**Institut Eurécom**

**Corporate Communications Department**

**2229, route des Crêtes**

**B.P. 193**

**06904 Sophia Antipolis**

**FRANCE**

**Research Report RR-03-094**

# White Paper:

# Alert Correlation: Technical Report[1]

**December 30, 2003**

**Fabien Pouget**

**Institut Eurécom**

**Email: {pouget@eurecom.fr}**

[1] This research is supported by a research contract with France Telecom R&D, contract N. 425-17044

# Alert Correlation: Technical Report

Fabien Pouget
Email: {pouget}@eurecom.fr
Eurecom
2229, Route des Crêtes ; BP 193
06904 Sophia Antipolis Cedex ; France

**Abstract:**
In this paper, we report on an experiment run with three alert correlation tools at Eurecom. The motivation of this work resides in our wish to experiment three tools, representative of the three categories we previously defined in [PoDa03]. A testbed was developed to compare them, and to evaluate their capacities. We describe each tool in details, as well as their installation modalities. We then present our testbed and discuss results obtained and lessons learned thanks to these experiments.

**Keywords:**
Alert Correlation, AlertSTAT, Simple Event Correlator, ACID.

**Corresponding Author:**
Fabien Pouget (pouget@eurecom.fr)
Tél: +33 (4) 93 00 29 26
Fax: +33 (4) 93 00 26 27

**TABLE OF CONTENTS**

## 1 Introduction

Alert correlation tools can be basically classified into three main categories, respectively named “Log Analysis Tools”, “Management Consoles” and “Experimental Tools”. We report the interested reader to [PoDa03] for more detailed information on this topic. A review of the state of the art is presented as well as common tools which aim at correlate alerts produced by diverse security elements.

Following this classification, we present in this report three tools that are representative of each category. Our choice was mainly guided by simple criteria, such as their costs, the installation requirements and their functionalities. The first tool is the Simple Event Correlator (SEC), which belongs to the ‘Log Analysis Tool’ category. The second tool is AlertSTAT, which represents the ‘Experimental tools’ category. Finally, the third tool is an ‘Alert Management Console’ called ACID.

The paper is organized as follows. Section 2 proposes a more detailed presentation of these three tools, as well as their installation procedures. Section three describes our experimental testbed, and Section four presents our resulting observations. The tools are compared to test their advantages, as well as some of their limitations.

## 2 Presented tools

### *2.1 Simple Event Correlator (SEC)*

#### *2.1.1 Tool Presentation*

The Simple Event Correlator (SEC) is a correlation tool written by Risto Vaarandi, a professional software developer from Estonia. It is available from SourceForge.net at the following URL: http://simple-evcorr.sourceforge.net. R. Vaarandi has provided a Manual page, a FAQ (Frequently Asked Questions) section and some Examples that are helpful in developing advanced solutions [Sec].

Basically, SEC is a PERL script which reads an input stream from a file or pipe and applies pattern matching operations to the input looking for patterns specified by rules, found in configuration files. It was originally conceived as a system for correlating HP OpenView network events, but it has also been used to correlate intrusion alerts generated by Snort. Actually, the system is flexible enough to be used for correlating almost anything.

This section introduces basic SEC use and operations. First, the expert knowledge is expressed through SEC rules. There is no limit to the number of rules, but there are nine distinct rule types. Each rule can be used to trigger one of fifteen different actions. What adds complexity is that a rule action can be used to generate an event that is used as input to another rule. In this way, rules can be strung together to perform complex correlation. We propose to briefly describe the different rule types as well as possible actions. However, let's beforehand have a look at a very simple rule example illustrated in figure 1:

```
type = Single ❶
ptype = RegExp ❷
pattern = foo\s+(\S+) ❸
desc = $0 ❹
action = logonly ❺
```

**Figure 1: a Simple SEC rule example**

This example means the following:

❶ *Single* is the rule type. SEC includes several different types of rules. This one is the simplest (see below for more information on type differences).

❷ *RegExp* is the pattern type. SEC allows two types: *RegExp* (for "Regular Expression") matching or *SubStr* for simpler string matching (matching words only)

❸ *foo\s+(\S+)* is the actual pattern- in this case a Perl regular expression pattern. This pattern matches the word *foo* followed by one or more spaces, followed by one or more non-space characters, such as *bar*, *toto*, etc… We invite the interested reader to [Clarks] for more information on regular expressions.

❹ *desc* is a variable definition for the pattern description. In this case, a Perl numbered variable, *$0*, is set to the entire matched pattern.

❺ The *action* statement describes the action taken when the pattern is recognized. In this case, the *logonly* action simply writes the pattern to the logfile if one is indicated on the command line, or to standard output if not.

In this example, we have created a SEC rule that matches a regular expression. The rule is ‘single’-type, the simplest one. However, eight other types are currently provided by SEC. We list a short summary of them below and we report the interested reader to Annex A for rule examples of each type:

- *SingleWithScript:* The *SingleWithScript* rule combines matching a pattern and the execution of a separate program to determine if the rule is matched. Running a separate program to validate or confirm whether an event is valid is often necessary. For example, matching an IP address in a rule and checking whether the IP address is on a list of valid addresses can not be done by pattern matching in a rule alone. A separate program is required to determine if the matched IP is on the list.
- *SingleWithSuppress:* With the *SingleWithSuppress* rule, it is possible to become alerted to an event the first time it is seen, then ignore the same event within a time window.
- *Pair:* The *Pair* rule handles two different events, matched by two different patterns in its rule definition. The rule uses time window which is set upon the first occurrence of event A. If event B occurs within the time window, events A and B are considered correlated, and the entire rule is considered matched. Otherwise, the correlation operation for the pair terminates. There are two action statements, each corresponding to its own pattern. Action 1 is executed when A is matched. Action 2 is executed if event B occurs within the time window.
- *PairWithWindow*: The *PairWithWindow* rule appears identical to the *Pair* rule. Both contain two patterns, two actions, and a time window. The difference is that, in *PairWithWindow* the action 2 is executed if events A and B both occur within the time window. If A occurs, but B does not occur, then action 1 is executed.
- *SingleWithThreshold:* The *SingleWithThreshold* rule is used to ‘count’ the number of matched events within a time window. If the number exceeds the threshold, the *action* is executed. If the number of matched events does not exceed the threshold within the time window, the time window ‘slides’; that is, start time for the correlation window is moved to the second occurrence of the matched pattern. This process repeats, until the time window expires with no matched events.

- *SingleWith2Thresholds*: The *SingleWith2Thresholds* rule is very similar to the *SingleWithThreshold* rule, except that we can now definitely determine when events stop. This is done with a second threshold and a second timer window. *SingleWith2Thresholds* counts the number of matched events and executes *action1* when the number is above *thresh* events. Once this low threshold (watermark) is reached, SEC starts *window2* and counts additional matched events. When the number of events falls below *thresh2* events within *window2*, SEC executes *action2*. Note that both windows are sliding windows; that is, the beginning time of the window moves to the time of the next match if the time window of the first match expires.
- *Suppress*: The *Suppress* rule is very intuitive. Events matching the rule are suppressed. Since the rule has no *action* statement, it does nothing.
- *Calendar*: The *Calendar* rule is another easy to understand rule. It executes *action* statements at specific times. The time specification is similar to that used by cron, and is detailed in crontab.

In addition to these different types, rules are associated with specific actions. SEC has over a dozen different actions it can perform. They include:

- *write*: The *write* action writes the specified text to a given file.
- *Shellcmd*: The *Shellcmd* action causes SEC to execute a shell command. The shell command can be any executable program permitted by normal user privileges.
- *Spawn*: The *Spawn* action is identical to the *Shellcmd* action, except that output from the command is fed back into SEC for pattern matching.
- *Assign and Eval*: Special variables have global scope across multiple SEC rule files. If an assignment is made by either the *assign* or *eval* actions, SEC maintains that assignment for the life of the program or until the next assignment to that variable.
- *Event*: The *event* action allows the insertion of input to SEC from inside SEC itself. It is a simple feedback mechanism- one controlled by SEC's own rules. A *time* parameter specifies the number of seconds to wait before inserting the event text into SEC's input stream.

Finally, SEC presents other features that are worth being mentioned. Between them, we distinguish:

- *Multiple Input Stream*: thanks to the *spawn* action, SEC can obtain input from multiple input streams. The *tail* program is often used to read multiple files, for instance. An example is (see Annex A for one example).
- *Pipe Output*: SEC allows writing to a 'named'pipe, also called a 'fifo'. This feature provides a simple method of inter-process communication (IPC). Most Unix systems already have the ability to create and use named pipes. With SEC, the only requirement is that the named pipe must exist before writing to it (typically, this is performed with the *mkfifo* and *mknod* commands).
- *Contexts*: SEC has the ability to define and use *contexts* with rules. Contexts are "the interrelated conditions in which something exists or occurs". In SEC, a context exists when it is created by a rule action. Contexts can act as event stores. Events can be added to contexts as they occur. A collection of events in a context can be input to a script to be saved in a file. Thus, many actions can be performed using contexts. They are not listed in this paper for size conciseness.

### *2.1.2 Documentation and Installation*

SEC installation is very easy. It suffices to decompress the file from [Sec] into a given folder. No more effort is required. In a *nix environment, commands should be similar to:

```
[]# mkdir Sec_folder
```

Then, you must copy the sec-2.2.beta2.tar.gz file from [Sec] and decompress it into the newly created folder:

```
[]# cd Sec_folder
[] tar –zxf sec-2.2.beta2.tar.gz
```

The Simple Event Correlator is now installed.

SEC has many parameters that control its operation. These are viewed by simply calling SEC with no parameters:

```
% perl sec.pl

        Version: 2.2.beta2
```

```
Usage:

  sec.pl -input=<inputfile> -conf=<conffile pattern> ...

Optional flags:

  -input_timeout=<input timeout>

  -timeout_script=<timeout script>

  -reopen_timeout=<reopen timeout>

  -poll_timeout=<poll timeout>

  -blocksize=<io block size>

  -log=<logfile>

  -debug=<debuglevel>

  -pid=<pidfile>

  -dump=<dumpfile>

  -cleantime=<clean time>

  -bufsize=<input buffer size>

  -evstoresize=<event store size>

  -quoting, -noquoting

  -tail, -notail

  -fromstart, -nofromstart

  -detach, -nodetach

  -intevents, -nointevents

  -testonly, -notestonly
```

All options are fully described in the SEC Manual Page. Options of the form `-name=value` are required to have a value. As noted above, the ``-conf=<conffile pattern>" and ``-input=<inputfile>"

options are required when executing *perl sec.pl*. A brief review of some of the more common options follows:

| Option | Description |
|---|---|
| -log=<logfile> | This option specifies the location of a logfile that SEC uses to track its operation, such as pattern matches, actions, etc. The volume of information is controlled by the -debug option. |
| -debug=<debuglevel> | This option controls how verbose SEC is as it tracks its operation. The values range between 1 (critical messages) and 6 (debug messages). Each level includes output from lower levels. |
| -pid=<pidfile> | This option provides for the location of a process ID file. SEC will write it's process ID to this file upon startup. |
| -dump=<dumpfile> | This option provides for the location of a dump file where SEC can dump its internal data structures, variables and other information upon receipt of the USR1 signal. The default location is /tmp/sec.dump. |
| -detach | Specifying this option causes SEC to detach itself from the controlling terminal and run as a daemon process. The default is -nodetach. |
| -intevents | This option causes SEC to perform special processing at startup. This special processing is described in the SEC man page. |
| -testonly | The ``-testonly" option can be used to test for syntax errors in configuration files. It does not start SEC for operation. |

The above options are the most common in ordinary usage. See the Manual Page for more information on these and other options.

### *2.1.3 Usages*

The Simple Event Correlator (SEC) is a powerful and flexible tool. The SEC web site has an example SEC rule set for Snort that demonstrates even more of SEC's capabilities. It shows how to configure SEC to:

- Create a portscan report
- Detect the start of a priority 1 attack, and send an email notification
- Handle incidents by thresholding
- Report IPs that have been active for a certain amount of time
- Send a daily incident report
- Etc

This sample snort ruleset is presented in Annex B. We invite the interested reader to have a deeper look at it. Finally, we would like to point out that despite the rule names, their functionality is quite limited.

## *2.2 AlertSTAT*

### *2.2.1 Tool Presentation*

AlertSTAT is a STAT-based system designed by the University of Santa-Barbara (California). The State Transition Analysis Technique (STAT) has been described in [PoDa03]. In short, it is a methodology to describe computer penetrations as *attack scenarios*. Attack scenarios are represented as a sequence of transitions that characterize the evolution of the security state of a system. In an attack scenario, *states* represent snapshots of a system's security-relevant properties and resources. They are characterized by means of *assertions*, which are predicates. *Transitions* between states are annotated with *signature actions* that represent the key actions that, if omitted from the execution of an attack scenario, would prevent the attack from completing successfully.

Vigna et Al. have developed a family of tools around this technique. Such a framework is presented in [Vign03]. AlertSTAT belongs to it. Its task is to fuse, aggregate and correlate alerts from intrusion detection systems (or sensors), such as USTAT, NetSTAT, WinSTAT, LinSTAT, etc. Therefore, AlertSTAT uses the alerts produced by other sensors as input and matches them with respect to attack scenarios that describe complex and multi-step attacks. A simple alertSTAT scenario is presented in Annex C.

AlertSTAT operates on alerts formatted according to the IETF's Intrusion Detection Message Standard (IDMEF) proposed standard [Idmef]. The application is built by composing an IDMEF-based Language Extension with an Event Provider that reads IDMEF events from files and /or remote connections and feeds the resulting event stream to the STAT core. A number of attacks scenarios have been developed, including the detection of complex scans, "many-to-one" and "one-to-many" attacks, island hopping attacks and privilege escalation attacks.

### *2.2.2 Documentation and Installation*

The installation is quite simple. We propose to install AlertSTAT (release 2.0) on our Red hat 7.3 machine as follows:

First, the tool requires having libxml2 installed. Furthermore, we must link */usr/include/libxml* to */usr/include/libxml2/libxml* like:

```
[]# ln –s /usr/include/libxml2/libxml /usr/include/libxml
```

A version of libxml2 is provided in alertSTAT's home page (the libxml2-v. 2.4.26 version).

Then, we decompress the alertSTAT downloaded file from [Ale]:

```
[]# tar –zxf alertSTAT-2.0.tar.gz
```

Finally, we follow the install README:

```
[]# cd STAT-1.0
[STAT-1.0]# ./configure
[STAT-1.0]# make
[STAT-1.0] make install (with root privileges)
```

The INSTALL file provides more details about installing the alertSTAT package using GNU build tools or rpm facilities (RedHat Package manager).

### 2.2.3 Usages

AlertSTAT is executed by typing:

```
[STAT-1.0] ./alertstat –alertfile [alert_filename].
```

The /etc/alertstat/ directory contains the configuration files for extensions, scenarios and providers.

A default setup is provided, which does the following tasks:

- The IDMEF is loaded
- The IDMEF provider is loaded and activated. It is configured to process the audit file passed as command line parameter.
- Seven IDMEF scenarios are provided and ready to be used
- The IDMEF response, which responds by sending higher level alerts, is loaded.

## 2.3 Analysis Console for Databases (ACID)

### 2.3.1 Tool Presentation

The Analysis Console for Intrusion Databases (ACID) is a PHP-based analysis engine to search and process a database of security events generated by various IDSs, firewalls, and network monitoring tools. It was developed by R. Danyliw at the CERT Coordination Center, initially as a part of the

AIRCERT project. It is currently maintained in the context of this project and in the author's free time. ACID is open-source and released under the GPL licensing. It is portable without modification to any operating system that can support PHP.

ACID features currently include:

- *Query-builder and search interface* for finding alert matching on alert meta information (such as signature, detection time), as well as the underlying network evidence (e.g. source/destination addresses, ports, payload or flags). We report the interested reader to the database Entity Relationship Diagram (ERD) provided in Annex D for further details on the Snort/ACID database.
- *Packet viewer (decoder)* graphically display the layers 3and 4 packet information of logged alerts
- *Charts and statistics generation* based on time, sensor, signature, protocol, IP address, TCP/UDP ports, or classification.

### *2.3.2 Documentation and Installation*

The following is a step-by-step list of installing ACID v. 0.9.6b23.

First of all, ACID has many dependencies that must be installed beforehand. We report the interested reader to http://www.snort.org/docs/snort_acid_rh9.pdf for more detailed on their installation and configuration. A very good step-by-step installation procedure is provided to help installing ACID and all its dependencies with Snort. We list below the ACID requirements for a MySQL environment.

- PHP-4.3.3: any home-grown script that understands the underlying DB format
- LibPcap-0.7.2: a network library required by Snort
- Apache-2.0.47: our web server
- MySQL-4.0.15a: the database in which to store the information from Snort.
- Snort-2.0.2: the IDS that generates alerts and fill the database.
- ADODB-1.2: a PHP database abstraction library
- JP-Graph-1.13: the PHP chart library
- Zlib-1.1.4: a compression library

PHPlot can be used instead of JPGraph for older PHP versions [PHPlot]. The installation process must be done with root privileges. It is quite tedious and our own installation of these dependencies is presented in Annex C. We provide below the installation and configuration of ACID only: However, this is only the visible part of the global installation iceberg.

First, the ACID downloaded file must be placed in a specific web server folder, named /www/html. Then, the commands must be from the downloads directory:

```
[]# cp acid-0.9.6b23.tar.gz /www/html (The Apache web server is installed in "/www" directory)
[]# cd /www/html
[]# tar –zxf acid-0.9.6b23.tar.gz
```

Now we can configure Acid. In the /www/html/acid/ directory, the acid_conf.php file should look like:

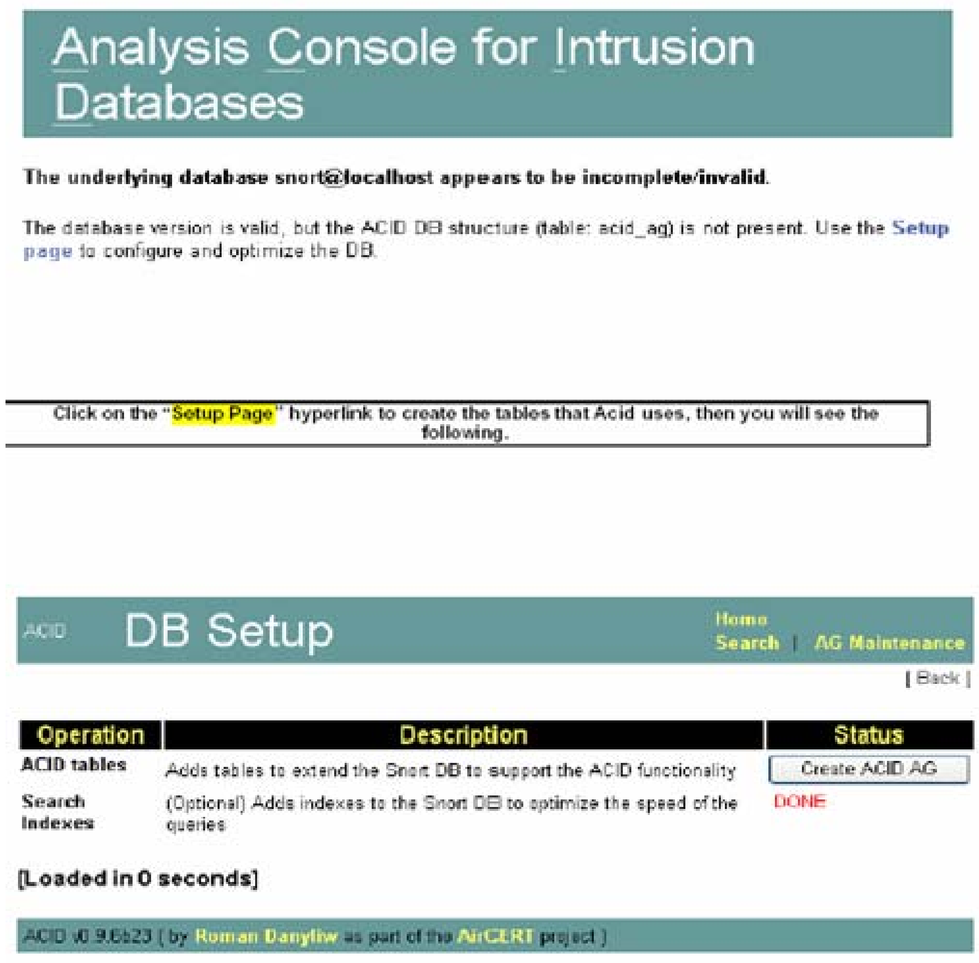
```
$DBlib_path = "/www/html/adodb";
/* The type of underlying alert database
*
* MySQL : "mysql"
* PostgresSQL : "postgres"
* MS SQL Server : "mssql"
*/
$DBtype = "mysql";
/* Alert DB connection parameters
* - $alert_dbname : MySQL database name of Snort alert DB
* - $alert_host : host on which the DB is stored
* - $alert_port : port on which to access the DB
* - $alert_user : login to the database with this user
* - $alert_password : password of the DB user
*
* This information can be gleaned from the Snort database
* output plugin configuration.
```

```
*/
$alert_dbname = "snort";
$alert_host = "localhost";
$alert_port = "";
$alert_user = "snort";
$alert_password = "new_password";
/* Archive DB connection parameters */
$archive_dbname = "snort";
$archive_host = "localhost";
$archive_port = "";
$archive_user = "snort";
$archive_password = "new_password ";
```

And a little further down

```
$ChartLib_path = "/www/html/jpgraph-1.13/src";
/* File format of charts ('png', 'jpeg', 'gif') */
$chart_file_format = "png";
```

We can now start Apache and go to http://yourhost/acid/acid_main.php. We get a message from the browser that looks like the one in figure 2:



Analysis Console for Intrusion Databases

**The underlying database snort@localhost appears to be incomplete/invalid.**

The database version is valid, but the ACID DB structure (table: acid_ag) is not present. Use the Setup page to configure and optimize the DB.

Click on the "Setup Page" hyperlink to create the tables that Acid uses, then you will see the following.

ACID DB Setup

Home
Search | AG Maintenance

[ Back ]

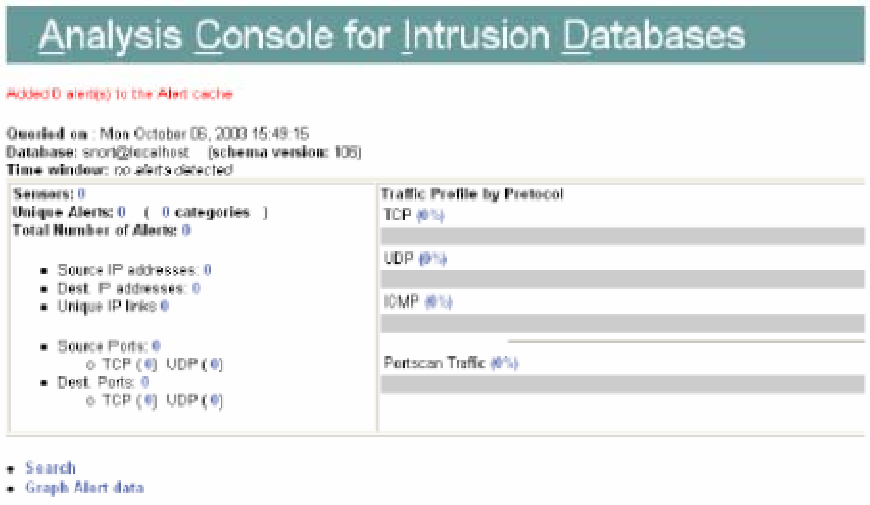
| Operation | Description | Status |
|---|---|---|
| ACID tables | Adds tables to extend the Snort DB to support the ACID functionality | Create ACID AG |
| Search Indexes | (Optional) Adds indexes to the Snort DB to optimize the speed of the queries | DONE |

[Loaded in 0 seconds]

ACID v0.9.6b23 ( by Roman Danyliw as part of the AirCERT project )

**Figure 2: Screenshot of ACID configuration**

Then, we click on "Create Acid AG". Thus, when we go to http://yourhost/acid, we see the ACID homepage as illustrated on figure 3:



Analysis Console for Intrusion Databases

Added 0 alert(s) to the Alert cache

**Queried on** : Mon October 06, 2003 15:49:15
**Database:** snort@localhost (**schema version:** 106)
**Time window:** no alerts detected

**Sensors:** 0
**Unique Alerts:** 0 ( 0 **categories** )
**Total Number of Alerts:** 0

- Source IP addresses: 0
- Dest. IP addresses: 0
- Unique IP links 0

- Source Ports: 0
  - TCP ( 0) UDP ( 0)
- Dest. Ports: 0
  - TCP ( 0) UDP ( 0)

**Traffic Profile by Protocol**
TCP (0%)
UDP (0%)
ICMP (0%)

Portscan Traffic (0%)

- Search
- Graph Alert data

**Figure 3: Screenshot ACID homepage**

There are some ways to secure the ACID directory. We report the interested reader to [Acid] for more details. Some of them are also mentioned at the end of the document available at http://www.snort.org/docs/snort_acid_rh9.pdf.

#### *2.3.3 Usages*

ACID is an alert management console, and its usage is limited to database queries from its GUI. However, its graphical interface is convenient to obtain simple queries in a fast way. Some screenshots are presented in section 3.5.

# 3 Experimentation

## *3.1 Associated Tools*

### *3.1.1 Introduction*

We need alerts to test the three previously mentioned tools. One solution consists in downloading existing alert files, which are used in many research papers to compare Intrusion Detection System efficiency and accuracy (such as the Cyber panel Grand Challenge Problem-GCP, etc…). Another solution consists in generating ourselves our own alerts. We chose this alternative for many reasons. First, a simple glance at alertSTAT shows that it might be simpler to test home-made alerts on existing scenarios than the opposite (this will be confirmed in the following section). Secondly, we want alerts in the IDMEF standard presented by IETF [Idmef]. Freely available alert files are scarce in IDMEF format, if not non-existant. Most of those we found are in pcap or snort formats.

Furthermore, a Snort plugin has been released recently. It modifies traditional snort alerts to generate IDMEF alerts. As a consequence, we decided to produce our own alerts thanks to this utility. Attacks were launched from 192.168.1.1 to 192.168.1.3, thanks to Nessus, a convenient but dangerous vulnerability scanner. These attacks are expected to trigger alerts from Snort, installed in promiscuous mode on 192.168.1.2 (see figure 4). A more detailed description of both utilities and their installation is given in the two following sub-sections.

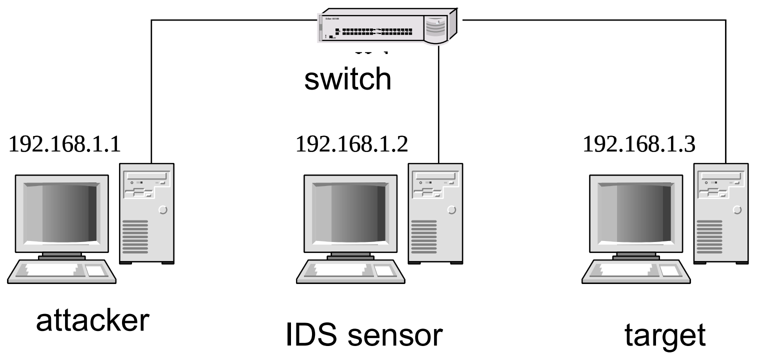


**Figure 4: Our testbed architecture**

### *3.1.2 NESSUS*

Nessus is a security scanner. It is software which audits remotely a given network and determines whether a malicious person may break into it, or misuse it in some way. It does not consider that a given service is running on a fixed port. Actually, it detects it and tests its security.

Each Nessus test is written as an external plugin. This way, many tests can be lead, without having to modify the code of the nessusd engine. Such tests are written through a particular language called NASL (Nessus Attack Scripting Language). We report the interested reader to [Nessus] for test descriptions. The Nessus project offers a large variety of tests which are daily updated and freely downloadable.

Nessus is built following client-server architecture. To make things clearer, Nessus is made up of two parts: a server, which performs the attacks, and a client which is the front end (GUI). They can be deployed in multiple configurations reducing management costs (one server can be used by multiple clients). Both can run on different Operating Systems.

The Nessus security scanner relies on the following items (dependences):

- *GTK* (the Gimp Toolkit v1.2). GTK is a set of widgets which are used by many open-sourced programs. It is used by the POSIX client Nessus. It can be downloaded at ftp://ftp.gimp.org/pub/gtk/v1.2.
- *OpenSSL*, optional but heavily recommended. OpenSSL is used for the client-server communication as well as in the testing of SSL-enabled services. It can be downloaded at: http://www.openssl.org.

It is installation is very easy. One way consists in downloading on Nessus web page the following *nessus-installer.sh* file. Then, it suffices to execute it after having placed it in a dedicated folder:

```
[]# mkdir nessus
[] # cd nessus
[nessus] # cp /path/to/nessus-installer.sh .
[nessus] # sh nessus-installer.sh (as root)
```

Before using Nessus, we must configure the server. The initial step consists in creating a user account. Indeed, the Nessus server has its own users database, each user having a set of restrictions. This allows sharing a single nessusd server for a whole network and different administrators who will only test their part of the network. In our case, this characteristic is not important, and we only create one user (login: tintin, password: milou). It is done like:

```
[]# nessus-adduser
```

Then, we follow instructions to obtain:

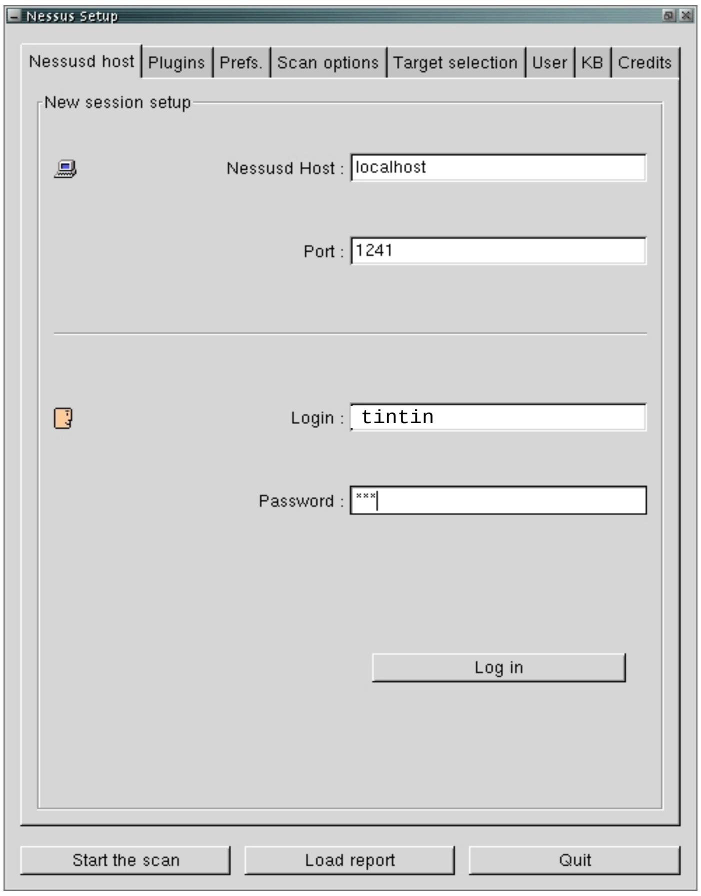
```
# nessus-adduser

Addition of a new nessusd user
------------------------------

Login : tintin
Authentication (pass/cert) [pass] : pass
Password : milou

User rules
----------
nessusd has a rules system which allows you to
restrict the hosts
that tintin has the right to test. For instance, you
may want
him to be able to scan his own host only.

Please see the nessus-adduser(8) man page for the
rules syntax

Enter the rules for this user, and hit ctrl-D once
you are done :
(the user can have an empty rules set)

deny 192.168.1.1 (attacker)
accept 10.168.1.3 (target)
default deny

Login            : tintin
Password         : milou
DN                   :
Rules            :

deny 10.168.1.1
accept 10.168.1.3
default deny


Is that ok (y/n) ? [y] y

user added.
```

Finally, the Nessus daemon (nessusd) can be configured. In the file /usr/local/etc/etc/nessus/nessusd.conf, several options can be set. This is typically where we can specify the resources we want Nessus to use, the speed at which it should read data, etc. In our case, we do not change this file. Nessus provides a default one.

That is it. We start Nessus with the following command:

[]# nessud –D

Now, we must configure the client side. It is simpler as everything can be done through a graphic interface. By simply typing *nessus*, the following interface appears like in figure 5:



New session setup

Nessusd Host : localhost

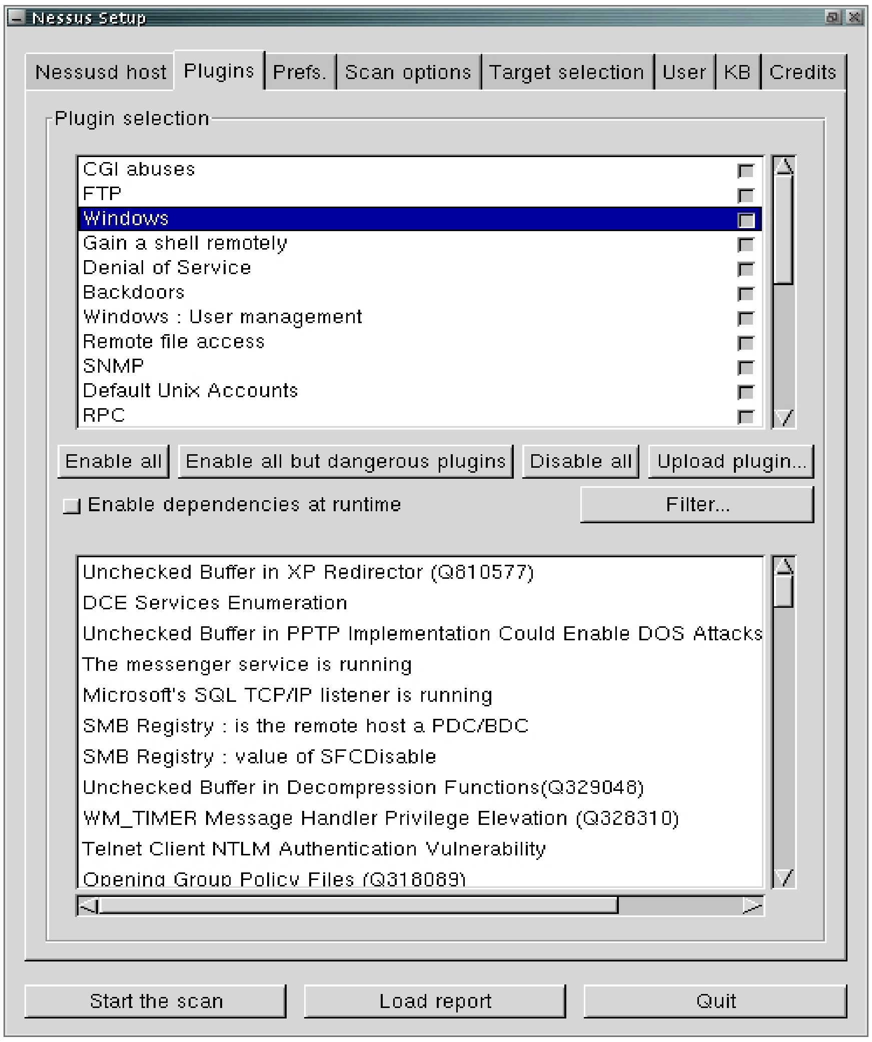
Port : 1241

Login : tintin

Password : ***

**Figure 5: Nessus Client configuration**

We simply connect to it as user *tintin*. Then, we can choose the tests (attack plugins) to perform on the remote target. It looks like figure 6:



CGI abuses
FTP
Windows
Gain a shell remotely
Denial of Service
Backdoors
Windows : User management
Remote file access
SNMP
Default Unix Accounts
RPC

Unchecked Buffer in XP Redirector (Q810577)
DCE Services Enumeration
Unchecked Buffer in PPTP Implementation Could Enable DOS Attacks
The messenger service is running
Microsoft's SQL TCP/IP listener is running
SMB Registry : is the remote host a PDC/BDC
SMB Registry : value of SFCDisable
Unchecked Buffer in Decompression Functions(Q329048)
WM_TIMER Message Handler Privilege Elevation (Q328310)
Telnet Client NTLM Authentication Vulnerability
Opening Group Policy Files (Q318089)

**Figure 6: Nessus client, attack plugins setup**

We are not really concerned by Nessus reports, as we only want to generate attacks that can trigger Snort alerts. We invite the interested reader to [Nessus] for more information of this tool reporting capabilities.

### *3.1.3 SNORT*

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 has three primary uses. It can be used as a straight packet sniffer like tcpdump, a packet logger (useful for network traffic debugging, etc), or as a full blown network intrusion detection system. We are interested in this last use. Snort uses a flexible rules language to describe traffic that it should collect or pass, as well as a detection engine that utilizes modular plugin architecture. Snort has a real-time alerting capability as well, incorporating alerting mechanisms for syslog, a user specified file, a UNIX socket, or WinPopup messages to Windows clients using Samba's smbclient. However, we are more interested in a specific output plugin that was recently released (v. 1.2.2a-2.1.0 released in November 2003). It has been tested with Snort version 2.1.0.

Therefore, we describe below the installation steps required to have Snort and libidmef running. The Operating System in use on 192.168.1.2 is Linux Red Hat 7.3.

First, the libidmef library installation is mandatory but very simple. Once downloaded, it suffices to untar/unzip the given file and to execute the makefile:

```
[]# tar –zxf libidmef-0.7.2.tar.gz
[]# cd libidmef-0.7.2
[libidmef-0.7.2]# ./configure
[libidmef-0.7.2]# make
[libidmef-0.7.2]# make install.
```

Secondly, we must decompress snort (v.2.1.0) and snort-idmef plugin like:

```
[]# tar –zxf snort-2.1.0.tar.gz
[]# tar –zxf snort-idmef-plugin-1.2.2alpha2.1.0.tar.gz
```

Then, some patches must be applied in Snort. A script is provided with the snort-idmef-plugin distribution (‘install-idmef.sh’). However, this can be done as follows:

```
[]# cd snort-idmef-plugin-1.2.2alpha2.1.0
[snort-idmef-plug..]# patch ../snort-2.1.0/configure.in configure.in.diff
[snort-idmef-plug..]# ../snort-2.1.0/src/plugbase.c src_plugbase.c.diff
[snort-idmef-plug..]# ../snort-2.1.0/src/plugin_enum.h src_plugin_enum.h.diff
[snort-idmef-plug..]# ../snort-2.1.0/src/output-plugins/Makefile.am src_output-plugins_Make
[snort-idmef-plug..]# cp spo_idmef.c spo_idmef.h ../snort-2.1.0/src/output-plugins/
```

Now, we have to run *autoconf* at snort’s root directory:

```
[snort-idmef-plug..]# cd ../snort-2.1.0
```

```
[snort-2.1.0]# autoconf
```

Finally, we can install Snort, following the traditional lines:

```
[snort-2.1.0]# ./configure –enable-idmef –with-libxml2-includes=/usr/include/libxml2 –with-mysql
[snort-2.1.0]# .make
[snort-2.1.0]# .make install
```

In addition, existing rules can be installed:

```
[]# .mkdir /etc/snort
[]# . cp snortrules-stable.tar.gz /etc/snort
[/etc/snort]# .tar –zxf snortrules-stable.tar.gz
[etc/snort]# .mv * ..
[/etc/snort]# rmdir rules
```

The configuration file */etc/snort/snort.conf* must be modified accordingly:

- the $RULE_PATH variable can be deleted. ("include XXXX.rules")
- We must specify the IDMEF output such as:

```
# idmef: log alerts to idmef format
#
output idmef: $HOME_NET logto=/var/log/snort/idmef_alerts.log analyzerid=109 dtd=/usr/local/etc/idmef-message.dtd output=log indent=true facility_default=file|idmef-messages.log alert_id=/var/log/alert_id_num
```

We should not forget to create the log directory:

```
[]# mkdir /var/log/snort
```

Snort is now operational. It can be executed with simple the command line:

```
[]# snort –c /etc/snort/snort.conf
```

### *3.2 Input Generation*

Once Snort and Nessus are installed, we can generate IDMEF alerts by launching attacks from 192.168.1.1. In the following, attacks are simple stealth scans. The idea consists in sending a TCP packet on well-chosen ports with all flags turned off. This is equivalent to the '*–sN'* scan mode of nmap or the '*–c 2'* option of hping [Nmap, Hping]. This approach was first described as a port scanning technique in [Fyo00]. The idea is that closed ports are required to reply to the probe packet with a RST, while open ports must ignore the packets in question (see RFC 793 pp. 64). Unfortunately, Microsoft IP/TCP layers do not behave as expected as they are often configured to send RST packets, independently of the port state (opened or closed). Thus, this scan type does not work against systems running Windows. As a result, this attack is currently used in active fingerprinting to determine Microsoft stations (see [Nmap] for further details).

Snort detects such an attack in stateful mode. Indeed, its preprocessor called stream4 provides a TCP stream reassembly and stateful analysis capabilities. Each three way handshake is recorded. Thus, when an incoming TCP packet is received, the preprocessor checks if it really closes an existing connection. Otherwise, an alert is generated, similar to the one illustrated below:

```
<IDMEF-Message/>
<?xml version="1.0"?>
<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFC XXXX IDMEF v1.0//EN" "/usr/local/etc/idmef-message.dtd">
<IDMEF-Message version="1.0">
  <Alert ident="289">
    <Analyzer analyzerid="109" model="snort" version="2.0.5">
      <Node>
        <name>chaplin</name>
      </Node>
    </Analyzer>
    <CreateTime ntpstamp="0xc36cc187.0xd3aa9b49">2003-11-24T17:42:31Z</CreateTime>
    <Source>
      <Node>
        <Address category="ipv4-addr">
          <address>192.168.1.1</address>
        </Address>
      </Node>
      <Service>
        <port>22</port>
        <protocol>tcp</protocol>
      </Service>
    </Source>
    <Target>
      <Node>
        <Address category="ipv4-addr">
          <address>192.168.1.3</address>
        </Address>
      </Node>
      <Service>
        <port>22</port>
        <protocol>tcp</protocol>
      </Service>
    </Target>
    <Classification origin="vendor-specific">
      <name>msg=(spp_stream4) STEALTH ACTIVITY (NULL scan) detection</name>
      <url>none</url>
    </Classification>
  </Alert>
</IDMEF-Message>
```

The alert is uniquely identified by the '*Alert ident'* attribute. The *service* section describes network services on targets. In our case, it contains two attributes, namely protocol (tcp) and port (22). The target node address is specified by the *target* element and the alert message is given by the

*Classification name* attribute. This alert simply reports a stealth scan on port 22 from 192.168.1.1 to 192.168.1.3.

### *3.3 SEC Configuration*

We decide to write a rule similar to the one presented in Annex A (SingleWithThreshold type) and apply it to existing alert logs. For each alert, we check two information patterns:

- the target address
- the alert message

Indeed, we find that stealth scan alerts are quite common. Thus, SEC can be used to correlate such alerts, and to issue a specific alert when the number of these alerts in a time window exceeds a certain threshold. We experiment two SEC features: its capability to read input streams online and offline. They are described in the following subsections.

#### *3.3.1 Offline log analysis*

We first try to apply SEC to offline log files, which were obtained previously, thanks to NESSUS. However, we realize that it is not so obvious to get target addresses with SEC. Indeed, rules are called for each incoming event. Each event is a line in the log file. So how can we only get the source address? A regular expression like:

“pattern=<address>(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})</address>”

is clearly not sufficient, as it would match both source and destination addresses. For instance:

```
type=Single
ptype=RegExp
continue=takenext
pattern =<address>(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})</address>
desc = Observed address $1
action= event

type=Single
ptype=RegExp
pattern= STEALTH ACTIVITY
desc = Stealth activity $1
action= logonly
```

These two rules lead to the following result:

```
# SEC output file
Wed Dec 17 17:15:35 2003: Simple Event Correlator version 2.2.beta2
Wed Dec 17 17:15:35 2003: Reading configuration from sec.conf
Wed Dec 17 17:15:35 2003: 2 rules loaded from sec.conf
Wed Dec 17 17:15:35 2003: Creating event ‘Observed address 192.168.1.1'
Wed Dec 17 17:15:35 2003: Creating event ‘Observed address 192.168.1.3'
Wed Dec 17 17:15:35 2003: Stealth activity 1
Wed Dec 17 17:15:35 2003: Creating event ‘Observed address 192.168.1.1'
Wed Dec 17 17:15:35 2003: Creating event ‘Observed address 192.168.1.3'
Wed Dec 17 17:15:35 2003: Stealth activity 1
Wed Dec 17 17:15:35 2003: Creating event ‘Observed address 192.168.1.1'
Wed Dec 17 17:15:35 2003: Creating event ‘Observed address 192.168.1.3'
Wed Dec 17 17:15:35 2003: Stealth activity 1
Wed Dec 17 17:15:35 2003: Creating event ‘Observed address 192.168.1.1'
Wed Dec 17 17:15:35 2003: Creating event ‘Observed address 192.168.1.3'
Wed Dec 17 17:15:35 2003: Stealth activity 1
Wed Dec 17 17:15:35 2003: Creating event ‘Observed address 192.168.1.1'
Wed Dec 17 17:15:35 2003: Creating event ‘Observed address 192.168.1.3'
Wed Dec 17 17:15:35 2003: Stealth activity 1
```

It illustrates our previous remark. The rule configuration file is checked for each log line. An additional work would be possible: it consists in analyzing SEC output (with SEC?) and group information per attack as (ip_src, ip_dst, Stealth activity 1).Furthermore, it seems difficult to correlate alerts within some time constraints. Indeed, the timestamp pattern needs to be extracted so that an additional script records and analyzes it. Consequently, writing SEC rules with time correlation constraints is not practical.

We would have wanted SEC to see each IDMEF alert as one event. However, this is not possible at this stage. Consequently, we decided to write the whole IDMEF alert on one line, thanks to a simple PERL concatenation file. The IDMEF alerts are then similar to figure 6:

**Figure 6: IDMEF alert (on one line)**

```
<IDMEF-Message/><?xml version="1.0"?>
<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFC XXXX IDMEF v1.0//EN" "/usr
/local/etc/idmef-message.dtd"><IDMEF-Message version="1.0"><Alert ident="289"
><Analyzer analyzerid="109" model="snort" version="2.0.5"><Node><name>chapli
n</name></Node></Analyzer><CreateTime ntpstamp="0xc36cc187.0xd3aa9b49">2003-11-
24T17:42:31Z</CreateTime><Source><Node><Address category="ipv4-addr">
<address>192.168.1.1</address></Address></Node><Service><port>22</port>
<protocol>tcp</protocol></Service></Source><Target><Node><Address
category="ipv4-addr"><address>192.168.1.3</address></Address></N
ode><Service><port>22</port><protocol>tcp</protocol></Service>
</Target><Classification origin="vendor-specific"><name>ms
g=(spp_stream4) STEALTH ACTIVITY NULL scan) detection</name><url>
none</url></Classification></Alert></IDMEF-Message>
```

It does not really simplify the whole system, as shown in figure 7. However, this is simpler to write SEC rules. One event corresponds to one alert.

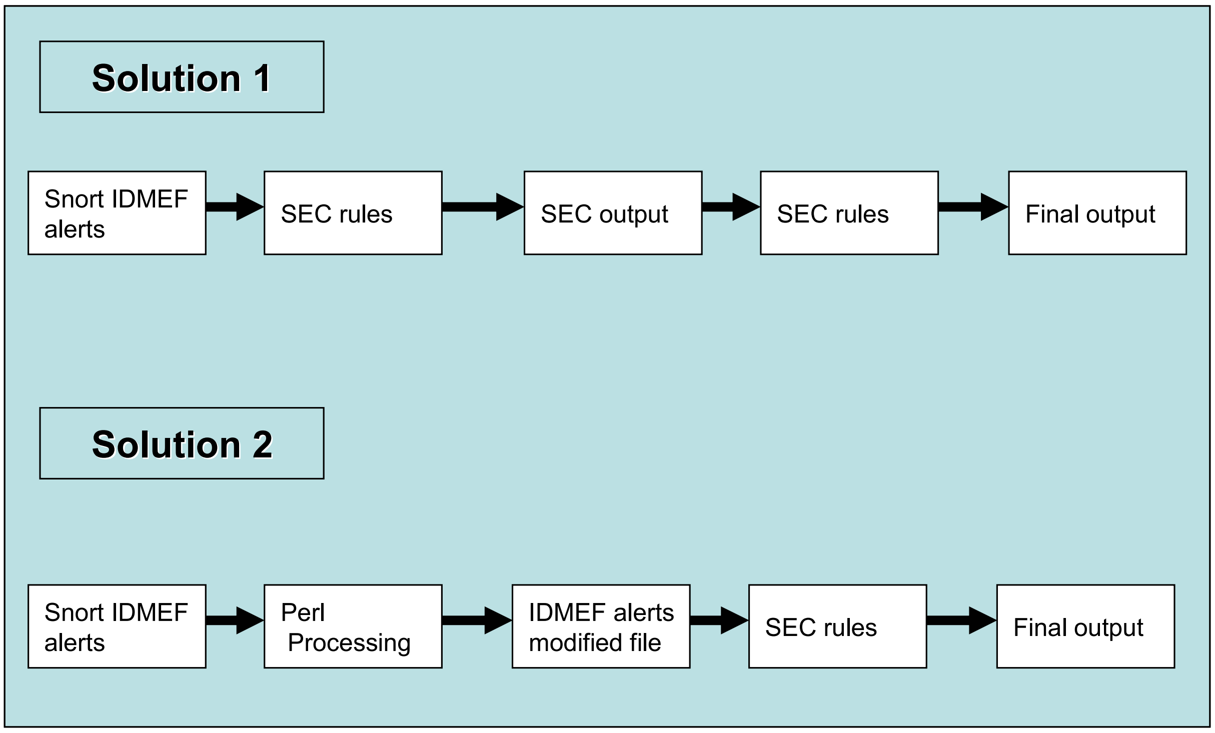


**Figure 7: SEC options we tested**

A simple configuration file would be:

```
##  rules, to be placed in sec.conf file
##
## first rule to get the target IP of our machine
type=SingleWithThreshold
ptype=RegExp
pattern =   <Source><Node><Address category="ipv4-addr">
            <address>(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})</address>
            </Address></Node><Service><port>22</port>
            <protocol>tcp</protocol></Service></Source><Target>
            <Node><Address category="ipv4-addr">
            <address>(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})</address>
            </Address></Node><Service><port>22</port>
            <protocol>tcp</protocol></Service></Target>
            <Classification origin="vendor-specific">
            <name>msg=(spp_stream4)
           STEALTH ACTIVITY (NULL scan) detection</name>
desc= STEATH Port scan from $1to $2
action=logonly
window=300
thresh=3
```

It is working this way. SEC generates two lines in the terminal window (default output) as the alert file contains more than six alerts containing this pattern.

```
# SEC output file
Wed Dec 17 17:25:55 2003: Simple Event Correlator version 2.2.beta2
Wed Dec 17 17:25:55 2003: Reading configuration from sec.conf
Wed Dec 17 17:25:55 2003: 2 rules loaded from sec.conf
Wed Dec 17 17:25:55 2003: Stealth activity 1 from 192.168.1 to 192.168.3
Wed Dec 17 17:25:55 2003: Stealth activity 1 from 192.168.1 to 192.168.3
```

The time window is useless in this case as everything is done offline. We are limited by the pattern field. One solution consists in breaking the rule into three smaller rules: two dedicated to the source and target addresses, and one for the alert message. The *takeNext* parameter allows such a rule cascade.

A similar rule was developed for snort portscan alerts (see Annex B). However, the previously presented one is richer as we manage to get the source address. This rule can be used to aggregate stealth portscan alerts, when they exceed a certain number (or threshold).

In our example, we generated alert scans at different frequencies to test the threshold. It is reinitialized each time its value is equal to the given parameter.

### 3.3.2 Online log analysis

The experiment is similar to the previous one. SEC is applied to Snort output file, while Snort is still logging alerts. We try the last rule on it, but we obviously face the same problem. Each new alert comes in the log file as multiple lines. Consequently, the rule is inefficient and never matches the expected pattern. One solution consists in applying SEC dynamically to an intermediate file: a Perl script is in charge of modifying snort output file, so that each IDMEF alert appears in one line (one SEC event). However, this is not convenient at all, and the traditional snort alert format seems more adapted to SEC parsing. We tried this solution. Results are very similar to the offline mode. The only change is the time information in the SEC output file:

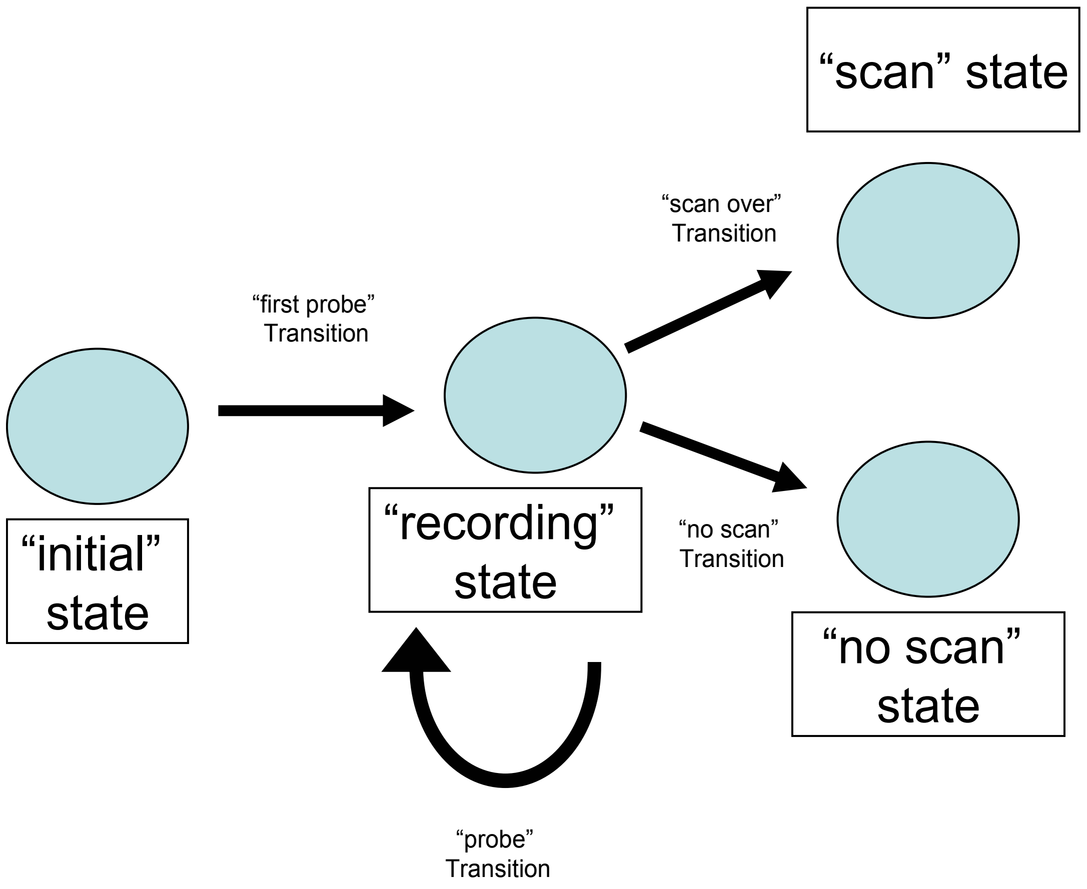
```
# SEC output file
Wed Dec 17 18:38:50 2003: Simple Event Correlator version 2.2.beta2
Wed Dec 17 18:38:50 2003: Reading configuration from sec.conf
Wed Dec 17 18:38:50 2003: 2 rules loaded from sec.conf
Wed Dec 17 18:41:15 2003: Stealth activity 1 from 192.168.1 to 192.168.3
Wed Dec 17 18:41:15 2003: Stealth activity 1 from 192.168.1 to 192.168.3
Wed Dec 17 18:41:15 2003: Stealth activity 1 from 192.168.1 to 192.168.3
Wed Dec 17 18:44:25 2003: Stealth activity 1 from 192.168.1 to 192.168.3
Wed Dec 17 18:44:25 2003: Stealth activity 1 from 192.168.1 to 192.168.3
```

10 snort stealth scans were detected at 18:41:15. However, only three SEC alerts were written corresponding to the first 9 events (threshold=3). Then, 5 stealth scans were sent by snort at 18:44:25. The window size was set to 300 ms (not expired since 18:41:15), so SEC took into account the last tenth event, and wrote two lines.

## 3.4 AlertSTAT Configuration

We need a scenario plugin (or module, depending on the STAT documents): A scenario plugin is a shared library that describes an attack scenario. The scenario plugin is usually generated from a STATL description, but could be theoretically developed manually. Our first idea was to design our own scenario. Seven scenarios are provided with AlertSTAT distribution and we tried to build something similar (One scenario, named 'Portscan' is presented in Annex C').

Our first remark is that none of them is documented. There is no state-transition graph or any other comment that would illustrate the scenarios. The documentation systematically refers to a non existing pdf file. Our second remark concerns the scenario complexity. On one hand, the theoretical scenario creation is quite easy. It suffices to write a file, following the STATL syntax. This file has a '.stat'extension. Then, it is compiled into modules. Finally, a link to this module must be added to the alertSTAT configuration file, in order to have the new scenario operational. Surprisingly enough, there is no '.stat'file provided in the AlertSTAT distribution (nor in the STAT web site, except a test.stat file given in Annex E). More strangely, compiled modules are written in C++, while STAT core is written in Java. For instance, the idmef_portscan module file is presented in Annex C. From the C++ declarations, we can hardly draw back the scenario graph, which should be similar to:



Disappointed by the scenario writing complexity (or more precisely its obscurity), we decide to test this already built scenario. It is already included in the AlertSTAT *scenario* file (file which specifies activated scenarios), so the operation does not require lots of efforts. The scenario module needs three parameters that are specified in the same *scenario* file like:

```
<IDMEF-Message>
<x-stat from="" to="">

<x-stat-scenario-load id="8" name="idmef_portscan" version="1.0"
     library="idmef_portscan.so">
</x-stat-scenario-load>
<x-stat-scenario-activate id="10" scenario_name="idmef_portscan"
     prototype_name="prototype_idmef_portscan">
<parameter name="threshold">3</parameter>
<parameter name="timeout">20</parameter>
<parameter name="flood_threshold">500</parameter>
</x-stat-scenario-activate>
```

To obtain a definition of these variables, one needs to plunge into the C++ code copied in Annex C. The timeout defines the attack time upper bound limit (in seconds) from the first received event. The threshold defines the lower-bound value of events number before considering there is a scan attempt. The flood_threshold variable sets the upper-bound value of events number before considering a flooding attack.

To test alertSTAT on our log files, we type the simple command:

```
[]# alertstat –alertfile <path_to_alert_file>
```

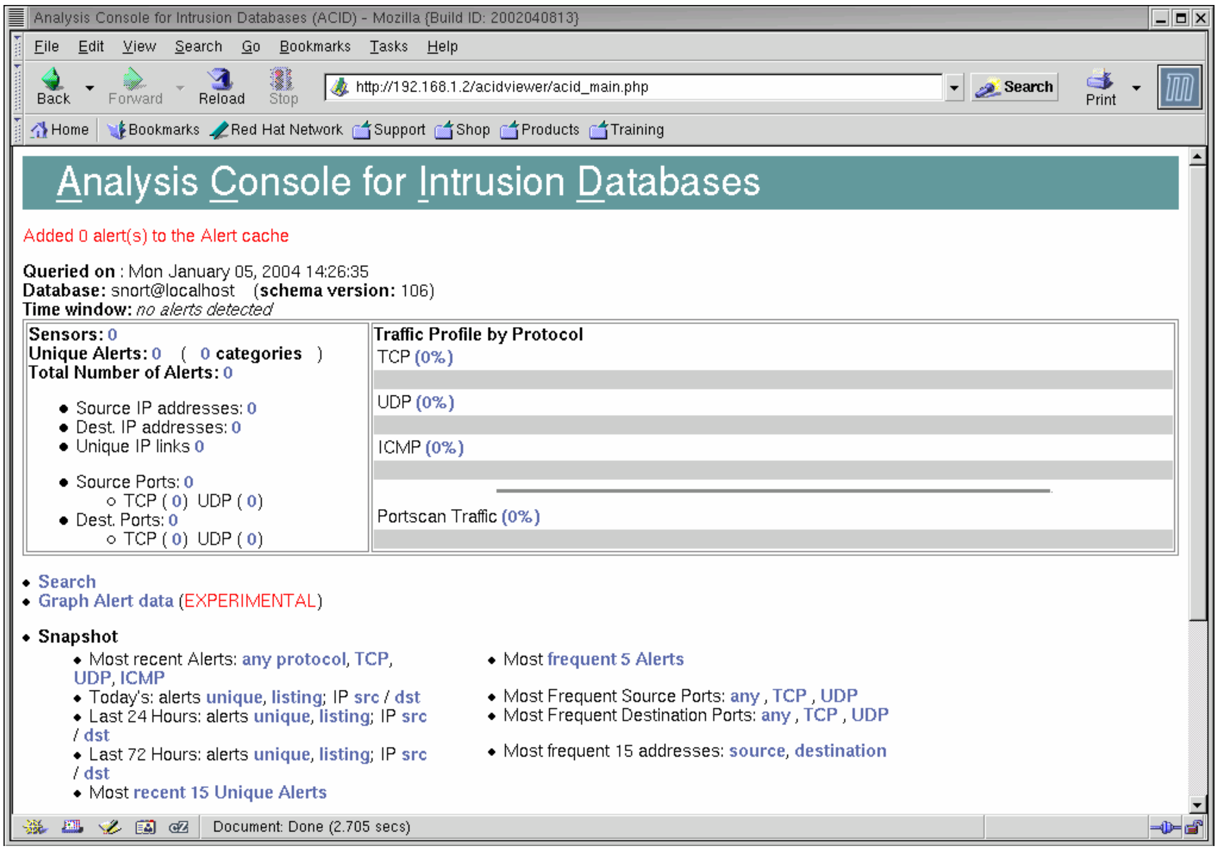
Finally, AlertSTAT results are obtained by default in the /usr/local/start/responses file. This can be changed in the etc/alertstat.cfg file. They are written as IDMEF-messages:

```
<IDMEF-Message>
<x-stat from="" to="">
<x-stat-response-load id="1" name="response_IDMEF" version="1.0" library="libresponse.so"/>
<x-stat-response-activate id="7" response_name="response_IDMEF"
                    response_func="response_send" prototype_name="prototype_idmef_portscan" scenario_state="scan"/>
(…)
</x-stat>
</IDMEF-Message>
```

The most interesting part is the *scenario_state* value, which indicates the current state of the system from the given scenario (prototype_idmef_portscan). SEC can be used online to detect such final states and beep or email the administrator.

## *3.5 ACID Configuration*

There is nothing to be done with ACID. Snort information is logged into the mysql database (output defined snort.conf file). Moreover, ACID is connected to this database to send queries. This is a background activity we are not really aware of, as we only interact with the ACID GUI. It looks like:


Analysis Console for Intrusion Databases (ACID) - Mozilla {Build ID: 2002040813}
File Edit View Search Go Bookmarks Tasks Help
Back Forward Reload Stop
http://192.168.1.2/acidviewer/acid_main.php
Search Print
Home Bookmarks Red Hat Network Support Shop Products Training
Document: Done (2.705 secs)

Analysis Console for Intrusion Databases

Added 0 alert(s) to the Alert cache

**Queried on** : Mon January 05, 2004 14:26:35
**Database:** snort@localhost (**schema version:** 106)
**Time window:** *no alerts detected*

| | |
|---|---|
| **Sensors:** 0<br>**Unique Alerts:** 0 ( 0 **categories** )<br>**Total Number of Alerts:** 0<br>• Source IP addresses: 0<br>• Dest. IP addresses: 0<br>• Unique IP links 0<br>• Source Ports: 0<br>  ○ TCP ( 0) UDP ( 0)<br>• Dest. Ports: 0<br>  ○ TCP ( 0) UDP ( 0) | **Traffic Profile by Protocol**<br>TCP (0%)<br>UDP (0%)<br>ICMP (0%)<br>Portscan Traffic (0%) |

- Search
- Graph Alert data (EXPERIMENTAL)
- **Snapshot**
  - Most recent Alerts: any protocol, TCP, UDP, ICMP
  - Today's: alerts unique, listing; IP src / dst
  - Last 24 Hours: alerts unique, listing; IP src / dst
  - Last 72 Hours: alerts unique, listing; IP src / dst
  - Most recent 15 Unique Alerts
  - Most frequent 5 Alerts
  - Most Frequent Source Ports: any , TCP , UDP
  - Most Frequent Destination Ports: any , TCP , UDP
  - Most frequent 15 addresses: source, destination

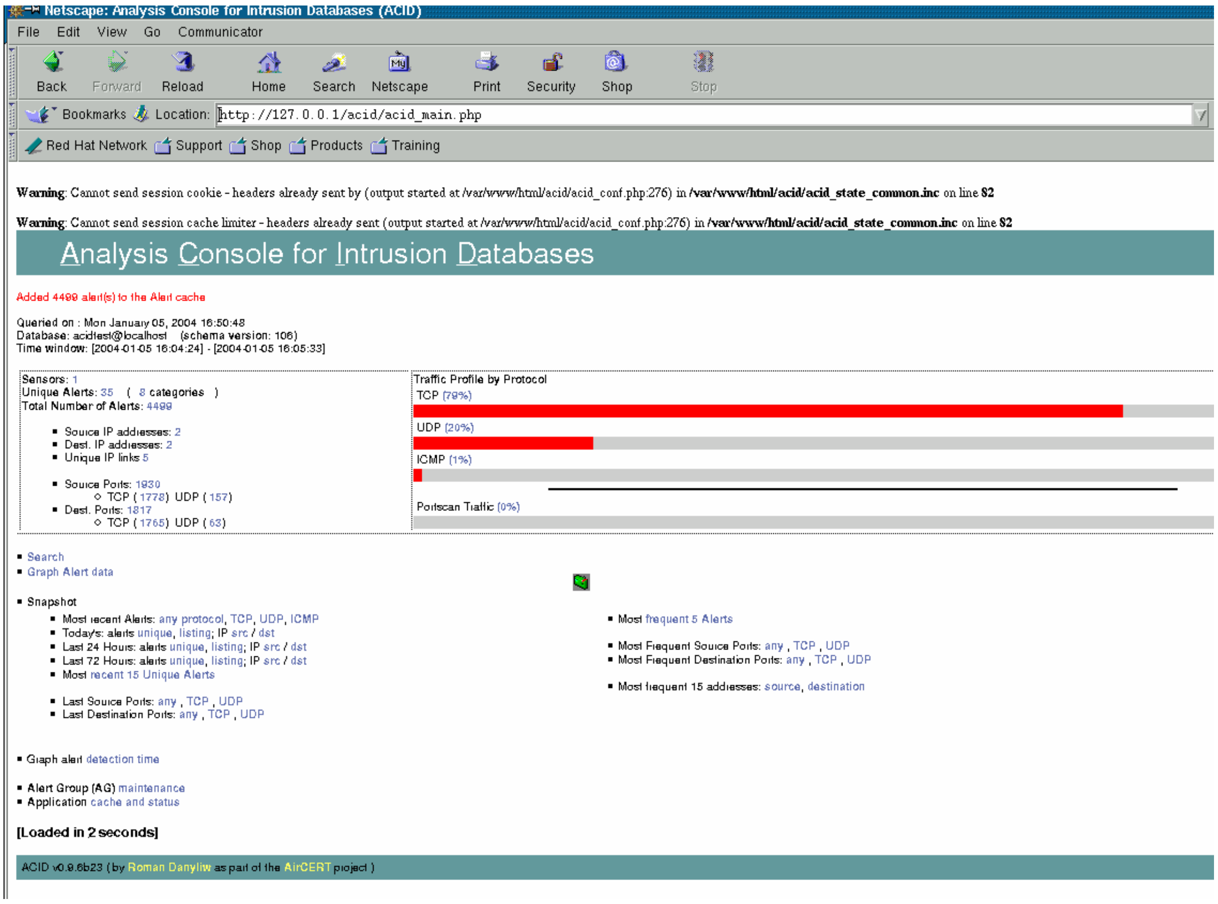
**Figure 8: ACID home page**

Figure 8 shows the ACID home page. Here is summarized general information on the corresponding database, such as the traffic profile by protocol (limited by snort to tcp, udp and icmp transport protocols). In our example, the database is still empty, as illustrated by null values. In general cases, it provides the following information:

- The number of distinct source/destination IP addresses
- The number of distinct source/destination ports for each protocol (UDP or TCP)
- The list of most frequent alerts
- Some snapshots of the database.

In figure 9 is presented the same home page, but with a non-empty database. These logs correspond to the Nessus traffic observed from 192.168.1.2. In the Nessus client configuration, we ticked by the option “all attacks”. As a consequent, we can determine that 79% of the Nessus total traffic is TCP, 20% is UDP and 1% is ICMP.

There are obviously two source and destination addresses observed (192.168.1.1 and 192.168.1.3), and 35 unique alerts (coming from 8 snort rule sets categories).


Netscape: Analysis Console for Intrusion Databases (ACID)
File Edit View Go Communicator
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://127.0.0.1/acid/acid_main.php
Red Hat Network Support Shop Products Training

**Warning**: Cannot send session cookie - headers already sent by (output started at /var/www/html/acid/acid_conf.php:276) in **/var/www/html/acid/acid_state_common.inc** on line **82**

**Warning**: Cannot send session cache limiter - headers already sent (output started at /var/www/html/acid/acid_conf.php:276) in **/var/www/html/acid/acid_state_common.inc** on line **82**

Analysis Console for Intrusion Databases

Added 4499 alert(s) to the Alert cache

Queried on : Mon January 05, 2004 16:50:48
Database: acidtest@localhost (schema version: 106)
Time window: [2004-01-05 16:04:24] - [2004-01-05 16:05:33]

Sensors: 1
Unique Alerts: 35 ( 8 categories )
Total Number of Alerts: 4499

- Source IP addresses: 2
- Dest. IP addresses: 2
- Unique IP links 5

- Source Ports: 1930
  - TCP ( 1778) UDP ( 157)
- Dest. Ports: 1817
  - TCP ( 1765) UDP ( 63)

Traffic Profile by Protocol
TCP (79%)
UDP (20%)
ICMP (1%)
Portscan Traffic (0%)

- Search
- Graph Alert data

- Snapshot
  - Most recent Alerts: any protocol, TCP, UDP, ICMP
  - Today's: alerts unique, listing; IP src / dst
  - Last 24 Hours: alerts unique, listing; IP src / dst
  - Last 72 Hours: alerts unique, listing; IP src / dst
  - Most recent 15 Unique Alerts
  - Last Source Ports: any , TCP , UDP
  - Last Destination Ports: any , TCP , UDP
  - Most frequent 5 Alerts
  - Most Frequent Source Ports: any , TCP , UDP
  - Most Frequent Destination Ports: any , TCP , UDP
  - Most frequent 15 addresses: source, destination

- Graph alert detection time

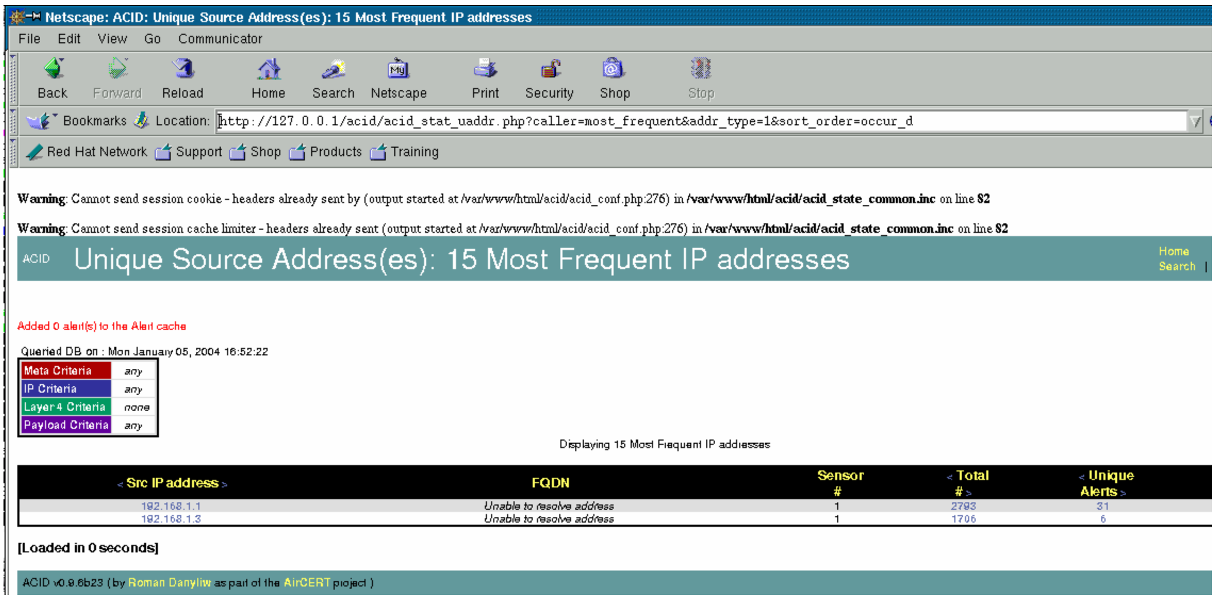
- Alert Group (AG) maintenance
- Application cache and status

**[Loaded in 2 seconds]**

ACID v0.9.6b23 ( by Roman Danyliw as part of the AirCERT project )

**Figure 9: another ACID home page**

With no real surprise, we obtain figure 10, while clicking on ‘source IP addresses’. All IP sources are listed, with their corresponding number of events and alerts.



Netscape: ACID: Unique Source Address(es): 15 Most Frequent IP addresses
File Edit View Go Communicator
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location:
Red Hat Network Support Shop Products Training

```
http://127.0.0.1/acid/acid_stat_uaddr.php?caller=most_frequent&addr_type=1&sort_order=occur_d
```

**Warning**: Cannot send session cookie - headers already sent by (output started at /var/www/html/acid/acid_conf.php:276) in **/var/www/html/acid/acid_state_common.inc** on line **82**

**Warning**: Cannot send session cache limiter - headers already sent (output started at /var/www/html/acid/acid_conf.php:276) in **/var/www/html/acid/acid_state_common.inc** on line **82**

ACID Unique Source Address(es): 15 Most Frequent IP addresses

Home | Search

Added 0 alert(s) to the Alert cache

Queried DB on : Mon January 05, 2004 16:52:22

| | |
|---|---|
| Meta Criteria | any |
| IP Criteria | any |
| Layer 4 Criteria | none |
| Payload Criteria | any |

Displaying 15 Most Frequent IP addresses

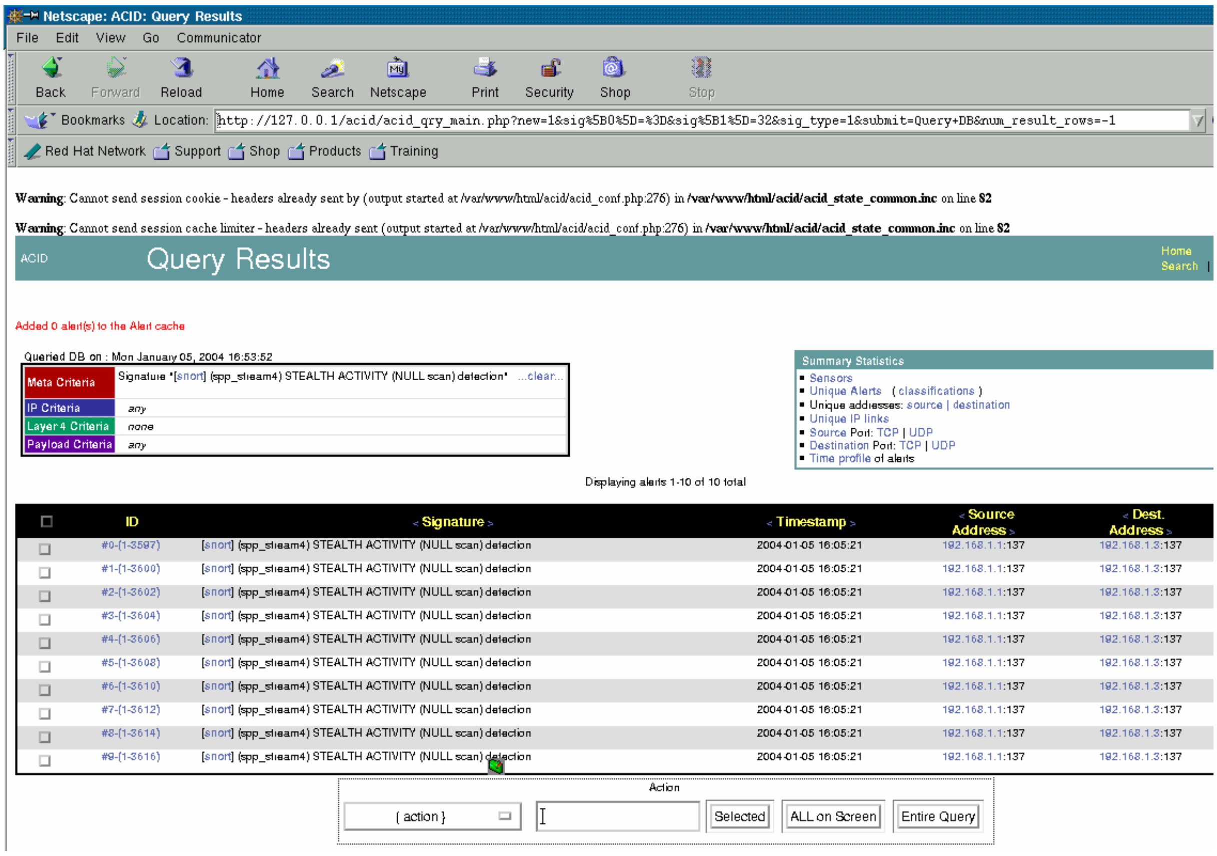
| Src IP address | FQDN | Sensor # | Total # | Unique Alerts |
|---|---|---|---|---|
| 192.168.1.1 | Unable to resolve address | 1 | 2793 | 31 |
| 192.168.1.3 | Unable to resolve address | 1 | 1706 | 6 |

[Loaded in 0 seconds]

ACID v0.9.6b23 ( by Roman Danyliw as part of the AirCERT project )

Figure 10: ACID source IP addresses

We worked in the previous experiments on a given Nessus attack: the stealth scan on port 22. Snort detects such attacks thanks to its spp_stream4 preprocessor module. They are represented in ACID as follows:



Netscape: ACID: Query Results
File Edit View Go Communicator
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location:
Red Hat Network Support Shop Products Training

```
http://127.0.0.1/acid/acid_qry_main.php?new=1&sig%5B0%5D=%3D&sig%5B1%5D=32&sig_type=1&submit=Query+DB&num_result_rows=-1
```

**Warning**: Cannot send session cookie - headers already sent by (output started at /var/www/html/acid/acid_conf.php:276) in **/var/www/html/acid/acid_state_common.inc** on line **82**

**Warning**: Cannot send session cache limiter - headers already sent (output started at /var/www/html/acid/acid_conf.php:276) in **/var/www/html/acid/acid_state_common.inc** on line **82**

ACID Query Results

Home | Search

Added 0 alert(s) to the Alert cache

Queried DB on : Mon January 05, 2004 16:53:52

| | |
|---|---|
| Meta Criteria | Signature "[snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection" ...clear... |
| IP Criteria | any |
| Layer 4 Criteria | none |
| Payload Criteria | any |

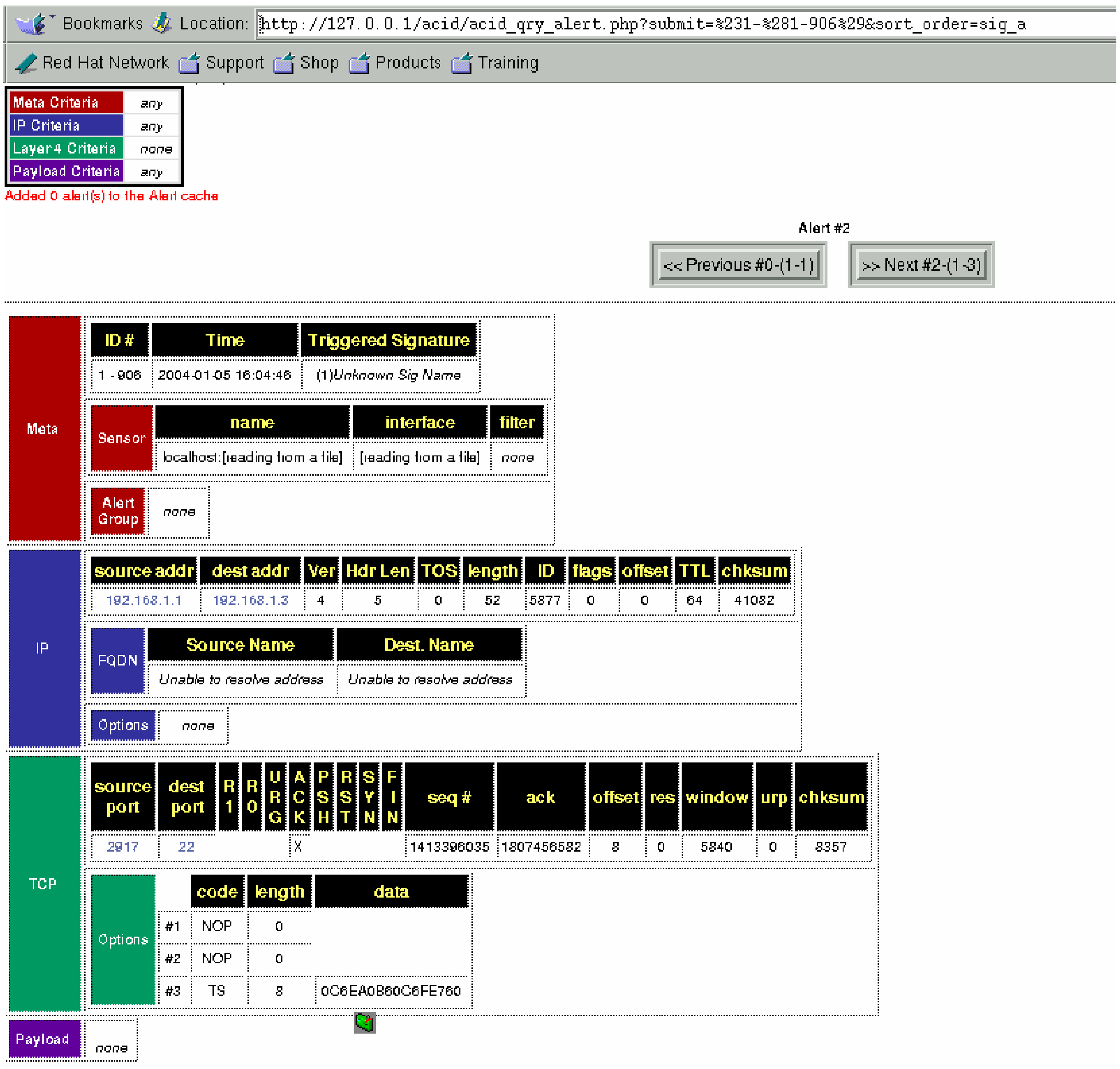
Summary Statistics
- Sensors
- Unique Alerts ( classifications )
- Unique addresses: source | destination
- Unique IP links
- Source Port: TCP | UDP
- Destination Port: TCP | UDP
- Time profile of alerts

Displaying alerts 1-10 of 10 total

| ☐ | ID | Signature | Timestamp | Source Address | Dest. Address |
|---|---|---|---|---|---|
| ☐ | #0-(1-3597) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-01-05 16:05:21 | 192.168.1.1:137 | 192.168.1.3:137 |
| ☐ | #1-(1-3600) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-01-05 16:05:21 | 192.168.1.1:137 | 192.168.1.3:137 |
| ☐ | #2-(1-3602) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-01-05 16:05:21 | 192.168.1.1:137 | 192.168.1.3:137 |
| ☐ | #3-(1-3604) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-01-05 16:05:21 | 192.168.1.1:137 | 192.168.1.3:137 |
| ☐ | #4-(1-3606) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-01-05 16:05:21 | 192.168.1.1:137 | 192.168.1.3:137 |
| ☐ | #5-(1-3608) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-01-05 16:05:21 | 192.168.1.1:137 | 192.168.1.3:137 |
| ☐ | #6-(1-3610) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-01-05 16:05:21 | 192.168.1.1:137 | 192.168.1.3:137 |
| ☐ | #7-(1-3612) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-01-05 16:05:21 | 192.168.1.1:137 | 192.168.1.3:137 |
| ☐ | #8-(1-3614) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-01-05 16:05:21 | 192.168.1.1:137 | 192.168.1.3:137 |
| ☐ | #9-(1-3616) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-01-05 16:05:21 | 192.168.1.1:137 | 192.168.1.3:137 |

Figure 11: ACID, Stealth Scan on port 22

If we click on the ‘snort’ word (in blue at each alert lines), we get the rule-description page currently available at Snort web site. Additionally, a click on the event ID (left column) gives more complete information on the selected event, such as the different protocol headers and payloads. One example is presented in figure 12, with one event listed in figure 11.



```
http://127.0.0.1/acid/acid_qry_alert.php?submit=%231-%281-906%29&sort_order=sig_a
```

| | |
|---|---|
| Meta Criteria | any |
| IP Criteria | any |
| Layer 4 Criteria | none |
| Payload Criteria | any |

Added 0 alert(s) to the Alert cache

Alert #2

<< Previous #0-(1-1)   >> Next #2-(1-3)

Meta

| ID # | Time | Triggered Signature |
|---|---|---|
| 1 - 906 | 2004-01-05 16:04:46 | (1)Unknown Sig Name |

| Sensor | name | interface | filter |
|---|---|---|---|
| | localhost:[reading from a file] | [reading from a file] | none |

| Alert Group | none |
|---|---|

IP

| source addr | dest addr | Ver | Hdr Len | TOS | length | ID | flags | offset | TTL | chksum |
|---|---|---|---|---|---|---|---|---|---|---|
| 192.168.1.1 | 192.168.1.3 | 4 | 5 | 0 | 52 | 5877 | 0 | 0 | 64 | 41082 |

| FQDN | Source Name | Dest. Name |
|---|---|---|
| | Unable to resolve address | Unable to resolve address |

| Options | none |
|---|---|

TCP

| source port | dest port | R1 | R0 | URG | ACK | PSH | RST | SYN | FIN | seq # | ack | offset | res | window | urp | chksum |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2917 | 22 | | | | X | | | | | 1413396035 | 1807456582 | 8 | 0 | 5840 | 0 | 8357 |

| Options | | code | length | data |
|---|---|---|---|---|
| | #1 | NOP | 0 | |
| | #2 | NOP | 0 | |
| | #3 | TS | 8 | 0C6EA0B60C6FE760 |

| Payload | none |
|---|---|

**Figure 11: ACID event details**

As a more general remark, it is quite surprising to see the ACK flag set in the TCP header of the presented packet. This packet triggers the NULL Scan alert of Snort. However, a NULL Scan is defined as: “all flags are turned off” [Nmap]. Thus, does it mean that NULL Scan definition of Snort differs from the previous one, or does it simply reveal another bug in the spp_stream4 preprocessor?

# 4 Observations

## *4.1 Results*

The three tools were configured to accomplish a given task, and they all did it. However, we can make several observations for each of the tools:

- About SEC: we would like to point that SEC is not really adapted to complex log analysis. It is a convenient Swiss Army knife to help finding information, but it is definitely not designed to perform deep analysis. As we showed in Section 3.2, SEC is not really suited to IDMEF standard. It would even become more interesting when it is used with traditional snort alerting format. Furthermore, SEC is more adapted to online log analysis. In offline modes, its time windowing capabilities are useless (except for a real time replay). As a consequence, we would advocate SEC usage in very specific cases: this tool can be incorporated in another richer tool that would use its rules flexibility, or to answer a specific question that cannot be found with traditional consoles like ACID. Even in this case, we are not convinced that it can replace a simpler Perl script.
- About AlertSTAT: AlertSTAT was initially classified in [PoDa03a] in the 'Experimental tool' category. We are more and more convinced of our classification after our experiments. Indeed, AlertSTAT is theoretically interesting, but its usage is currently restricted, for two main reasons:
    - The AlertSTAT package is incomplete. The installation files refer to documents that do not exist. Many files are missing (with the furnished examples), which make the tool even less comprehensive.
    - Writing and using a scenario is potentially possible. However, we think that a larger scenario database should be distributed with this tool. The provided examples are not really interested, as they can often be replaced by simpler analysis tools. A larger scenarii database would permit to use AlertSTAT as a 'meta-alert' generator.
- About ACID: ACID is the most famous alert correlation console. Designed along with Snort, it is widely used by many network administrators to prevent them from being overwhelmed

with Snort alerts. ACID can be easily used to find simple 'snapshots' of the IDS output database. Generally speaking, it is convenient in a multi-sensors network to analyze centralized alerts. However, its simplicity is also its limitation. It prevents the administrator from tediously querying the database. But it does not support real complex queries. In this case, the solution consists in directly logging to the database and write down queries. Moreover, Snort database is not really adapted to complex queries, and a new database scheme must be considered.

## *4.2 Comparison*

SEC is ideally suited for performing real-time monitoring. While it can take offline log file as input (see Section 3.2), it has really been designed to process active log files. SEC excels at event aggregation. It is easily configured to detect multiple similar events and report them as a single composite event, thereby reducing the amount of data the analyst has to review.

SEC has a facility for real-time notification. It can feed reports to any program or script that is capable of processing file streams. It can send email, write to a file, or send pager notifications.

Despite its many good points, SEC does have its drawbacks- namely its complexity and limited installation base. The learning curve for SEC is steep, and while it is fairly well documented, writing relevant rules may require some time. Furthermore, since SEC was originally intended for use with Network management systems such as HP OpenView, the amount of Snort-specific information available is more limited. Finally, we are convinced that IDMEF format is too complex and SEC would perform as well with traditional format. Using IDMEF standard implies extra-processing of the output file. Furthermore, pattern matching on long lines is not convenient, and we are often compelled to break each rule into sub-rules in order to clarify rules and to avoid mistakes.

On the other side, AlertSTAT would be more interesting for complex scenarios which can hardly be described by the mean of SEC rules or database requests. However, there is no complex scenario offered with the tool package. AlertSTAT is provided with seven simple and not documented ones. Furthermore, it seems difficult to write a multitude of them. Even if Vigna et Al. praise the STAT framework in [Vign03], we find their tools currently not practical. The concept itself is interesting

However, the solutions they freely distribute in their site are hardly usable. Moreover, they promise scenario documentation in their tool distribution which is non-existent. A large scenario library would make this tool more applicable. As far as we know, such a library does not exist yet. This is tedious work for one network administrator.

To conclude, we would say that the three alert correlation tools we analyzed do not provide redundant information. On the contrary, they are more or less complementary and do not fulfill the same tasks. As we show for illustration in Section 3.4, we can use SEC to analyze alertSTAT output.

Table 4 summarizes their usages. We find that four main criteria may help choosing between the three tools:

- The input format
- The tool usage
- The easiness changing tool configuration.
- Correlation based on some time properties

A cross means that the tool is more adapted to this criteria category.

| **Criteria** | **Criteria categories** | **SEC** | **ACID** | **AlertSTAT** |
|---|---|---|---|---|
| **Input format** | snort alert format | X | X | |
| | IDMEF format | | | X |
| **Usage** | Simple queries on alerts files or databases | X | X | |
| | Complex queries – meta-alert information | | | X |
| **Tools:** | Frequent modifications | | X | |

| | | | | |
|---|---|---|---|---|
| **Configuration modifications** | Rare modifications | X | | X |
| **Correlation with time constraints** | Permits time constraints | X | | X |
| | Does not permit time constraints | | X | |

Some basic correlation operations are presented in [PoDa03]. We report the interested reader to this document for more information on their definition. We suggest specifying for each tool which operations they are adapted to:

| **Basic correlation operations** | **SEC** | **AlertSTAT** | **ACID** |
|---|---|---|---|
| Compression | X | | X |
| Filtering | | | X |
| Selective Suppression | | | |
| Thresholding | X | X | |
| Modification | X | | |
| Generalization | X | X | |
| Specialization | | | |
| Enrichment | | X | |

We observe that the three tools cover distinct operation groups. For instance, SEC can handle many basic operations, thanks to its flexibility. However, it might be quite hard to write its corresponding rules. On the other hand, ACID is very limited. It is restricted to simple filtering and compression operations. Finally, we note that 'Selective Suppression' and 'Specialization' operations are not covered.

## 5 Conclusion

Alerts often come from multiple sensors, spanning multiple complex subsystems. This complexity implies that such systems require constant monitoring and maintenance. Human capacities are not sufficient, and some tools try to address their issues.

In this paper, we have presented and evaluated three of them, respectively SEC, AlertSTAT and ACID. They all have characteristics that make them original and useful. SEC, for instance, is not particularly adapted to IDMEF standard, but can be used within a more specified tool. AlertSTAT tool lacks of maintenance, but is very promising. It can generate some meta-alerts that would be impossible to obtain with traditional tools. Finally, ACID is a very basic console. We are not convinced that it really helps correlating alerts. It gives a better overview of centralized alerts, but performs limited requests on the database. In other words, this is a 'convenient but simple tool for database mining'. Moreover, it is built on Snort database which is not really optimized for complex sql queries. As a consequence, we think it is the least promising tool of those we tested.

With regards to these experiments, we conclude that so-called alert-correlation tools have not reached a satisfactory mature level. Many solutions exist with interesting features. Thus, the next step consists in grouping them into a more coherent correlation architecture. However, we are not sure this trend is prevalent today.

## 6 References

| | |
|---|---|
| [ADO] | Abodb home page: http://php.weblogs.com/adodb/ |
| [Ale] | AlertSTAT home page: http://www.cs.ucsb.edu/~rg/STAT/software/alertstat.html |
| [Bro03] | J. Brown. “Working with SEC- the Simple Event Correlator”, 2003. Available at: http://sixhooter.v6.thrupoint.net/SEC-examples/article.html |
| [Cla03] | N. Clarks. “Perl regular expressions tutorial”, in *perlretut*, 2003. Available at: http://search.cpan.org/ñwclark/perl-5.8.2/pod/perlretut.pod |
| [Gd] | Gd home page: http://www.boutell.com/gd/ |
| [Idmef] | Intrusion Detection Message Exchange format. Draft 10 available at http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-10.txt |
| [Idwg] | Intrusion Detection Working Group (IDWG) home page: http://www.ietf.org/html.charters/idwg-charter.html |
| [Nessus] | Nessus home page: http://www.nessus.org |
| [Nmap] | Nmap security scanner home page: http://www.insecure.org/nmap/ |
| [PHPlot] | PHPlot home page : http://www.phplot.com |
| [PoDa03] | F. Pouget, M. Dacier. “White Paper: Alert Correlation: Review of the state of the art”. Insitute Eurecom, Sophia-Antipolis, France. Nov. 2003. |
| [RHupd] | RedHat Modules updates available at: ftp://updates.redhat.com/7.3/ |
| [SEC] | SEC home pages: http://simple-evcorr.sourceforge.net or http://www.estpak.ee/~risto/sec/ |
| [Snorta] | Snort rules available at: http://snort.org/dl/signatures/ |
| [Snortb] | Snort home page: http://www.snort.org |
| [Vign03] | G. Vigna, F. Valeur, R.A. Kemmerer. “Designing and Implementing a Family of Intrusion Detection Systems”. In Proceedings of ESEC/FSE’03, Finland. Sept. 2003. |

# 7 Annexes

## 7.1 *Annex A*

SEC contains several rule types for event correlation. They are illustrated below with examples:

### SingleWithScript

Copy the following to example.conf:

```
# Example conf

# Single with script.  Pass matched IP address

# to script for validation. If valid, execute

# action 1; if not valid execute action2.

#

# Note: change script path (and possibly perl path)

#       to match your system.


type=SingleWithScript

ptype=RegExp

pattern=(\d+)\.(\d+)\.(\d+)\.(\d+)

script=/usr/bin/perl /home/SEC-examples/example.pl $0

desc=$0

action=write - IP address $0  matches.

action2=write - IP address $0  does NOT match.
```

Note that while this RegExp pattern used will match an IP address, it will also match expressions that are not real IP addresses, such as ``9999.8888.7777.6666".

Also note that this rule takes two action statements. SEC checks the return value of the called program. If the program returns a zero value, the *action* is executed, if non-zero *action2* is executed.

Next, copy the following to script `example.pl`

```
#!/usr/bin/perl

#

# Script example.pl - check if IP argument

# matches a short list of IP addresses.

# Return zero on success, 1 on failure.

@match_list = ( '1.2.3.4',

                '2.3.4.5',

                '3.4.5.6'

              );

$ip_addr = $ARGV[0] or die "No IP address passed on command line";

foreach $ip (@match_list)

{

  exit (0) if $ip_addr eq $ip;

}

exit 1;
```

Script `example.pl` accepts a single IP address on the command line passed from the matched rule. If the address matches one of the IPs on its small list of IP addresses, it returns zero, else it returns 1. If there is no IP address at all, the script dies and returns a non-zero value.

Run with:

```
% perl sec.pl -conf=example.conf -input=-
```

Output looks like this:

```
toto@toto:~/SEC-examples$perl sec.pl -conf=example.pl -input=-

Simple Event Correlator version 2.1.11

Reading configuration from example.pl

Can't open configuration file example.pl (No such file or directory)

^C

toto@toto:~/SEC-examples$perl sec.pl -conf=example.conf -input=-

Simple Event Correlator version 2.1.11

Reading configuration from example.conf

1 rules loaded from example.conf

1.2.3.4

Child 16396 created for command '/usr/bin/perl /home/SEC-examples/example.pl 1.2.3.4'

Child 16396 terminated with exitcode 0

Writing event 'IP address 1.2.3.4 matches.' to file -

IP address 1.2.3.4 matches.
```

```
5.6.7.8

Child example.pl 5.6.7.8'

Child 16398 terminated with non-zero exitcode 1

Writing event 'IP address 5.6.7.8 does NOT match.' to file -

IP address 5.6.7.8 does NOT match.

^C
```

More robust IP address matching is possible with the **Net::IP_Addr** perl module.

---

## SingleWithSuppress

Copy the following to example.conf:

```
# Example example.conf

# Example of SingleWithSuppress



type=SingleWithSuppress

ptype=RegExp

pattern=foo

desc=$0

action=write - $0 suppressed for 5 seconds at %t

window=5
```

Run with:

```
% perl sec.pl -conf=example.conf -input=-
```

and continuously enter ``foo'' as rapidly as possible.

Output:

```
toto@toto:~/SEC-examples$perl sec.pl -conf=example.conf -input=-

Simple Event Correlator version 2.1.11

Reading configuration from example.conf

1 rules loaded from example.conf

foo

Writing event 'foo suppressed for 5 seconds at Sat Nov 15 17:04:38
2003' to file -

foo suppressed for 5 seconds at Sat Nov 15 17:04:38 2003

foo

foo

foo

foo

foo

foo

Writing event 'foo suppressed for 5 seconds at Sat Nov 15 17:04:44
2003' to file -

foo suppressed for 5 seconds at Sat Nov 15 17:04:44 2003

foo

foo

foo

foo
```

```
foo

foo

Writing event 'foo suppressed for 5 seconds at Sat Nov 15 17:04:50
2003' to file -

foo suppressed for 5 seconds at Sat Nov 15 17:04:50 2003

foo

^C
```

---

## Pair

Copy the following to `example.conf`:

```
# Example example.conf

# Example Pair rule.

# Match event A and B within window.



type=Pair

ptype=RegExp

pattern=foo

desc=$0

action=write - foo matched at %t. Start window of 5 seconds for
bar ...

ptype2=RegExp

pattern2=bar

desc2=$0
```

```
action2=write - bar matched at %t.  bar is within window!

window=5
```

Run with:

```
% perl sec.pl -conf=example.conf -input=-
```

When running this rule, first enter ``foo" and ``bar" close together (i.e. within 5 seconds). Then enter ``foo" and wait to enter ``bar" until the window is past (i.e. more than 5 seconds.) The first time the *Pair* rule will correlate them together, while the second time they are not correlated.

Output will look similar to:

```
toto@toto:~/SEC-examples$perl sec.pl -conf=example.conf -input=-

Simple Event Correlator version 2.1.11

Reading configuration from example.conf

1 rules loaded from example.conf

foo

Writing event 'foo matched at Sat Nov 15 18:17:07 2003. Start
window of 5 seconds for bar ...' to file -

foo matched at Sat Nov 15 18:17:07 2003. Start window of 5 seconds
for bar ...

bar

Writing event 'bar matched at Sat Nov 15 18:17:09 2003.  bar is
within window!' to file -
```

```
bar matched at Sat Nov 15 18:17:09 2003.  bar is within window!

foo

Writing  event  'foo  matched  at  Sat  Nov  15  18:17:14  2003.  Start
window of 5 seconds for bar ...' to file -

foo matched at Sat Nov 15 18:17:14 2003. Start window of 5 seconds
for bar ...

bar

^C
```

---

## PairWithWindow

Copy the following to `example.conf`:

```
# Example example.conf

# Example PairWithWindow rule.

# Match both events A and B within window executes action2.

# If event B does not occur within window, execute action.



type=PairWithWindow

ptype=RegExp

pattern=foo

desc=$0

action=write - foo matched, bar NOT matched within window.

ptype2=RegExp

pattern2=bar
```

```
desc2=$0

action2=write - foo and bar both matched within 5 second window!

window=5
```

Run with:

```
% perl sec.pl -conf=example.conf -input=-
```

When running this rule, first enter ``foo" and ``bar" close together (i.e. within 5 seconds). Then enter ``foo" and wait to enter ``bar" until the window is past (i.e. more than 5 seconds.) The first time the *PairWithWindow* rule will correlate them together, while the second time they are not correlated.

Output looks like:

```
toto@toto:~/SEC-examples$perl sec.pl -conf=example.conf -input=-

Simple Event Correlator version 2.1.11

Reading configuration from example.conf

1 rules loaded from example.conf

foo

bar

Writing event 'foo and bar both matched within 5 second window!'
to file -

foo and bar both matched within 5 second window!



foo
```

```
Writing event 'foo matched, bar NOT matched within window.' to
file -

foo matched, bar NOT matched within window.

^C
```

---

## SingleWithThreshold

Copy the following to `example.conf`:

```
# Example example.conf

# Example SingleWithThreshold rule.

# Match event A thresh number of times in window

# and execute action.  Slide window if needed

# until window expires.



type=SingleWithThreshold

ptype=RegExp

pattern=foo

desc=$0

action=write - foo matched three times in 10 seconds!

window=10

thresh=3
```

Run with:

```
% perl sec.pl -conf=example.conf -input=-
```

When running this rule, first enter ``foo" three times close together (i.e. within 10 seconds). The *action* will execute.

Then enter ``foo" slowly, waiting five to eight seconds between each entry. Since there are never three entries (*thresh=3*) entered within the sliding window, the rule is not matched and the *action* is not executed.

The first time the *SingleWithThreshold* rule will correlate them together, while the second time they are not correlated.

Output looks similar to:

```
toto@toto:~/SEC-examples$perl sec.pl -conf=example.conf -input=-

Simple Event Correlator version 2.1.11

Reading configuration from example.conf

1 rules loaded from example.conf

foo

foo

foo

Writing event 'foo matched three times in 10 seconds!' to file -

foo matched three times in 10 seconds!

foo

foo

foo

foo
```

```
foo

foo

^C
```

---

## SingleWith2Threshds

Copy the following to `example.conf`:

```
# Example example.conf

# Example SingleWith2Threshholds rule.

# Match thresh A events (go above low watermark) and execute
action.

# Then switch to thresh2 and window2 to count more A events.

# If less than thresh2 A events occur in window2 (stay under high

# watermark), execute action2.



type=SingleWith2Thresholds

ptype=RegExp

pattern=foo

desc=$0

action=write - foo hit low watermark (3) at time %t

window=5

thresh=3
```

```
desc2=$0

action2=write - foo stayed under high watermark (5) at time %t

window2=10

thresh2=5
```

Run with:

```
% perl sec.pl -conf=example.conf -input=-
```

In this example, it will be necessary to time your entries fairly closely. The first example shows three matched events followed by *action*, then less than *thresh2* (5 events) within *window2* (10 seconds).

The second example shows multiple events after the low watermark then a marked slowing of entries that result in *action2*.

Output from this example:

```
toto@toto:~/SEC-examples$perl sec.pl -conf=example.conf -input=-

Simple Event Correlator version 2.1.11

Reading configuration from example.conf

1 rules loaded from example.conf

foo

foo

foo

Writing event 'foo hit low watermark (3) at time Mon Nov 17
09:42:05 2003' to file -
```

```
foo hit low watermark (3) at time Mon Nov 17 09:42:05 2003
foo
foo
Writing event 'foo stayed under high watermark (5) at time Mon Nov 17 09:42:16 2003' to file -
foo stayed under high watermark (5) at time Mon Nov 17 09:42:16 2003
foo
foo
foo
Writing event 'foo hit low watermark (3) at time Mon Nov 17 09:42:22 2003' to file -
foo hit low watermark (3) at time Mon Nov 17 09:42:22 2003
foo
foo
foo
foo
foo
foo
foo
foo
foo
foo
```

```
foo

foo

foo

foo

foo slowing way down...

foo

foo

Writing event 'foo stayed under high watermark (5) at time Mon Nov
17 09:42:49 2003' to file -

foo stayed under high watermark (5) at time Mon Nov 17 09:42:49
2003

^C
```

---

## Suppress

Copy the following to `example.conf`:

```
# Example example.conf

# Example of Suppress.

# First rule suppresses 'foo'.

# Second rule matches any pattern and

# executes write action.



type=Suppress

ptype=RegExp
```

```
pattern=foo

desc=$0



type=Single

ptype=RegExp

pattern=(.*)

desc=$0

action=write - entry was: $0
```

Run with:

```
% perl sec.pl -conf=example.conf -input=-
```

In this example, the first rule suppresses ``foo'' while the second rule matches any pattern and writes it to standard output. Since ``foo'' is already suppressed by the first rule, it will never be written by the second rule.

Output looks similar to:

```
toto@toto:~/SEC-examples$perl sec.pl -conf=example.conf -input=-

Simple Event Correlator version 2.1.11

Reading configuration from example.conf

2 rules loaded from example.conf

bar

Writing event 'entry was: bar' to file -
```

```
entry was: bar
baz
Writing event 'entry was: baz' to file -
entry was: baz
foo
foo
foo
bar
Writing event 'entry was: bar' to file -
entry was: bar
baz
Writing event 'entry was: baz' to file -
entry was: baz
^C
```

---

## Calendar

Copy the following to `example.conf`:

```
# Example example.conf
# Example calendar rule.
# Write a message every minute.

type=Calendar
```

```
time=* * * * *
desc=$0
action=write - The time is now: %t
```

This example takes no user input. However, the ``-input" parameter must still be present on the command line. Run with:

```
% perl sec.pl -conf=example.conf -input=-
```

Output is similar to:

```
toto@toto:~/SEC-examples$perl sec.pl -conf=example.conf -input=-
Simple Event Correlator version 2.1.11
Reading configuration from example.conf
1 rules loaded from example.conf
Writing event 'The time is now: Mon Nov 17 10:40:42 2003' to file
-
The time is now: Mon Nov 17 10:40:42 2003
Writing event 'The time is now: Mon Nov 17 10:41:00 2003' to file
-
The time is now: Mon Nov 17 10:41:00 2003
Writing event 'The time is now: Mon Nov 17 10:42:00 2003' to file
-
The time is now: Mon Nov 17 10:42:00 2003
^C
```

Using the ``-debug=4" parameter removes the informational debug statements and results in just:

```
The time is now: Mon Nov 17 10:46:35 2003

The time is now: Mon Nov 17 10:47:00 2003

The time is now: Mon Nov 17 10:48:00 2003

The time is now: Mon Nov 17 10:49:00 2003

The time is now: Mon Nov 17 10:50:00 2003
```

Note also that SEC invokes the *action* of all calendar rules at startup, but only at the top of each minute thereafter. Actions that must not occur too closely together must take this into account.

Running applications from SEC is similar. This example runs a script that checks MD5 checksums on a list of files every five minutes. The script takes a single parameter- ``MD5_CHECK":

```
#

# Run the SystemCheck.sh script every five minutes.

#

type=Calendar

time=0,5,10,15,20,25,30,35,40,45,50,55 * * * *

desc=MD5_CHECK

action=shellcmd /home/jpb/SEC-examples/SystemCheck.sh %s
```

## Multiple Input Streams

To set up SEC to read multiple files, the **tail** program is often used as in the following example.

Copy the following to `example.conf`:

```
#
# Example example.conf
# Multiple input files with spawn.
#
#
type=Single
ptype=RegExp
pattern=foo
continue=TakeNext
desc=$0
action=spawn /usr/bin/tail -f ./aaa.in ;\
       spawn /usr/bin/tail -f ./bbb.in ;\
       spawn /usr/bin/tail -f ./ccc.in ;


# Match lines beginning with aaa:
type=Single
ptype=RegExp
```

```
pattern=^aaa:(.*)

desc=$0

action=write aaa.out %s



# Match lines beginning with bbb:

type=Single

ptype=RegExp

pattern=^bbb:(.*)

desc=$0

action=write bbb.out %s



# Match lines beginning with ccc:

type=Single

ptype=RegExp

pattern=^ccc:(.*)

desc=$0

action=write ccc.out %s





# Match all other lines

type=Single

ptype=RegExp
```

```
pattern=(.*)

desc=$0

action=write other.out %s
```

In this example, the *spawn* action is part of a rule that matches input ``foo". This means that the spawn actions will not occur until ``foo" is recognized in the input stream.

After the *tail* commands will forward input from their respective files into SEC. SEC will treat all input streams the same, and parse input from all streams according to all rules.

Note that the input files `aaa.in`, `bbb.in`, and `ccc.in` must exist before running the example. Use the **touch** command to create these empty files as follows:

```
% touch aaa.in bbb.in ccc.in
```

Note also that the last rule is a catch-all rule: if the input does not get recognized by any other rule, it will be written to `other.out`

Run with:

```
% perl sec.pl -conf=example.conf -input=-
```

The session starts as follows:

```
toto@toto:~/SEC-examples/tmp$perl  ../sec.pl  -conf=example.conf  -
input=-

Simple Event Correlator version 2.1.11

Reading configuration from example.conf

5 rules loaded from example.conf

foo
```

```
Spawning shell command '/usr/bin/tail -f ./aaa.in'

Child 15940 created for command '/usr/bin/tail -f ./aaa.in'

Spawning shell command '/usr/bin/tail -f ./bbb.in'

Child 15941 created for command '/usr/bin/tail -f ./bbb.in'

Spawning shell command '/usr/bin/tail -f ./ccc.in'

Child 15942 created for command '/usr/bin/tail -f ./ccc.in'

Writing event 'foo' to file other.out

aaa:input from terminal

Writing event 'aaa:input from terminal' to file aaa.out

bbb:input from terminal

Writing event 'bbb:input from terminal' to file bbb.out

ccc:input from terminal

Writing event 'ccc:input from terminal' to file ccc.out

ddd:input from terminal

Writing event 'ddd:input from terminal' to file other.out
```

So far all input has been from the terminal. In another window or session in the same directory, perform the following commands:

```
% echo "aaa:from other session copied into ccc.in" >> ccc.in

% echo "bbb:from other session copied into aaa.in" >> aaa.in

% echo "ddd:from other session copied into bbb.in" >> bbb.in
```

SEC processes these inputs as well:

```
Creating event 'aaa:from other session copied into ccc.in'
(received from child 15956)

Writing event 'aaa:from other session copied into ccc.in' to file
aaa.out

Creating event 'bbb:from other session copied into aaa.in'
(received from child 15954)

Writing event 'bbb:from other session copied into aaa.in' to file
bbb.out

Creating event 'ddd:from other session copied into bbb.in'
(received from child 15955)

Writing event 'ddd:from other session copied into bbb.in' to file
other.out

^C
```

Examine each output file to determine its contents:

```
toto@toto:~/SEC-examples/tmp$cat aaa.out

aaa:input from terminal

aaa:from other session copied into ccc.in



toto@toto:~/SEC-examples/tmp$cat bbb.out

bbb:input from terminal

bbb:from other session copied into aaa.in
```

```
toto@toto:~/SEC-examples/tmp$cat ccc.out

ccc:input from terminal



toto@toto:~/SEC-examples/tmp$cat other.out

foo

ddd:input from terminal

ddd:from other session copied into bbb.in
```

As shown above, SEC parsed the input, regardless of where it came from, and performed the actions indicated on each matched rule.

## 7.2 *Annex B*

```
####################################################################
#                Sample SEC ruleset for Snort IDS
####################################################################

# -----------------------------------------------------------------
# Handle portscans
# -----------------------------------------------------------------

# For every completed portscan, add an entry to the PORTSCAN_REPORT;
# also generate a meta-event ACTIVITY_FROM for the IP

type=Single
ptype=RegExp
pattern=End of portscan from (([\d\.]+).*)
desc=Portscan from $1
action=add PORTSCAN_REPORT %t: %s; event ACTIVITY_FROM_$2: %s



# -----------------------------------------------------------------
# Recognize snort alert message and generate corresponding SEC event
# -----------------------------------------------------------------

# recognize snort alert message; also generate
# a meta-event ACTIVITY_FROM for the IP

type=Single
ptype=RegExp
```

```
pattern=snort(?:\[\d+\])?: \[[0-9:]+\] (.+) \[(.+)\] \[.*Priority: (\d+)\]: \
\S+ ([\d\.]+):?\d* -> ([\d\.]+):?\d*
desc=PRIORITY $3 INCIDENT FROM $4 TO $5: $1 [$2]
action=event %s; event ACTIVITY_FROM_$4: $1


# -----------------------------------------------------------------
# Handle priority 1 incidents
# -----------------------------------------------------------------

# Detect the beginning of priority 1 attack from a certain source IP,
# and send a warning e-mail message that a new attack has begun;
# also create a context for storing a detailed information about the attack

type=Single
ptype=RegExp
pattern=PRIORITY 1 INCIDENT FROM (\S+) TO \S+: .+
context=!ATTACK_FROM_$1
continue=TakeNext
desc=Priority 1 attack started from $1
action=create ATTACK_FROM_$1; add ALERT_REPORT %t: %s; pipe '%t: %s' \
       mail -s 'SNORT: priority 1 attack from $1 (alert)' root@localhost


# For every priority 1 incident, add an entry to the context by its IP;
# if the IP has been quiet for 5 minutes, report the whole attack

type=Single
ptype=RegExp
pattern=PRIORITY 1 INCIDENT FROM (\S+) TO (\S+): (.+)
```

```
context=ATTACK_FROM_$1
continue=TakeNext
desc=Priority 1 incident from $1 to $2: $3
action=add ATTACK_FROM_$1 %t: %s; \
       set ATTACK_FROM_$1 300 ( report ATTACK_FROM_$1 \
       mail -s 'SNORT: priority 1 attack from $1 (report)' root@localhost )


# -----------------------------------------------------------------
# Handle incidents by thresholding
# -----------------------------------------------------------------

# Count how many _certain type_ of incidents are coming from one source
# if the threshold has been crossed, reset the counting operation started
# by the next rule, in order to avoid duplicate alerts for the same IP

type=SingleWithThreshold
ptype=RegExp
pattern=PRIORITY (\d+) INCIDENT FROM (\S+) TO \S+: (.+)
continue=TakeNext
desc=Snort has seen >= 30 priority $1 incidents from $2: $3
action=add ALERT_REPORT %t: %s; \
       reset +1 Snort has seen >= 150 incidents from $2; \
       create TURNOFF_$2 3600
thresh=30
window=3600


# Count how many incidents come from one source

type=SingleWithThreshold
```

```
ptype=RegExp
pattern=PRIORITY \d+ INCIDENT FROM (\S+) TO \S+: .+
context=!TURNOFF_$1
desc=Snort has seen >= 150 incidents from $1
action=add ALERT_REPORT %t: %s
thresh=150
window=7200


# -----------------------------------------------------------------
# Report IPs that have been active for some time
# -----------------------------------------------------------------

# Set up activity contexts for the IP; if the IP has been active for 2
hours,
# and there have been no gaps longer than 30 minutes, report its activities

type=Single
ptype=RegExp
pattern=ACTIVITY_FROM_(\S+):
context=!ACTIVITY_LIST_FOR_$1
continue=TakeNext
desc=Create activity contexts for $1
action=create ACTIVITY_LIST_FOR_$1_LIFETIME; \
       create ACTIVITY_LIST_FOR_$1 7200 ( report ACTIVITY_LIST_FOR_$1 \
       mail -s 'SNORT: $1 has been active for 2 hours' root@localhost; \
       delete ACTIVITY_LIST_FOR_$1_LIFETIME )


# Add the activity event to the context of a given IP, and extend
# the lifetime of activity contexts for 30 minutes for the IP
```

```
type=Single
ptype=RegExp
pattern=ACTIVITY_FROM_(\S+): (.*)
context=ACTIVITY_LIST_FOR_$1
desc=Activity from $1: $2
action=add ACTIVITY_LIST_FOR_$1 %t: %s; \
       set ACTIVITY_LIST_FOR_$1_LIFETIME 1800 ( delete ACTIVITY_LIST_FOR_$1
)


# ------------------------------------------------------------------
# Send reports every day at 9:00 am
# ------------------------------------------------------------------

# send daily report about regular alerts

type=Calendar
time=0 9 * * *
desc=Sending alert report...
action=report ALERT_REPORT \
       mail -s 'SNORT: daily alert report' root@localhost; \
       delete ALERT_REPORT



# send daily report about portscans

type=Calendar
time=0 9 * * *
desc=Sending portscan report...
action=report PORTSCAN_REPORT \
       mail -s 'SNORT: daily portscan report' root@localhost; \
       delete PORTSCAN_REPORT
```

## *7.3 Annex C*

```
/* portscan.cpp */

/* plugin generated by STATL v1.0a15 */


#include "STAT/stat_scenario.h"

#include "idmef1lib.h"



extern "C" {


namespace {


  /***********************************************************/

  /***             GLOBAL/PROTOTYPE ENVIRONMENT            ***/

  /***********************************************************/


  /* Structure that contains the global environment */

struct prototype_env {

  int timeout;

  int threshold;

  int flood_threshold;

  HashTable attackers;

  IDMEFMerger *merger;

};


u_char *prototype_env_new(struct stat_core* stat,

                          struct scenario_prototype* prototype)

{

  prototype_env *g_env = new prototype_env();
  if (prototype->argc < 3) {
    stat_error(stat,"wrong number of arguments (%d) for scenario %s",
             prototype->argc, prototype->name);
    return NULL;
  }
  int i,j;
  for(i=0,j=0; j<prototype->argc; i+=2,j++) {
    if (!strcmp(prototype->argv[i],"timeout"))
      g_env->timeout = atoi(prototype->argv[i+1]);
    if (!strcmp(prototype->argv[i],"threshold"))
      g_env->threshold = atoi(prototype->argv[i+1]);
    if (!strcmp(prototype->argv[i],"flood_threshold"))
      g_env->flood_threshold = atoi(prototype->argv[i+1]);
  }
  g_env->merger = (IDMEFHelperFactory::getMerger("ScanMerger"));

  return (u_char*)g_env;
}

void prototype_env_del(struct stat_core* stat, u_char* p_env) {
  if(p_env == NULL) return;

  prototype_env *g_env = (prototype_env*)p_env;



  delete g_env;
  return;
}

void prototype_env_dump(struct stat_core *stat,
             char   *sample,
             int     size,
             u_char *p_env,
             int     level)
{
  prototype_env *g_env;
  char *indent = get_indent_string(level);
  g_env = (prototype_env*)p_env;

  if (sample != NULL) {
    sprintf(sample, "\
%s  timeout: %d\n\
%s  threshold: %d\n\
%s  flood_threshold: %d\n\
%s  attackers: %s\n\
%s  merger: %s\n\
",
       indent, g_env->timeout,
       indent, g_env->threshold,
       indent, g_env->flood_threshold,
       indent, g_env->attackers.toString(),
       indent, g_env->merger->toString());
  } else {
    fprintf(stat->dump, "\
%s  timeout: %d\n\
%s  threshold: %d\n\
%s  flood_threshold: %d\n\
%s  attackers: %s\n\
%s  merger: %s\n\
",
       indent, g_env->timeout,
       indent, g_env->threshold,
       indent, g_env->flood_threshold,
       indent, g_env->attackers.toString(),
       indent, g_env->merger->toString());
  }
  del_indent_string(indent);
}

u_char  *prototype_env_restore(struct  stat_core*  stat,char*  dump,  int
dumpsize) {
  prototype_env *g_env = new prototype_env();


  return (u_char*)g_env;
}

  /***********************************************************/
  /***             LOCAL/INSTANCE ENVIRONMENT             ***/
  /***********************************************************/

  /* Structure that contains the local environment */
struct instance_env {
  IDMEF_Message *IDMEF_ALERT;
  u_long attacker_address;
  string analyzer_id;
  STATVector sub_alerts;
  int count;
#define TIMER_t1 1
  int t1;
};
```

```
u_char *instance_env_new(struct stat_core* stat,
                         u_char* p_env)
{
  instance_env *l_env = new instance_env();
  prototype_env *g_env = (prototype_env*)p_env;

  l_env->IDMEF_ALERT = NULL;
  l_env->count = 0;
  l_env->t1 = TIMER_t1;

  return (u_char*)l_env;
}

u_char *instance_env_clone(struct stat_core* stat, u_char* i_env){
  instance_env *new_env;

  if (i_env == NULL) return NULL;
  new_env = new instance_env();
  instance_env *old_env = (instance_env*)i_env;
  new_env->IDMEF_ALERT = old_env->IDMEF_ALERT;
  new_env->attacker_address = old_env->attacker_address;
  new_env->analyzer_id = string(old_env->analyzer_id);
  new_env->sub_alerts = old_env->sub_alerts;
  new_env->count = old_env->count;
  new_env->t1 = old_env->t1;

  return (u_char*)new_env;
}

void instance_env_del(struct stat_core* stat, u_char* i_env) {
  if(i_env == NULL) return;
  instance_env *l_env = (instance_env*)i_env;



  delete l_env;
  return;
}

void instance_env_dump(struct stat_core *stat,
             char   *sample,
             int     size,
             u_char *i_env,
             int     level)
{
  instance_env *l_env;
  char *indent = get_indent_string(level);
  l_env = (instance_env*)i_env;

  if (sample != NULL) {
    sprintf(sample, "\
%s  IDMEF_ALERT: %s\n\
%s  attacker_address: %ul\n\
%s  analyzer_id: %s\n\
%s  sub_alerts: %s\n\
%s  count: %d\n\
%s  t1: %d\n\
",
       indent, l_env->IDMEF_ALERT->toString(),
       indent, l_env->attacker_address,
       indent, (l_env->analyzer_id).c_str(),
       indent, l_env->sub_alerts.toString(),
       indent, l_env->count,
       indent, l_env->t1);
  } else {
    fprintf(stat->dump, "\
%s  IDMEF_ALERT: %s\n\
%s  attacker_address: %ul\n\
%s  analyzer_id: %s\n\
%s  sub_alerts: %s\n\
%s  count: %d\n\
%s  t1: %d\n\
",
       indent, l_env->IDMEF_ALERT->toString(),
       indent, l_env->attacker_address,
       indent, (l_env->analyzer_id).c_str(),
       indent, l_env->sub_alerts.toString(),
       indent, l_env->count,
       indent, l_env->t1);
  }
  del_indent_string(indent);
}

u_char   *instance_env_restore(struct   stat_core*   stat,char*   dump,   int
dumpsize) {
  instance_env *l_env = new instance_env();



  return (u_char*)l_env;
}

  /*************************************************************/
  /***               RESPONSE INITIALIZATION               ***/
  /*************************************************************/
```

```
void instance_resp_getparam(struct stat_core* stat,
             struct scenario_instance* instance,
             int* r_argc,
             char*** r_argv)
{
  char **args;
  instance_env *l_env=(instance_env*)(instance->environment);
  prototype_env *g_env=(prototype_env*)(instance->prototype->environment);
  char tmp[16];

  *r_argc = 22;
  args = (char**)new_chunk((*r_argc+1) * sizeof(char*));
  args[0] = stat_strdup("timeout");
  snprintf(tmp,16,"%i",g_env->timeout);
  args[1] = stat_strdup(tmp);
  args[2] = stat_strdup("threshold");
  snprintf(tmp,16,"%i",g_env->threshold);
  args[3] = stat_strdup(tmp);
  args[4] = stat_strdup("flood_threshold");
  snprintf(tmp,16,"%i",g_env->flood_threshold);
  args[5] = stat_strdup(tmp);
  args[6] = stat_strdup("attackers");
  args[7] = stat_strdup((char*)g_env->attackers.toString());
  args[8] = stat_strdup("merger");
  args[9] = stat_strdup((char*)g_env->merger->toString());
  args[10] = stat_strdup("IDMEF_ALERT");
  args[11] = stat_strdup((char*)l_env->IDMEF_ALERT->toString());
  args[12] = stat_strdup("attacker_address");
  snprintf(tmp,16,"%i",l_env->attacker_address);
  args[13] = stat_strdup(tmp);
  args[14] = stat_strdup("analyzer_id");
  args[15] = stat_strdup((char*)(l_env->analyzer_id).c_str());
  args[16] = stat_strdup("sub_alerts");
  args[17] = stat_strdup((char*)l_env->sub_alerts.toString());
  args[18] = stat_strdup("count");
  snprintf(tmp,16,"%i",l_env->count);
  args[19] = stat_strdup(tmp);
  args[20] = stat_strdup("t1");
  snprintf(tmp,16,"%i",l_env->t1);
  args[21] = stat_strdup(tmp);
  args[22] = NULL;
  *r_argv = args;
  return;
}

void instance_resp_delparam(struct stat_core* stat,
             struct scenario_instance* instance,
             int r_argc,
             char** r_argv)
{
  for (int i=0; i<r_argc; i++) {
    free_chunk((u_char*)r_argv[i]);
  }
  free_chunk((u_char*)(r_argv));
  return;
}

  /************************************************************/
  /***          STATE CALLBACK FUNCTION DEFINITIONS        ***/
  /************************************************************/
  /* state s0 */


  /* state recording */
static void state_recording_code(struct stat_core* stat,
             struct scenario_instance* instance,
             struct stat_state* state)
{
  prototype_env *g_env;
  instance_env  *l_env;
  g_env = (prototype_env *)instance->prototype->environment;
  l_env = (instance_env *)instance->environment;

  {
    timer_start(stat, instance, TIMER_LOCAL, l_env->t1, g_env->timeout, 0);
  }
}


  /* state scan */


  /* state noscan */


  /**********************************************************/
  /***      TRANSITION CALLBACK FUNCTION DEFINITIONS       ***/
  /**********************************************************/


  /* transition firstprobe */
static int trans_firstprobe_assertion(struct stat_core* stat,
             struct scenario_instance* instance,
             struct stat_transition* transition,
             struct stat_event* event)
{
  prototype_env *g_env;
  instance_env  *l_env;
  int result;
  IDMEF_Message* e = (IDMEF_Message*)event->data;
  g_env = (prototype_env *)instance->prototype->environment;
  l_env = (instance_env *)instance->environment;

  result = ((((e->alert->source) && (e->alert->source->node)) && (e->alert->source->node->address))  &&  (!g_env->attackers.contains(HashKey(e->alert->analyzer->analyzerid, e->alert->source->node->address->get_address()))));
  return result;
}

static void trans_firstprobe_code(struct stat_core* stat,
             struct scenario_instance* instance,
             struct stat_transition* transition,
             struct stat_event* event)
{
  prototype_env *g_env;
  instance_env  *l_env;
  IDMEF_Message* e = (IDMEF_Message*)event->data;
  g_env = (prototype_env *)instance->prototype->environment;
  l_env = (instance_env *)instance->environment;

  {
    l_env->attacker_address        =        e->alert->source->node->address->get_address();
    l_env->analyzer_id = e->alert->analyzer->analyzerid;
    l_env->count = 1;
    g_env->attackers.put(HashKey(l_env->analyzer_id.c_str(),         l_env->attacker_address), NULL);
    IDMEF_Message *e2 = e->clone();
    l_env->sub_alerts.add(e2);
  }
}

  /* transition probe */
static int trans_probe_assertion(struct stat_core* stat,
             struct scenario_instance* instance,
             struct stat_transition* transition,
             struct stat_event* event)
{
  prototype_env *g_env;
  instance_env  *l_env;
  int result;
  IDMEF_Message* e = (IDMEF_Message*)event->data;
  g_env = (prototype_env *)instance->prototype->environment;
  l_env = (instance_env *)instance->environment;

  result = (((((e->alert->source) && (e->alert->source->node)) && (e-
>alert->source->node->address)) && (e->alert->source->node->address-
>get_address() == l_env->attacker_address)) && (!strcmp(l_env-
>analyzer_id.c_str(), e->alert->analyzer->analyzerid)));
  return result;
}

static void trans_probe_code(struct stat_core* stat,
             struct scenario_instance* instance,
             struct stat_transition* transition,
             struct stat_event* event)
{
  prototype_env *g_env;
  instance_env  *l_env;
  IDMEF_Message* e = (IDMEF_Message*)event->data;
  g_env = (prototype_env *)instance->prototype->environment;
  l_env = (instance_env *)instance->environment;

  {
    l_env->count += 1;
    IDMEF_Message *e2 = e->clone();
    l_env->sub_alerts.add(e2);
  }
}

  /* transition scan_over */
static int trans_scan_over_assertion(struct stat_core* stat,
             struct scenario_instance* instance,
             struct stat_transition* transition,
             struct stat_event* event)
{
  prototype_env *g_env;
  instance_env  *l_env;
  int result;
  struct stat_event* t1 = event;
  g_env = (prototype_env *)instance->prototype->environment;
  l_env = (instance_env *)instance->environment;

  result = ((l_env->count >= g_env->threshold));
  return result;
}

static void trans_scan_over_code(struct stat_core* stat,
             struct scenario_instance* instance,
             struct stat_transition* transition,
             struct stat_event* event)
{
  prototype_env *g_env;
  instance_env  *l_env;
  struct stat_event* t1 = event;
  g_env = (prototype_env *)instance->prototype->environment;
  l_env = (instance_env *)instance->environment;

  {
    int i;
    g_env->attackers.remove(HashKey(l_env->attacker_address));
    IDMEF_Message *aggregate = NULL;
    if (l_env->sub_alerts.size() > g_env->flood_threshold)
    {
      aggregate = get_default_idmef(stat, "flood attack");
    }
    else
    {
      aggregate = get_default_idmef(stat, "portscan");
    }
    for (i = 0;i < l_env->sub_alerts.size();++i)
    {
      IDMEF_Message *m = (IDMEF_Message*)l_env->sub_alerts.elementAt(i);
      g_env->merger->merge(aggregate, m);
      delete(m);
    }
    l_env->sub_alerts.removeAllElements();
    aggregate->clevel = 2;
    struct stat_event *stat_evt_aggregate =
        stat_event_new(stat,    aggregate->getType(),    0,    stat->time,
(u_char*)aggregate);
    stat_event_prepend_to_q(stat,stat_evt_aggregate);
    l_env->IDMEF_ALERT = aggregate;
  }
}

  /* transition no_scan */
static int trans_no_scan_assertion(struct stat_core* stat,
             struct scenario_instance* instance,
             struct stat_transition* transition,
             struct stat_event* event)
{
  prototype_env *g_env;
  instance_env  *l_env;
  int result;
  struct stat_event* t1 = event;
  g_env = (prototype_env *)instance->prototype->environment;
  l_env = (instance_env *)instance->environment;

  result = ((l_env->count < g_env->threshold));
  return result;
}

static void trans_no_scan_code(struct stat_core* stat,
             struct scenario_instance* instance,
             struct stat_transition* transition,
             struct stat_event* event)
{
  prototype_env *g_env;
  instance_env  *l_env;
  struct stat_event* t1 = event;
  g_env = (prototype_env *)instance->prototype->environment;
  l_env = (instance_env *)instance->environment;

  {
    g_env->attackers.remove(HashKey(l_env->attacker_address));
    int i;
    for (i = 0;i < l_env->sub_alerts.size();++i)
    {
      IDMEF_Message *m = (IDMEF_Message*)l_env->sub_alerts.elementAt(i);
      m->clevel = 2;
      struct stat_event *stat_evt_m =
            stat_event_new(stat, m->getType(), 0, stat->time, (u_char*)m);
      stat_event_prepend_to_q(stat,stat_evt_m);
    }
  }
}
  /***********************************************************/
  /***       FUNCTION TO LOAD THE SCENARIO DEFINITION      ***/
  /***********************************************************/

void prototype_init(struct stat_core *stat,
             struct scenario_prototype *prototype)
{
  struct     stat_state     *state_s0,     *state_recording,     *state_scan,
*state_noscan;
  struct stat_transition *trans_firstprobe, *trans_probe, *trans_scan_over,
*trans_no_scan;
  struct stat_event_spec *es_firstprobe_e;
  struct stat_event_spec *es_firstprobe;
  struct stat_event_spec *es_probe_e;
  struct stat_event_spec *es_probe;

  state_s0 = state_new(stat,
             "s0",
             STATE_INITIAL,
             NULL,
             NULL,
             NULL);
  prototype_add_state(stat, prototype, state_s0);

  state_recording = state_new(stat,
             "recording",
             0,
             NULL,
             state_recording_code,
             NULL);
  prototype_add_state(stat, prototype, state_recording);

  state_scan = state_new(stat,
             "scan",
             0,
             NULL,
             NULL,
             NULL);
  prototype_add_state(stat, prototype, state_scan);

  state_noscan = state_new(stat,
             "noscan",
             0,
             NULL,
             NULL,
             NULL);
  prototype_add_state(stat, prototype, state_noscan);

  es_firstprobe_e = event_spec_new(stat,IDMEF_Message_ID);
  es_firstprobe = es_firstprobe_e;
  trans_firstprobe = transition_new(stat,
             TRANSITION_NON_CONSUMING,
             "firstprobe",
             es_firstprobe,
             0,
             0,
             trans_firstprobe_assertion,
             trans_firstprobe_code,
             state_s0,
             state_recording,
             NULL);

  es_probe_e = event_spec_new(stat,IDMEF_Message_ID);
  es_probe = es_probe_e;
  trans_probe = transition_new(stat,
             TRANSITION_CONSUMING,
             "probe",
             es_probe,
             0,
             0,
             trans_probe_assertion,
             trans_probe_code,
             state_recording,
             state_recording,
             NULL);
```

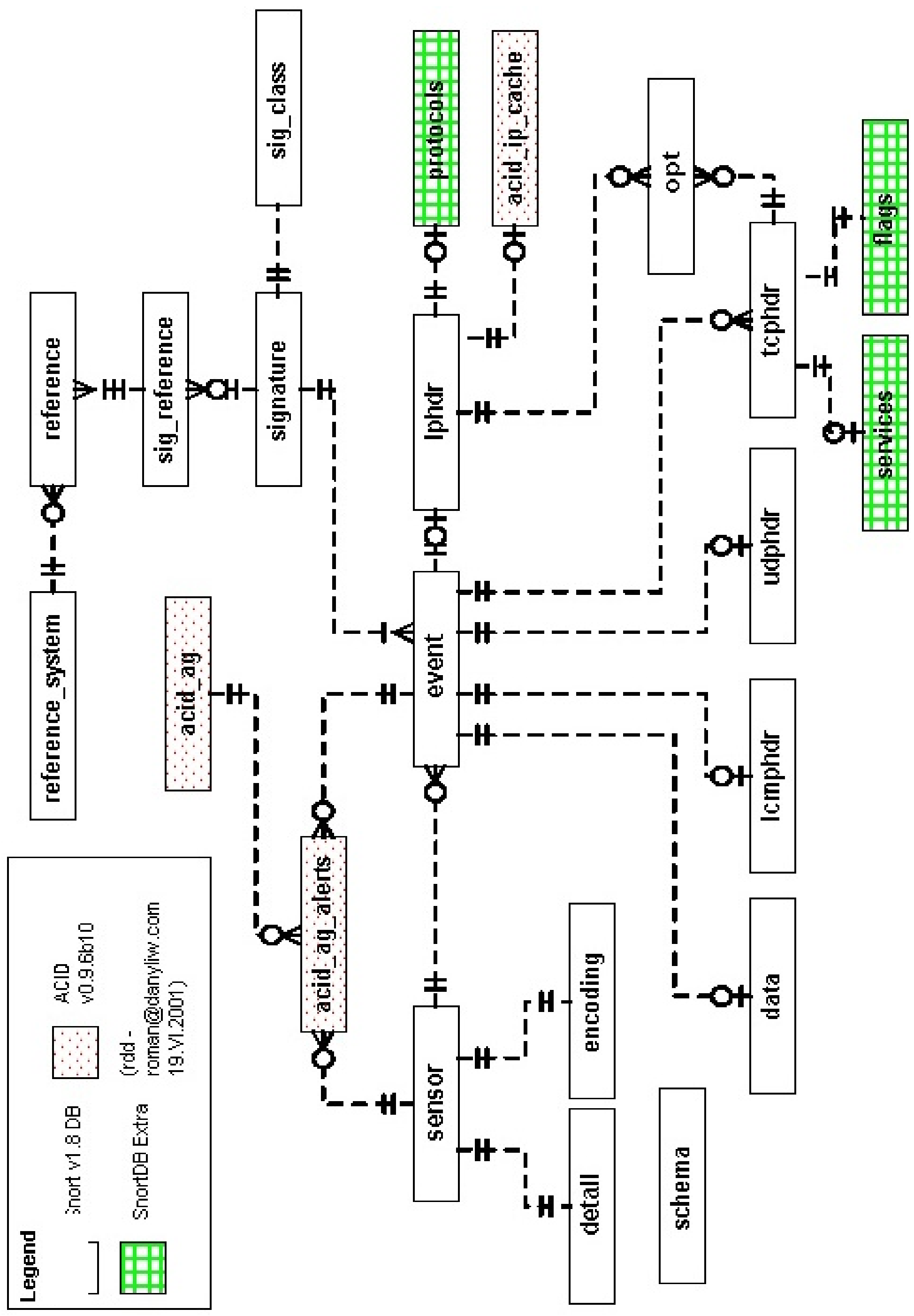
```
  trans_scan_over = transition_new(stat,
             TRANSITION_CONSUMING,
             "scan_over",
             NULL,
             TIMER_LOCAL,
             TIMER_t1,
             trans_scan_over_assertion,
             trans_scan_over_code,
             state_recording,
             state_scan,
             NULL);

  trans_no_scan = transition_new(stat,
             TRANSITION_CONSUMING,
             "no_scan",
             NULL,
             TIMER_LOCAL,
             TIMER_t1,
             trans_no_scan_assertion,
             trans_no_scan_code,
             state_recording,
             state_noscan,
             NULL);

}

} /* end namespace */
} /* end extern "C" */
```

## *7.4 Annex D*

# ACID: Database (v100-103) ER Diagram

Snort (and other devices) log to database with the following schema:

**ACID 0.9.6b10 Database ER diagram**



| Table | Component | Description |
|---|---|---|
| schema | Snort | Self-documented information about the database |
| sensor | Snort | Sensor name |

| | | |
|---|---|---|
| event | Snort | Meta-data about the detected alert |
| signature | Snort | Normalized listing of alert/signature names, priorities, and revision IDs |
| sig_reference | Snort | Reference information for a signature |
| reference | Snort | Reference IDs for a signature |
| reference_system | Snort | (lookup table) Reference system list |
| sig_class | Snort | Normalized listing of alert/signature classifications |
| data | Snort | Contents of packet payload |
| iphdr | Snort | IP protocol fields |
| tcphdr | Snort | TCP protocol fields |
| udphdr | Snort | UDP protocol fields |
| icmphdr | Snort | ICMP protocol fields |
| opt | Snort | IP and TCP options |
| `detail` | Snort | (lookup table) Level of detail with which a sensor is logging |
| `encoding` | Snort | (lookup table) Type of encoding used for the packet payload |
| `protocols` | SnortDB extra | (lookup table) Layer-4 (IP encoded) protocol list |
| `services` | SnortDB extra | (lookup table) TCP and UDP service list |
| `flags` | SnortDB extra | (lookup table) TCP flag list |
| acid_ag | ACID | Meta-data for alert groups |

|  |  |  |
|---|---|---|
| acid_ag_alert | ACID | Alerts in each alert group |
| acid_ip_cache | ACID | Cached DNS and whois information |

---

*schema*

---

```
+-------+------------------+------+-----+---------------------+------------------------+
| Field | Type             | Null | Key | Default             | Description            |
+-------+------------------+------+-----+---------------------+------------------------+
| vseq  | int(10) unsigned |      | PRI | 0                   | Database schema ID     |
| ctime | datetime         |      |     | 0000-00-00 00:00:00 | Timestamp              |
+-------+------------------+------+-----+---------------------+------------------------+
```

---

*sensor*

---

```
+-----------+------------------+------+-----+---------+---------------------------------+
| Field     | Type             | Null | Key | Default | Description                     |
+-----------+------------------+------+-----+---------+---------------------------------+
| sid       | int(10) unsigned |      | PRI | NULL    | Sensor ID                       |
| hostname  | text             | YES  |     | NULL    | Hostname of the sensor          |
| interface | text             | YES  |     | NULL    | Network interface (e.g. eth0)   |
| filter    | text             | YES  |     | NULL    | BPF filter                      |
| detail    | tinyint(4)       | YES  |     | NULL    | Detail level of the logging     |
| encoding  | tinyint(4)       | YES  |     | NULL    | Encoding format of the payload  |
+-----------+------------------+------+-----+---------+---------------------------------+
```

---

*event*

---

```
+-----------+------------------+------+-----+---------------------+---------------------+
| Field     | Type             | Null | Key | Default             | Description         |
+-----------+------------------+------+-----+---------------------+---------------------+
| sid       | int(10) unsigned |      | PRI | 0                   | Sensor ID           |
| cid       | int(10) unsigned |      | PRI | 0                   | Event ID            |
| signature | int(10) unsigned |      | MUL | 0                   | Signature ID        |
| timestamp | datetime         |      | MUL | 0000-00-00 00:00:00 | Timestamp           |
+-----------+------------------+------+-----+---------------------+---------------------+
```

---

*signature*

---

```
+--------------+------------------+------+-----+---------+-----------------------+
| Field        | Type             | Null | Key | Default | Description           |
+--------------+------------------+------+-----+---------+-----------------------+
| sig_id       | int(10) unsigned |      | PRI | NULL    | Signature ID          |
| sig_name     | varchar(255)     |      | MUL |         | Signature Name        |
| sig_class_id | int(10) unsigned | YES  | MUL | NULL    | Classification ID     |
| sig_priority | int(10) unsigned | YES  |     | NULL    | Priority              |
| sig_rev      | int(10) unsigned | YES  |     | NULL    | Revision number       |
| sig_sid      | int(10) unsigned | YES  |     | NULL    | Internal signature ID |
+--------------+------------------+------+-----+---------+-----------------------+
```

*sig_reference*

*7.4.1*

```
+---------+------------------+------+-----+---------+----------------------------------+
| Field   | Type             | Null | Key | Default | Description                      |
+---------+------------------+------+-----+---------+----------------------------------+
| sig_id  | int(10) unsigned |      | PRI | 0       | Signature ID                     |
| ref_seq | int(10) unsigned |      | PRI | 0       | Reference sequence number        |
| ref_id  | int(10) unsigned |      |     | 0       | Reference ID                     |
+---------+------------------+------+-----+---------+----------------------------------+
```

*reference*

```
+---------------+------------------+------+-----+---------+------------------------------+
| Field         | Type             | Null | Key | Default | Description                  |
+---------------+------------------+------+-----+---------+------------------------------+
| ref_id        | int(10) unsigned |      | PRI | NULL    | Reference ID                 |
| ref_system_id | int(10) unsigned |      |     | 0       | Reference system ID          |
| ref_tag       | varchar(20)      |      |     |         | Reference tag CVE-CAN )      |
+---------------+------------------+------+-----+---------+------------------------------+
```

*reference_system*

```
+-----------------+------------------+------+-----+---------+---------------------------+
| Field           | Type             | Null | Key | Default | Description               |
+-----------------+------------------+------+-----+---------+---------------------------+
| ref_system_id   | int(10) unsigned |      | PRI | NULL    | Reference system ID       |
| ref_system_name | varchar(20)      | YES  |     | NULL    | Reference system name     |
+-----------------+------------------+------+-----+---------+---------------------------+
```

*sig_class*

```
+----------------+------------------+------+-----+---------+----------------------------+
| Field          | Type             | Null | Key | Default | Description                |
+----------------+------------------+------+-----+---------+----------------------------+
| sig_class_id   | int(10) unsigned |      | PRI | NULL    | Signature classification ID|
| sig_class_name | varchar(60)      |      | MUL |         | Classification name        |
+----------------+------------------+------+-----+---------+----------------------------+
```

*data*

```
+--------------+------------------+------+-----+---------+------------------------------+
| Field        | Type             | Null | Key | Default | Description                  |
+--------------+------------------+------+-----+---------+------------------------------+
| sid          | int(10) unsigned |      | PRI | 0       | Sensor ID                    |
| cid          | int(10) unsigned |      | PRI | 0       | Event ID                     |
| data_payload | text             | YES  |     | NULL    | Packet payload encoded       |
+--------------+------------------+------+-----+---------+------------------------------+
```

*iphdr*

```
+----------+----------------------+------+-----+---------+----------- -----------------+
| Field    | Type                 | Null | Key | Default | Description                  |
+----------+----------------------+------+-----+---------+------------------------------+
| sid      | int(10) unsigned     |      | PRI | 0       | Sensor ID                    |
| cid      | int(10) unsigned     |      | PRI | 0       | Event ID                     |
| ip_src   | int(10) unsigned     |      | MUL | 0       | Source IP address            |
| ip_dst   | int(10) unsigned     |      | MUL | 0       | Destination IP address       |
| ip_ver   | tinyint(3) unsigned  | YES  |     | NULL    | IP version                   |
| ip_hlen  | tinyint(3) unsigned  | YES  |     | NULL    | IP Header length             |
| ip_tos   | tinyint(3) unsigned  | YES  |     | NULL    | IP type-of-service           |
| ip_len   | smallint(5) unsigned | YES  |     | NULL    | IP datagram length           |
| ip_id    | smallint(5) unsigned | YES  |     | NULL    | IP ID                        |
| ip_flags | tinyint(3) unsigned  | YES  |     | NULL    | IP flags                     |
| ip_off   | smallint(5) unsigned | YES  |     | NULL    | IP fragment offset           |
| ip_ttl   | tinyint(3) unsigned  | YES  |     | NULL    | IP time-to-live              |
| ip_proto | tinyint(3) unsigned  |      |     | 0       | IP protocol                  |
| ip_csum  | smallint(5) unsigned | YES  |     | NULL    | IP checksum                  |
+----------+----------------------+------+-----+---------+------------------------------+
```

*tcphdr*

```
+-----------+----------------------+------+-----+---------+----------------------+
| Field     | Type                 | Null | Key | Default | Description          |
+-----------+----------------------+------+-----+---------+----------------------+
| sid       | int(10) unsigned     |      | PRI | 0       | Sensor ID            |
| cid       | int(10) unsigned     |      | PRI | 0       | Event ID             |
| tcp_sport | smallint(5) unsigned |      | MUL | 0       | TCP source port      |
| tcp_dport | smallint(5) unsigned |      | MUL | 0       | TCP destination port |
| tcp_seq   | int(10) unsigned     | YES  |     | NULL    | TCP sequence number  |
| tcp_ack   | int(10) unsigned     | YES  |     | NULL    | TCP ACK number       |
| tcp_off   | tinyint(3) unsigned  | YES  |     | NULL    | TCP offset           |
| tcp_res   | tinyint(3) unsigned  | YES  |     | NULL    | TCP reserved         |
| tcp_flags | tinyint(3) unsigned  |      | MUL | 0       | TCP flags            |
| tcp_win   | smallint(5) unsigned | YES  |     | NULL    | TCP window           |
| tcp_csum  | smallint(5) unsigned | YES  |     | NULL    | TCP checksum         |
| tcp_urp   | smallint(5) unsigned | YES  |     | NULL    | TCP urgent pointer   |
+-----------+----------------------+------+-----+---------+----------------------+
```

*udphdr*

```
+-----------+----------------------+------+-----+---------+----------------------+
| Field     | Type                 | Null | Key | Default | Description          |
+-----------+----------------------+------+-----+---------+----------------------+
| sid       | int(10) unsigned     |      | PRI | 0       | Sensor ID            |
| cid       | int(10) unsigned     |      | PRI | 0       | Event ID             |
| udp_sport | smallint(5) unsigned |      | MUL | 0       | UDP soure port       |
| udp_dport | smallint(5) unsigned |      | MUL | 0       | UDP destination port |
| udp_len   | smallint(5) unsigned | YES  |     | NULL    | UDP length           |
| udp_csum  | smallint(5) unsigned | YES  |     | NULL    | UDP checksum         |
+-----------+----------------------+------+-----+---------+----------------------+
```

*icmphdr*

```
+-----------+----------------------+------+-----+---------+----------------------+
| Field     | Type                 | Null | Key | Default | Description          |
+-----------+----------------------+------+-----+---------+----------------------+
| sid       | int(10) unsigned     |      | PRI | 0       | Sensor ID            |
| cid       | int(10) unsigned     |      | PRI | 0       | Event ID             |
| icmp_type | tinyint(3) unsigned  |      | MUL | 0       | ICMP type            |
| icmp_code | tinyint(3) unsigned  |      |     | 0       | ICMP code            |
| icmp_csum | smallint(5) unsigned | YES  |     | NULL    | ICMP checksum        |
| icmp_id   | smallint(5) unsigned | YES  |     | NULL    | ICMP ID              |
| icmp_seq  | smallint(5) unsigned | YES  |     | NULL    | ICMP sequence number |
+-----------+----------------------+------+-----+---------+----------------------+
```

*opt*

```
+-----------+---------------------+------+-----+---------+---------------- ------------+
| Field     | Type                | Null | Key | Default | Description                  |
+-----------+---------------------+------+-----+---------+------------------------------+
| sid       | int(10) unsigned    |      | PRI | 0       | Sensor ID                    |
| cid       | int(10) unsigned    |      | PRI | 0       | Event ID                     |
| optid     | int(10) unsigned    |      | PRI | 0       | Option ID                    |
| opt_proto | tinyint(3) unsigned |      |     | 0       | Option protocol (IP, TCP)    |
| opt_code  | tinyint(3) unsigned |      |     | 0       | Option code                  |
| opt_len   | smallint(6)         | YES  |     | NULL    | Option length
| opt_data  | text                | YES  |     | NULL    | Option data                  |
+-----------+---------------------+------+-----+---------+------------------------------+
```

*acid_ag*

```
+----------+-----------------+------+-----+---------+---------------------------------+
| Field    | Type            | Null | Key | Default | Description                     |
+----------+-----------------+------+-----+---------+---------------------------------+
| ag_id    | int(10) unsigned |      | PRI | NULL    | Alert Group (AG) ID             |
| ag_name  | varchar(40)     | YES  |     | NULL    | AG name                         |
| ag_desc  | text            | YES  |     | NULL    | AG description                  |
| ag_ctime | datetime        | YES  |     | NULL    | Timestamp of AG creation time   |
| ag_ltime | datetime        | YES  |     | NULL    | Timestamp of last AG modification|
+----------+-----------------+------+-----+---------+---------------------------------+
```

*acid_ag_alert*

```
+--------+-----------------+------+-----+---------+---------------------+
| Field  | Type            | Null | Key | Default | Description         |
+--------+-----------------+------+-----+---------+---------------------+
| ag_id  | int(10) unsigned |      | PRI | 0       | Alert Group (AG) ID |
| ag_sid | int(10) unsigned |      | PRI | 0       | Sensor ID           |
| ag_cid | int(10) unsigned |      | PRI | 0       | Event ID            |
+--------+-----------------+------+-----+---------+---------------------+
```

*acid_ip_cache*

```
+--------------------+-----------------+------+-----+---------+-----------------------+
| Field              | Type            | Null | Key | Default | Description           |
+--------------------+-----------------+------+-----+---------+-----------------------+
| ipc_ip             | int(10) unsigned |      | PRI | 0       | IP address (32-bit)   |
| ipc_fqdn           | varchar(50)     | YES  | MUL | NULL    | FQDN                  |
| ipc_dns_timestamp  | datetime        | YES  |     | NULL    | DNS lookup timestamp  |
| ipc_whois          | text            | YES  |     | NULL    | whois information     |
| ipc_whois_timestamp | datetime        | YES  |     | NULL    | whois lookup time     |
+--------------------+-----------------+------+-----+---------+-----------------------+
```

## *7.5 Annex E*

```
use teststat;

scenario xtest2 ()
{
 global string CLASSIFICATION_NAME = "teststat_scenario2";
 global string CLASSIFICATION_URL  = "http://www.cs.ucsb.edu/~rsg";

 string SOURCE_USERNAME;
 string TARGET_USERNAME;
 string TARGET_PROC_PATH;
 string ADDITIONAL_DATA;

 transition trans1 (s0->s1) nonconsuming
 {
    [MESSAGE m1]
    {
       SOURCE_USERNAME = m1.from.username;
       TARGET_USERNAME = m1.to.username;
       ADDITIONAL_DATA = m1.body;
    }
 }


 transition trans2 (s1->slast) nonconsuming
 {
    [ACTION a1] : (a1.subject.username ==  TARGET_USERNAME)
    {
            TARGET_PROC_PATH = a1.object.oname;
    }

 }

 initial
 state s0 { }

 state s1 { }

 state slast
 {
   {
       log("Last state reached (teststat_scenario2)");
   }
 }
}
```