, is highly specific to each developers' sensitivity to security concerns...

1.1.2 Pollers

Pollers are a specific type of event generators. Their function is to generate an event when a specific state is detected on a third-party system. The most simple analogy is to be made with Network Management Systems. In this case, a polling station checks systems status (via ping, snmp for example). If a system appears to be down, the polling station generates an alert to the Management Station.

In a security specific context, pollers will be mainly responsible for checking service status (in order to detect DoS) and data integrity (usually web page content).

The main limitation encountered with pollers is performance, as it may be difficult to setup systems that would be able to poll hundreds of targets at short intervals whilst non-disturbing the managed systems operations. Continuous polling may impact system operations, leading, in the extreme, to CPU (or, in the worst case, network) resource starvation.

1.2 C Boxes and D Boxes

Collection boxes' purpose is to gather information from different sensors and translate them into a standard format, in order to have an homogeneous base of messages.

Once again availability and scalability of these boxes appears to be a major concern. However, such aspects can be managed in a way similar to that used for any server-side service, using clusters, high availability and load balanced dedicated hardware / appliances, etc.

The standard formatting of collected data (the second point described above), appears far more theoretical and still subject to controversy around the security community. The IETF [define?] is working on standards for message formatting as well as transmission protocols [7]. However, unofficial extensions seems already necessary for correlation purposes [8] as well as distributed sensor management [9]...

D Boxes are the more standard modules we find in a SOC architecture. They are databases.

The only SOC specific operation to be performed by this Box type is a basic level of correlation in order to identify and fuse duplicates either from the same or different sources.

Beside classical concerns regarding database availability, integrity and confidentiality, D Boxes will mainly face the problem of performance as sensors may generate dozens of messages each second. Those messages will have to be stored, processed and analyzed as quickly as possible, in order to allow a timely reaction to intrusion attempts or success.

C and D Boxes concepts will be detailed in Chapter 3.

1.3 A Boxes and K Boxes

Those modules are responsible for the analysis of events stored in D Boxes. They are to perform various operations in order to provide qualified alert messages.

This kind of operation is probably the one on which most current researche focuses [10], be it in terms of correlation algorithms, false-positive message detection, mathematical representation [11] or distributed operating [12].

However, the diversity of such research and the early stage of implementations (mostly limited to proof of concept) lead to the design of a module that is the most proprietary and non-standard part of the SOC. We will thus present an approach dealing with the structural analysis of intrusion attempts, as well as behavior analysis i.e. alignment with the security policy.

It is evident that the analysis process needs inputs from a database in which intrusion path characteristics, protected system model and security policy are stored. This is the very purpose of K Boxes.

A and K Boxes concepts will be detailed in chapter 4.

1.4 R Boxes

R Box is a generic term used to define the ensemble of reaction and reporting tools used to react against the offending events taking place on or within the supervised systems.

Experience shows that this is a very subjective concept, as it involves GUI ergonomics, security policy enforcement strategy, legal constraints and contractual SLAs by the supervising team towards the client.

These subjective-lead constraints make it virtually impossible to define anything else than advice and best practice based on real-life experiences over time.   . However, the importance of R Boxes should not be under estimated, as an intrusion attempt may well be perfectly analyzed and qualified but the whole operations would be rendered useless if no appropriate reaction could be launched within the appropriate delay.   The only possible reaction would then be the infamous “post-mortem analysis”...

2 SOC global architecture

The SOC global architecture implements the different box types defined in the preceding chapter. However, beside the pure technical aspects involved in such an implementation, it is necessary to consider the supervision of an IT infrastructure as a full operational project. We will thus follow the functional steps of such a project in order to describe both the purpose and the concepts of selected parts of the architecture described in figure 2.

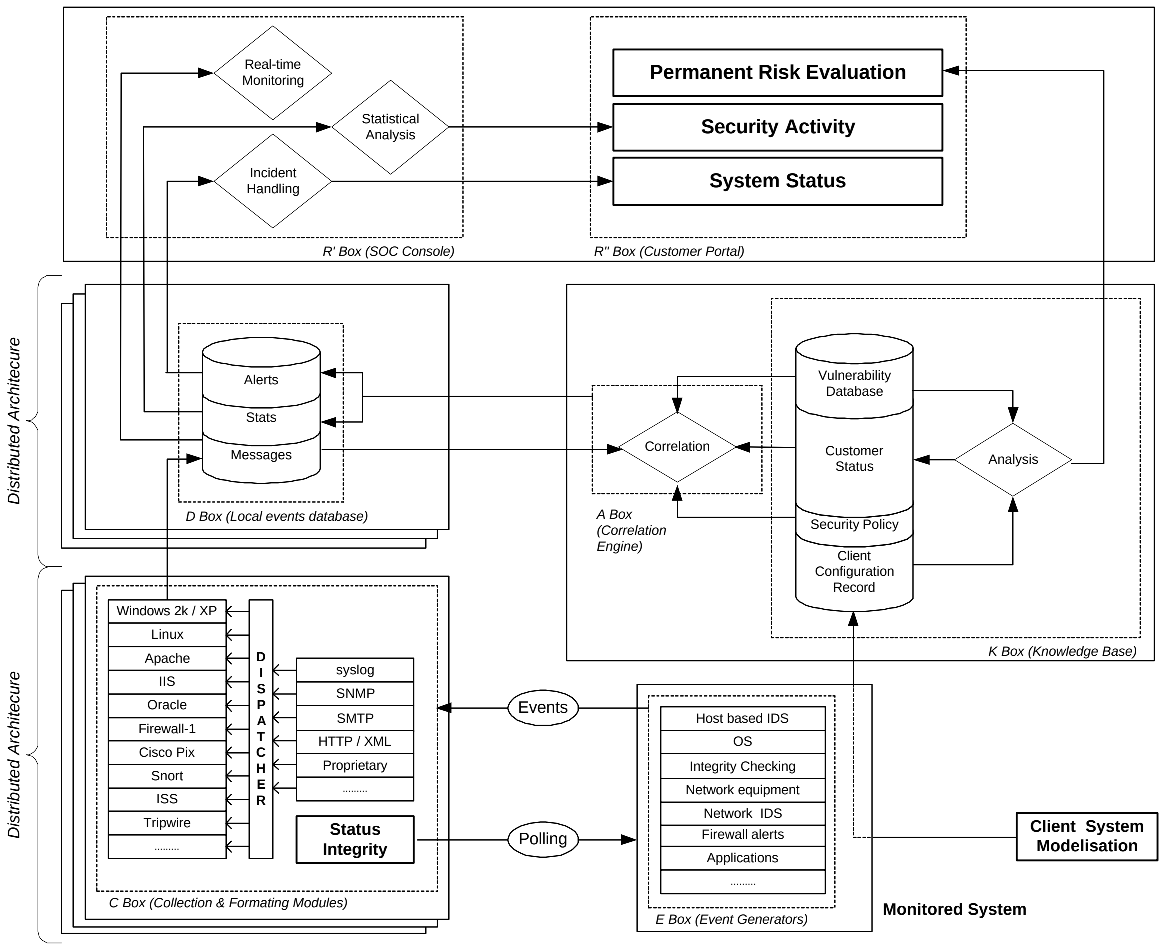

Figure 2: SOC Architecture

2.1 Data acquisition

Before setting up sensors and designing any correlation or analysis rule, it is necessary to evaluate the overall security level of the IT infrastructure to be supervised. This will make it possible to determine if an intrusion path may effectively lead to an intrusion on the target system and the criticity associated to such an intrusion attempt.

Another point to be defined is the security policy, mostly in terms of access rights, permitted operations, etc.

2.1.1 Technical and organizational inventory

Security level evaluation can be divided into two parts: vulnerability assessment and system criticity. This data should be stored in a specific module of the Knowledge Base: the Client Configuration Record (CCR).

Acquisition of this data may be performed in two different ways, the Black Box approach and the White Box approach. The former is a typical output from a blind penetration testing process. Such a process is widely implemented and quickly provides results. However, the latter approach, as described in [13], seems more appropriate to handle exhaustive inventory of supervised systems and provide intrusion path generation.

System criticity is to be defined according to the relative impact that an intrusion can have for each type of consequence. . As such an approach is very subjective, the work must be performed using a standard method for attack taxonomy [14] and classification. However the lack of definitions (be they a list of terms or matrices-based) that meet the six characteristics of an acceptable taxonomy [15], enforces an arbitrary choice. It is then possible to either use a specific definition or, to rely on an external source such as "unauthorized results" defined in [16].

2.1.2 Vulnerability database

The vulnerability database holds information about security breaches and insecure behavior that would either impact the overall security level or that could be exploited by an attacker in order to perform an intrusion. The database format must make it possible to include the following types of vulnerabilities.

- structural vulnerabilities, ie. vulnerabilities internal to a specific software such as a buffer overflow, format string, race conditions, etc. This part of the database is obviously the easiest to implement, feed and maintain. The majority of these processes can be scripted, as information is widely available from public sources such as public mailing lists, software editor advisories and web sites. However a validation and correlation step (if multiple sources are used) should be mandatory and performed by a expert team.

- functional vulnerabilities, depending on configuration, operational behavior, users, etc. These vulnerabilities differ from the previous ones as they deeply depend on the environment in which they live. As an example, an NFS mount should be considered a functional vulnerability given that an intruder can get into an account/host allowing them to mount the file system. Therefore, it will be assumed that many such vulnerabilities are present on systems but may be considered as "inactive" as long as at least one of the needed conditions is not satisfied. The hardest part is the definition / formatting of such vulnerabilities and the feeding of the database. The need for expert teams in each field (OS specific, applications, network, etc.) is obvious.

- topology-based vulnerabilities, including networking impact on intrusions and their consequences. This part of the database includes network-based vulnerabilities (sniffing, spoofing, etc.) as well as the impact of filters on the path to an intrusion. Such vulnerabilities cannot fit into a vulnerability database unless it supports a minimum of topology modeling.

2.1.3 Security policy

The next step of the supervised system inventory is an organizational one and, more specifically, a review of security policy aspects that would impact either event generation and / or the reaction-reporting processes.

It is clear that the two major aspects of security policy that need to be reviewed are authorization and testing / audit procedures. Those two aspects will provide information concerning behavior that sensors would detect. Events generated (administrator login, portscans, etc.) will then be marked as matching with security policy criteria. Others will be analyzed as possible part of an intrusion attempt.

Those pieces of information are stored in the Knowledge Base.

2.1.4 Status evaluation

The last part of the Knowledge Base is a detailed security level evaluation of the systems to be monitored. The objective is to process such an evaluation through an analyzing engine capable of integrating the three kinds of vulnerabilities given in §2.1.2, as well as security policy constraints. The engine should provide a list of vulnerabilities each system is exposed to, the relative impact of each vulnerability and intrusion paths leading to activation of “inactive” vulnerabilities.

In order to be reliable, such an evaluation must be re-generated each time a new vulnerability is found or one of the monitored system is changed.

2.2 Event generation, collection and storage

E Boxes should be setup to generate as much raw information as possible. This information can be sent in “real-time” to C Boxes and / or can be stored locally for future collection by C Boxes, behaving in the same way that an RMON [17] probe does.

2.2.1 Exhaustivity and performances

As described above, a maximum amount of raw information should be made available to C boxes. Indeed discrimination and qualification of events will be made by the correlation engine, which will discard unimportant information. However, this theoretical point of view clearly has limitations in terms of performance. If such an approach can reasonably be implemented for IDS systems, it quickly appears to be unacceptable in the case of OS events as well as those of most applications. Collection of each html access log for a large web farm is one of the most clear examples of such a limitation...

It is thus necessary to pre-filter information at the source, ie. on the E Box. Such a filter will drastically reduce the amount of data collected. However, applying a filter BEFORE generating events means that a first qualification is performed. This qualification may be driven by two factors.

- structural specifications; in this case some event will not be generated as they concern components (hardware, OS, applications, etc.) that are not present on the supervised system. This kind of filter is typically set on IDS’s and firewalls / filtering equipments.

- security policy pre-filters; those filters are set in order not to generate events as they comply with the security policy. As an example “su –“ command could be allowed to a user within a specific time range, or a portscan initiated from a specific IP address, etc.

Pre-filters significantly reduce resources needed by collectors, however they have two main drawbacks.

The first is the difficulty to maintain such distributed filters. Rigorous change management procedures must be put in place in order to make sure that any change on the supervised systems or the security policy will be reported to the relevant pre-filter. What is more, most of those pre-filters are set at the application level, therefore using heterogeneous configuration files and thus increasing the complexity of management.

The second one is the lack of exhaustivity concerning security related events on the systems. If this makes statistics far less reliable, it also may make things harder for post-mortem analysis.

2.2.2 Collection and storage

The main operations performed by collectors are the reception of raw messages through different protocols and source type identification / formatting. Once a message is formatted, it is stored into an event database. Performances and availability issues naturally imply the design of a scalable architecture which allows large distribution of collectors and databases around a network.

Collection and storage will be detailed in chapter 3.

2.3 Data analysis and reporting

2.3.1 Structural and behavior-lead alerts

The main operations performed that generate alerts are the following: correlation, structural analysis, intrusion path analysis and behavior analysis.

Correlation is a stand-alone operation leading to the creation of contexts against which further analysis will be made, in order to check if they match the characteristics of an intrusion attempt.

Structural analysis may be compared to an advanced pattern matching process, used to determine if events stored within a certain context lead to a known intrusion path (or attack tree [18]). Intrusion path analysis is the next step whose output will provide information about the exposure of the target system to the intrusion attempt detected. Then, the behavior analysis will integrate elements from the security policy in order to determine if the intrusion attempt is allowed or not.

The purpose of such operations is to generate alerts that not only match the structural path of intrusion (ie. scan, fingerprinting, exploiting, backdooring & cleaning), but also take care of the security policy defined, as well as criticity of targets systems.

Data analysis will be detailed in chapter 4.

2.3.2 Interfaces

Two kind of interfaces are made available: The SOC console and the End-user portal.

2.3.2.1 SOC Console

The SOC console (R Box) is designed for internal analysis and presents mostly unformatted data from different parts of the K Boxes. The three interfaces are:

- real-time monitoring interfaces, which provide raw data from the messages part of the K box. This allows basic filtering functions such as “egrep” in order to isolate specific messages and is used for debugging, in depth analysis of specific events and replay of events.

- incident handling interface, is the internal engine used for generation and follow-up of incident tickets and reaction procedures described below. It provides qualified alert information as well as numerous debugging data and checkpoints. It is the more complex interface, as it must fit either with operational performance, ergonomics and advanced filters or, research and identification functions. Such an interface is the very corner-stone of a timely and appropriate human reaction to intrusions.

- statistical analysis interface, provides raw data of security activity statistics over short, medium and long term periods. This is mainly used as an under-layer for graphical representation.

2.3.2.2 End-user portal

The end-user portal provides formatted data of activity. It is designed in order to provide multi-level reporting, for targets ranging from security engineers to high-level management through Security Officers. It is divided into three main parts:

- permanent risk evaluation interface, gives information about the current security level of supervised systems configuration and software versions. It provides information on the overall security level, vulnerability characteristics and criticity, intrusion scenarios and patch or configuration details.

- security activity, is a mid-term to long-term reporting, providing macro data about intrusion types, frequency, sources and consequences on the supervised system. At a lower level, it is to be used in order to determine trends and identify specific items such as a recurring attack sources or mostly targeted services to watch for.

- system status, which is the “pseudo real-time” interface for end-user, allowing a close follow-up of open incidents, systems under attack and intrusion paths activated by intruders. It also provides information about the reaction and escalation procedure currently occurring in order to circumscribe the attack.

2.3.3 Reaction and escalation procedures

Eventually, reacting appropriately to an attack is mostly a question of organization and procedures to be applied by the incident response teams [19]. Reaction ranges from passive monitoring for further information through to target system emergency halt through CERT incident reporting [20]. Of course, appropriate reaction should be determined before an attack takes place and procedures must be validated then securely (mainly in terms of integrity) stored and made accessible to supervision teams.

In simple terms, a certain level of escalation must be defined in order to ensure quick and effective reaction, in parallel with the use of appropriate human resources. Escalation procedures are given in figure 3. Another aspect to be specified is the delay, defined as t1 in the figure above, in which the reaction procedure must be launched, according to attack criticity. Once this delay is exhausted, escalation to the next (upper) level should be automatic.

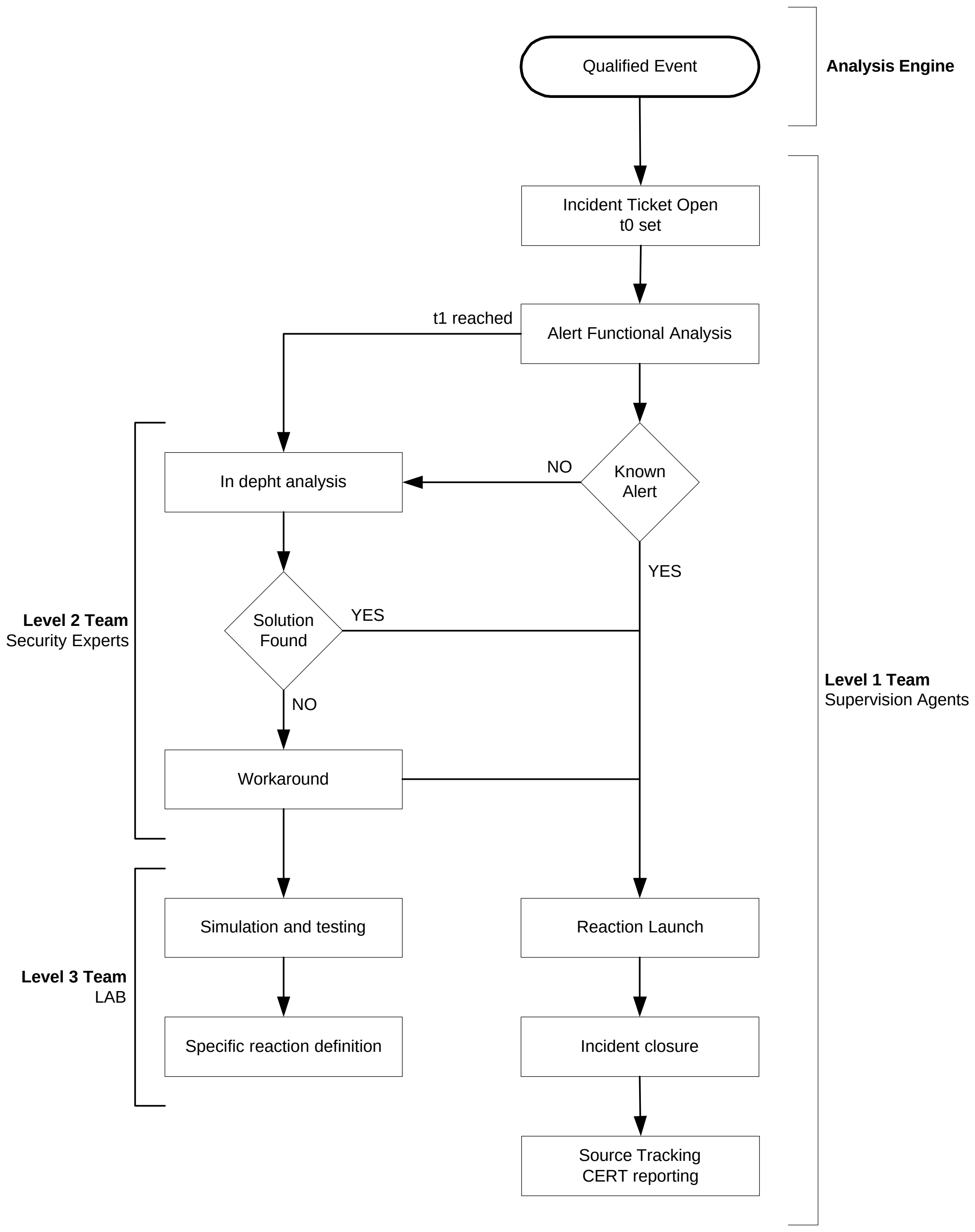

Figure 3: Escalation Procedure

As we can see three escalation levels have been defined:

- the first level should be what we refer to as agents, i.e. mid-technical level staff, which are able to understand events generated by A Boxes as well as the reaction procedure to apply (this is necessary as it is important to be able to know when the application of such a procedure failed). Agents escalate incidents to level two, if the event does not match “known events” or “pre-defined reaction” criteria or if the time limit (t1) is reached depending upon the incident criticity.

- the second level should be a team of technical experts. These experts are responsible for the analysis of intrusion events that have not been defined *a priori*. Their priority is to qualify events with the help of SOC console interfaces (cf. §2.3.2.1) and provide a workaround to be applied by level one agents, pending further research or permanent solutions.

- the third level should be a “laboratory” in which suspicious packets, system operations and so on will be re-played, in order to determine the nature of the unknown intrusion and provide a fully qualified reaction procedure. The lab will also be responsible for contacting vendors of OS, applications, hardware, etc. for patch design and / or their application.

3 Collection and storage

3.1 Data collection

Collecting data from heterogeneous sources implies the setup of two kinds of agents: protocol and application. The former collect information from E Boxes, the latter parses information for storage in a “pseudo-standard” format. Those two modules are connected by a dispatcher. Such an architecture allows high-availability and load-balancing systems to be set at any level into the architecture.

Figure 4 shows some architecture examples, based on the details provided below.

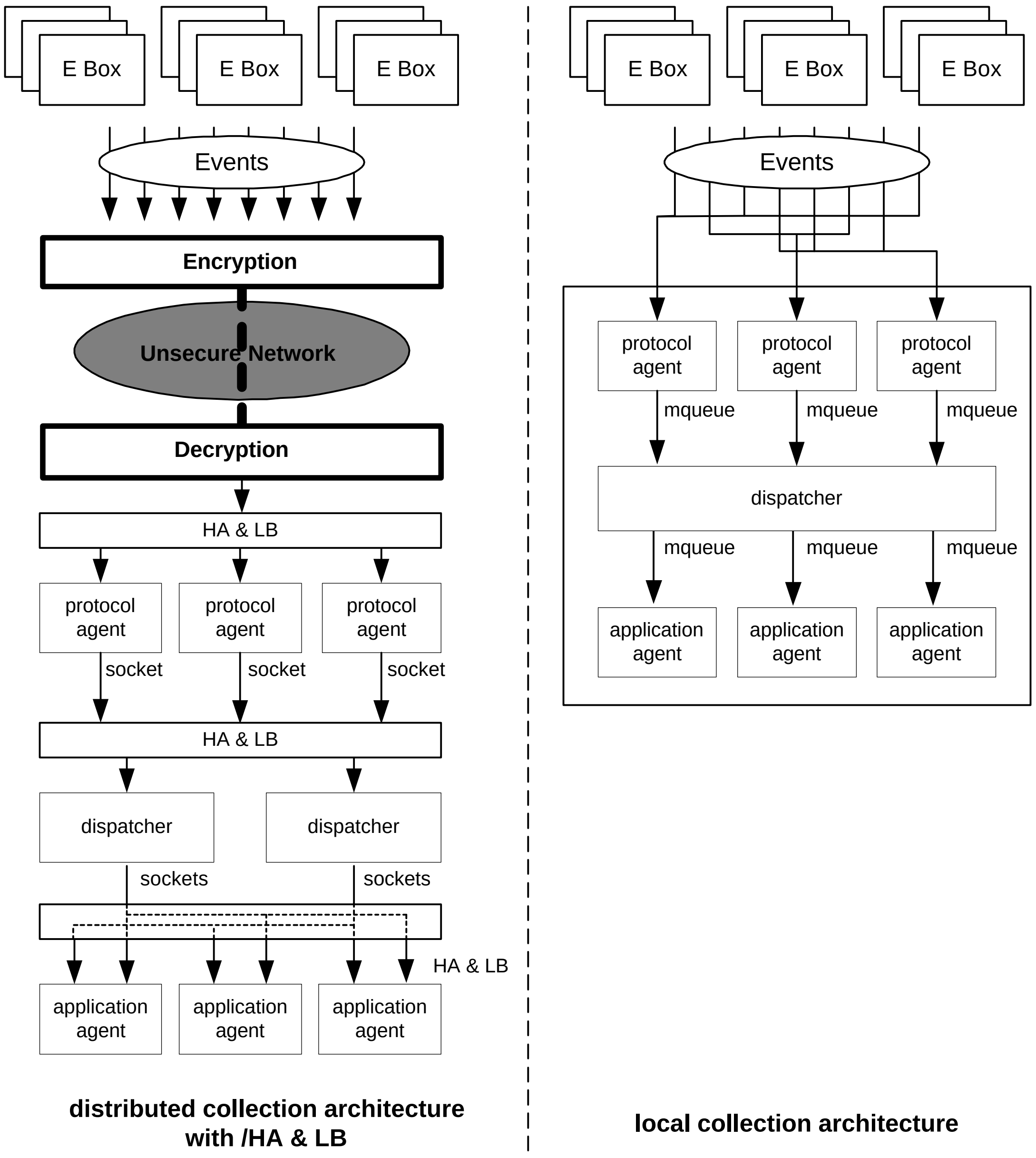

Figure 4: Collection Macro Architecture Examples

3.1.1 Protocol agents

3.1.1.1 Basic functions

Protocol agents are designed to receive information from specific transport protocols, such as syslog, snmp, smtp, html, etc. They act like server side applications and their only purpose is to listen to incoming connections from E Boxes and make collected data available to the dispatcher (cf. §3.1.3).

The simplicity of such agents make them easy to implement and maintain.

The raw format storage is usually a simple file, though direct transfer to the dispatcher through named pipes, sockets or shared memory ensures better performance.

3.1.1.2 Performance and availability

An interesting part of this approach, is the ease with which one can deploy farms of agents, as they are very simple applications that don not share data. It is thus possible, for very large systems to deploy syslog, snmp, smtp, . servers array, etc., which will be served by standard HA & LB equipment. Cluster architecture is also a possibility.

The objective is to provide a scalable and available collection platform, whichever approach is taken.

3.1.1.3 Security

From a security point of view, the most important point is to ensure the integrity of data collected by agents. This is particularly important if data is to be transferred on a shared or un-trusted network.

Obviously, most protocols used to collect information, sit on top of the unreliable UDP layer. It seems therefore necessary to encapsulate such data into a secure tunnel in order to ensure that data will reach the collection agent and that it will not be altered during transport. This last point also concerns data sent over TCP (just like through smtp or http). However, in order to maintain a high performance level and allow better HA and LB operations, it would be intelligent to perform encryption-related operations on dedicated equipment, on each side of the communication.

3.1.2 Dispatcher and application agents

3.1.2.1 The dispatcher

The dispatcher’s purpose is to determine the source-type of an incoming event and then forward the original message to the appropriate application agent. Once again, implementation is relatively trivial, once a specific pattern has been found for each source-type from which the data may be received.

Autonomous operations performed by the dispatcher are the following:

- listen to an incoming channel from protocol agents, such as socket, named pipe, system V message queue, etc.

- check pattern matching against a patterns database that should be pre-loaded in memory for performances considerations. This database contains patterns specific to each couple (E Box type, Xmit protocol), as numerous event generators use different messages formats depending on the transmission protocol.

- send the original message to an E Box specific application agent through any suitable outgoing channel.

3.1.2.2 Application agents

Application agents are specific to each (E Box, Xmit protocol) couple. They perform formatting of messages so that they match with the generic model of the message database.

Autonomous operations performed by application agents are:

- listen to an incoming channel from dispatchers, such as socket, named pipe, system V message queue etc.

- parse the original message into standard fields.

- transmit the formatted message to corresponding D Boxes. Once again any kind of channel can be used, depending on the D Box nature (database, chained pointers, etc.).

3.1.2.3 Dispatchers and application agents merging

, As fully distributed architecture theoretically appears to be the ultimate solution for scalability and high-availability, some implementations may need redundant operations to perform functions of dispatchers and application agents.

As an example, based on a regexp parsing in perl, the following operation would be performed by the dispatcher in order to identify a Snort 1.8.x alert in syslog format:

```
if($line =~ /.*snort: \[\d+:\d+:\d+\] .*) {
      send_to_snort_1.8_syslog_agent($line)
}
```

and the application agent would perform the following operation:

```
if($line =~ /.*\[\d+:\d+:\d+\] (.*) \[Classification: (.*)\] \[Priority:.*\]: 
\{(.*)\} (.*) -> (.*)/) {
            # fill formatted messages fields cf.  §3.2
            $msgtype = "Snort 1.8 – Alert";
            $proto = getprotobyname($3);
            $src = $4;
            $dst = $5;
            $intrusion_type = $intrusion_type[SnortIntrusionType($2)];
            $info = $1;
      }
```

It appears that these two operations overlap and that, in the case of a centralized platform, the following operation would reduce resource usage and provide exactly the same result:

```
if($line =~ /.*snort: \[\d+:\d+:\d+\] (.*) \[Classification: (.*)\]
\[Priority:.*\]: \{(.*)\} (.*) -> (.*)/) {
```

```
            $msgtype = $msgtype[1];
            $proto = getprotobyname($3);
            $src = $4;
            $dst = $5;
            $intrusion_type = $intrusion_type[SnortIntrusionType($2)];
            $info = $1;
      }
```

In some cases, it is then probable that dispatcher and application agents will be merged for the sake of performance and simplicity.

3.2 Data formatting and storage

Two kind of data have to be formatted in a “standard” manner (ie. homogeneous and understandable by any module of the SOC): host entry and collected messages.

3.2.1 Host entry

3.2.1.1 Unique host identification

The need for a standardized host data structure appears as:

- sensors may transmit host information in IP address format or FQDN (Full Qualified Domain Name) format.

- multi-homing techniques provides multiple IP address for the same physical system.

- virtual host techniques provides multiple FQDN for the same physical system.

- HA & LB systems may hide multiple systems behind a single IP address or FQDN.

It appears that identifying a host either by its IP address or its FQDN is not reliable. What is more, in the never-ending need for performance, (reverse) DNS lookup cannot be performed for each new (IP address) FQDN detected in logs.

It is then necessary to rely on a third-party ID, IP address and FQDN independent: the host token.

3.2.1.2 Host entry data structure

The data structure to store host information follows the scheme given in Figure 5.

The function and implementation of each part is trivial and should not need further explanation.

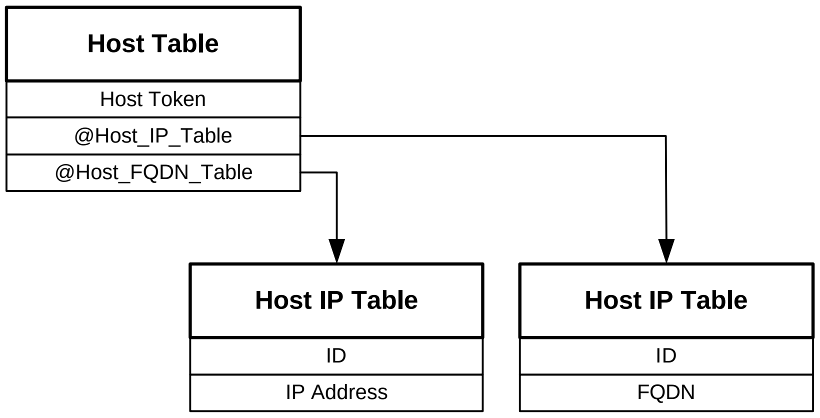

Figure 5: Host Entry Data Structure

3.2.1.3 Data management and maintenance

The best way to handle such data without acceding to a database or browsing a tree of chained structures, is to generate a hash-table holding corresponding host tokens. Such a hash-table should be created in memory at system startup and updated each time a new address or FQDN is identified.

An update of the original database should, of course, be planned in order to save new data. The delay between each synchronization is to be defined according to performances constraints.

3.2.2 Messages

3.2.2.1 Homogeneous messages format structure

Working on data generated by the different types of equipment, transmitted via different protocols requires “standard” formatting. Although an effort has been made to define a worldwide standard with IDMEF [21], it appears that the XML bus used is too heavy and resources consuming, for our purposes of event correlation. However, a separate translation process MUST be implemented for IDMEF compliance.

Structure of a formatted message is the following

Field	Attributes	Description
id	Unique	Unique message ID
sensor_id	Not Null	Unique Sensor ID
msg_type	Not Null	Type of message (ipchains, snort-1.8.x-alert etc.)
epoch_time	Not Null	Date in epoch format of event generation
source		Intrusion Source Host Token
target		Intrusion Target Host Token
proto		Protocol number [22]
src_port		Intrusion source port number
tgt_port		Intrusion target port number
info		Additional info
int_type_id	Not Null	Intrusion type ID (Filter, Access etc.)
int_id		Intrusion ID
message	Not Null	Original message

Table 1: Formatted message structure

3.2.2.2 Third-party structures

As described in Table 1, other data is involved in the creation of formatted messages. Relations between each structure are given in Figure 6.

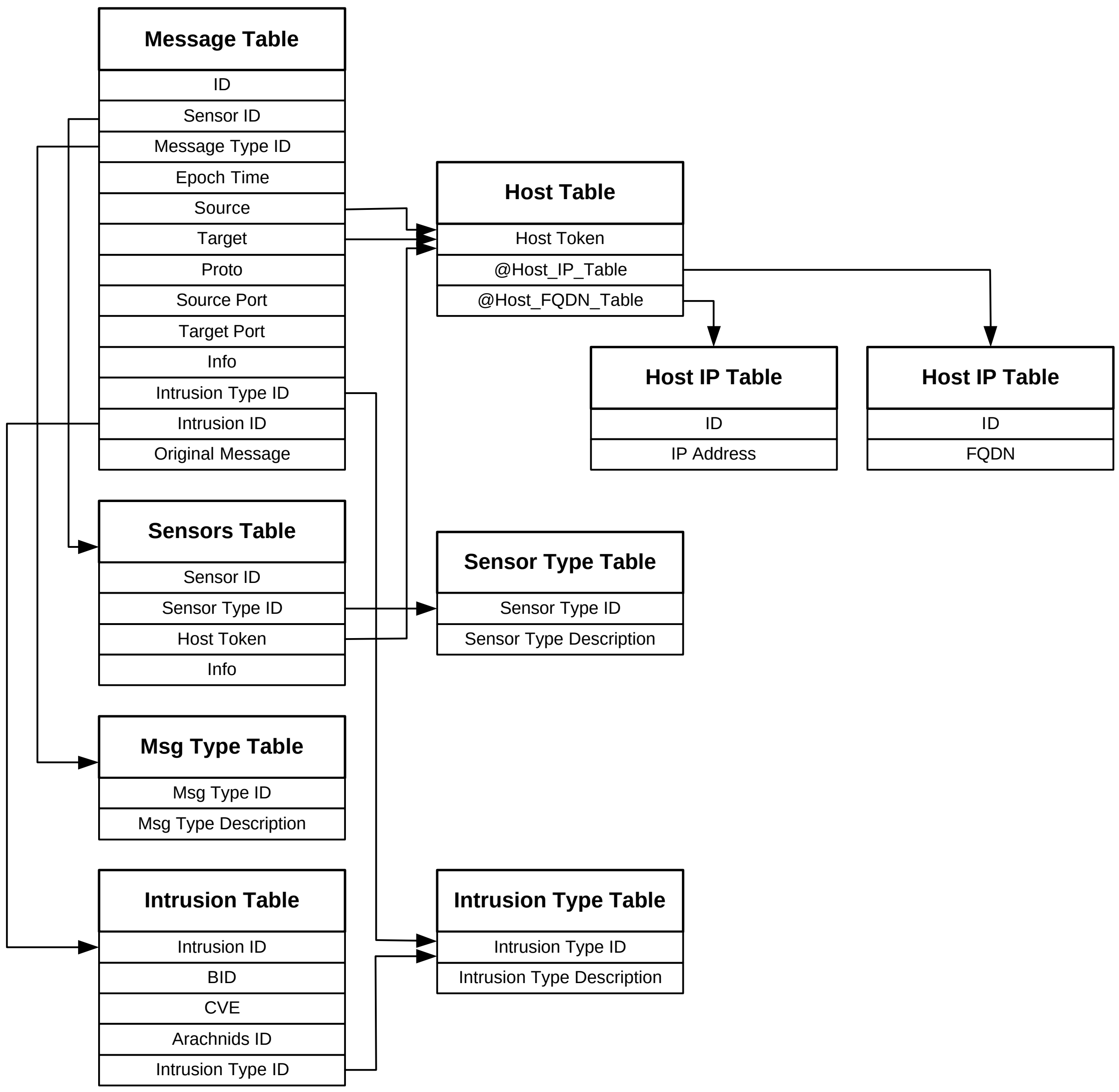

Figure 6: Formatted message definition structures

Apart of the Host Table data structure described in §3.2.1, the tables involved in the building of a formatted message are the following:

- Sensor Table, this table is designed to identify each sensor on the supervised system. This sensor is given a unique ID and a sensor type. Additional optional data, are a host token (if the sensor is IP) and a description.

- Sensor Type Table, this table is only designed to provide a human readable description of each sensor type.

- Message Type Table, this table provides a human readable description of Message Type ID.

- Intrusion Table, the purpose of this table is to provide matches from different references, to a similar attack. As examples, we give BID (Bugtraq ID), CVE ID and Arachnids ID. Others can be integrated.

- Intrusion Type Table, this table defines major intrusion type families such as filter, scan, fingerprinting, exploit, access, etc.

As for the Host Entry Table, most data involved in the building of a formatted message should be loaded in memory at startup and regularly synchronized.

4 Correlation

4.1 Overview

4.1.1 Operating the correlation

The correlation’s purpose is to analyze complex information sequences and produce simple, synthesized and accurate events. In order to generate such qualified events, five operations are to be performed:

- duplicates identification, the first, obvious, operation is to identify duplicates and set a specific flag in order to keep the information and continue without the need keep multiple identical messages.

- sequence patterns matching, is the most common operation performed by a correlation engine. Its purpose is to identify a sequence of messages which would be characteristic of an intrusion attempt. It makes it possible to identify on-going intrusion processes, as well as complex intrusion scenarios.

- time pattern matching, is designed to include another important dimension in intrusion analysis: time. This is mainly used for context (see below) management, as well as slow and distributed intrusion processes.

- system exposure and criticity analysis, provides information about the target system’s vulnerability to detected intrusion attempts. Indeed, it seems inappropriate to have a SOC generating alarms concerning an intrusion scenario based on a vulnerability that the target system is not exposed to. Another piece of information is the criticity of the intrusion i.e. its overall impact on the supervised system. This helps to manage the priorities in terms of reaction to multiple incidents.

- security policy matching, is a behavior-based filter that eliminates specific events if they match security policy criteria such as administrator login, identification processes and authorizations / restrictions.

A global overview of correlation operations is given in figure 7 below.

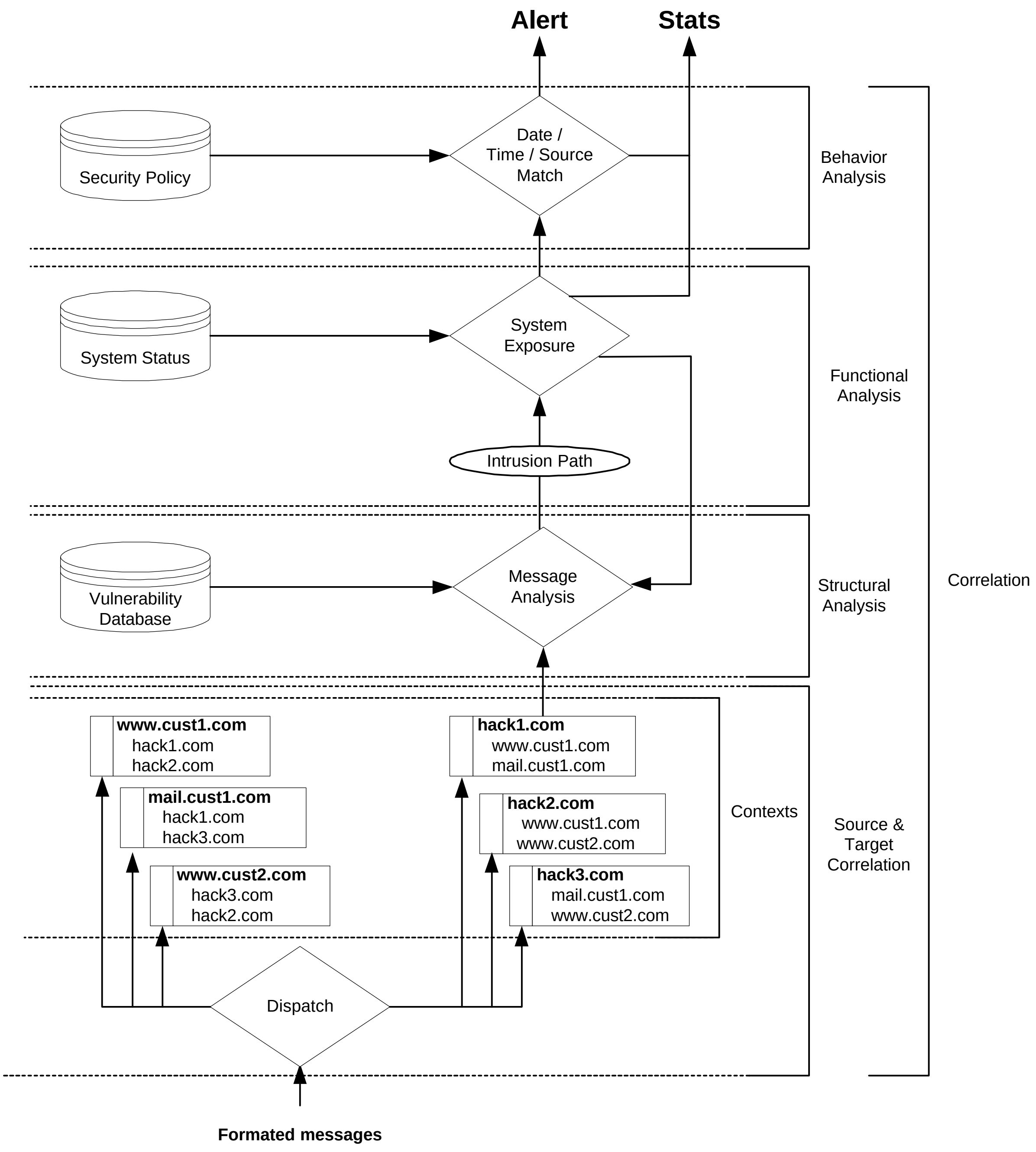

Figure 7: Main correlation operations

4.1.2 Introduction to contexts

The analysis defined above is based upon a specific structure called contexts. All correlation operations are performed against these structures. In simple terms, the definition of a context is the following: a container of formatted data matching a common criteria.

Therefore, any message stored in the formatted message database is to be part of one or more contexts. Correlation operations will be done in parallel so that they can be run simultaneously on each context.

Two kinds of context management approach can be implemented:

The first one is to define independent and distinct contexts. Each context will contain messages matching every criteria. We define such an architecture as an array of contexts.

The second approach is a hierarchical one. Top level contexts matching a limited number of criteria are defined. Then sub-contexts, based on different criteria, are created and so on. This will be defined hereafter as context tree.

As is to be expected, none of the preceding approaches meet all needs, be they in terms of performance or functionality. A mixed architecture will thus have to be defined.

4.2 Contexts

4.2.1 Context definition criteria

Defining context criteria must be done according to security related events that the SOC must react to, be they distributed scanning operations, finger-printing, massive exploit testing, account brute forcing, spamming and so on.

A full functional architecture of contexts is given in figure 8 (§4.2.2.1).

4.2.1.2 Source and target

The first obvious criteria is the attacking and attacked host’s ID, which has to be standardized (cf. §3.2.1).

- source, defining source as a context creation criteria allows sweeps detection, identification of systems used as attack relays or compromised by worms and targeted hack-proofing.

- target, contexts created by target criteria will provide information on scans (be they distributed, slow or “normal”) and, should it even be noticed, intrusion attempts and system compromises.

Two arrays of context should then be defined, one with context matching sources, another matching targets. Each context of each array should then be considered as a top-level context for the context trees. The criteria to be matched by the smallest “branches” would be target ID (for contexts created by the source ID match) or source ID (for contexts created by the source ID match).

4.2.1.3 Protocol and ports

Whilst working with trivial data, the protocols and the ports of the targeted systems should form the criteria for the next level of context ”branches”. This is mainly done in order to isolate single scanning operations from heavy repetitive attempts to compromise a system through a specific application. What is more, it helps to identify the various steps of an intrusion.

Indeed one of the most common intrusion scenario, is a wide portscan sweep followed by fingerprinting / version identification on open ports followed by an exploit launched on systems believed to be vulnerable.

4.2.1.4 Intrusion type

In order to identify which type of message is stored, thus starting a more accurate analysis of messages, a next level of context generation is performed according to the intrusion type ID (defined in §3.2.2.2). An example of intrusion type ID definition is given in table 2 below.

Intrusion Type ID	**Description**
0 / Unknown	Unknown intrusion type
1xx - Identification	**Target identification**
100 / Filtered	Packets filtered by firewalls, ACLs etc.
110 / Scan	Basic portscan detection
120 / Fingerprinting	Target identification
2xx - Exploits	**Bulk intrusion attempts**
200 / Exploit	Exploit launch detection
3xx – Denial of Service	**Successful DoS attacks**
300 / Denial of Service	Partial DoS attack
310 / Denial of Service	Global DoS attack
4xx – Security Bypass	**Security policy bypass attempts**
400 / Spoofing	IP / MAC spoofing
410 / Content	Content filtering bypass
420 / Privileges	Privilege gaining attempts
5xx – System compromise	**Attempts to compromise the target system**
510 / Account Success	Account access
520 / Data Failure	Access to private data attempts
530 / Integrity	System integrity compromise

Table 2: Intrusion type ID

4.2.1.5 Intrusion ID

The last “branch” of contexts contains specific intrusion ID, i.e. the characterization of each message. At this level we reach the atomic (except for duplicates – cf. §4.2.2.2) dimension of each message. This field refers to the Intrusion Table (cf. §3.2.2.2) and will be responsible for the link between the correlation engine and system status information stored in the Knowledge Base (cf. figure 2 and §2.1.1).

4.2.2 Contexts structure

As any correlation operation is exclusively performed on contexts, it appears that their structure is probably one of the most important aspects of the SOC.

4.2.2.1 Functional architecture

The functional architecture has been described in the preceding chapter. It is made up of an array of context trees. Each tree contains 4 level of branches, as described in figure 8.

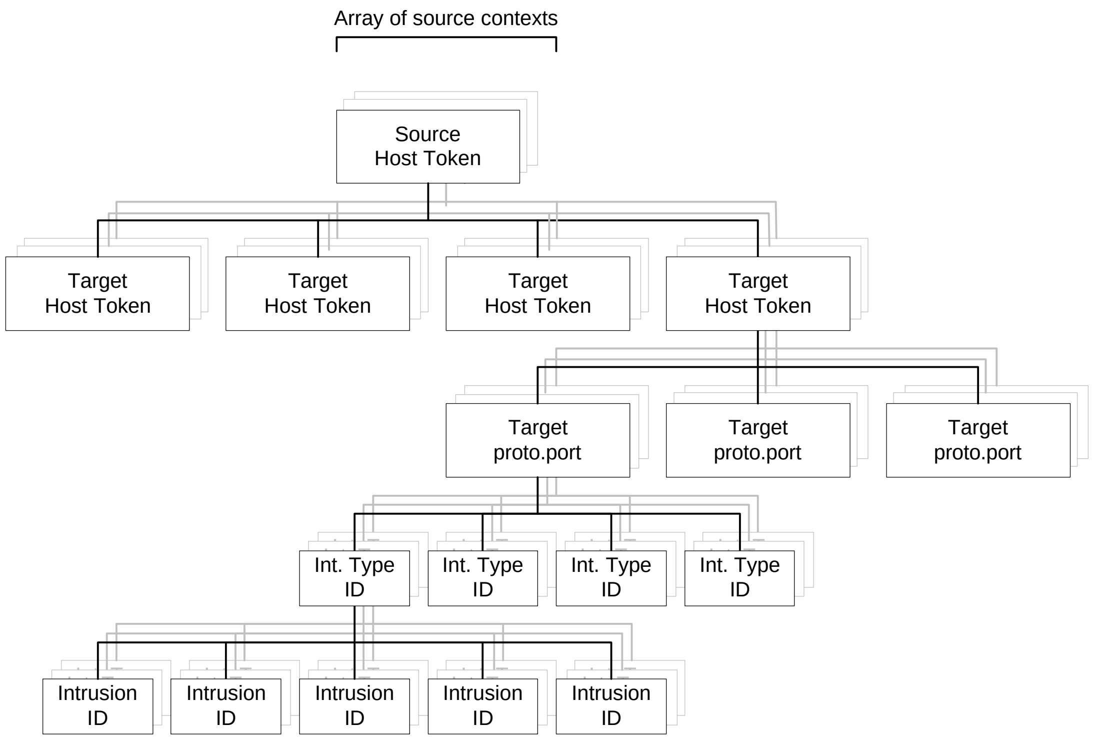

Figure 8: Contexts functional architecture

4.2.2.2 Data structure

In order to handle the architecture defined above, we need to implement a structure that will ensure proper storage and access to information.

Figure 9 describes a scheme for context implementation. As PERL natively implements arrays and hash-tables we will use the PERL notation. However, this is not necessarily a recommended implementation.

In this implementation example the following fields are defined.

- start_time and stop_time, these fields are found in each branch of the context structure. It provides information on time the first and last messages concerning a given branch (and dependent sub-branches) were generated.

- duplicate, gives the number of duplicate messages. Duplicate messages contain exactly the same information from one message to another except for the time information.

Other fields are self-explanatory or have been explained already.

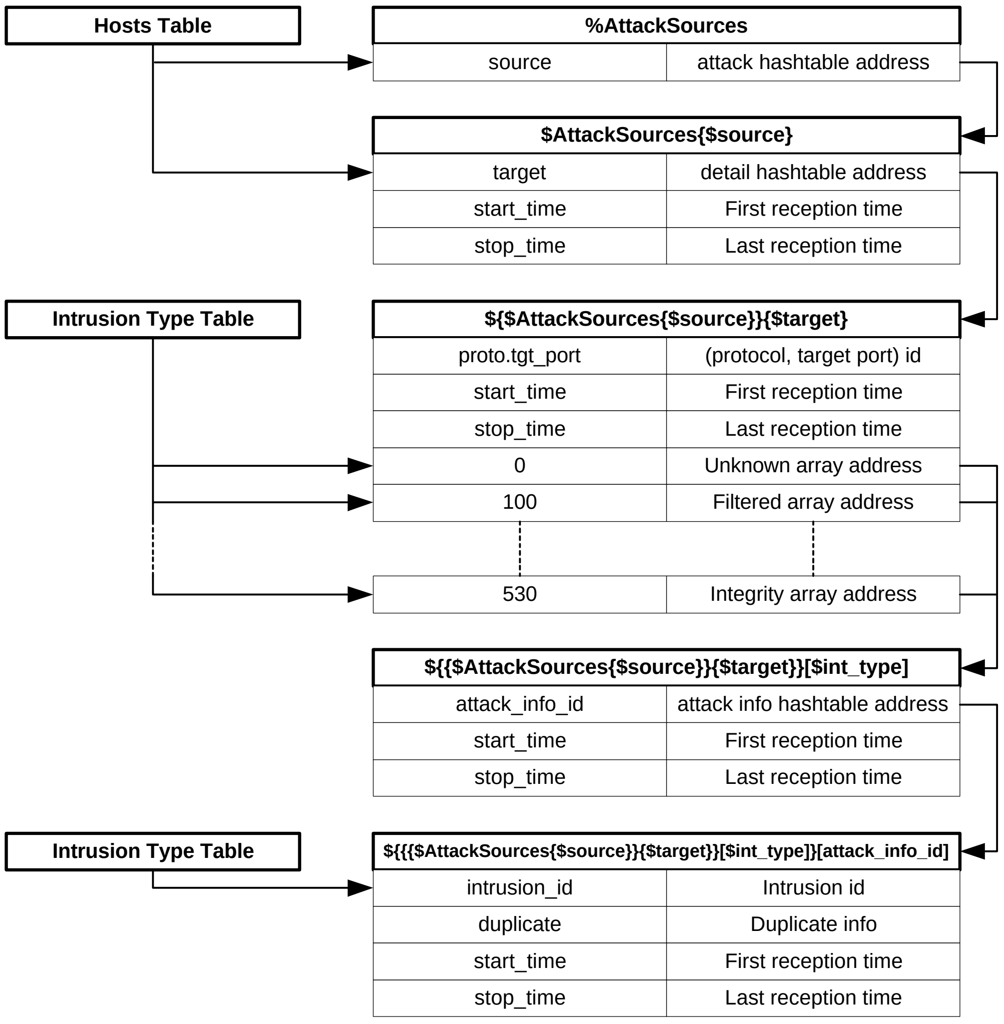

Figure 9: Contexts implementation scheme

4.2.3 Contexts status

Another important characteristic of context is its status. We define three distinct statusesas detailed below:

- active, context matches specific criteria (usually based on time but could be any other criteria), which could be characteristic of an on-going intrusion process. Typically, such a context appears to be under a heavy load from the arrival of new message and its analysis by the correlation engine should be set to the highest priority possible.

- inactive, such a context either does not match "active" criteria or did not receive a specific closure code. This means that it is no longer analyzed by the correlation engine, but that it can be reactivated by the next message matching the same context criteria.

- closed, in this state a context is completed. Any new message matching this context criteria will create a new context.

Context status management is summarized in figure 10.

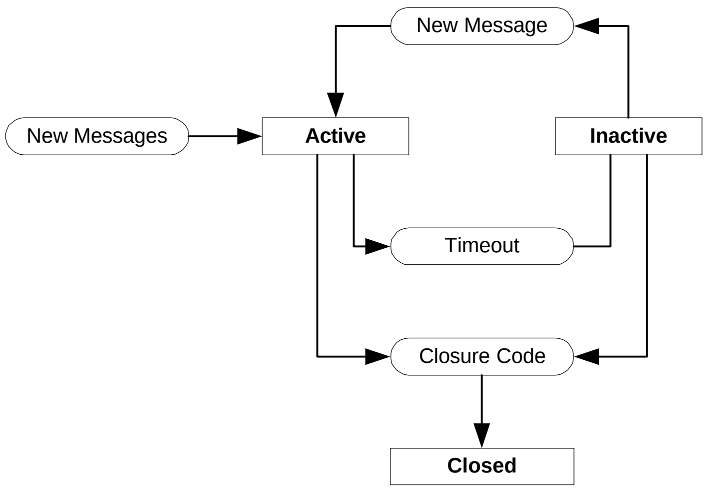

Figure 10: Context status management

4.3 Analysis

4.3.1 Structural analysis

The purpose of structural analysis is to identify on going intrusion attempts, manage context inactivity status and context closure conditions. In simple terms, structural analysis is a set of operations performed by independent modules on each context. Each module is activated by a specific message and performs analysis using a "standard" semantic.

4.3.1.1 Analysis modules structure

The output of the analysis modules is the result of several logical operations between autonomous conditions against fields of contexts. Figure 11 describes members of such operations.

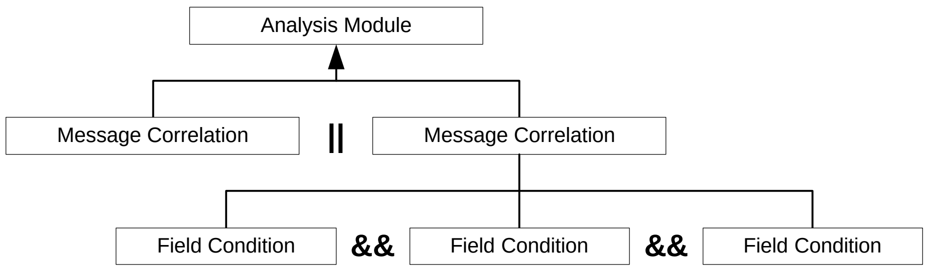

Figure 11: Analysis module structure

Field conditions have the following structure:

```
field operator <field | value> [!]
```

It appears that the power of structural analysis relies on the number of operators made available. However, the very structure of contexts provides embedded operations such as source, target, port correlation. This not only increases the number of “native” operators but also improves significantly the performances of structural analysis.

The ! sign indicates that the field condition is to be matched in order to activate the module (cf. below)

4.3.1.2 Analysis module activation
Two kind of events can activate analysis modules: messages and time.

- messages, as described above, some field conditions must be matched in order to activate an analysis module. A header containing field conditions to be met, is then generated for each analysis module. Given the structure of analysis module, it appears that the header will be a set of logical OR operations, whose members will be field conditions that require the least amount of resources to be evaluated.

- time, the analysis module header may also contain timer information forcing the correlation to be evaluated. This is mainly used for context closure and time dependent intrusions detection such as (slow) portscans, brute forcing, etc.

4.3.2 Advanced correlation
Advanced correlation operations are performed in order to define the criticity of an intrusion attempt and evaluate if such an intrusion attempt is permitted according to the security policy.

4.3.2.1 Functional analysis
This second correlation step is performed in order to evaluate system exposure to the intrusion and the overall impact of such an intrusion on the supervised system.

Once the structural analysis has provided information about an occurring intrusion attempt, a request is made to Customer Status part of the K Box. This request contains the Intrusion ID and the Host token of the target. The response provides the following pieces of information:

- criticity, is a value from an arbitrary scale, typically info-warning-minor-major-critical based.

- closure code, if the context is to be closed, usually because the target is not impacted by the intrusion attempt.

- message, a new formatted message to be appended to the actual context, that may activate additional analysis modules.

4.3.2.2 Behavior analysis
The purpose of this last analysis is to define if the attempts match the security policy. This is mainly used to manage access to accounts but can also be implemented in the case of pre-programmed audits, portscans, etc.. In such a situation a closure code is sent to the context.

Technically, this analysis is performed in exactly the same way as structural analysis i.e. via specific modules whose structure is loaded from the security policy part of the K Box.

5 Conclusion

The complexity of SOC setup is more a question of the integration than the implementation of individual modules. The emerging standards should provide help in reducing gaps between theoretical approaches, proprietary implementations and stand-alone systems.

In the meantime, intrusions are clearly taking place and there is thus a need for operational supervision systems today. Experience shows that a pragmatic approach needs to be taken in order to implement a professional SOC that can provide reliable results. The theory described above forms the framework for deploying such a SOC.

Glossary

A Box:	Events analysis module
C Box:	Event collection & Formatting module
CCR:	Client Configuration Record
D Box:	Events databases module
E Box:	Events generators module
FQDN:	Full Qualified Domain Name
HA:	High-Availability
K Box:	Knowledge base
LB:	Load-Balancing
R Box:	Events reaction module
SOC:	Security Operation Center

Bibliography

[1] Network Intrusion Detection – An Analyst's Handbook – Second Edition – Stephen Northcutt, Judy Novak – New Riders – ISBN: 0-7357-1008-2

[2] Intrusion detection using autonomous agents – Eugene H. Spafford, Diego Zamboni – Computer Networks 34 (2000) 547-570

[3] Bypassing Intrusion Detection Systems – Ron Gula - Network Security Wizards

[4] Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection - Tom Ptacek, Timothy Newsham

[5] Application-Integrated Data Collection for Security Monitoring – Magnus Almgren, Ulf Lindqvist – SRI International

[6] Interfacing Trusted Applications with Intrusion Detection Systems – Marc Welz, Andrew Hutchison – University of Cape Town

[7] IETF Intrusion Detection Working Group – M. Erlinger, S. Staniford-Chen, et al.

[8] Probabilistic Alert Corelation – Alfonso Valdes and Keith Skinner – SRI International

[9] Designing a Web of High-Configurable Intrusion Detection Sensors – Giovanni Vigna, Richard A. Kemmerer and Per Blix – Reliable Software Group – University of California Santa Barbara

[10] Recent Advances in Intrusion Detection - W. Lee (Editor), L. Me (Editor), A. Wespi (Editor) - ISBN: 3-5404-2702-3

[11] A Visual Mathematical Model for Intrusion Detection - Greg Vert, Deborah A. Frincke, Jesse C. McConnel – Center for Secure and Dependable Software – University of Idaho

[12] A Framework for Cooperative Intrusion Detection – Deborah Frincke, Don Tobin, Jesse McConnel, Jamie Marconi, Dean Polla - Center for Secure and Dependable Software – University of Idaho

[13]Modeling and Simulation of Intrusion Risk – Renaud Bidou – Intexxia

[14]The IEEE Standard Dictionary of Electrical and Electrics Terms – John Radatz - IEEE

[15]Fundamentals of Computer Security Technology – Edward G. Amoroso

[16]A Common Language for Computer Security Incidents – John D. Howard, Thomas A. Longstaff – Sandia Report – SAND98-8667

[17]Remote Network Monitoring Management Information Base – IETF RFC 1757 - S. Waldbusser - Carnegie Mellon University

[18]Attack trees - Bruce Schneier – Counterpane Labs

[19]Survivable Networks Systems: An Emergency Discipline – R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. Longstaff, N. R. Mead – CERT Technical Report - Carnegie Mellon University

[20]Incident Reporting Guidelines – CERT Coordination Center - Carnegie Mellon University

[21]Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition – D. Curry, H. Debar – IETF Intrusion Detection Working Group

[22]Assigned Numbers - RFC 1700 – J. Reynolds, J. Postel – Internet Working Group

[23]Analysis Techniques for Detecting Coordinated Attack and Probes – John Green, David Marchette, Stephen Northcutt, Bill Ralph

[24]A Pattern Matching Based Filter for Audit Reduction and Fast Detection of Potential Intrusions – Josué Kuri, Gonzalo Navarro, Ludovic Mé, Laurent Heye