Intelligent Correlator for NIDS

GIAC Security Essentials Practical Assignment
Version 1.4b
Option 2
by

Marco Bove

¨

Abstract

In today NIDS the number of alerts may be huge and the delay in between an alert is generated and the system administrator analyzes it, can be too long and the situation can be changed, e.g. with dual boot Unix-Windows machines. Therefore we would like to give a low priority or to filter out not relevant alerts. We would like also to gather more information about the target of the attack at the time the attack has been performed. The goal of this work is the realization of a prototype of a system that reduces the number of false positives of a NIDS by triggering a real time collects for information upon alert reception.

Index

1 Introduction

In today NIDS the number of alerts may be huge and the delay in between an alert is generated and the system administrator analyzes it, can be too long and the situation can be changed, e.g. with dual boot Unix-Windows machines. Therefore we would like to give a low priority or to filter out not relevant alerts. We would like also to gather more information about the target of the attack at the time the attack has been performed.

The goal of this work is the realization of a prototype available for future enhancement that reduces the number of false positives of a NIDS by triggering in real time a collection for information upon alert reception. More technically the NIDS is based on Snort: the generation of a Snort alert triggers a “real time” gathering of information, such as a Nessus scan, to see if the target of the seen attack is sensitive to the vulnerability as defined in the Snort alert. A correlation with other recent event, concerning e.g. the same target or the same vulnerability, may also be performed. Depending on the results of the scan and the correlation, the alert will be given a certain priority, e.g. a *red alert* priority in case the machine is sensible to the attack at the time of the attack itself or the same machine has been the target of many kinds of attack in a reasonable short time. Optionally an action can be triggered after a *red alert* has been detected.

The final result of this work, as previously said, is a prototype. That means that it future enhancement and development are required to make it productive. It has the only purpose to show how a system based on this kind of implementation may improve intrusion detection eliminating false positives and gathering information on the network at the time the attack has been performed

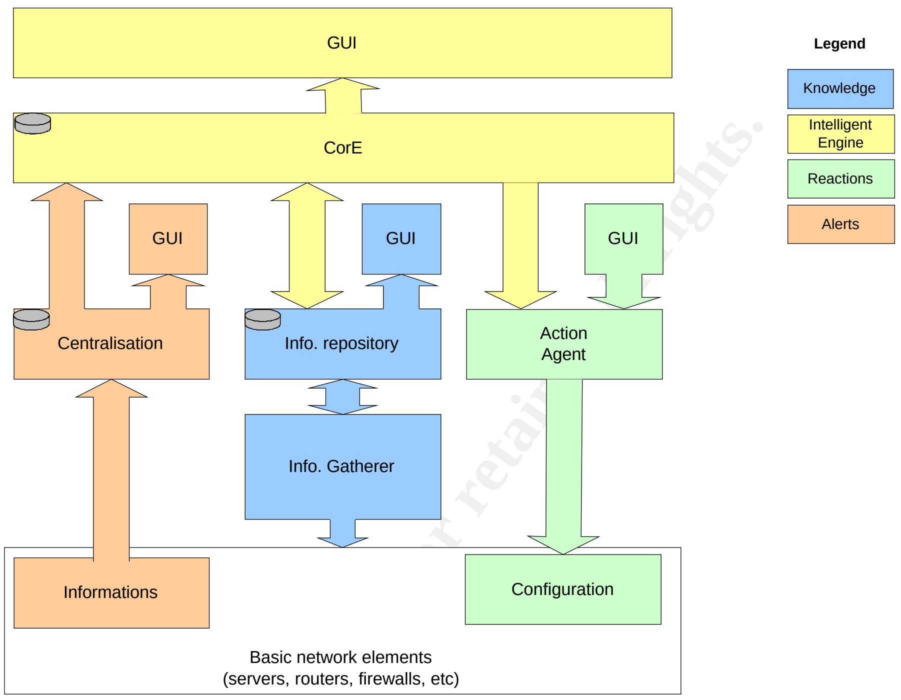
1.1 Security Note

For security purpose in the pictures as well as in the body text all real IP addresses have been substituted by imaginary IP addresses e.g. 0.0.0.1; in order to distinguish machines among imaginary IP addresses, e.g. 0.0.0.1 for machine A, 0.0.0.2 for machine B etc. . Also the real names of the machines have been wiped.

2 Basic Elements

In this section we will describe the elements used in the implementation of the prototype. In Figure 1 we can see an overview of the whole architecture.

Figure 1. Architecture Overview



It is possible to distinguish four main blocks:

1. The **Alerts** block: this block deals with the intrusion detection. When a suspicious event happens and is detected, then an alert is generated and sent to the Centralization database (Snort database).
2. The **Knowledge** block: this block has the task to provide to the Correlation Engine information on demand about the underlying network.
3. The **Reaction** block: this block has the task to put in place countermeasures after a *red alert* has been generated and a reaction has been decided.
4. The **CorE**: this block as well as the Knowledge block is the real target of this work. It analyzes incoming alerts, it asks the Info Repository for fresh information, it evaluates the severity of the alerts and finally it decides for eventual countermeasures. A GUI will show to the system administrator the results.

The goal of this work is to build a prototype for the CorE and the Knowledge block based on an existing Alerts block.

Now follows a description of the tools used in the development of the prototype. This description aims to give a short introduction about the tools used for this work to readers who are not familiar with them.

2.1 Alerts block

The alerts block has the task to log events in the network. In our system, alerts are generated by many tools: Snort, Samhain, Syslog, Tripwire, but we will work only on a subset of Snort alerts and more precisely alerts based on the CVE system.

2.1.1 Snort

Accordingly to [1] Snort "is an open source network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more. Snort uses a flexible rules language to describe traffic that it should collect or pass, as well as a detection engine that utilizes a modular plugin architecture."

The vulnerabilities detected by Snort refer to well-known lists of vulnerabilities such as CVE, ArachNIDS, Bugtraq that provide more additional information about a given alert identified by a tag.

2.1.1.1 CVE, arachNIDS, Bugtraq

CVE, arachNIDS and Bugtraq have the common purpose to unify under a single identification well know vulnerabilities.

CVE (Common Vulnerabilities & Exposures): accordingly to [2] CVE "is a list or dictionary that provides common names for publicly known information security vulnerabilities and exposures". It is possible to find two different types of classification in the CVE list of vulnerabilities: vulnerabilities identified by the *CVE* tag and the vulnerabilities identified by the *CAN* tag. The *CVE* tag identifies entries in the list that are effectively vulnerabilities, while *CAN* entries accordingly to [3] represents "those vulnerabilities or exposures under consideration for acceptance into CVE".

arachNIDS (advanced reference archive of current heuristics for network intrusion detection systems): accordingly to [4] arachNIDS is a "comprehensive database of network attack signature information that can dynamically create and export signature strings compatible with IDS software such as Snort". arachNIDS is fully compatible with CVE.

Bugtraq: Bugtraq is a security mailing list. More precisely, accordingly to [5] "BugTraq is a full disclosure moderated mailing list for the *detailed* discussion and announcement of computer security vulnerabilities: what they are, how to exploit them, and how to fix them".

As already stated we are using only alerts that reference CVE vulnerabilities (both CVE and CAN) because of ease of implementation of the prototype. CVE tags are easy identified in the structure of an alert in the Snort database as well as in the description of the plugin used to Nessus to collect fresh information.

2.2 Knowledge block

The Knoledge block relies on network discovery tools such as Nessus and Nmap. Also if these are the only tools used by the prototype other tools such as NetMap have been tested. Nmap is not used directly, but through Nessus (see 3.2.1).

2.2.1 Nessus

Nessus is the security scanner used to get fresh information about the target of a hypothetical attack at the same time the attack has been performed. Accordingly to [6] Nessus is "a security scanner, i.e. a software which will audit remotely a given network and determine whether bad guys (aka 'crackers') may break into it, or misuse it in some way". Nessus uses different type of portscanner such as Nmap. Nmap is also the portscanner that we will use configuring Nessus for an OS fingerprint of the target.

2.2.1.1 Nmap

Accordingly to [7] Nmap "is a utility for network exploration or security auditing. It supports ping scanning (determine which hosts are up), many port scanning techniques (determine what services the hosts are offering), and TCP/IP fingerprinting (remote host operating system identification)".

2.2.2 Netmap

Accordingly to [8] Netmap "is a tool for discovering and analyzing computer networks. NetMap includes a discovery tool, a network database and a viewer and analyzer application. Network discovery is done by using existing tools, both commercially available and in the freeware domain. Given a query, the set of tools that best answers it is automatically chosen. The tools are runned, and the results are fused together."

Also if it has not been included in the prototype Netmap has been tested. It revealed to be a powerful tool. It provides an SQL database to store information about the network topology and a graphical front end GUI to the database. On the other hand it also revealed to be quite slow and for this reason not adequate for this work.

3 Implementation

This section describes the implementation of the prototype.

3.1 The CorE (Correlation Engine)

The functionalities of the Correlation Engine (CorE) are multiple:

- Check the existence of new alerts;
- Analyze the alerts in order to define the required information needed to process the alert;
- Collect the information from the Info Repository;
- Compute the severity of the alerts and store it in the CorE database;
- Optionally take countermeasures against the attack referred by the alert.

The structure of the CorE is depicted in Figure 2.

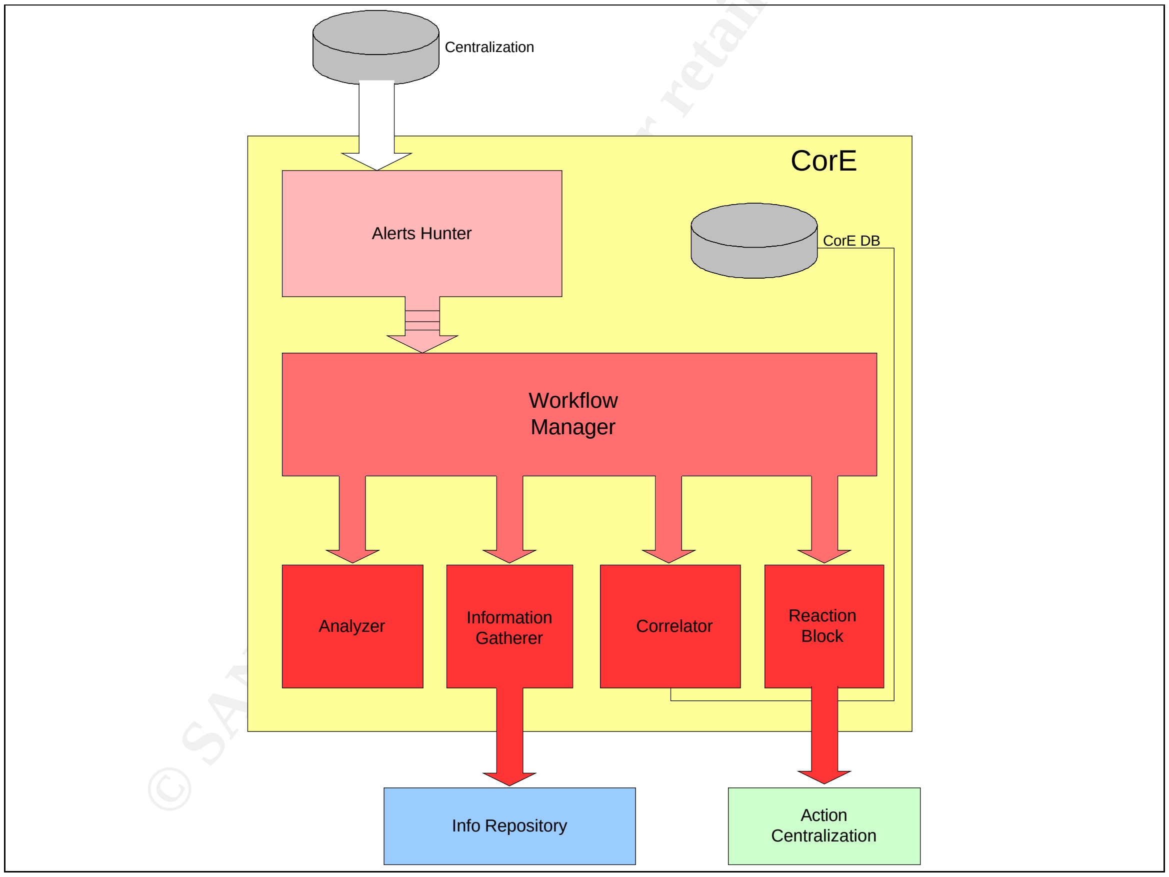


Figure 2. The Correlator Engine (CorE)

In this section we analyze in detail the tasks and the implementation details of each block.

3.1.1 Implementation Details: Alerts Hunter

The Alerts Hunter has the task to check the existence of new alerts from the Snort database.

Originally the prototype was thought to run on top of a Prelude Manager, using the Prelude XML output as input for the CorE in order to process the alerts almost real time. As the insertion of Prelude in our architecture caused many problems (i.e. failure in the installation and configuration processes) the installation of Prelude itself has been postponed. As a consequence also the concept of the Alerts Hunter has been redesigned. At the moment the Alerts Hunter is searching the Snort database each minute for new alerts. This has been considered a valid delay based on the frequency alerts arrive in the Snort database at the time of the implementation, but must be based on the network that is being analyzed. This is still a temporary solution.

In this temporary implementation the Alerts Hunter searches the Snort database each minute. It pulls out all the alerts that have been generated since the previous pull operation and sends them one by one to the Workflow Manager starting from the oldest one.

Till now the prototype has been tested only in the lab, not on a real network. A problem rises in condition of overloading. When the Workflow manager is not fast enough to treat all the alerts before the following pull operation made by the Alerts Hunter. Since we are implementing a prototype, this problem can be postponed. First, with the use of a Prelude Manager, alerts will not arrive anymore in burst and secondly, the alerts may be treated in parallel by multiple Workflow Manager.

3.1.2 Implementation Details: Workflow Manager

The Workflow manager has the task to synchronize the modules of the CorE. Its structure is represented in Figure 3 with a flow diagram that explains the sequence of the tasks inside the CorE itself.

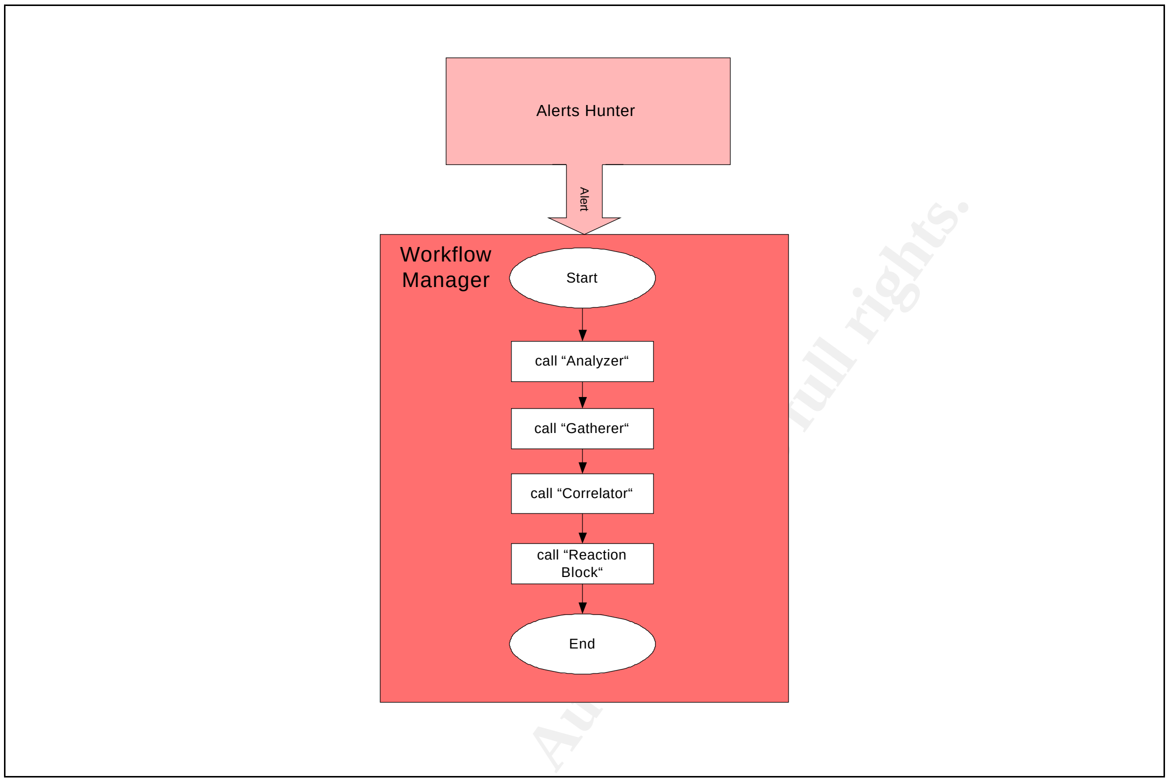


Figure 3. The Workflow Manager

Upon receiving an alert, the Alerts Hunter passes it to the Workflow Manager that begins the procedure of classification based on the following four steps:

1. Analysis of the alert: we need to retrieve from the alert the information we need to investigate, i.e. IP address of the target, identification of the sensor that generated the alert, type of alert, timestamp. (The Analyzer is described in 3.1.3).
2. Gathering of information: this module has the task to search for fresh information about the target of the attach, e.g. if the target is alive, if it is vulnerable to the attack and some information about its status such as the OS (for more details see 3.1.4 and 3.2).
3. After fresh information about the target has been collected we can re-evaluate the severity of the alert depending on the status of the target (the Correlator block is described in 3.1.5).
4. Once the severity has been evaluated a reaction may be triggered.

3.1.3 Implementation Details: Analyzer

As this work is a prototype, only some alerts are considered. These alerts are the ones referenced in the CVE system (see 2.1.1.1). The Analyzer has a really simple structure (as it appears in Figure 4).

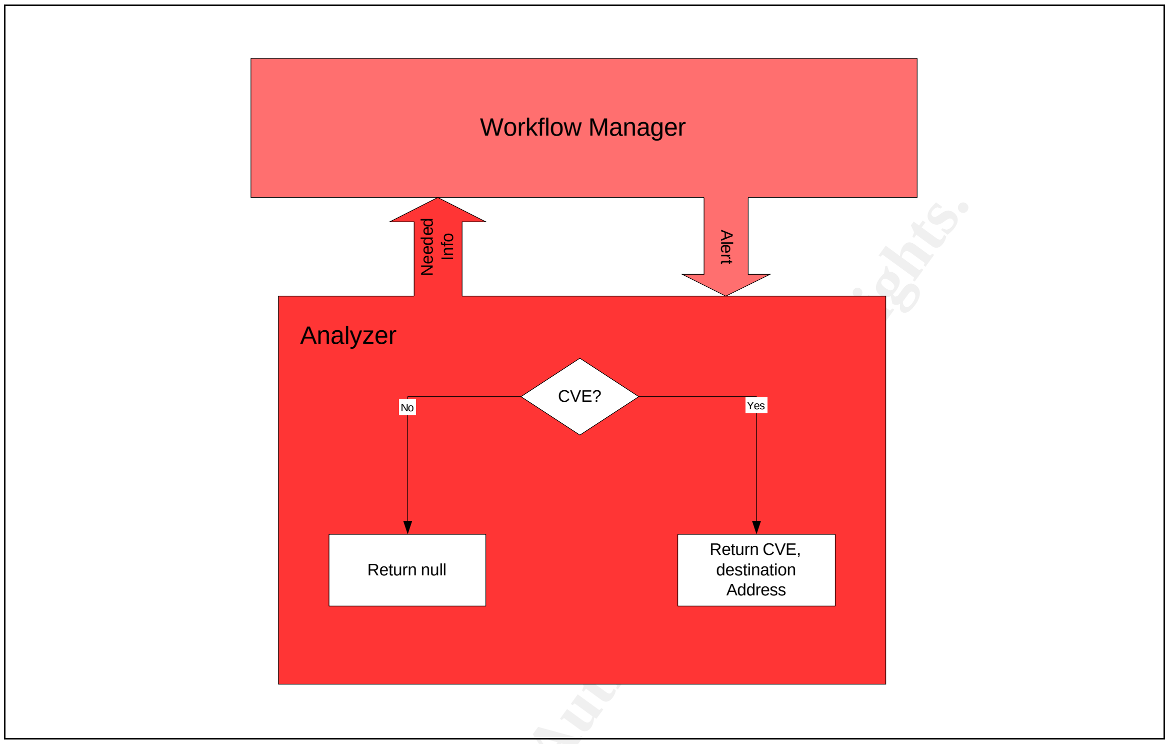


Figure 4. The Analyzer

The Analyzer checks for a CVE tag in the alerts. In case a CVE is found, the CVE number, the IP address of the target and the Snort sensor's IP address are extracted from the alert and returned the Workflow Manager.

In case a CVE tag is not found the alerts is discarded, a *null* value is returned to the workflow manager that stops the treatment of the alert and wait for a new one.

3.1.4 Implementation Details: Info Gatherer

The Info Gatherer (whose architecture is in Figure 5) works as interface between the CorE and the Info Repository. It receives the information from the Workflow manager and decides which information is needed. In our simplified case it asks the Info Repository to collect fresh information about the target IP address related with CVE tag.

It is the task of the Info repository to find the required information and to decide how to get it from the network underneath (see 3.2).

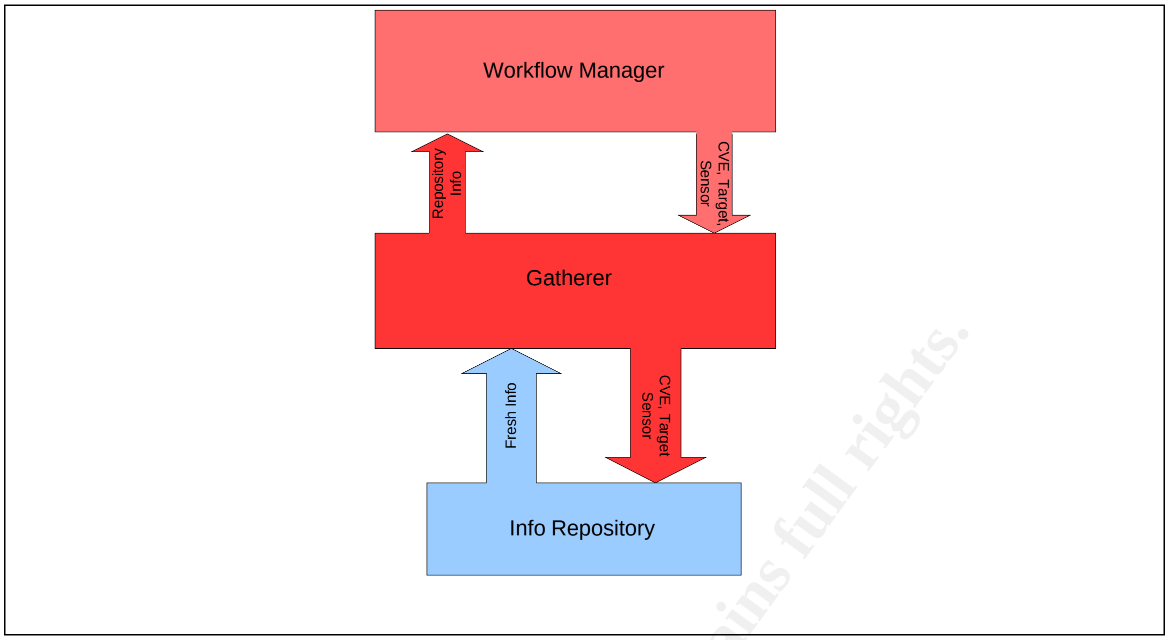


Figure 5. The Info Gatherer

3.1.5 Implementation Details: Correlator

The structure of the Correlator is in Figure 6.

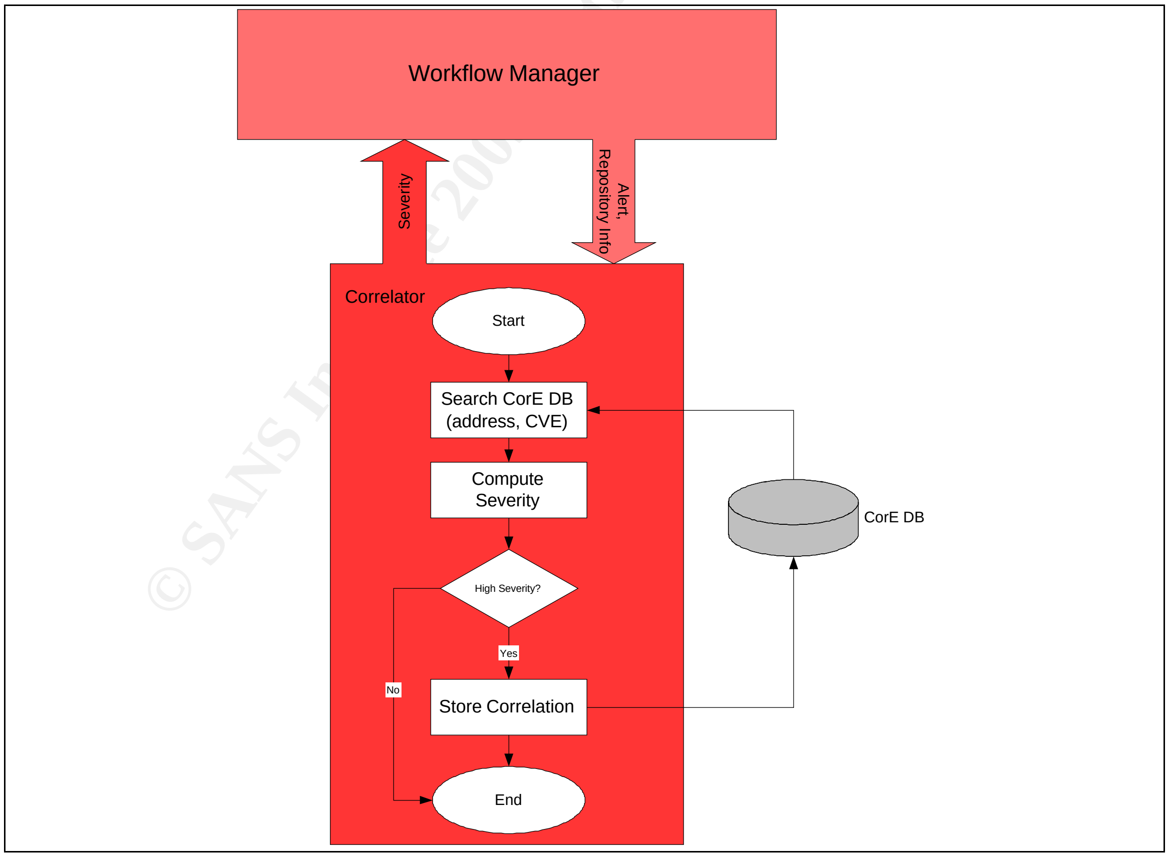


Figure 6. The Correlator

The Correlator has the task to evaluate the severity of the alerts. In the current version of the prototype this block only checks if the machine, which appears to be the target, is sensitive to the type of attack reported by Snort, using the results provided by the Gatherer (3.1.4). The flow diagram of this block in Figure 6 illustrates the original idea to correlate the alert with other alerts concerning e.g. the same target machine, the same CVE, etc. In the current simplified version, if the target machine appears to be vulnerable, then a new entry is generated in the CorE database. This entry simply links the original alert with the results of the Nessus scan in the Info Repository.

3.1.6 Implementation Details: Reaction block

As previously stated the Reaction block is actually empty as no reaction has been planned. The block exists only for future purposes and developments. Many kinds of reactions may be planned depending on the severity of the alert, the criticality of the target and may vary from an email or SMS to the system administrator up to the isolation of the target machine or subnet. Particular attention has to been taken on this point: if the planned countermeasure is to isolate a target machine, we must avoid DoS in case of induced false positives.

3.2 Info Repository

The Info Repository has the task to provide network information on demand for the Gatherer. For this purpose it relies on Nessus.
In this prototype the info repository receives the CVE tag of the Snort alert and the IP address of the target machine and returns the results of a Nessus scan of the target based on the CVE tag received. Its structure is depicted below in Figure 7.

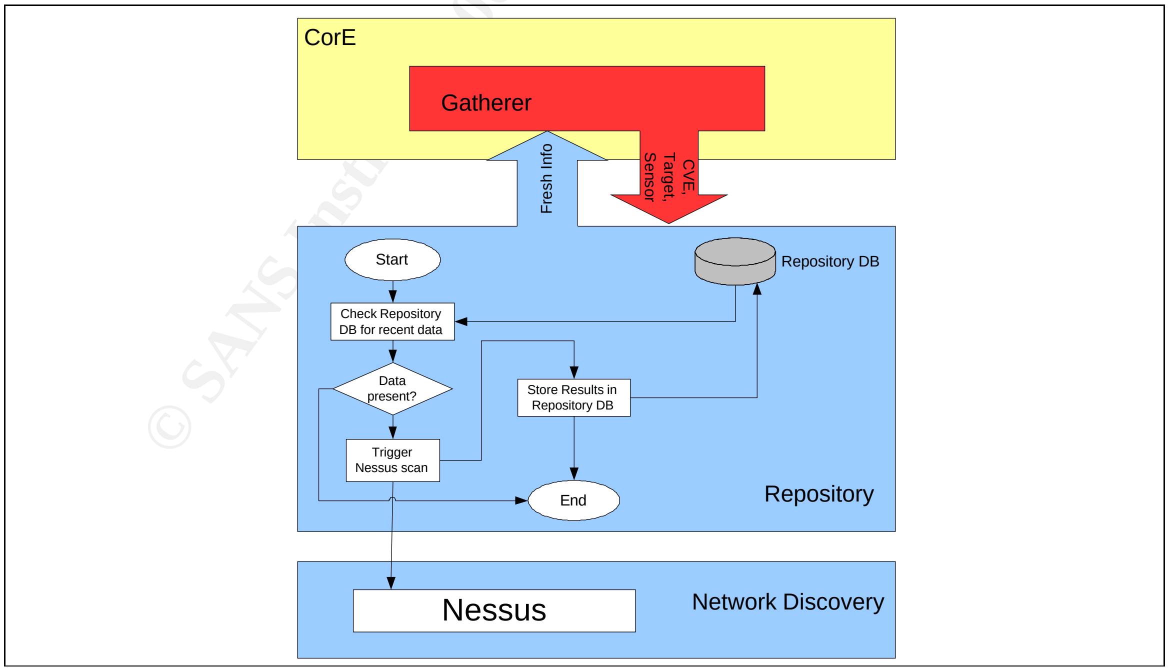


Figure 7. The Info Repository

First, there is a check on the local database to check if fresh information is already available in the local database. There will be the possibility to set the desired freshness of the information. Currently it is hard coded to the default value of 5 minutes. This parameter is obviously linked to the network underneath: if the network is very dynamic, e.g. users with dual boot machines often switching operative system, then the smaller it is the most accuracy we can obtain. In the opposite case, if the network contains static servers never changing OS configuration, a good accuracy can be achieved even for high values of these parameters.

If fresh information is not available in the Repository Database then a Nessus scan will be triggered.

The xml output that is generated by Nessus is parsed into the SQL Repository database and is made available for the Correlator block.

3.2.1 Nessus Configuration

Before using Nessus to check if the machine target of the attack is sensitive to that particular attack, we need to generate a specific configuration file. Nessus will executes only the plugin that checks the vulnerability identified by the CVE tag describing the attack. The generation of the configuration file is done using a shell script in two steps.

First we have to find which plugin is required for the desidered CVE. This is done using a *cat* operation on a file called *plugins* that contains a list of all the Nessus plugins with a description of what they are used for.

```
ATTACKID=`cat /run_nessus/plugins | grep $CVE | awk -F ',' '{print
\$1}'`;
```

($CVE is the variable containing the desired CVE tag).

The second step is to generate a configuration file enabling the plugin just identified. This operation is done by modifying a *template* file using a simple *sed* command:

```
sed -e "s/$ATTACKID = no/$ATTACKID = yes/" /run_nessus/template
>/run_nessus/.nessusrc;
```

The template file has been generated using the Nessus Graphical Interface disabling all the plugins and enabling the following parameters (listed as they appear in the *.nessusrc* configuration file):

PING section:

```
Ping the remote host[checkbox]:Do an ICMP ping = yes
```

```
Ping the remote host[checkbox]:Make the dead hosts appear in the
report = yes
```

```
Ping the remote host[entry]:Number of retries (ICMP) : = 3
```

NMAP section (The only port scanner that has been enabled is NMAP.)**:**

```
Nmap[radio]:TCP scanning technique : = SYN scan
Nmap[checkbox]:Ping the remote host = yes
Nmap[checkbox]:Identify the remote OS = yes
Nmap[checkbox]:Use hidden option to identify the remote OS = yes
Nmap[radio]:Port range = Fast scan (nmap-services)
```

Knowledge Base:

```
only_test_hosts_whose_kb_we_dont_have = no
```

These settings allow a fast scan to check whether the machine is alive and the OS fingerprint.

In the *.template* file the plugins are listed in the following way:

Plugin number = Boolean value   e.g. 10370 = no

where *no* indicates that Nessus will not use the plugin 10370, *yes* otherwise. In the *.template* file all the plugins are disabled, that means that the whole list of plugins is set to *no*.

Now we are ready to trigger Nessus using the following command line:

```
nessus -c /run_nessus/.nessusrc -q 0.0.0.1 1241 nessus_user secret
/run_nessus/.targets /run_nessus/results/results.xml -T xml
```

Where 0.0.0.1 is the server host running *nessusd* listening on port 1241 (nessus default) to which we connect as user *nessus_user* with password *secret*. The output will be stored in the /run_nessus/results/*results.xml* file in the *xml* format because among all the possible output formats the xml is the one that is most easy to parse into an SQL database. Finally we define the path to the *.nessusrc* configuration file and to the file containing the list of targets (*.target* file).

Encountered problems with Nessus: using the same configuration file (i.e. same nessus parameters) and same target machine in the same conditions sometimes the information about the OS running on the target machine does not appear in the nessus results.

3.3 Databases

Three databases have been planned:

1. The Snort Database: it contains the alerts to feed the CorE. These alerts are still to be analyzed in order to check if they can be potentially dangerous or only false positives.
2. The Info Repository Database: this database contains the results of the Nessus scans triggered by Snort alerts.
3. The CorE database: this database contains the links to the other two databases in order to correlate the snort alert with the on demand Nessus scan.

In this first step of implementation the databases have been reduced to only one database with different tables. More in details the tables containing the Nessus results and the CorE correlation have been added to the pre existing Snort database.

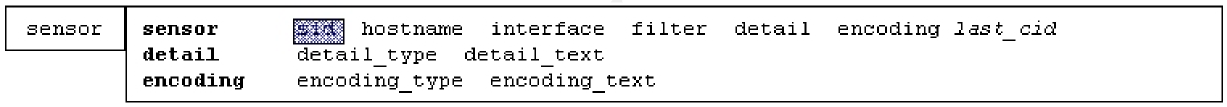
The database is a MySQL database.

In the future we will continue to refer to three different databases because they are logically different.

3.3.1 The Centralization (Snort) Database

The Snort database is the starting point of this work. This is a pre existing database whose implementation is not part of this work. Here follows a short overview of the tables of this database that have been at least partially used in this project.

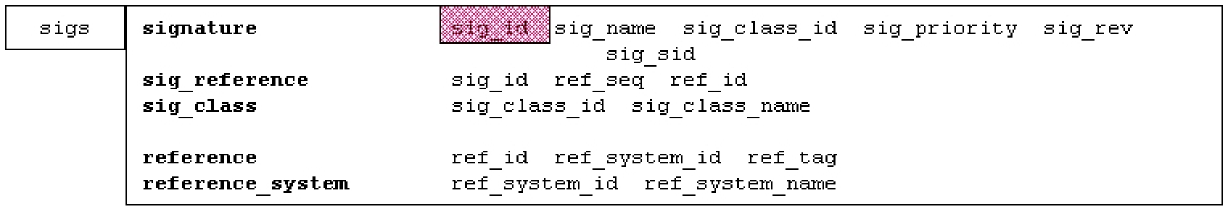
The Sensor tables (Figure 8) contain data about the Snort sensor that generated the alert. This is relevant related with 3.5. We need in fact the information about the sensor to know which Nessus server we have to connect to in order to trigger the Nessus scan.



sensor	**sensor**	sid hostname interface filter detail encoding *last_cid*
	detail	detail_type detail_text
	encoding	encoding_type encoding_text

Figure 8. The Sensor Tables

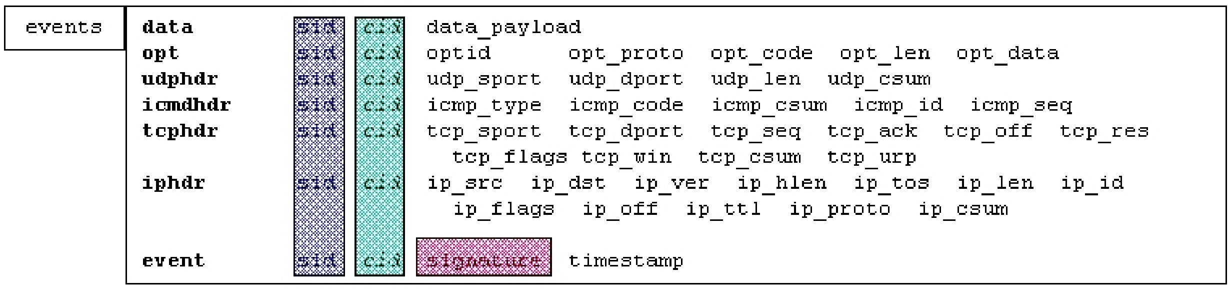
The Signature tables (Figure 9) contain the information required to classify the alert. In our prototype we only need to check the existence of a CVE tag in the *ref_tag* field.



sigs	**signature**	sig_id sig_name sig_class_id sig_priority sig_rev sig_sid
	sig_reference	sig_id ref_seq ref_id
	sig_class	sig_class_id sig_class_name
	reference	ref_id ref_system_id ref_tag
	reference_system	ref_system_id ref_system_name

Figure 9. The Signature Tables

The Event tables (Figure 10) contain the information about the event that triggered the generation of the Snort alert. At the moment we need the *ip_dst* value, i.e. the IP address of the target machine, the *cid* (i.e. an identifier of the event), and the *sid* (i.e. the sensor identifier).



```
events  data      sid  cid  data_payload
        opt       sid  cid  optid      opt_proto  opt_code  opt_len  opt_data
        udphdr    sid  cid  udp_sport  udp_dport  udp_len  udp_csum
        icmdhdr   sid  cid  icmp_type  icmp_code  icmp_csum  icmp_id  icmp_seq
        tcphdr    sid  cid  tcp_sport  tcp_dport  tcp_seq  tcp_ack  tcp_off  tcp_res
                              tcp_flags tcp_win  tcp_csum  tcp_urp
        iphdr     sid  cid  ip_src  ip_dst  ip_ver  ip_hlen  ip_tos  ip_len  ip_id
                              ip_flags  ip_off  ip_ttl  ip_proto  ip_csum

        event     sid  cid  signature  timestamp
```

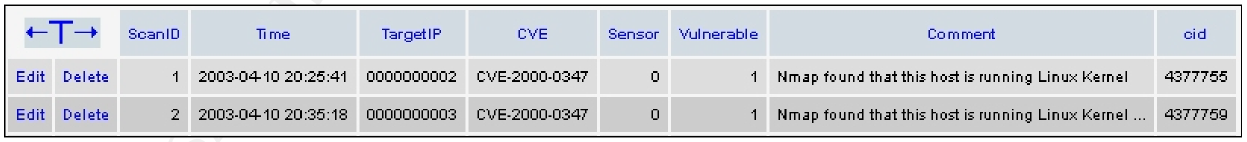
Figure 10. The Event Tables

3.3.2 The Repository Database

The Repository database is a table containing the following values:

- ScanID: this is a unique value that idenfies the nessus scan and that will be referenced in the CorE database.
- CVE: this field contains information about the vulnerability that has been tested by Nessus.
- Target: this value contains the IP address of the target machine, i.e. the machine that appears to be the target in the Snort alert and that has been tested by Nessus against the vulnerability identified by the CVE tag.
- Timestamp: this field contains the information about when the Nessus scan has been executed.
- Vulnerable: this boolean value indicates if the target is sensible or not to the vulnerability identified by the specified CVE.
- Comments: this field contains text about additional information provided by Nessus, e.g. OS fingerprint and whether the machine is alive or not.
- cid: this field is an identifier of the event.

An example of entries in this table is in the screenshot of Figure 11 (as previously stated some information has been formatted).



←T→		ScanID	Time	TargetIP	CVE	Sensor	Vulnerable	Comment	cid
Edit	Delete	1	2003-04-10 20:25:41	0000000002	CVE-2000-0347	0	1	Nmap found that this host is running Linux Kernel	4377755
Edit	Delete	2	2003-04-10 20:35:18	0000000003	CVE-2000-0347	0	1	Nmap found that this host is running Linux Kernel ...	4377759

Figure 11. The Repository Database

3.3.2.1 Nessus SQL

Alternative to the temporary solution described above, Nessus SQL is being tested. Accordingly to [9] Nessus SQL will use an SQL database as backend database for its output. This solution allows to solve two temporary hacks of the prototype:

- The temporary and extremely poor database used in the Repository Info;
- The hack used to parse the xml output into an SQL database.

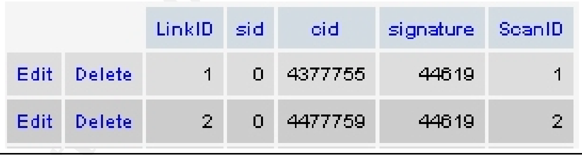
Another solution is to use the hack proposed in [10]. In this case it is sufficient to rebuild the Nessus with the modified *nsr_output.c*, but it strongly depends on Nessus implementation and a change in the parameters of the Info Repository database implies a recompilation of Nessus.

3.3.3 The CorE Database

The CorE Database contains only the link for the correlation between the Nessus results and Snort alert in case the target resulted to be vulnerable to the attack after the nessus scan. An entry in the CorE database has the following structure:

- LinkID: a unique identifier;
- ScanID: a link to the ScanID value of the Repository database;
- cid: idenfifier of the event;
- sid: identifier of the Snort sensor;
- Signature: used to retrieve information about the alert.

An example is in the screenshot of Figure 12.



		LinkID	sid	cid	signature	ScanID
Edit	Delete	1	0	4377755	44619	1
Edit	Delete	2	0	4477759	44619	2

Figure 12. The CorE Database

3.4 The CorE GUI

A simple GUI has been implemented for the CorE. This GUI shows only relevant alerts to the system administrator, i.e. the alerts related to a vulnerable target machine.

In the administrator GUI the information is organized in the following way:

- Timestamp of the alert;
- Snort sensor that generated the alert;
- Source of the attack;
- Target of the attack;
- Timestamp of the Nessus scan;

- Vulnerability status of the target: this field is actually useless as only alerts related to a vulnerable machine are linked in the CorE database In future development of this prototype other network discovery tools may be used at the same time of Nessus and could be usefull to distinguish the results, i.e. which tool has found the vulnerability;
- OS Fingerprint detected by Nessus: in this way the administrator is able to know which OS wa running on the target machine at the moment of the attack.
- Status of the target: if the target is alive or not.

3.5 Network Architecture

An example of a network architecture using the prototype is in Figure 13.

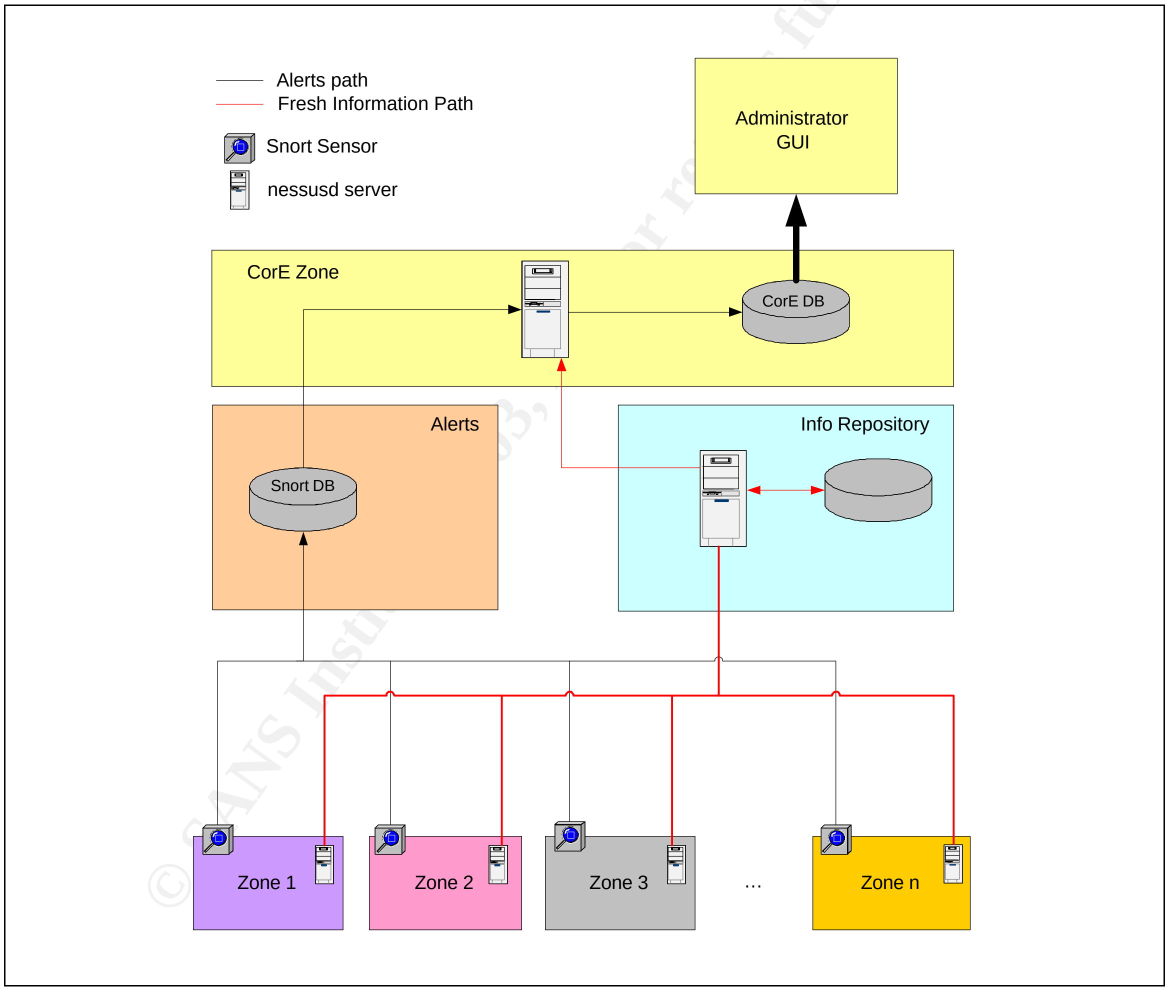


Figure 13. Architecture Implementation

It is relevant to note that each zone needs at least two elements:

- A Snort sensor capable of performing the network based intrusion detection, that analyzes the traffic, generates the alerts and sends them to the centralization database;

- A Nessus server that can scan all the possible targets in that zone and that must be reachable by the Nessus client in the Info Repository zone.

Of course the Snort sensor and the Nessus server may run on the same machine.

Note: as this work is dealing uniquely with the implementation of the prototype, the intrinsic security of each zone (firewalls, rules, access lists), as well as where to implement the databases in order to have them in high security zone, but at the same time reachable by the sensors, has not been discussed.

4 Conclusions and Results

The results obtained by this prototype are definitely encouraging to continue its development. Not only the administrator has a clear view of the really potentially dangerous alerts, avoiding searching a huge number of false positive, but also receives information about the status of the target at the time the attack has been executed.

On the other hand this prototype is far from being useful as it takes into account only alerts referring to the CVE list, therefore the administrator cannot avoid to look to the whole Snort database.

Even if the results obtained are encouraging they are only coming from lab simulations, as the prototype has not yet been tested on a real network.

4.1 Future works

This work is still a prototype, so it has many limits and needs to be enhanced.

- First of all it checks only alerts based on the reference of CVE. Future development must also contain the other systems of reference such as Bugtraq and arachNIDS in order to be able to evaluate the severity of the other alerts received and screening false positives.
- The prototype is actually correlating the information of the on-demand Nessus scans with the Snort alerts. A more intelligent correlation needs to be implemented in order to link previous alerts concerning e.g. the same target or the same source, the same vulnerability etc. A further correlation with Syslog, Samhain, Tripwire events will also provide more information.
- The implementation of the parser that takes Nessus results and stores them in the Info Repository database is currently a simple hack that uses *grep* operations on the xml output and copy the results into the sql database. To improve this point we are currently following the development of Nessus SQL (see 3.3.2.1).
- Also related to Nessus SQL is the actual status of the databases. As said in 3.3, in the current implementation only the Snort database (not goal, but base of this work) has a productive shape while the CorE and the Info Repository databases are only tables temporarily included in it.
- The implementation of the Alerts Hunter is related to the Prelude status. In the current state it searches the Snort database each *X* minutes for new alerts. This is far from being an optimal solution. Built on top of a Prelude Manager generating an XML output, the Alerts Hunter will have no need to search itself the alerts, while receiving them in real time directly from the Prelude Manager.
- Automatic reactions to network attacks are still in the research phase. They represent indeed a high potential for DoS in case of induced false positives.

- The Network Discovery is based on Nessus only. Other tools, e.g. an optimization of Netmap, etc., may be integrated in the Info Gatherer block in order to check the status of the target at the same time an attack is detected.
- Except the simple GUI for the CorE no other GUIs have been implemented for the other blocks. These GUIs may help in managing the databases e.g. cleaning up old entries, archivieng old alerts etc.

Table of Figures

Table of references

[1] Snort. “The Open Source Network Intrusion Detection System”. URL: http://www.snort.org/about.html (16 April 2003).

[2] Mitre Corporation. “Common Vulnerabilities & Exposures”. URL: http://www.cve.mitre.org/docs/docs2000/key_to_info_shar.pdf

[3] Mitre Corporation. “CVE Candidates Explained”. URL: http://www.cve.mitre.org/about/candidates.html

[4] Whitehats, Inc. “arachNIDS center”. URL: http://www.whitehats.com/ids/index.html

[5] Insecure.org. “Bugtraq mailing list”.URL: http://lists.insecure.org/about/bugtraq.txt (16 December 2002)

[6] Renaud Deraison.”The Nessus Project”. URL: http://www.nessus.org/intro.html (1998-2003)

[7] Insecure.org. “Nmap Free Stealth Network Port Scanner, Linux/Windows/UNIX/Solaris Tools & Hacking”. URL: http://www.insecure.org/ (19 March 2003)

[8] The NetMap project. “Network Modeling, Discovery, and Analysis”. URL: http://www.cs.ucsb.edu/~rsg/NetMap/

[9] Javier Fernandez-Sanguino. Nessus devel mailing list. “[Database devel] Current status (README and schema)”. URL: http://archives.neohapsis.com/archives/apps/nessus/2003-q1/0289.html (24 March 2003)

[10] Galitz, SecurePoint - Nessus Archive. “Nessus -> MySQL hack”. URL: http://msgs.securepoint.com/cgi-bin/get/nessus-0101/53.html (12 January 2001)