Working with SEC- the Simple Event Correlator, Part Two

Jim Brown

jpb@jimby.name

Copyright © 2004 Jim Brown

July 24, 2004

SEC by Risto Vaarandi, is a powerful event correlation engine written entirely in Perl that is capable of handling a wide variety of event correlation tasks.

Part One of this article introduced **SEC** and discussed the basics of setup and use. **Part Two** provides more in-depth examples of **SEC** usage and includes a complete example of setting up a database (**MySQL**) for use with **SEC** processing of logfiles on BSD systems.

Table of Contents

1 Introduction
2 Contexts Revisited
3 Miscellaneous SEC Topics
4 SEC Performance
5 Database Integration
6 Other Uses For SEC
7 Conclusion

1 Introduction

SEC is now (July, 2004) at version 2.2.5. Some of the examples in **Part Two** require features beyond 2.2. Check your version before attempting the examples below.

It's worth noting that event correlation is a complex subject. Events can be synchronous or asynchronous in nature, and trying to develop a framework where they can be well understood whenever they occur is difficult. **SEC** is quite capable of handling complexity assuming, of course, that a suitable ruleset can be designed.

In **Part Two**, only the example files and output will be given for each example. The default execution statement is:

```
$ perl sec.pl -conf=example.conf -input=- -debug=4
```

(no informative debug from SEC) unless otherwise noted.

---

2 Contexts Revisited

Contexts, as explained in **Part One**, are logical instances of data that relate to events. They exist, or they do not exist. In this section, we explore contexts in more depth.

2.1 Logical Operations With Contexts

The earlier examples used contexts as simple boolean flags. If a context existed, and a pattern was matched, a rule was considered matched and any actions were triggered.

Contexts can be combined with the following perl logical operators- !, &&, || and parentheses. Consider the following example:

```
# Example C5.1.01.conf
# Context logical operations
# Create FOO_CONTEXT on pattern 'foo'
# Create BAR_CONTEXT on pattern 'bar'
# Create BAZ_CONTEXT on pattern 'baz'
# On pattern 'checkme', write out message if all three exist.
#

type=Single
ptype=RegExp
pattern=foo
desc=$0
action=create FOO_CONTEXT; write - creating context FOO_CONTEXT;

type=Single
ptype=RegExp
pattern=bar
desc=$0
action=create BAR_CONTEXT; write - creating context BAR_CONTEXT;

type=Single
ptype=RegExp
pattern=baz
desc=$0
action=create BAZ_CONTEXT; write - creating context BAZ_CONTEXT;

type=Single
ptype=RegExp
pattern=checkme
context=FOO_CONTEXT && BAR_CONTEXT && BAZ_CONTEXT
desc=$0
action=write - FOO_CONTEXT && BAR_CONTEXT && BAZ_CONTEXT is TRUE
```

When run, the output looks like:

```
$ perl sec.pl -conf=C5.1.01.conf -input=- -debug=4
Simple Event Correlator version 2.1.11
Reading configuration from C5.1.01.conf
foo
creating context FOO_CONTEXT
bar
creating context BAR_CONTEXT
baz
creating context BAZ_CONTEXT
checkme
FOO_CONTEXT && BAR_CONTEXT && BAZ_CONTEXT is TRUE
checkme
FOO_CONTEXT && BAR_CONTEXT && BAZ_CONTEXT is TRUE
^C
```

This example required all three contexts to exist to produce output on the ``*checkme*'' input. A different logical condition on the ``*checkme*'' rule such as:

```
type=Single
ptype=RegExp
pattern=checkme
context=!FOO_CONTEXT && (BAR_CONTEXT || BAZ_CONTEXT)
desc=$0
```

```
action=write - !FOO_CONTEXT && (BAR_CONTEXT || BAZ_CONTEXT) is TRUE
```

is possible.

Output:

```
$ perl sec.pl -conf=C5.1.01.conf -input=- -debug=4
Simple Event Correlator version 2.1.11
Reading configuration from C5.1.01.conf
bar
creating context BAR_CONTEXT
baz
creating context BAZ_CONTEXT
checkme
!FOO_CONTEXT && (BAR_CONTEXT || BAZ_CONTEXT) is TRUE
foo
creating context FOO_CONTEXT
checkme
checkme
^C
```

More complex logical expressions are possible. Use of parentheses to disambiguate logical conditions is recommended. Note that the above examples contained contexts with infinite lifetimes. Complex timing scenarios are possible, and often depend on asynchronous input from multiple sources.

2.2 Verifying Contexts and State Information

With the rapid rise in complexity using logical operators, there is a need to be able to verify the state of all **SEC** variables, contexts and state information. The is where the ``*USR1*'' signal and the *-dump* parameter come in.

Sending the **SEC** process the USR1 signal directs **SEC** to dump information about the current state of its contexts, internal lists, rule usage statistics and other useful information. By default, **SEC** will dump this internal information to `/tmp/sec.dump`. Use the *-dump=filename* parameter to change the default setting.

To see how this works, re-run the above example and, after the first input (`foo`) send the perl process running the **SEC** program the USR1 signal:

```
$ perl sec.pl -conf=C5.1.01.conf -input=- -debug=4
Simple Event Correlator version 2.1.11
Reading configuration from C5.1.01.conf
foo
creating context FOO_CONTEXT
```

Then, from another window or terminal session, signal **SEC** with the USR1 signal. This example is from a BSD based system:

```
$ ps -ax | grep sec
  379  p0  S+     0:00.18 perl sec.pl -conf=C5.1.01.conf -input=- -debug=4
$ kill -USR1 379
```

Upon returning to the original window, notice that **SEC** has written a notification that signal USR1 was received:

```
$ perl sec.pl -conf=C5.1.01.conf -input=- -debug=4
Simple Event Correlator version 2.1.11
Reading configuration from C5.1.01.conf
foo
creating context FOO_CONTEXT
SIGUSR1 received: dumping data to /tmp/sec.dump
```

Examining `/tmp/sec.dump` we find:

```
$ cat /tmp/sec.dump
Date of dump: Fri Dec  5 16:18:57 2003
Program version: 2.1.11

Performance statistics:❶
============================================================
Run time: 108 seconds
User time: 0.2109375 seconds
System time: 0.0546875 seconds
Child user time: 0 seconds
Child system time: 0 seconds
Processed input lines: 1

Rule usage statistics:
============================================================

Statistics for the rules from C5.1.01.conf
------------------------------------------------------------
Rule 1 at line 9 ($0) has matched 1 events❷
Rule 2 at line 15 ($0) has matched 0 events
Rule 3 at line 21 ($0) has matched 0 events
Rule 4 at line 27 ($0) has matched 0 events

Content of input buffer:
------------------------------------------------------------

foo❸
------------------------------------------------------------

Pending events that are generated by rules:
------------------------------------------------------------
------------------------------------------------------------

List of active event correlation operations:
============================================================
Total: 0 elements

List of active contexts:❹
============================================================
Context Name: FOO_CONTEXT
Creation Time: Fri Dec  5 16:17:12 2003
Lifetime: infinite
------------------------------------------------------------
Total: 1 elements

Running children:❺
============================================================
Total: 0 elements
```

Note the following:

❶

General usage and performance statistics reveal various details of **SEC** operation.

❷

This section shows how many times each rule has been matched. In this example, Rule 1 (starting on line 9 of the configuration file) has been matched exactly once.

❸

Contents of the input buffer, showing the last several lines. The size of the input buffer is contolled by the *-bufsize=n* parameter. The default is 10 lines. (Note- this example was shortened for readability.)

❹

The list of active contexts are shown here. There is one entry for each context which lists all the details about the context- lifetime, expiration actions, etc.

❺

Any child processes started by **SEC** will be listed in this section. There are none in this example.

Using dumps to determine the exact state of **SEC** is quite useful in situations involving many complex, interrelated events.

2.3 More Mini-Program Examples

Part One of this article introduced **SEC** mini-programs- small perl programs that can function as a logical context. This section presents two more examples of using perl mini-programs- one that interrupts normal event processing, and one that utilizes external perl modules with SEC.

The first example creates a rule that interrupts normal event processing. When the keyword ``*foo*'' is entered, the mini-program is entered and the user is prompted for a secret word:

```
# Example C5.2.01.conf
# First rule interrupts event processing until
# secret word is entered.

type=Single
ptype=RegExp
pattern=foo
context= = \
({ my $inp; \
   my $t1; \
   while(1) \
   { \
     print "Enter secret keyword to continue:"; \
     $inp = <>;  \
     chop $inp; \
     print "you said [$inp]\n";  \
     last if ($inp eq "ZuLu"); \
   } \
   1; \
};)
desc=$0
action=write - Continuing...

# Echo input.
type=Single
ptype=RegExp
pattern=.*
desc=$0
action=write - Entry was [%s]
```

Note that in this example, when ``*foo*'' is entered and the while loop is entered, input taken from the keyboard does *not* pass into SEC's input buffer and get evaluated against the ruleset. This is confirmed by signaling **SEC** to dump its internals as discussed above. Output from this example is:

```
$ perl sec.pl -conf=C5.2.01.conf -input=- -debug=4
Simple Event Correlator version 2.1.11
Reading configuration from C5.2.01.conf
ready
Entry was [ready]
set
Entry was [set]
foo
Enter secret keyword to continue:grok
you said [grok]
Enter secret keyword to continue:zonk
you said [zonk]
Enter secret keyword to continue:ZuLu
you said [ZuLu]
Continuing...
one
Entry was [one]
two
Entry was [two]
SIGUSR1 received: dumping data to /tmp/sec.dump
^C
```

Output from the dumpfile confirms that our incorrect gueses (and the correct guess as well) are not in the input buffer:

```
...
Statistics for the rules from C5.2.01.conf
------------------------------------------------------------
Rule 1 at line 3 ($0) has matched 1 events
Rule 2 at line 23 ($0) has matched 4 events
```

```
Content of input buffer:
------------------------------------------------------------
ready
set
foo
one
two
------------------------------------------------------------
...
```

The second example shows a different use of mini-programs from the **SEC** mailing list archives. William Gertz has demonstrated how it is possible to load a complete perl module at **SEC** startup (combining mini-programs with the special **SEC** internal event *SEC_STARTUP*).

Extending SEC Rules With Perl Modules (William Gertz).

Suppose we need to check whether an IP address or network is within a CIDR block range. Creating a regular expression covering a large group of addresses seems like carpentry with scalpel. Consider:

```
172.16.20.128/26
```

prefix notation for *172.16.20.128 - 172.16.20.191*. A regular expression that will match this range is:

```
pattern = \s(172\.16\.20\.1(?:2[89])|(?:[3-8][0-9])|(?:9[01])),\s
```

Unfortunately, adding more ranges to check makes the pattern even more complex. We need something more fitting like a band saw rather than a surgical instrument. Fortunately there is **NetAddr::IP** from *CPAN*.

If NetAddr::IP was already loaded and a range set, we could do the match using a context Perl miniprogram:

```
...
pattern=\s(\S+),\s
ptype=RegExp
context= =(NetAddr::IP->new('$1')->within($range))
...
```

If the module can be loaded, *$range* can be used within a **SEC** rule. But we don't want a rule that loads the module for each line of input- we need a rule that only loads the module at startup. For that we'll set up a rule that matches only the *SEC_STARTUP* event:

```
type=Single
desc=Module load and range setup
ptype=SubStr
pattern=SEC_STARTUP ❶
context=[SEC_INTERNAL_EVENT] ❷
action=assign %a 0; \❸
    eval %a (require NetAddr::IP; \
    $range = NetAddr::IP->new('172.16.20.128/26'); 1;); \❹
    eval %a (exit(1) unless %a);
```

Note the following:

❶ The event *SEC_STARTUP* is inserted by **SEC** at startup when SEC is executed up with the -*intevents* parameter.

❷ SEC also generates a special context, *SEC_INTERNAL_EVENT*, just before an internal event (in this case *SEC_STARTUP*) is processed. This contex is deleted immediated after the internal event is processed. The square brackets tell **SEC** to test the context before evaluating the pattern. **SEC** will work without them, but it runs faster if this check is made first.

❸

The action loads the **NetAddr::IP** module but forces perl to exit if the load or *$range* setup fails.

❹

Loads the module through the *require* function, and sets *$range* through the new method call. It sets %a to 1 (the success indicator) only if the eval action runtime compile succeeds. If the eval action fails, **SEC** doesn't change %a from the default failed indicator (0).

For this example, **SEC** version 2.2 or greater must be used. Here is the complete ruleset for this example:

```
# Example C5.2.02.conf
# Rule 1 load NetAddr::IP on SEC startup.
# Rule 2 uses the NetAddr->within() method
#        to check an incoming IP
# Rule 3 matches any IP address input not found by Rule 2
type=Single
desc=Module load and range setup
ptype=SubStr
pattern=SEC_STARTUP
context=[SEC_INTERNAL_EVENT]
action=assign %a 0; \
    eval %a (require NetAddr::IP; \
    $range = NetAddr::IP->new('172.16.20.128/26'); 1;); \
    eval %a (exit(1) unless %a);

type=Single
desc=$0
ptype=RegExp
pattern=(\d+\.\d+\.\d+\.\d+)
context= =(NetAddr::IP->new('$1')->within($range))
action=write - Matched %s.

type=Single
desc=$0
ptype=RegExp
pattern=.*
action=write - %s was not matched.
```

Run this example with the following command (includes informative debug output):

```
$ perl sec.pl -conf=C5.2.02.conf -input=- -intevents
```

Example output:

```
$ perl sec.pl -conf=C5.2.02.conf -input=- -intevents
Simple Event Correlator version 2.2.beta2
Reading configuration from C5.2.02.conf
3 rules loaded from C5.2.02.conf
Creating SEC internal context 'SEC_INTERNAL_EVENT'
Creating SEC internal event 'SEC_STARTUP'
Assigning value '0' to variable '%a'
Evaluating code 'require NetAddr::IP;     $range = NetAddr::IP-
>new('172.16.20.128/26'); 1;' and setting variable '%a'
Assigning value '1' to variable '%a'
Evaluating code 'exit(1) unless 1' and setting variable '%a'
Assigning value '1' to variable '%a'
Deleting SEC internal context 'SEC_INTERNAL_EVENT'
172.16.20.128
Writing event 'Matched 172.16.20.128.' to file -
Matched 172.16.20.128.
1.2.3.4
Writing event '1.2.3.4 was not matched.' to file -
1.2.3.4 was not matched.
172.16.20.191
Writing event 'Matched 172.16.20.191.' to file -
Matched 172.16.20.191.
172.16.20.192
Writing event '172.16.20.192 was not matched.' to file -
172.16.20.192 was not matched.
^C
```

3 Miscellaneous SEC Topics

3.1 More Uses of SEC Variables

This section includes additional examples of variable use in conjunction with the **SEC** internal startup event.

Consider the following rule set:

```
# Example C6.1.01.conf
# Setting variables through the environment.
#
type=Single
ptype=RegExp
pattern=(SEC_STARTUP|SEC_RESTART)
desc=$0
context=SEC_INTERNAL_EVENT
action=eval %P ( $ENV {'SEC_MAINDIR'} );\
       eval %S ( $ENV {'SEC_SCRIPTSDIR'} );\
       eval %F ( $ENV {'SEC_FIFO'} );\
       spawn /usr/bin/tail -f %P/SEC_input2

#
# Test rule.
#
type=Single
ptype=RegExp
pattern=testme
desc=$0
action=write - %P ; \
       write - %S ; \
       write - %F
```

This example shows how variables can be set through environment variables for use in later rules.

To be useful, the environment variables *SEC_MAINDIR*, *SEC_SCRIPTSDIR*, and *SEC_FIFO* should be set before running this example. In **sh** and derived shells, set these variables as follows:

```
$ SEC_MAINDIR=foo/main ; export SEC_MAINDIR
$ SEC_SCRIPTSDIR=foo/scripts ;export SEC_SCRIPTSDIR
$ SEC_FIFO=foo/fifo ; export SEC_FIFO
```

The *spawn* action in the above rule executes the **tail** program. This is used to enable **SEC** to gather input from more than one input stream at a time. **tail** requires that the file exist, so:

```
$ mkdir -p $SEC_MAINDIR
$ touch $SEC_MAINDIR/SEC_input2
```

before using.

This example depends on using the -*intevents* parameter:

```
$ perl sec.pl -conf=C6.1.01.conf -input=- -intevents
```

Output with informative debug:

```
$ perl sec.pl -conf=C6.1.01.conf -input=- -intevents
Simple Event Correlator version 2.1.11
Reading configuration from C6.1.01.conf
2 rules loaded from C6.1.01.conf
Creating context 'SEC_INTERNAL_EVENT'
Creating SEC internal event 'SEC_STARTUP'
Evaluating code '$ENV {'SEC_MAINDIR'} ' and setting variable '%P'
Assigning value 'foo/main' to variable '%P'
Evaluating code '$ENV {'SEC_SCRIPTSDIR'} ' and setting variable '%S'
Assigning value 'foo/scripts' to variable '%S'
Evaluating code '$ENV {'SEC_FIFO'} ' and setting variable '%F'
Assigning value 'foo/fifo' to variable '%F'
Spawning shell command '/usr/bin/tail -f foo/main/SEC_input2'
Child 1445 created for command '/usr/bin/tail -f foo/main/SEC_input2'
Deleting context 'SEC_INTERNAL_EVENT'
```

```
testme
Writing event 'foo/main' to file -
foo/main
Writing event 'foo/scripts' to file -
foo/scripts
Writing event 'foo/fifo' to file -
foo/fifo

    (Received input from second input stream- see below.)

Creating event 'testme' (received from child 1445)
Writing event 'foo/main' to file -
foo/main
Writing event 'foo/scripts' to file -
foo/scripts
Writing event 'foo/fifo' to file -
foo/fifo
^C
```

The first instance of `testme` was entered at the keyboard. The second instance was entered from:

```
$ echo testme >> foo/main/SEC_input2
```

3.2 Notes on Logging Windows Systems

Microsoft Windows systems do not use **syslog** for event logging. Instead, Windows sends event messages to an internal event logging system. These event logs can be viewed with the **Event Viewer** application located in the Administrative Tools area. Three types of logs are available, Application, System, and Security. In this section, we will discuss how to get Windows systems to log with syslog, and how to read them with SEC.

A full treatment of Windows logging is beyond the scope of this article. (See the Microsoft Knowledge Base article `` How To Enable and Apply Security Auditing in Windows 2000'' for a more extensive treatment of event logs.) In order to convert Microsoft event logs to syslog messages, third-party applications are ususally used. For this discussion, we will use the freely available **Snare for Windows** product from Intersect Alliance available here. The general principles discussed here are applicable to other products as well.

Install the **Snare** application on your Windows system. Once installed, the **Snare** application starts up as a service. By default, it sends syslog messages to the local host (127.0.0.1). To configure **Snare** to send logs to another host, click ``Setup'', then ``Audit Configuration''. Enter the local host name (your Windows system name) and the IP or DNS address of the remote syslog server. You may also change the ``Facility'' (default ``User'') and ``Level'' (default ``Notice''). Be sure your `/etc/syslog.conf` configuration is configured to handle the Facility and User specified.

Once these initial configurations are accepted, **Snare** will begin sending log entries to the syslog host. Ensure that the syslog host is configured to allow syslog entries from remote hosts, and restart **syslogd** if necessary. By default, **Snare** has ``detailed'' logging turned on, and you should begin to see a fair amount of logs arriving at the log server. The entries are tab delimited and look similar to the following (folded for readability):

```
Jul 28 20:50:09 10.1.1.50 78LYH8R MSWinEventLog 0       Security        406     Wed
Jul 28 20:50:09 2004
  592     Security        JBrown  User    Success Audit 78LYH8R Detailed Tracking
A new
  process has been created:     New Process ID: 2004     Image File Name: \WINNT\
system32\notepad.exe
  Creator Process ID: 1416     User Name: JBrown     Domain: 78LYH8R     Logon ID:
(0x0,0x15923)   0
```

```
Jul 28 20:51:31 10.1.1.50 78LYH8R MSWinEventLog 0       Security        413     Wed
Jul 28 20:51:25 2004
  593     Security        JBrown  User    Success Audit 78LYH8R Detailed Tracking
A process
  has exited:     Process ID: 888     User Name: JBrown     Domain: 78LYH8R  Logon
ID: (0x0,0x15923)   7

Jul 28 20:53:17 10.1.1.50 78LYH8R MSWinEventLog 0       Security        420     Wed
Jul 28 20:53:17 2004
  592     Security        JBrown  User    Success Audit 78LYH8R Detailed Tracking
A new
  process has been created:     New Process ID: 1660     Image File Name: \WINNT\
system32\CMD.EXE
  Creator Process ID: 1416     User Name: JBrown     Domain: 78LYH8R     Logon ID:
(0x0,0x15923)   14


Jul 28 20:51:36 10.1.1.50 78LYH8R MSWinEventLog 0       Security        415     Wed
Jul 28 20:51:36 2004
 593     Security        JBrown  User    Success Audit 78LYH8R Detailed Tracking
A process
 has exited:     Process ID: 2004     User Name: JBrown     Domain: 78LYH8R  Logon
ID: (0x0,0x15923)   9
```

(The fields and their meaning are described in the above Microsoft article.)

The first entry is a notification that the **notepad** application was started (process ID 2004). The second entry indicates that process ID 888 exited, but there is no way to tell the name of the process that exited without tracking all processes as they start up and recording the process name along with the process ID. This is a very resource intensive operation when there are thousands of processes per minute in a busy system. The third entry is the start of the Windows **CMD.EXE** the command line interface. The last entry is the shutdown of the **notepad** application noted in the first entry.

Clearly, there is a *lot* of data being generated by the event subsystem in Windows. Careful tuning of the Local Security Policy, as well as the **Snare** ``Audit Reporting Objectives'' can reduce the overall load.

If all you want to track with **SEC** is the ``Process Creation'' event type, the following rule should detect these events. The long pattern line contains escaped newlines for readability.

```
#
# Snare syslog Rule for Process Creation
#

type=Single
ptype=RegExp
pattern=\S+\s+\d+\s+\S+\s+(\S+)\s+(\S+)\s+\
MSWinEventLog\t\d+\tSecurity\t\d+\t.*?\t(\d+)\t\
Security\t\S+\t\S+\tSuccess Audit\t(\S+)\t\
Detailed Tracking.*?Process ID: (\d+).*?Name:\s+(.*?)\
Creator Process ID: (\d+)\s+User Name: (\S+)\s+\
Domain: (\S+)\s+Logon ID: (\S+)\s+\d+.*
desc=$0
action=write - Found pattern [%s]
```

with pattern backreferences Host or IPaddr, System Name, Windows Event ID, Hostname, new process ID, new process name, parent process ID, User Name, Windows domain, and Logon ID. If this is too much, perhaps just the essential:

```
pattern=\S+\s+\d+\s+\S+\s+(\S+)\s+(\S+)\s+.*?New Process ID: (\d+).*?Name:\s+
(.*?)Creator.*
```

will do with backreferences Host or IPaddr, System Name, new process ID, and new process name.

---

4 SEC Performance

An important question to ask before building a logging and monitoring system based on **SEC** is, *"What kind of load is **SEC** capable of handling?"* This section investigates this question and provides some tools for answering it in your local situation. It presents a framework of some test data and scripts to test SEC.

4.1 SEC Performance Issues

Many factors influence the performance of a logging and monitoring system. A short list includes:

- Hardware platform
- Ruleset size
- Perl regular expression complexity
- SEC actions
- Velocity of data at **SEC** input
- **syslogd** (if used)
- Processor load
- Backend storage system (database or equivalent)
- Network congestion

Keep this list in mind during the discussion of the test framework below.

4.2 The SEC Performance Test Framework

This section presents a framework for testing **SEC** performance. It consists of:

- *Test Data* a collection of raw syslog messages captured during a typical project.
- *SEC Rules* to recognize messages in the test data. For this framework, complicated actions have been removed.
- *Test Scripts* to start and stop the test, and capture performance metrics.

The *Test Data* is simply a collection of messages. This data can be any collection of text, including the text of this article. However, the goal of this test is to measure **SEC** performance reading syslog messages. If your test requirements are different, you can still use this framework. Simply adjust the data files for size and prepare the end message as described below.

The tests in this section used three data files; a 1 MB message file, a 10MB message file, and a 100MB message file. These files contain normal (and abnormal) syslog messages. The files are used in two ways; (1) fed directly into **SEC** input, and (2) sent to **syslogd** from a special program, *File2Syslog.pl*. More details on how this works are contained in the next section.

The files also contain a special end message: *EOF*. This text string identifies the end of the file and is matched by a special rule:

```
#
# SEC Performance Test Rule
# Look for EOF at the end of the line, and send ourselves
# a USR1 signal to dump statistics, and a TERM signal to
#end the program.
type=Single
ptype=RegExp
pattern=EOF\s*$
desc=$0
action=eval %k ( $pid=$$$; kill (USR1, $pid); kill(TERM, $pid));
```

When the above rule is matched, the process id of the current program is read, and used to generate a statistics dump, and then end the program. It's important to test for the string ``EOF'' at the end of the input line to test the effort required to scan an entire line. Otherwise, the perl regular expression code will not be as fully exercised.

Making small **SEC** rules files is easy. Coming up with large rule sets requires automation. Below is a simple rule generating program that creates a set of rules to use with this test framework. The script reads a dictionary and stores it in memory. It then outputs a configurable number of rules each formatted like a sample syslog entry with the dictionary words placed in the text portion of the entry. A large data set of rules can be generated for **SEC** performance testing using this script:

```
#!/usr/bin/perl
#
# rulemaker.pl
#
#
# Create SEC rules file from template using dictionary wordlist.
# usage: perl rulemaker.pl NumberOfRules
#

sub usage() {
  print "\nusage: perl rulemaker.pl NumberOfRules\n\n";
  exit 1;
}

# Get argument.
if($#ARGV == -1)
{
  &usage;
}
else
{
  $counter = $ARGV[0];
}

# Change to common dictionary file.
$filename = "/usr/share/dict/web2";

# Get words
open WORDLIST, $filename or die "Can't open wordlist [$filename]: $!";
@words = <WORDLIST>;
close WORDLIST;

#Solaris template
$template="\\S+\\s+\\d+\\s+\\S+\\s+(\\S+)\\s+(\\S+):\\s+\\[.*?\\]\\s+WORDS";

#Generate many rules
for ($i=1; $i<$counter ; $i++)
{

  $tmpl_words = "";

  # Pick up just two words.
  # This can also be made variable if desired.
  for($j=1; $j<=2; $j++)
  {
    #
    # No random seed here...
    #
    $randval = rand($#words);
    $randword = $words[$randval];
```

```
    chop $randword;
    $tmpl_words .= "$randword ";
  }

  chop $tmpl_words;  #remove blank at end

  $t_template = $template;

  # Make substitution into template.
  $t_template =~ s/WORDS/$tmpl_words/;

  #print "[$t_template]\n";

  print  << "END";
#Rule $i
type=Single
ptype=RegExp
pattern=$t_template
desc=\$0
action=none

END

}

#
# Print the Last Rule
#

print  << "LAST";
# End of File Rule
type=Single
ptype=RegExp
pattern=EOF\\s*\$
desc=\$0
action=eval %k ( \$foo=\$\$\$; kill (USR1, \$foo); kill(TERM, \$foo));
LAST

exit 0;
```

The *Test Script* performs a single test, starting **SEC** with one rules file and one data file. The script calls *File2Syslog.pl* to push the data into **syslogd**. Once the last line of the data file is read, SEC performs the action on the rule, sending both the dump signal, and the termination signal. By pairing different sizes of rules files and data files, you can get a very good idea of how **SEC** performs with small and large data sets and with small and large rules files. The results are shown in the next section.

```
#!/bin/sh
#
# SEC Performance Test Driver Script
#
# Clear out messages file  (BSD)
cp /dev/null /var/log/messages
sleep 1

# Start SEC with appropriate rules file
time perl sec.pl -conf=T01.conf -input=/var/log/messages &
#time perl sec.pl -conf=T50.conf -input=/var/log/messages &
#time perl sec.pl -conf=T100.conf -input=/var/log/messages &
#time perl sec.pl -conf=T500.conf -input=/var/log/messages &

# Sleep to give SEC time to read large rule files.
sleep 4

perl File2Syslog.pl test.1MRealMsgs
#perl File2Syslog.pl test.10MRealMsgs
#perl File2Syslog.pl test.100MRealMsgs
```

You may find it necessary to restart syslog so as to allow writing to **syslogd** from a local process. Under BSD, starting **syslogd** without any options meets this requirement. You may also find it necessary to disable any syslog rotation facilities that are active during these tests. BSD systems usually use *newsyslog* to rotate logs. This can be commented out under FreeBSD in the `/etc/newsyslog.conf` file as follows:

```
...
/var/log/maillog                        640  7     *    @T00  Z
```

```
/var/log/sendmail.st                    640  10    *    168   B
#/var/log/messages                      644  5     100  *     Z
/var/log/all.log                        600  7     *    @T00  Z
...
```

4.3 SEC Test Results

Using the scripts, and rules described in the last section, several tests were performed. The following table lists results for data directly copied into the **SEC** input file (not through **syslogd**):

Table 1. SEC Performance With Data Copied Directly Into Input

No. Rules per File	Input Lines	No. Bytes	Run Time	User Time	Sys Time	Msgs/ Second	Notes
1	10436	1000004	1.47	0.36	0.05	7099.32	test.1MRealMsgs
1	104351	10000004	4.73	3.01	0.43	22061.52	test.10MRealMsgs
1	1043501	100000004	37.66	29.31	4.17	27708.47	test.100MRealMsgs

The next table lists results for data sent via File2Syslog.pl to **syslogd**. **SEC** was configured to read the syslog messages file:

Table 2. SEC Performance With Data Processed Through syslogd

No. Rules per File	Input Lines	No. Bytes	Run Time	User Time	Sys Time	Msgs/ Second	Notes
1	10436	1000004	3.37	0.36	0.04	3096.74	test.1MRealMsgs
1	104351	10000004	13.49	2.72	0.45	7735.43	test.10MRealMsgs
1	1043501	100000004	122.04	28.66	4.29	8550.48	test.100MRealMsgs
50	10436	1000004	6.69	3.75	0.06	1559.94	test.1MRealMsgs
50	104351	10000004	50.00	38.57	0.43	2087.02	test.10MRealMsgs
50	1043501	100000004	524.14	426.74	4.42	1990.88	test.100MRealMsgs
100	10436	1000004	10.41	7.42	0.05	1002.50	test.1MRealMsgs
100	104351	10000004	89.23	77.77	0.47	1169.46	test.10MRealMsgs
100	1043501	100000004	932.67	830.51	4.89	1118.83	test.100MRealMsgs
500	10436	1000004	44.25	38.28	0.09	235.84	test.1MRealMsgs
500	104351	10000004	411.69	395.35	0.65	253.47	test.10MRealMsgs
500	1043501	100000004	9358.96	8931.46	12.86	111.50	test.100MRealMsgs

The above tests were performed on an IBM T41, 1.Ghz, 512MB Ram, internal 60GB IDE disk, running FreeBSD 4.10 with the KDE desktop. All syslog was local (not connected to a network). All tests were performed five times and the results were averaged resulting in the values shown above.

---

5 Database Integration

There is considerable effort in getting **SEC** integrated with a database. The effort requires shifting attention from SEC to the mechanics of getting a database installed and creating scripts to write to and read from the database. This section outlines the following general steps:

- Install database software and perl *DBI* and *DBD* modules
- Create database schema
- Create database insert code
- Connect **SEC** and database insert code with IPC

To fully combine **SEC** with a database, you should have a working knowledge of SEC, and the perl *DBI* and *DBD::MySQL* modules.

Inter-process communication (IPC) using named pipes (FIFOs) was discussed in Part One, Section 3.4. where programs for reading and writing to named pipes were demonstrated. This section expands on that background, building a data base to capture and manage syslog entries.

Because a lot of these steps are system specific only a high level discussion of how these steps are performed are included. Read the database documentation on installation and configuration for your system before proceeding.

5.1 Installing MySQL and required Perl Modules

The **MySQL** database will be used for this section. Follow the download and installation procedures for **MySQL** for your system. Start up the data base engine and create a new data base for use called **sedb**, (for ``Simple Event Data Base") as follows:

```
$ mysql.sh start
$ mysqladmin -u root -p create sedb
Enter password: dummy
$
```

Next, follow the instructions on CPAN for installing the Perl *DBI::* and *DBD::mysql* driver modules.

To verify that your installation is complete, you should now be able to run the following program without error:

```
#!/usr/bin/perl
# MySQL installation verification.
# Open database 'sedb', then just exit.
# No errors should occur.

use strict;
use Socket;
use DBI();

# Connect to the database.
my $dbh = DBI->connect("DBI:mysql:database=sedb;host=localhost",
            "root", "dummy",
            {'RaiseError' => 1});
exit;
```

This ensures that you have a database correctly installed, and that connectivity to the database with the *DBI* and *DBD::mysql* perl modules works correctly. If errors occur in running the above program, fix any problems in your installation before proceeding.

It is also an excellent idea to read the Perl DBI:: documentation thoroughly. There are many pieces to this module and it will pay off in the long run to have a ``more than passing acquaintance '' with it.

5.2 Connecting SEC to MySQL

Before going further, it is best to test exactly how **SEC** and **MySQL** will function together. The communication between the two programs will be via FIFOs (named pipes). Part One already showed how **SEC** can write to a FIFO. It also demonstrated a small program to read from a FIFO. All that is left to do, is to enable the reading program to insert records into the database.

The script below (adapted from the Perl::DBI bundle) has been modified to read from a FIFO and enter textlines into the database.

```
#!/usr/bin/perl
#
# Example S8.2.01.pl  - Script to read data out of a named pipe
#                       and write to MySQL database.
#
#
$| = 1;

use strict;
use DBI();

my $filename;
my $inputline;
my $linenumber;


# Open FIFO first.  FIFO must already exist.
$filename = "./SEC_fifo";
open(FIFO, "+< $filename") or die "FIFO error on $filename $!";

# Connect to MySQL. Database 'sedb' must already exist.

my $dbh = DBI->connect("DBI:mysql:database=sedb;host=localhost",
            "root", "dummy",
            {'RaiseError' => 1});


# Drop table 'foo'. This may fail, if 'foo' doesn't exist.
# Thus we put an eval around it.
eval { $dbh->do("DROP TABLE foo") };
print "Dropping table foo failed: $@\n" if $@;

# Create a new table 'foo'. This must not fail, thus we don't
# use eval.
$dbh->do("CREATE TABLE foo (id INTEGER, textline VARCHAR(80))");

print "Reading from FIFO...\n";

while (<FIFO>)
{
  $inputline = $_;
  # Quit read loop when requested.
  last if($inputline =~ /quit/i);

  chop $inputline;
  $linenumber++;
  print "Got: [$inputline], ";

  # Insert data into table 'foo', using placeholders
  $dbh->do("INSERT INTO foo VALUES (?, ?)",
                undef,
                $linenumber,
                $inputline);

  print "inserted it.\n";

}


# Now retrieve data from the table.
my $sth = $dbh->prepare("SELECT * FROM foo");
```

```
$sth->execute();
while (my $ref = $sth->fetchrow_hashref()) {
  print "Found a row: id = $ref->{'id'}, line = $ref->{'textline'}\n";
}
# Drop table 'foo'. At this point, table 'foo' exists, so
# we won't use an eval.
$dbh->do("DROP TABLE foo");

$sth->finish();

# Disconnect from the database.
$dbh->disconnect();

exit;
```

Before running this example, ensure that you have a FIFO named `SEC_fifo` in the directory where the above script is run. Also, it may be helpful to have at least three terminal sessions opened for this example.

To begin, make sure that **MySQL** is running on your system:

```
$ ps -ax | grep mysql
 1116  p2  I+     0:00.01 mysql -u root -p sedb
 1045  p5  I      0:00.01 /bin/sh /usr/local/bin/mysqld_safe --user=mysql --
datadir=/var/db/mysql --pid-file=/var/
 1063  p5  S      0:01.97 /usr/local/libexec/mysqld --basedir=/usr/local --
datadir=/var/db/mysql --user=mysql --pi
```

Next, start up the reading script:

```
$ perl S8.2.01.pl
DBD::mysql::db do failed: Unknown table 'foo' at S8.2.01.pl line 30.
Dropping table foo failed: DBD::mysql::db do failed: Unknown table 'foo' at
S8.2.01.pl line 30.

Reading from FIFO...
```

The error output is OK in this case because the table does not exist in the newly created database. The script actually creates the table it is about to use. The reading script is now reading from the FIFO, waiting for input.

The first input test is simple. In another window, enter the following:

```
$ echo blah blah blah >> SEC_fifo

$ echo blah blah blah2 >> SEC_fifo
$ echo quit >> SEC_fifo
```

which causes the script to read and process the input:

```
Reading from FIFO...
Got: [blah blah blah], inserted it.
Got: [blah blah blah2], inserted it.
Found a row: id = 1, line = blah blah blah
Found a row: id = 2, line = blah blah blah2
$
```

The script read from the FIFO and inserted the text into the database table. When *quit* was entered, the main input terminated, and the script printed out all the rows in the database table. This test exercises the FIFO read and database insert and retrieval code.

Next, restart the script and, in another session, start up **SEC** with the same configuration file used in the FIFO example in Part One and enter some test data:

```
# Example C3.4.01.conf
# Recognize any pattern and write to a FIFO.
#
type=Single
```

```
ptype=RegExp
pattern=(.*)
desc=$0
action=write SEC_fifo %s
```

```
$ perl sec.pl -conf=C3.4.01.conf -input=-
Simple Event Correlator version 2.1.11
Reading configuration from C3.4.01.conf
1 rules loaded from C3.4.01.conf
Database insert test line A.
Writing event 'Database insert test line A.' to file SEC_fifo
Database insert test line B.
Writing event 'Database insert test line B.' to file SEC_fifo
Here is line C.
Writing event 'Here is line C.' to file SEC_fifo
Line D is the last line...
Writing event 'Line D is the last line...' to file SEC_fifo
quit
Writing event 'quit' to file SEC_fifo
```

As each line is processed by SEC, it is written to `./SEC_fifo`, and received by the reader script. The reader script inserts the whole line into the database, along with a line id number:

```
Got: [Database insert test line A.], inserted it.
Got: [Database insert test line B.], inserted it.
Got: [Here is line C.], inserted it.
Got: [Line D is the last line...], inserted it.
```

When ``quit'' is received, the reader script quits the read loop, and falls through to the second part of the script where it reads out all records in table 'foo', drops the table, disconnects from the **sedb** database and exits:

```
Found a row: id = 1, line = Database insert test line A.
Found a row: id = 2, line = Database insert test line B.
Found a row: id = 3, line = Here is line C.
Found a row: id = 4, line = Line D is the last line...
$
```

Note that **SEC** is still running however, and any further attempts to process input and write to the now-closed FIFO will fail:

```
Line E is entered after reader script closed.
Writing event 'Line E is entered after reader script closed.' to file SEC_fifo
Can't open pipe SEC_fifo for writing event 'Line E is entered after reader script
closed.'!
```

This illustrates one of the key problems with this type of interprocess communication- both sides must always be up and ready to receive input. A production quality system would normally use some sort of process control monitor for each of these processes, restarting them if they fail.

It may also be necessary to tune the operating system to account for bursty data which might overflow a FIFO. Procedures for this are operating system dependant. Check the documentation for your system.

5.3 The Simple Event Data Base

Using a database to collect, manage, and report log entries is not new. Many commercial products and some Open Source products now utilize a database backend. But, with all the flexibility a database provides, it also introduces complexity in installation, operation and on-going maintenance.

For our purposes, we will keep the design goals of *sedb* deliberately simple. We want to keep raw text, and we also want to aggregate information on different *types* and *levels* of logged events. So:

- *sedb* should capture raw log text and apply an MD5 checksum to each record as it is stored.
- *sedb* should store event correlation information from SEC, regarding the *threat type* and the *threat level*.

Threats can be categorized in two ways- a *threat type* is a category that determines how to group similar events. Section 5.2 in Part One introduced several threat types that will be used in this section. These are primarily security oriented types, but adding new types is straightforward- simply create a new, unique category and create rules to match.

The following table contains the threat types used for *sedb*:

```
Type      Description
-------   -----------------------------------------------------------
MONITOR - Messages that concern syslogd itself, or messages that
          infer that the logging system itself has been
          enabled/disabled
PHYSMOD - Messages that concern physical modification to the
          system such as PCMCIA card, cable or storage
          device insertion/removal
USERACT - Messages that concern unusual user activities such as
          privilege escalation, account modifications, or other
          unusual activities
NETWACT - Messages that concern network related activities such
          as connection attempts, IDS events, DoS events, known
          network attacks
COMPROM - Messages inferring a compromise may have occurred such
          as suspicious core dumps, and integrity check failures
PROCESS - Messages inferring that unexpected processes are running
          such as a known trojan, P2P binaries, etc.
SHUTRST - Messages that infer the system has been restarted
```

These types will be passed from **SEC** to the *sedb* database as described later in this section.

A *threat level* is an indication of the severity of the threat. This example will use the conventional *RED*, *ORANGE*, *YELLOW*, and *GREEN* color scheme to indicate decreasing levels of severity.

The overall system then-

1. Receives raw log entries from syslog
2. Analyzes raw log entries and generates an ``analyzed'' entry
3. Stores both raw and analyzed entries in *sedb*

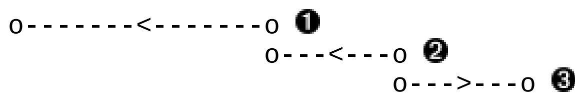
Collection is only one half of a real working system. Reporting is also important. We will provide some SQL queries that report on the level and severity of the data. Fancy reporting (including graphs, charts, etc.) is left as an exercise for the reader. Tobi Oetikers RRD Tool can be quite helpful in developing fancy charts and graphs.

5.3.1 The Correlation Map and Strategy

What exactly, do we want **SEC** to correlate, and how do we do that? This section provides one solution to these questions.

The following is a map of several items we want **SEC** to correlate. Two different threat types across the top are joined by a horizontal bar with a directional symbol indicating the direction of correlation. For example, if a *USERACT* event is followed by a *MONITOR* event, we will consider the events correlated. The direction of the arrow indicates time.

```
MONITOR PHYSMOD USERACT NETWACT COMPROM PROCESS SHUTRST
```



The defined correlation combinations are:

❶ *USERACT->MONITOR*

❷ *NETWACT->USERACT*

❸ *NETWACT->COMPROM*

Deriving these combinations is performed by asking how different combinations of events might be related. For instance the first example, *USERACT->MONITOR*, would indicate that a possible cause of **syslogd** being shutdown, or a network card being put into promiscuous mode, would be user activity (*USERACT*). Of course, there are other possible causes. Certain user activity, such as a failed login, may nave nothing to do with whether syslog was shutdown or a network card was put in to promiscuous mode. These examples show how valid combinations may be detected, but does not account for coincidences.

Just *how* **SEC** is to correlate events is a two-step process. With the First Level configuration file, we process raw syslog messages and assign them a type and level. We then have **SEC** use the *event* action to feedback the input into **SEC** to be read by the Second Level configuration file. It is the Second Level configuration that uses the PairWithWindow rule to pair up possible correlations.

(Note: This is not the only way to find correlations in the input. Another way this could be done is to create a context for each threat type and create rules that look for combinations of contexts as shown above in Section 2.1.)

In the configuration files shown below *%F* is a **SEC** variable that is set to the name of the output FIFO. The variable is set in the System configuration file shown in a later section.

5.3.2 The *sedb* First Level Configuration File

The rules immediately below are part of the First Level configuration file. The Second Level Rules configuration file is shown in the next section.

```
#
# First Level Configuration File - First.conf
#
#  Log Text Pass Thru Rule. This rule sends the text
#  to the database backend for storage in the logtext table.
# -------------------------------------------------------
#
# THIS RULE MUST BE THE FIRST RULE.  IT MUST HAVE A 'continue=TakeNext'
# STATEMENT.
#
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+(.*)
continue=TakeNext
desc=$0
action=write %F LOG|$1|$2


#
# The rules below assign a type and level to each log entry.
```

```
#
# MONITOR - SEC rules to pick up disruptive monitoring events.
#
#Sample BSD logs involving syslogd disabled or unusual promiscuous mode.
#----------------------------------------------------------------------
#Nov 15 20:02:48 foohost syslogd: exiting on signal 15
#Nov 22 02:00:02 foohost syslogd: restart
#Nov 11 15:58:55 foohost /kernel: de0: promiscuous mode enabled
#Nov 11 15:58:57 foohost /kernel: de0: promiscuous mode disabled
#

#
# Syslog Exit
# -----------
# RED event.  No context needed.
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+syslogd: exiting on signal (\d+)
desc=$0
action=event 0 $1 MONITOR:RED syslog exit on signal $2 at %t

#
# Syslog Restart
# ---------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+syslogd: restart
desc=$0
action=event 0 $1 MONITOR:YELLOW  syslog restart at %t


#
# Syslog Exit
# -----------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+/kernel: (\S+) promiscuous mode (\S+)
desc=$0
action=event 0 $1 MONITOR:RED  $2 promiscuous mode $3 at %t



# PHYSMOD - Events concerning physical modifications.
#
# Sample BSD logs involving physical modifications.
#-------------------------------------------------
#Nov 14 21:11:19 foohost /kernel: pccard: card inserted, slot 0
#Nov 14 22:28:09 foohost /kernel: pccard: card removed, slot 0
#Nov 12 19:46:31 foohost /kernel: de0: link down: cable problem?
#Nov 12 19:46:42 foohost /kernel: de0: autosense failed: cable problem?
#Oct 18 06:26:37 foohost pccardd[49]: ep0: 3Com Corporation (/3C589/) inserted.
#Oct 18 06:26:42 foohost pccardd[49]: pccardd started
#

#
# PCMCIA Card Insertion, Removal
# --------------------------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+/kernel: pccard: card (\S+), slot (\d+)
desc=$0
action=event 0 $1 PHYSMOD:ORANGE  pccard: card $2 in slot $3 at %t

#
# PCMCIA Card Daemon
# --------------------------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+pccardd\[\d+\]: (.*)
desc=$0
action=event 0 $1 PHYSMOD:YELLOW  pccardd: $2 at %t

#
# Cabling Problem
# ----------------
#
type=Single
ptype=RegExp
```

```
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+/kernel: (\S+)\s+(.*?:) cable problem
desc=$0
action=event 0 $1 PHYSMOD:ORANGE  cable problem on $2, text: $3 at %t


# USERACT - Events concerning user activities.
#
# Sample BSD logs involving logins, change of UID and privilege escalations.
#---------------------------------------------------------------------------
#Nov 14 12:14:58 foohost sshd[3388]: fatal: Timeout before authentication for
192.168.1.1
#Nov 14 19:58:34 foohost sshd[6597]: Bad protocol version identification
'^B^S^D^Q^L' from 192.168.1.100
#Oct 18 06:16:53 foohost sshd[131]: Accepted keyboard-interactive/pam for foouser
from 192.168.1.1 port 1077 ssh2
#Nov 15 04:02:24 foohost login: 1 LOGIN FAILURE ON ttyp2
#Nov 15 04:02:24 foohost login: 1 LOGIN FAILURE ON ttyp2, mysql
#Oct 18 03:20:46 foohost login: 2 LOGIN FAILURES ON ttyv0
#Oct 18 02:52:04 foohost login: ROOT LOGIN (root) ON ttyv1
#Oct 18 06:11:11 foohost login: login on ttyv0 as root
#Nov 10 19:40:03 foohost su: foouser to root on /dev/ttyp0
#Nov 18 09:37:38 foohost su: BAD SU foouser to root on /dev/ttyp3
#Nov 22 12:26:44 foohost su: BAD SU goodboy to root on /dev/ttyp0
#

#
# sshd Problems
# --------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+sshd\[\d+\]: (fatal|Bad)(.*)
desc=$0
action=event 0 $1 USERACT:YELLOW  sshd $2 problem, text: $3 at %t

#
# sshd Accepted
# --------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+sshd\[\d+\]: Accepted (.*)
desc=$0
action=event 0 $1 USERACT:GREEN  sshd accepted login, text: $2 at %t

#
# login FAILURES
# ---------------
# ORANGE
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+login: (.*?FAILURE.)(.*?ON) (.*)
desc=$0
action=event 0 $1 USERACT:YELLOW  login $2 on $4 at %t

#
# su  bad
# -----------------
# ORANGE
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+su: (BAD SU) (\S+) to (\S+) on (\S+)
desc=$0
action=event 0 $1 USERACT:YELLOW  su: $2 $3 to $4 on $5 at %t


#Nov 10 19:40:03 foohost su: foouser to root on /dev/ttyp0
#Nov 18 09:37:38 foohost su: BAD SU foouser to root on /dev/ttyp3
#Nov 22 12:26:44 foohost su: BAD SU badboy to root on /dev/ttyp0
#
#
# su  good to root
# -----------------
# ORANGE
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+su: (\S+) to root on (\S+)
desc=$0
action=event 0 $1 USERACT:YELLOW  su: $2 to ROOT on $4  at %t



# NETWACT - SEC rules to pick up suspicious network events.
```

```
#
# Sample BSD logs involving odd or suspicious network activity.
#---------------------------------------------------------------
#Jun  3 17:46:24 foohost named[38298]: client 10.12.127.176#3714: request has
invalid signature: tsig verify failure
#Apr 14 16:23:08 foohost /kernel: arp: 10.10.152.12 moved from 00:90:27:37:35:cf to
00:d0:59:aa:61:11 on de0
#Apr  1 11:23:39 jimby /kernel: Limiting closed port RST response from 368 to 200
packets per second


#
# named Dynamic DNS Update rejection
# ----------------------------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+named\[\d+\]: client (\S+): request has invalid
signature:(.*)
desc=$0
action=event 0 $1 NETWACT:YELLOW  dyndns attempt from $2, text: $3 at %t

#
# MAC address moved
# -----------------
# ORANGE
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+/kernel: arp: (\S+) moved from (\S+) to (\S+) on
(\S+)
desc=$0
action=event 0 $1 NETWACT:YELLOW  arp moved on $2, from: $3 to $4 on $5 at %t

#
# DoS RST rate limit
# ------------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+/kernel: Limiting closed port RST response from
(\d+) to (\d+)
desc=$0
action=event 0 $1 NETWACT:YELLOW  RST limit enforced: $2 to $3 at %t



# COMPROM - SEC rules to pick up potential system compromise events.
#
# Sample BSD logs involving potential system compromise.
#-------------------------------------------------------
#May 25 18:09:55 foohost ntpd[1325]: ntpd exiting on signal 11
#Jul 21 18:33:16 foohost /kernel: pid 55454 (ftpd), uid 1001: exited on signal 8
#Apr  9 12:57:06 foohost /kernel: pid 28039 (telnet), uid 0: exited on signal 3
(core dumped)

#
# ntpd crash
# ------------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+ntpd\[\d+\]: ntpd exiting on signal (\d+)
desc=$0
action=event 0 $1 COMPROM:RED  ntpd crashed on signal $2 at %t

#
# Process crash
# ------------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+/kernel: pid \d+ \(\S+\), uid (\d+): exited on
signal (\d+)
desc=$0
action=event 0 $1 COMPROM:RED  $2 crashed on signal $4, uid $3 at %t



# PROCESS - SEC rules to pick up suspicious process events.
#
# Sample BSD logs involving unusual processes.
#---------------------------------------------
#Mar 23 08:05:52 foohost thttpd[126]: thttpd/2.25b 29dec2003 starting on port 8090
```

```
#
# Suspicious processes
# --------------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+(thttpd)\[(\d+)\]:(.*)
desc=$0
action=event 0 $1 PROCESS:ORANGE  suspicious process  $2 pid $3, text: $4 at %t


# SHUTRST - SEC rules to pick up system shutdown, restart events.
#
# Sample BSD logs involving system shutdown and reset.
#---------------------------------------------------------
#Mar  6 16:28:13 foohost reboot: rebooted by foouser
#Jul 15 17:35:49 foohost halt: halted by root
#Mar  6 16:29:17 foohost /kernel: Copyright (c) 1992-2003 The FreeBSD Project.

#
# Reboot message
# --------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+reboot: rebooted by (\S+)
desc=$0
action=event 0 $1 SHUTRST:RED  rebooted by $2

#
# Halt message
# --------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+halt: halted by (\S+)
desc=$0
action=event 0 $1 SHUTRST:RED  halted by $2

#
# Restart message
# --------------
#
type=Single
ptype=RegExp
pattern=^\S+\s+\d+\s+\S+\s+(\S+)\s+/kernel: Copyright \(c\) (\S+) The FreeBSD
Project
desc=$0
action=event 0 $1 SHUTRST:RED  restart message at %t
```

Note how the *event* action contains the threat type and severity keywords encoded in the first token ('PHYSMOD:YELLOW', 'COMPROM:RED', etc). These tokens are decoded by the Second Level Configuration file, where the analysis takes place.

5.3.3 The *sedb* Second Level configuration File

In the Second Level configuration file, the objective is to decode the incoming events from the First Level configuration file and use the ``PairWithWindow'' rule type to match up events. Note the use of ``%number'' variables in the ``action2'' rule entry. These refer to the matched backreferences of ``pattern'' in the earlier match. In this case, the syslog text of both entries in being stored with a ``CORR:'' tag for easy retrieval.

```
#
# Second Level Configuration File - Second.conf
#
#
# Decode threat events coming in.
# Note continue=TakeNext where it applies.
#


# Anytime RED is passed it's an immediate keeper.
type=Single
```

```
ptype=RegExp
continue=TakeNext
pattern=^(\S+)\s+(\S+):RED\s+(.*)
desc=$0
action=write %F ANL|$1|$2|RED|$3


# OFF_HOURS ORANGE is promoted to a RED.
type=Single
ptype=RegExp
continue=TakeNext
pattern=^(\S+)\s+(\S+):ORANGE\s+(.*)
context=OFF_HOURS
desc=$0
action=write %F ANL|$1|$2|RED|PROM:$3



# Any other time  ORANGE is passed
type=Single
ptype=RegExp
pattern=^(\S+)\s+(\S+):ORANGE\s+(.*)
desc=$0
action=write %F ANL|$1|$2|ORANGE|$3



#
# Decode threat events coming in from First Level Configuration.
# See note for each type.  Note continue=TakeNext where it applies.
#

#Correlation Map entries.  Use PairWithWindow

# USERACT AND MONITOR
type=PairWithWindow
ptype=RegExp
pattern=^(\S+)\s+(USERACT):(\S+)\s+(.*)
desc=$0
continue=TakeNext
action=write - MONITOR event didn't follow USERACT within window
ptype2=RegExp
pattern2=^(\S+)\s+(MONITOR):(\S+)\s+(.*)
desc2=$0
action2=write %F ANL|$1|$2|$3|CORR:USERACT+MONITOR:(%4)+($4)
window=300



# NETWACT AND USERACT
type=PairWithWindow
ptype=RegExp
pattern=^(\S+)\s+(NETWACT):(\S+)\s+(.*)
desc=$0
continue=TakeNext
action=write - USERACT event didn't follow NETWACT within window
ptype2=RegExp
pattern2=^(\S+)\s+(USERACT):(\S+)\s+(.*)
desc2=$0
action2=write %F ANL|$1|$2|$3|CORR:NETWACT+USERACT:(%4)+($4) ; reset -1
window=300



# NETWACT AND COMPROM  (NETWACT first)
type=PairWithWindow
ptype=RegExp
pattern=^(\S+)\s+(NETWACT):(\S+)\s+(.*)
desc=$0
continue=TakeNext
action=write -  COMPROM event didn't follow NETWACT within window
ptype2=RegExp
pattern2=^(\S+)\s+(COMPROM):(\S+)\s+(.*)
desc2=$0
action2=write %F ANL|$1|$2|$3|CORR:NETWACT+COMPROM:(%4)+($4) ; reset -1
window=300
```

5.3.4 The *sedb* System Configuration File

The System Configuration file contains set of calendar rules that provides a context for ``BUSINESS_HOURS" and ``OFF_HOURS". In the case of an ORANGE level event occuring during OFF_HOURS, that event is promoted to a RED event. By using contexts and pairings together it is possible to create complicated correlations beyond the examples in this chapter.

The System Configuration file also contains a start-up rule. This rule sets various variables (such as `` %F" that will be used later. To enable these variables, **SEC** must be started with the -*intevents,* parameter. See the **SEC** man page for more information.

```
#
# System Configuration File - System.conf
# Configuration file for SEC internal events.
#
#
# This file contains a rule defining the SEC internal events processed
# by SEC on startup and signals.  It also contains the calendar
# rules used by the system.
#
# This file uses the internal events  SEC_STARTUP and SEC_RESTART
#
#
# %s Event description string (reserved by SEC)
# %t Textual timestamp (reserved by SEC)
# %d Numeric timestamp (reserved by SEC)
#
# Additional variables used by the system.
# -----------------------------------------
# %S Scripts directory     $ENV{'SECSCRIPTSDIR'}
# %T Temporary directory   $ENV{'SECTMPDIR'}
# %F fifo for output       $ENV{'SECFIFO'}
#
# NOTE: To enable these variables, SEC must be started
# with the -intevents  parameter.
# See the SEC man page for more information.
#


#
# Startup rule.
#
type=Single
ptype=RegExp
pattern=(SEC_STARTUP|SEC_RESTART)
desc=$0
context=SEC_INTERNAL_EVENT
action=eval %S ( $ENV {'SECSCRIPTSDIR'} );\
       eval %T ( $ENV {'SECTMPDIR'} );\
       eval %F ( $ENV {'SECFIFO'} );


# Contexts set by calendar rules.
#
#  BUSINESS_HOURS  - a context for business hours (6:00am to 7:59pm, M-F)
#  OFF_HOURS       - a context for all other hours.
#
# Business Hours- applies to all hosts. 6:00am - 7:59pm, M-F
# Off hours -  8:00pm - 5:59am M-F and all day Sat and Sun . Set by Calendar Rules.
#
# Three calendars are needed since calendar is run every minute
# and we don't actually know when SEC will be started (or restarted)
#
#
type=Calendar
time=* 6-20 * * 1,2,3,4,5
desc=BUSINESS_HOURS
context=!BUSINESS_HOURS
action=create %s;\
       write - Switched to Business Hours;\
       delete OFF_HOURS;


type=Calendar
time=* 0-5,21-23 * * *
context=!OFF_HOURS
desc=OFF_HOURS
action=create %s;\
```

```
        write - Switched to Off Hours;\
        delete BUSINESS_HOURS;


type=Calendar
time=* * * * 6,7
context=!OFF_HOURS
desc=OFF_HOURS
action=create %s;\
        write - Switched to Off Hours;\
        delete BUSINESS_HOURS;


#
# This calendar rule is a health check. Writes to stdout every
# five minutes.  This could be used for visual confirmation
# that SEC is still running.
#
type=Calendar
time=0,5,10,15,20,25,30,35,40,45,50,55 * * * *
desc=SEC check
action=write - %s at %t
```

5.3.5 The *sedb* Schema

The schema for *sedb* is quite simple- just two tables. The *logtext* table keeps the raw log text, along with an MD5 checksum of the entry. The *loganalysis* table keeps track of threat types (COMPROM,PHYSMOD, MONITOR, etc.) and threat levels (RED, ORANGE, etc.).

```
#
#
# Simple Event Data Base (sedb) schema.
#
# sedb contains two tables:
#
#  logtext     - used to store raw log text
#  loganalysis - used to store analysis of log text
#

DROP TABLE logtext;

CREATE TABLE logtext (lt_record_id        BIGINT NOT NULL AUTO_INCREMENT UNIQUE,
                      lt_dev_fqdn         VARCHAR(128),     # join with
la_dev_fqdn
                      lt_date_inserted    TIMESTAMP,
                      lt_log_text         TEXT,
                      lt_text_md5hash     VARCHAR(32),
                      FULLTEXT            (lt_log_text),
                     );
#
#-----------------------------------------------------------------------------------
-------
#
DROP TABLE loganalysis;

CREATE TABLE loganalysis (la_record_id        BIGINT NOT NULL AUTO_INCREMENT
UNIQUE,
                          la_dev_fqdn         VARCHAR(128),  # Device FQDN
                          la_date_inserted    TIMESTAMP,
                          la_threat_type      VARCHAR(20),   # PHYSMOD, MONITOR,
etc.
                          la_threat_text      TEXT,          # The analysis message
                          la_threat_level     VARCHAR(8),    # RED, ORANGE, YELLOW,
etc.
                          la_md5_hash         VARCHAR(32),
                          FULLTEXT           (la_threat_text)
                          );
#
# Index for threat type.
#
ALTER TABLE loganalysis ADD INDEX la_threat_type_idx (la_threat_type);

#
# Index for threat level.
#
ALTER TABLE loganalysis ADD INDEX la_threat_level_idx (la_threat_level);
```

Once the **MySQL** software is started up, log into the **MySQL** monitor program **mysql** and load the following data base schema:

```
$mysql -u root -p sedb
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.0.18

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show tables;
Empty set (0.00 sec)

mysql>
mysql>
mysql> \. sedb.sql
ERROR 1051: Unknown table 'logtext'      (Note: this error is OK. Dropped table
does not exist.)
Query OK, 0 rows affected (0.00 sec)

ERROR 1051: Unknown table 'loganalysis'  (Note: this error is OK. Dropped table
does not exist.)
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show tables;
+----------------+
| Tables_in_sedb |
+----------------+
| loganalysis    |
| logtext        |
+----------------+
2 rows in set (0.00 sec)

mysql> describe loganalysis;
+-------------------+---------------+------+-----+---------+----------------+
| Field             | Type          | Null | Key | Default | Extra          |
+-------------------+---------------+------+-----+---------+----------------+
| la_record_id      | bigint(20)    |      | PRI | NULL    | auto_increment |
| la_dev_fqdn       | varchar(128)  | YES  |     | NULL    |                |
| la_date_inserted  | timestamp(14) | YES  |     | NULL    |                |
| la_threat_type    | varchar(20)   | YES  | MUL | NULL    |                |
| la_threat_text    | text          | YES  | MUL | NULL    |                |
| la_threat_level   | varchar(8)    | YES  | MUL | NULL    |                |
| la_md5_hash       | varchar(32)   | YES  |     | NULL    |                |
+-------------------+---------------+------+-----+---------+----------------+
7 rows in set (0.38 sec)

mysql> describe logtext;
+-------------------+---------------+------+-----+---------+----------------+
| Field             | Type          | Null | Key | Default | Extra          |
+-------------------+---------------+------+-----+---------+----------------+
| lt_record_id      | bigint(20)    |      | PRI | NULL    | auto_increment |
| lt_dev_fqdn       | varchar(128)  | YES  |     | NULL    |                |
| lt_date_inserted  | timestamp(14) | YES  |     | NULL    |                |
| lt_log_text       | text          | YES  | MUL | NULL    |                |
| lt_text_md5hash   | varchar(32)   | YES  |     | NULL    |                |
+-------------------+---------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql> exit
Bye
```

5.3.6 IPC Data Format

We will use the same techniques discussed above for passing data from **SEC** to the data base insert script.

Along with the use of a *FIFO* there must be an agreed data format for communication. To keep things as simple as possible, we will use a pipe delimited format as follows:

```
Log Text Entry    : LOG|<hostname>|<log_text_entry_from_syslog>
Log Analysis Entry: ANL|<hostname>|<type>|<level>|<analyzed_entry_from_SEC>
```

This format is already shown in the Second Level Configuration file. The format is simple to create, and simple to parse. The only drawback, is that there is an interdependency on the format for both SEC, which creates the format, and the database insert script which reads and acts on the format. Any modifications to either side must consider the effect on the other side. There are, of course, other ways to do this, but this is a simple, easy to implement method.

The database insert code in the next section is similar to the pipe output example given above.

5.3.7 The *dbinsert.pl* Script

Create a directory for all the scripts needed for interacting with *sedb*. The following perl module, *dbinsert.pl* will be used to load data into the datbase:

```
#!/usr/bin/perl
################################################################################
#
# NAME:
#     dbinsert.pl  -  Enter log records in database
#
# SYNOPSIS:
#     perl dbinsert.pl
#
# DESCRIPTION:
#     This script reads records from a FIFO that is written to by SEC.  SEC
#     puts log text lines, and log analysis lines, in an agreed format, into
#     the FIFO.  This script reads those records, interprets each line and
#     enters the record into the database.
#
#     Input lines format is defined in the SEC configuration rule files.
#     SEC uses the 'write' statement to write to the FIFO defined as '%F',
#     taken from $ENV{'SECFIFO'} in the environment.  This fifo is setup at
#     runtime during the SEC startup event rules.  See the sedb System
#     configuration file for more information.
#
# RETURN CODES:
#     None- This script should operate as a background daemon process and
#     should never exit.
#
# SEE ALSO:
#     sedb configuration file
#
################################################################################

$| = 1;  # Set for unbuffered operation.


#
# Create our pid file
# --------------------

$piddir = $ENV{'SECRUNDIR'};

$pidfile = "${piddir}/dbinsert.pid";

open PIDFILE, "> $pidfile" or die "Can't open pidfile [$pidfile] : $!";
print PIDFILE "$$\n";
close PIDFILE;



use strict;
use Socket;
use DBI();
use Digest::MD5 qw(md5_hex);


my  $inputline;
my  $in_control_word;
my  $in_hostname;
my  $fifo_name;
```

```
my  $in_type;
my  $in_level;
my  $in_analysis;
my  $in_remainder;

print "Start of dbinsert.pl\n";

# Connect to the database.
my $dbh = DBI->connect("DBI:mysql:database=sedb;host=localhost",
            "root", "dummy",
            {'RaiseError' => 1});

#
# Always use lower case for all column names.
#
$dbh->{FetchHashKeyName} = 'NAME_lc';

#
# Create 'prepare' statements.
#
# Statement to insert a logalysis record.
# ----------------------------------------
my $la_insert_rec = $dbh->prepare("INSERT INTO loganalysis VALUES
( ?,?,?,?,?,?,? )")
    or die "Can't prepare la_insert_rec  statement: $DBI::errstr";

#
# Statement to insert a logtext record.
# -------------------------------------
my $lt_insert_rec = $dbh->prepare("INSERT INTO logtext VALUES ( ?,?,?,?,? )")
    or die "Can't prepare lt_insert_rec  statement: $DBI::errstr";

#
# The FIFO is defined through the environment in the sedb.sh  setup script.
# --------------------------------------------------------------------------
$fifo_name = $ENV{'SECFIFO'};

# We pretend to be a writer to the FIFO to keep it open across open/close by SEC
open(FIFO, "+< $fifo_name") or die "*** FATAL fifo [$fifo_name] problem!  $!";

while (<FIFO>)
{
    $inputline = $_;

    chop $inputline;
#   print "Got: [$inputline]\n";
   print ".";

    #
    # Format is:
    #     ANL|hostname|type|level|analyzed_text
    # or
    #     LOG|hostname|log_text
    #-------------------------------------------

    ($in_control_word,$in_hostname,$in_remainder) = split /\|/, $inputline, 3;

#
# Add a log text record
# ---------------------

    if ($in_control_word =~ /^LOG/)
    {
      &addlogtextrec($in_hostname,$in_remainder);
      next;
    }

#
# Add a loganalysis record
# ------------------------
    if ($in_control_word =~ /^ANL/)
    {
       ($in_type, $in_level, $in_analysis) = split /\|/, $in_remainder, 3;

       if ($in_level =~ /(^RED)|(^ORANGE)|(^YELLOW)/) # GREEN is not logged
       {
         &addloganalrec($in_hostname,$in_type,$in_level,$in_analysis);
       next;
       }
```

```
        else
        {
          print "***Error: unknown level: [$inputline]\n";
          next;
        }
    }

    if( $in_control_word !~ /^LOG/ &&
        $in_control_word !~ /^ANL/)
    {
      print "***Error: unknown control_word: [$inputline]\n";
      next;
    }

}  # end while(<FIFO>)

# Never reached...
exit(1);



#
#
#  addloganalrec()
# ---------------------------------------------------------------------------------
#


sub  addloganalrec()
{
  my $ins_hostname = shift @_;
  my $ins_type     = shift @_;
  my $ins_level    = shift @_;
  my $ins_rec      = shift @_;
  my $la_md5;

#   if ($ins_hostname ne "" )
#   {
#      print "addloganalrecord:[$ins_hostname] [$ins_type] [$ins_level] [$ins_rec]\
n";
#   }


  $la_md5 = md5_hex($ins_rec);

  $la_insert_rec->execute(undef,       # needed by DBI
                        $ins_hostname, # device fqdn
                        undef,         # date inserted
                        $ins_type,     # threat type  LoggingDisabled, etc.
                        $ins_rec,      # The Message  "bad things happened"
                        $ins_level,    # threat level ORANGE, RED, etc.
                        $la_md5)       # msg hash
      or die "Can't execute la_insert_rec statement: $DBI::errstr";

}


#
#
#  addlogtextrec()
# ---------------------------------------------------------------------------------
#


sub addlogtextrec()
{
  my $ins_hostname = shift @_;
  my $ins_logtext  = shift @_;
  my $lt_md5;


#   if ($ins_hostname ne "" )
#   {
#      print "addlogtextrec:[$ins_hostname] [$ins_logtext]\n";
#   }

  $lt_md5 = md5_hex($ins_logtext);

  $lt_insert_rec->execute(undef,
                          $ins_hostname, # Hostname
                          undef,         # Current date and time.
                          $ins_logtext,  # The Message
```

```
                          $lt_md5)       # msg hash
        or die "Can't execute lt_insert_rec statement: $DBI::errstr";

}
```

5.3.8 Startup Scripts

An script that provides the proper environment:

```
################################################################################
#
# NAME:
#     sedbvars.sh  - script to pull in needed environment variables.
#
#
# SYNOPSIS:
#     . ./sedbvars.sh
#
# DESCRIPTION:
#
#  This script contains environment variables used in the sedb logging and
#  monitoring system.  The environment variables can be pulled into the
#  current shell using the '. ./sedbvars.sh' command from the current directory.
#
#  Directories
#     TLDIR              Base directory for normal use.
#     SECCONFDIR         Directory for SEC configuration files.
#     SECSCRIPTSDIR      Directory for Logging and Monitoring system scripts.
#     SECRUNDIR          Directory for local process use (pid files).
#     SECTMPDIR          Temorary directory for SEC dump and log files.
#     SECWORKDIR         Directory for miscellaneous data files.
#
################################################################################

#
# Get argument if it exists.
#

ARG=$1


# Directories.

# Top level directory.
# SET THIS FOR YOUR SYSTEM BEFORE USING!

TLDIR=/home/foouser/SEC-examples/sedb
export TLDIR

SECCONFDIR=$TLDIR/conf
SECRUNDIR=$TLDIR/run
SECSCRIPTSDIR=$TLDIR/scripts
SECTMPDIR=$TLDIR/tmp
SECWORKDIR=$TLDIR/work
export SECCONFDIR SECSCRIPTSDIR SECRUNDIR SECTMPDIR SECWORKDIR


#
# Other important files- the input logfile and output fifo
#

#BSD SECINPUTDIR=/var/log
#Solaris SECINPUTDIR=/var/adm
#Testing SECINPUTDIR=./test

SECINPUTDIR=$TLDIR/work
SECINPUTFILE=messages
export SECINPUTDIR SECINPUTFILE


SECFIFO=$SECWORKDIR/SEC_fifo
export SECFIFO


#
# Common OS utils.
#

#BSD /bin/foo
```

```
#Solaris /usr/bin/foo

CAT=/bin/cat
COPY=/bin/cp
CHMOD=/bin/chmod
PERL=/usr/bin/perl

PS=/bin/ps
PSARGS=-ax


export CAT COPY CHMOD PERL PS PSARGS


#
# Take a look with showvars

showvars() {
echo
echo TLDIR=$TLDIR
echo SECCONFDIR=$SECCONFDIR
echo SECRUNDIR=$SECRUNDIR
echo SECSCRIPTSDIR=$SECSCRIPTSDIR
echo SECTMPDIR=$SECTMPDIR
echo SECWORKDIR=$SECWORKDIR
echo
echo
echo SECINPUTDIR=$SECINPUTDIR
echo SECINPUTFILE=$SECINPUTFILE
echo
echo SECFIFO=$SECFIFO
echo
echo CAT=$CAT
echo COPY=$COPY
echo CHMOD=$CHMOD
echo PERL=$PERL
echo
echo PS=$PS
echo PSARGS=$PSARGS
echo
echo "End of sedbvars.sh"
echo
}



[ "$ARG" = "show" ] && showvars
```

And another one to start it all up and provide some additional useful commands...

```
#!/bin/sh
####################################################################################
######
#
# NAME:
#      sedb.sh - script to start, stop and control SEC in the sedb logging and
#      monitoring system.
#
#
# SYNOPSIS:
#      sh sedb.sh   start | stop | dump | hup | abrt
#
# DESCRIPTION:
#
#      'start' starts up the system.
#      'stop'  shuts down the system.
#      'dump'  causes SEC to dump its internal tables to tmp/sec.dump
#      'hup'   causes SEC to reread its configuration files and
#              reinitialize everything
#      'abrt'  causes SEC to reread its configuration files without reinitilizing
#              contexts and variables.
#
####################################################################################
######

#set -x


# Pull in the environment from ./sedbvars.sh

if [ ! -f ./sedbvars.sh ]
then
```

```
   echo "*** Error- ./sedbvars.sh environment script not found.  Check
configuration."
   exit
fi


case "$1" in
'start')

        echo
        echo "Start of SEDB Logging and Monitoring System.  Please wait..."
        #
        # Get our environment and list it out
        #
#        . ./sedbvars.sh  list

        # Start dbinsert first to prevent annoying error messages about broken pipe
        if [ -f $SECSCRIPTSDIR/dbinsert.pl ]
        then
                #
                # Start the dbinsert script.

                echo "Starting dbinsert.pl..."

                $PERL $SECSCRIPTSDIR/dbinsert.pl   > /dev/null 2>&1  &

                sleep 2

                RC=`$PS $PSARGS | grep dbinsert | grep -v grep`
                if [ $? != 0 ]
                then
                   echo "****  ERROR:  dbinsert did not start."
                   echo "      To fix: start dbinsert by hand."
                fi
        fi


        if [ -f $SECSCRIPTSDIR/sec.pl ]
        then
                #
                if [ !  -f $SECCONFDIR/System.conf \
                     -a -f $SECCONFDIR/First.conf \
                     -a -f $SECCONFDIR/Second.conf  ]; then
                        echo "ERROR:  sec.pl configuration files missing!! Check
installation."

                        exit 1;
                fi
                #
                #
                $PERL  $SECSCRIPTSDIR/sec.pl  \
                                -conf=$SECCONFDIR/System.conf \
                                -conf=$SECCONFDIR/First.conf \
                                -conf=$SECCONFDIR/Second.conf \
                                -testonly

# Config checking is available in sec 2.1.9 and later ONLY.

               if [ $? -ne 0 ];
               then
                       echo "ERROR:  sec.pl configuration files faulty!! Check
configuration."

                       exit 1;
               fi
               #


                echo 'SEDB Logging and  Monitoring Service Starting....'

                $COPY /dev/null  $SECTMPDIR/sec.log

                $PERL    $SECSCRIPTSDIR/sec.pl  \
                                 -conf=$SECCONFDIR/System.conf \
                                 -conf=$SECCONFDIR/First.conf \
                                 -conf=$SECCONFDIR/Second.conf \
                                 -debug=4 \
                                 -dump=$SECTMPDIR/sec.dump \
                                 -log=$SECTMPDIR/sec.log \
                                 -pid=$SECRUNDIR/sec.pid \
                                 -intevents \
                                 -input=$SECINPUTDIR/$SECINPUTFILE \
                                 -detach

#
#Use one of the lines below for interactive input (for testing)
```

```
#                                  -input=$TPDIR/t.t \
#                                  -input=./t.t \
                sleep 1
                echo ""
                echo "SEC Monitoring System Started."
                echo ""

        else
                echo "ERROR:  $SECSCRIPTSDIR/sec.pl missing!! Check installation."
                exit 1;
        fi

        echo "Done."

        ;;

'stop')
        #
        # Get our environment and don't list it out
        #
#        . ./sedbvars.sh

        for prog in sec dbinsert
        do

            if [ -f $SECRUNDIR/${prog}.pid ]
            then
                    pid=`$CAT $SECRUNDIR/${prog}.pid`
                    [ "$pid" -gt 0 ] && kill -TERM $pid  && echo "Killed $prog pid
$pid"
                    rm -f $SECRUNDIR/${prog}.pid
            else
                    # Was pid file removed?  Check here and kill prog if running...
                    pid=`$PS $PSARGS | grep $prog | grep -v grep | awk '{print
$1}'`
                    [ $pid ] && kill -TERM $pid
                    echo
                    echo "$prog not running"
                    echo
            fi

        done

        ;;
'dump')
        #
        # Get our environment and don't list it out
        #
#        . ./sedbvars.sh

        if [ -f $SECRUNDIR/sec.pid ]; then
                secpid=`$CAT $SECRUNDIR/sec.pid`
                [ "$secpid" -gt 0 ] && kill -USR1 $secpid
        fi
        ;;
'hup')
        #
        # Get our environment and don't list it out
        #
        . ./sedbvars.sh

        if [ -f $SECRUNDIR/sec.pid ]; then
                secpid=`$CAT $SECRUNDIR/sec.pid`
                [ "$secpid" -gt 0 ] && kill -HUP $secpid
        fi
        ;;
'abrt')
        #
        # Get our environment and don't list it out
        #
#        . ./sedbvars.sh

        if [ -f $SECRUNDIR/sec.pid ]; then
                secpid=`$CAT $SECRUNDIR/sec.pid`
                [ "$secpid" -gt 0 ] && kill -ABRT $secpid
        fi
```

```
        ;;
'check')
        #
        # Get our environment and list it out
        #
#        . ./sedbvars.sh list

        for prog in sec dbinsert
        do

            if [ -f $SECRUNDIR/${prog}.pid ]; then
                    pid=`$CAT $SECRUNDIR/${prog}.pid`
                    [ "$pid" -gt 0 ] && $PS $PSARGS | grep $pid
            fi

        done

         ;;
*)
        echo "Usage: $0 { start | stop | dump | hup | abrt }"
        exit 1
        ;;
esac
```

5.4 Putting It All Together

Below is an example run of the *sedb* database. Comments are shown in parentheses:

```
$
$. ./sedbvars.sh
$
$showvars

TLDIR=/home/foouser/SEC-examples/sedb
SECCONFDIR=/home/foouser/SEC-examples/sedb/conf
SECRUNDIR=/home/foouser/SEC-examples/sedb/run
SECSCRIPTSDIR=/home/foouser/SEC-examples/sedb/scripts
SECTMPDIR=/home/foouser/SEC-examples/sedb/tmp
SECWORKDIR=/home/foouser/SEC-examples/sedb/work


SECINPUTDIR=/home/foouser/SEC-examples/sedb/work
SECINPUTFILE=messages

SECFIFO=/home/foouser/SEC-examples/sedb/work/SEC_Fifo

CAT=/bin/cat
COPY=/bin/cp
CHMOD=/bin/chmod
PERL=/usr/bin/perl

PS=/bin/ps
PSARGS=-ax

End of sedbvars.sh

$
$


(Stop the sedb system first...)


$sh sedb.sh stop

sec not running

dbinsert not running

$
$


(Start the sedb system...)


$sh sedb.sh start
```

```
Start of SEDB Logging and Monitoring System.  Please wait...
Starting dbinsert.pl...
Simple Event Correlator version 2.1.11
Reading configuration from /home/foouser/SEC-examples/sedb/conf/System.conf
5 rules loaded from /home/foouser/SEC-examples/sedb/conf/System.conf
Reading configuration from /home/foouser/SEC-examples/sedb/conf/First.conf
21 rules loaded from /home/foouser/SEC-examples/sedb/conf/First.conf
Reading configuration from /home/foouser/SEC-examples/sedb/conf/Second.conf
10 rules loaded from /home/foouser/SEC-examples/sedb/conf/Second.conf
SEDB Logging and  Monitoring Service Starting....
Simple Event Correlator version 2.1.11
Changing working directory to /
Reading configuration from /home/foouser/SEC-examples/sedb/conf/System.conf
Reading configuration from /home/foouser/SEC-examples/sedb/conf/First.conf
Reading configuration from /home/foouser/SEC-examples/sedb/conf/Second.conf

SEC Monitoring System Started.

Done.
$

(Now check again...)

$sh sedb.sh check
  869  ??  Ss      0:00.15 /usr/bin/perl
/home/foouser/SEC-examples/sedb/scripts/sec.pl
-conf=/home/foouser/SEC-examples/sedb/conf/System.conf -conf=/home/foouser/SEC-
example
  860  p0  I       0:00.05 /usr/bin/perl
/home/foouser/SEC-examples/sedb/scripts/dbinsert.pl
$
```

Once the system is running, it will process input from whereever $SECINPUTDIR/$SECINPUTFILE is set. In this case it's `/home/foouser/SEC-examples/sedb/work/messages`. To process syslog messages, set $SECINPUTDIR to your systems log directory and $SECINPUTFILE to the log messagesfile in sedbvars.sh and restart the system.

Below are some SQL queries that show the results of one example run. Comments again in parentheses:

```
  (Initialize data base...)
mysql> \. sedb.sql
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
mysql>
mysql> show tables;
+----------------+
| Tables_in_sedb |
+----------------+
| loganalysis    |
| logtext        |
+----------------+
2 rows in set (0.01 sec)

mysql>

mysql> describe logtext;
+------------------+----------------+------+-----+---------+----------------+
| Field            | Type           | Null | Key | Default | Extra          |
+------------------+----------------+------+-----+---------+----------------+
| lt_record_id     | bigint(20)     |      | PRI | NULL    | auto_increment |
| lt_dev_fqdn      | varchar(128)   | YES  |     | NULL    |                |
| lt_date_inserted | timestamp(14)  | YES  |     | NULL    |                |
| lt_log_text      | text           | YES  | MUL | NULL    |                |
| lt_text_md5hash  | varchar(32)    | YES  |     | NULL    |                |
```

```
+------------------+---------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql>
mysql> describe loganalysis;
+------------------+---------------+------+-----+---------+----------------+
| Field            | Type          | Null | Key | Default | Extra          |
+------------------+---------------+------+-----+---------+----------------+
| la_record_id     | bigint(20)    |      | PRI | NULL    | auto_increment |
| la_dev_fqdn      | varchar(128)  | YES  |     | NULL    |                |
| la_date_inserted | timestamp(14) | YES  |     | NULL    |                |
| la_threat_type   | varchar(20)   | YES  | MUL | NULL    |                |
| la_threat_text   | text          | YES  | MUL | NULL    |                |
| la_threat_level  | varchar(8)    | YES  | MUL | NULL    |                |
| la_md5_hash      | varchar(32)   | YES  |     | NULL    |                |
+------------------+---------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)

mysql>
```

(Start **sedb** as shown above and put some data into the **SEC** input file...)

```
echo "Nov 14 12:14:58 foohost sshd[3388]: fatal: Timeout before authentication for
192.168.1.1" >> messages
echo "Nov 22 02:00:02 foohost syslogd: restart" >> messages

echo "Jun  3 17:46:24 foohost named[38298]: client 10.12.127.176#3714: request has
invalid signature: tsig verify failure" >> messages
echo "Nov 14 12:14:58 foohost sshd[3388]: fatal: Timeout before authentication for
192.168.1.1" >> messages


echo "Jun  3 17:46:24 foohost named[38298]: client 10.12.127.176#3714: request has
invalid signature: tsig verify failure" >> messages
echo "May 25 18:09:55 foohost ntpd[1325]: ntpd exiting on signal 11" >> messages
```

(Get record counts from tables...)

```
mysql>
mysql> select count(*) from logtext;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from loganalysis;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql>
mysql>
```

(Show data with SQL query...)

```
mysql> select la_record_id,
la_dev_fqdn,la_date_inserted,la_threat_type,la_threat_text,la_threat_level
    -> from loganalysis ;
+--------------+-------------+-------------------+----------------
+---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
-------------------------------------------------------------+-----------------+
| la_record_id | la_dev_fqdn | la_date_inserted | la_threat_type | la_threat_text
| la_threat_level |
+--------------+-------------+-------------------+----------------
+---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
-------------------------------------------------------------+-----------------+
|            1 | foohost     |   20040725222340 | MONITOR        |
CORR:USERACT+MONITOR:(sshd fatal problem, text: : Timeout before authentication for
192.168.1.1 at Sun Jul 25 22:23:40 2004)+(syslog restart at Sun Jul 25 22:23:40
2004)                                                          | YELLOW          |
|            2 | foohost     |   20040725222357 | USERACT        |
CORR:NETWACT+USERACT:(dyndns attempt from 10.12.127.176#3714, text:  tsig verify
```

```
failure at Sun Jul 25 22:23:56 2004)+(sshd fatal problem, text: : Timeout before
authentication for 192.168.1.1 at Sun Jul 25 22:23:56 2004) | YELLOW          |
|            3 | foohost     |   20040725222410 | COMPROM        | ntpd crashed on
signal 11 at Sun Jul 25 22:24:10 2004
| RED         |
|            4 | foohost     |   20040725222410 | COMPROM        |
CORR:NETWACT+COMPROM:(dyndns attempt from 10.12.127.176#3714, text:  tsig verify
failure at Sun Jul 25 22:24:10 2004)+(ntpd crashed on signal 11 at Sun Jul 25
22:24:10 2004)                                                 | RED             |
|            5 | foohost     |   20040725222410 | COMPROM        |
CORR:NETWACT+COMPROM:(dyndns attempt from 10.12.127.176#3714, text:  tsig verify
failure at Sun Jul 25 22:23:56 2004)+(ntpd crashed on signal 11 at Sun Jul 25
22:24:10 2004)                                                 | RED             |
+--------------+-------------+------------------+----------------
+----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
--------------------------------------------------------------+-----------------+
5 rows in set (0.00 sec)

mysql>
mysql>
```

5.5 SEDB Caveats

As shown, *sedb* isn't all that useful. There are only a handful of syslog messages that are matched and only three rules that provide correlation. There are also some significant limitations:

- It doesn't properly correlate entries from multiple sources. A USERACT event from one system should not correlate with a COMPROM event from a different system.
- There is no way to ignore events from a system under test or maintenance. Ideally, we want a way to take a device temporarily 'out' of the logging and monitoring system, even though we still get syslog entries from the device.
- One backend database can quickly get overloaded in a busy environment. There should be a way to dynamically point the data insertion script (dbinsert.pl) to another database.
- There is no way to tell if the system is down, or data is not being received. A heartbeat or watchdog function is needed to ensure that someone is notified if the database is down, or the **SEC** process or data insert process dies.

Nevertheless, a system simlar to this (and addressing the above issues) has been developed and deployed at a major commercial institution. SEC has proven capable of handling syslog correlation tasks similar to commercial products costing thousands of dollars.

---

6 Other Uses For SEC

SEC can be used in other applications such as portknocking, and process control.

6.1 Portknocking

Portknocking combines event correlation and system security by recognizing a certain sequence of events- in this case attempted connections to closed ports. The idea is to have a system totally closed (no ports open) and have the system recognize a certain sequence of attempted connections (knocks) on closed ports- say port 25, port 9966, port 35, port 7000 all from the same source host and all within a

certain time. In effect, the sequence becomes a key. Once the key is recognized the system opens an incoming port for connection, such as sshd for limited period of time. The source host, and only the source host is permitted to connect.

A couple of notes before jumping right in: you must run these examples as root, since you will be attempting to start sshd, which binds to a priviliged port. Also, make sure sshd is not running before starting this example. Finally, remember that true computer security is composed of many layers. You should not depend exclusively on portknocking for security.

Portknocking is described in several articles at www.portknocking.org.

6.1.1 A Simple Portknocking Ruleset

To start, consider the following syslog messages from a BSD based system:

```
Jun 27 15:02:51 foohost /kernel: Connection attempt to TCP 127.0.0.1:7777 from
127.0.0.1:1806 flags:0x02
Jun 27 15:02:58 foohost /kernel: Connection attempt to TCP 127.0.0.1:5555 from
127.0.0.1:4356 flags:0x02
```

These messages are sent to syslog by the kernel in response to an attempted connection on a closed port. To see these message, two sysctl variables net.inet.tcp.blackhole and net.inet.udp.blackhole have to be set. The required commands are:

```
$ sysctl  net.inet.tcp.blackhole=1
```

and

```
$ sysctl  net.inet.udp.blackhole=1
```

A ruleset that recognizes these messages is:

```
#
#  Portknocking example 1
#
#BSD Jun 27 15:02:58 foohost /kernel: Connection attempt to TCP 127.0.0.1:5555 from
127.0.0.1:4356 flags:0x02
#
type=Single
ptype=RegExp
pattern=\S+\s+\d+\s+\S+\s+(\S+)\s+/kernel: Connection attempt to (\S+) (\S+):(\S+)
from (\S+):(\S+).*
desc=$0
action=write - Found a knock on $2 $3 port $4 from $5
```

The goal in this example is to start *sshd* with a 30 second window to accept a connection, and die if no connection has been received and processed. If a connection is accepted, sshd is to process it and then die.

Several sshd options are useful in this case. The *-d* option causes sshd to run in the foreground, process one connection and then die. The *-g seconds* option causes sshd to wait *seconds* from the start of the connection until the authentication is completed. If there is no authentication during this period, sshd dies.

The following script accomplishes the above requirements:

```
#!/bin/sh
#
#  SEC Portknock example startup script for sshd
#
# Start sshd in single shot mode with a 15 second
```

```
# authentication window.

#set -x
/usr/sbin/sshd -d  -g 15 > /dev/null 2>&1   &

# Get pid of backgrounded process
PID=$!
#echo PID is $PID

sleep 30

# Terminate above sshd process.  If a valid connection
# was received, and a child process was created to handle
# the connection, the child will continue.
kill -TERM $PID
exit 0;
```

The previous rule recognizes any single knock, writes the output to stdout, but does not perform any other action. To recognize a complete key, say TCP port 12345 followed within 30 seconds by TCP port 54321 we can use a Pair Rule. The correlated action (action2) can be to run the above script that starts sshd with apropriate options.

```
#  portknock.conf
#  Portknocking example 2
#
#BSD Jun 27 15:02:58 foohost /kernel: Connection attempt to TCP 127.0.0.1:5555 from
127.0.0.1:4356 flags:0x02
#

type=Pair
ptype=RegExp
pattern=\S+\s+\d+\s+\S+\s+(\S+)\s+/kernel: Connection attempt to (\S+) (\S+):12345
from (\S+):(\S+).*
desc=$0
action=write - first half of key received....
ptype2=RegExp
pattern2=\S+\s+\d+\s+\S+\s+(\S+)\s+/kernel: Connection attempt to (\S+) (\S+):54321
from (\S+):(\S+).*
desc2=$0
action2=write - second half received! Starting sshd for 30 seconds...; shellcmd
./opensesame.sh;
window=30
```

Run this example as follows (without informative debug):

```
$ perl sec.pl -conf=portknock.conf -input=/var/log/messages -debu=4
```

Once this example is running, switch to another screen and use telnet to knock on the key ports as follows:

```
$ telnet localhost 12345
```

Then quit immediately using Ctl-C. Then immediately:

```
$ telnet localhost 54321
```

and quit immediately again. All this activity should produce the following output on the **SEC** session:

```
# perl sec.pl -conf=t.conf -input=/var/log/messages -debug=4
Simple Event Correlator version 2.1.11
Reading configuration from t.conf
first half of key received....
second half received! Starting sshd for 30 seconds...
Terminated
```

During the 30 second window you should be able to ssh into your local machine with your own userid, i.e.:

```
$ ssh myuserid@localhost
```

You should receive a password prompt if you attempt to ssh within the window.

This example uses static port numbers, and recognizes a single key. There is also no outside authentication of the key. Using the **SEC** calendar rule, it would be easy to change keys on a regular basis. **SEC** contexts could also be used to provide a more robust recognition strategy. You could even bolt on a complete back-end authentication mechanism using the *shellcmd* action and elaborate scripts.

6.2 Process Control

Consider an engineering process that must control a motor or other device in response to several kinds of input, such as temperature, body position, or ambient light. These kinds of input can be modelled by text input into SEC. Suitable rules that determine how the device is to be controlled can be constructed according to a correlation map or other plan.

The goal of this example is to construct a running process that responds to some signal that is generated by **SEC** according to rules and text input.

The script below simulates a simple motor. It outputs a dot '.' at each inteval determined by the *$throttle* variable. Smaller values cause the motor to output it's dot faster, while larger values cause it to output dots slower. The script is set to resond to two signals- USR1 and USR1. A signal handler for USR1 adjusts the value of the throttle slower (smaller value), while another handler for USR2 adjusts the value of the throttle faster (smaller value).

```
#!/usr/bin/perl
#
# motor.pl  - simulate a motor with a throttle.
#

sub usage()
{
  print "\nUsage:  throttle.pl [num]\n";
  print "             Print a dot '.' every num seconds.\n";
  print "             Responds to signals USR1 (faster) and USR2 (slower).\n";
  exit 1;
}

$throttle = shift (@ARGV);

if ($throttle =~ /-(h|\?)/)
{
  &usage;
}

if (!($throttle =~ /\d+/))
{
  $throttle = 1;
}

# Attempt reset throttle up or down.

# Slower
$SIG{USR1} = sub {$throttle += ($throttle > .06) ? .05 :
                  (($throttle > .0025 ) ? .0025 : .0005);
             print STDERR $throttle;};

# Faster
$SIG{USR2} = sub {$throttle -= ($throttle > .06) ? .05 :
                  (($throttle > .0025 ) ? .0025 : .0005);
             print STDERR $throttle;};

print STDERR "ProcessID = $$\n";
sleep 1;

while(1)
```

```
{
  print STDERR ".";
  select undef, undef, undef, $throttle;
}
```

To operate, start up the script and observe the process id. In another session, send the process the USR2 signal. For example:

```
$ kill -USR2 10032
```

sends the USR2 signal to process 10032. When motor.pl receives the signal, it recalculates the throttle and runs faster. Repeat the above command several times to see how the process speeds up. Alternately, send the USR2 signal and see the process slow down.

To apply **SEC** in this case, consider the following ruleset:

```
# Example motor.conf
# Control a motor process from user input.
#
type=Single
ptype=RegExp
pattern=Start (\d+)
desc=$0
action=eval %p ($1)

# This rule speeds up the motor process.  Sends USR2 signal to process.
type=Single
ptype=RegExp
pattern=^f$
desc=$0
action=eval %r (kill USR2, %p)


# This rule slows down the motor process.  Sends USR1 signal to process.
type=Single
ptype=RegExp
pattern=^s$
desc=$0
action=eval %r (kill USR1, %p)
```

The first rule recognizes the process ID, input from the user. The second and third rules recognize keywords, ``f'' and ``s'' to make the motor go faster or slower. The action sends the appropriate signal to the process.

Note that in this case, the process control doesn't really start and stop the motor process- it just manipulates the throttle. However, it would not be hard to contruct a complete control system with a handful of rules similar to those above.

---

7 Conclusion

SEC is a verstile event correlation tool that can be used in many ways, and for many different applications. Hopefully, you now have the required knowledge and understanding to use **SEC** in your own applications. Be sure to check the **SEC** Users email list for up-to-date practices and answers to common (and not so common) questions.

Special thanks to Risto Vaarandi for providing us all with this unique tool!