Focus

Simple Event Correlator for real-time security log monitoring

Risto Vaarandi 

Difficulty

Over the past decade, event correlation has become a prominent event processing technique in many domains (network and security management, intrusion detection, etc.). However, existing open-source log monitoring tools don't support it well. In this paper, we will discuss how to employ SEC for monitoring and correlating events from security logs.

When it comes to the security of the IT system, event logs play a crucial role. Today, many applications, operating systems, network devices and other system components are capable of writing security related event messages to log files. The BSD *syslog* protocol is an event logging standard supported by majority of OS and network equipment vendors, which allows one to set up a central log server for receiving and storing event messages from the whole IT system. There also exist several flexible and powerful *syslog* server implementations that are suitable for use at the central log server, most notably Syslog-ng. Since event logging is a widely accepted and well-standardized practice, there is a high chance that after a security incident has occurred in an IT system, there is (are) also event log message(s) for it in some log file(s).

Because in most cases event messages are appended to event logs in real-time as they are emitted by system components, event logs are an excellent source of information for monitoring the system, including security conditions that arise in it. Over the past 10-15 years, a number of open-source tools have been developed for monitoring event logs in real-time, e.g., Swatch and Logsurfer. However, majority of these tools can accomplish simple tasks only, e.g., raise an alarm immediately after a certain message has been appended to a log file. On the other hand, many essential event processing tasks involve *event correlation* – a conceptual interpretation procedure where new meaning is assigned to a set of events that happen within

What you will learn...

- what event correlation is and what are the common approaches for event correlation,
- what was the motivation for developing SEC and what are its main features,
- how to employ SEC for real-time monitoring of security event logs.

What you should know...

- it is assumed that the reader is familiar with the regular expression language,
- the basic knowledge of Perl is helpful when reading the section *Integrating custom Perl code with SEC rules*.

a predefined time interval [Jakobson and Weissman, 1995]. A software application that implements event correlation is called *event correlator*, and during the interpretation procedure, the correlator might create new events and hide original events from the end user.

As an example of the importance of event correlation for security management, consider the processing of *login failure* events. Although an individual *login failure* event might be a symptom of a password cracking attempt, it could also indicate that the user accidentally typed a wrong password. Therefore, one can't simply configure the log file monitoring tool to send an immediate alert on the occurrence of *login failure* log message, since this could result in a high number of false positives. In order to reduce the number of false alarms, one or both of the following event correlation schemes can be used:

- once N *login failure for user X* events have been observed during the last T seconds, generate the *excessive number of login failures for user X* event and send it as an alarm to the security administrator,
- if the *login failure for user X* event appears and during the next T seconds no *successful login for user X* event will appear, generate the *login failure not followed by success for user X* event and send it as an alarm to the security administrator.

Over the past decade, a number of approaches have been proposed for event correlation, including rule-based [Froehlich et al., 2002], codebook based [Yemini et al., 1996], graph based [Gruschke 1998], neural network based [Wietgrefe et al., 1997; Wietgrefe 2002], and probabilistic [Meira 1997; Steinder and Sethi, 2002] methods. There are also a number of event correlator products available on the market, like HP ECS, SMARTS, NetCool, NerveCenter, LOGEC, and RuleCore.

The codebook based method (used by SMARTS) works as follows – if a set of events `e1, …, ek` must be interpreted as event A, then `e1, …, ek` are stored to the *codebook* as a bit vector pointing to A. If the correlator needs to correlate a set of events, it finds the most closely matching vector(s) from the codebook and reports the interpretation(s) corresponding to the vector(s). With the graph based method, the human analyst identifies all dependencies between system components (services, hosts, network devices, etc.) and constructs a graph with each node representing a system component and each edge a dependency between two components. When a set of events occurs, the graph is used for finding the root cause of events (e.g. *HTTP server not responding* events were caused by the failure of a single network link). With the neural network based method, a neural network is trained for the identification of anomalies in the event stream, for root cause detection, etc.

Listing 1. *SEC rule for correlating SNMP public access udp messages from Snort IDS*

```
# Sample matching input line:
# Mar  1 00:36:32 snorthost.mydomain [auth.alert] snort[17725]: [1:1411:10]
# SNMP public access udp [Classification: Attempted Information Leak]
# [Priority: 2]: {UDP} 192.168.115.34:54206 -> 192.168.52.179:161

type=SingleWithSuppress
ptype=RegExp
pattern=snort\[\d+\]: \[[\d:]+\] SNMP public access udp.*\{UDP\} \
([\d\.]+):\d+ -> ([\d\.]+):\d+
desc=SNMP public access from $1 to $2
action=pipe '%s' mail -s 'Snort alert' root
window=300
```

Rule-based approach is common for event correlation and has been employed in several products like HP ECS and RuleCore. In the case of this approach, events are correlated according to the rules *condition → action* specified by the human analyst. One of the main advantages of the rule-based event correlation is the fact that humans find it usually natural to express their knowledge in terms of rules. For example, it is easy to describe temporal relations between events with rules, while it could be cumbersome with other methods. Furthermore, unlike some other event correlation methods (e.g. neural network based correlation), the rule-based event correlation is clear and transparent to the end user. As argued in [Rich and Knight, 1991], if end users do not understand why and how the application reached its output, they tend to ignore the results computed by that application.

Although event correlation has become a prominent event processing technique in many domains (including network and security management, intrusion detection, etc.), existing open-source log file monitoring tools don't support it well. Despite the fact that event correlation systems that are currently available on the market have been highly successful and are used worldwide by many larger companies, they suffer from a number of drawbacks. Firstly, existing systems are often heavyweight solutions that have a complicated design and user interface. This means that their deployment and maintenance is time-consuming, and they require extensive user training. Also, their complexity and resource requirements often make them unsuitable for employment in smaller IT systems and for event correlation on nodes with limited computing resources. Secondly, since existing systems are mostly commercial, they are platform-dependent-customers supplied with program binaries that run on a limited number of operating systems. Furthermore, several commercial systems have been designed

for one particular network management platform only (e.g., HP OpenView). Some systems also suffer from the fact that they have been designed specifically for network fault management, and their application in other domains (including event log monitoring) is cumbersome. Thirdly, existing systems tend to be quite expensive, and therefore, many institutions with a more limited budget are unable to use them for daily security and system management tasks.

In this paper, we will discuss SEC (Simple Event Correlator) – an open-source tool developed by the author for lightweight and platform-independent event correlation – and we will analyze several real-life examples of how to employ SEC for monitoring and correlating events from security logs.

SEC basics

SEC is an open-source event correlation tool that uses rule-based approach for processing events. This approach was chosen because of its naturalness of knowledge representation and transparency of the event correlation process. The main design objectives for SEC were platform independence, lightweight build and ease of configuration, applicability for a wide variety of event correlation tasks, and low consumption of system resources.

In order to achieve independence from operating system platforms, the author decided to write SEC in Perl. Since Perl runs on almost every operating system flavor and has become a standard part of many OS distributions, Perl applications are able to run on a wide range of operating systems. In addition, well-written Perl programs are fast and memory-efficient.

SEC does not need much disk space and is very easy to install, since its current size is only about 250KB, and its configuration is stored in regular text files (the size of each file is typically a few kilobytes). Also, since SEC is written entirely in Perl and does not depend on other software packages, it can be used instantly after its source distribution has been unpacked, without any additional preparations (such as compiling and linking the source or installing other software).

Listing 2. *SEC ruleset for correlating sshd authentication failure and success messages on Solaris*

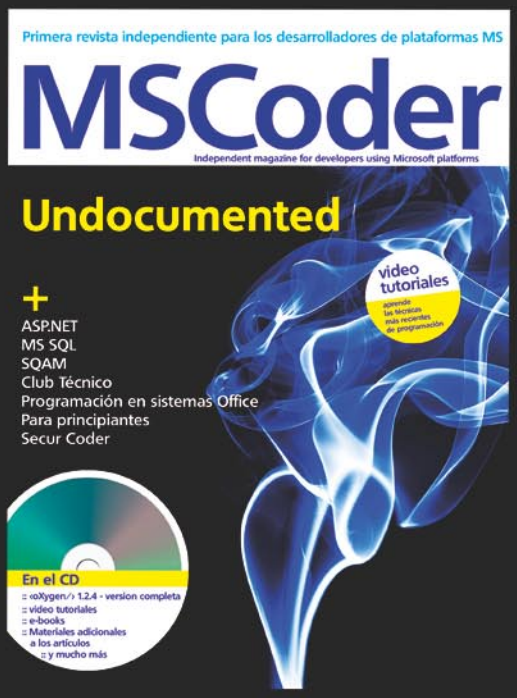
```
# Sample matching input lines:
# Apr  3 14:20:19 myhost sshd[25888]: [ID 800047 auth.error] error:
# PAM: Authentication failed for risto from myhost2
# Apr  3 14:20:23 myhost sshd[25888]: [ID 800047 auth.info] Accepted
# keyboard-interactive/pam for risto from 192.168.27.69 port 9729 ssh2

type=PairWithWindow
ptype=RegExp
pattern=sshd\[\d+\]: \[ID \d+ auth\.error\]\
 error: PAM: Authentication failed for (\S+) from \S+
desc=PAM authentication failed for $1
action=event PAM_AUTHENTICATION_FAILED_FOR_$1
ptype2=RegExp
pattern2=sshd\[\d+\]: \[ID \d+ auth\.info\]\
Accepted keyboard-interactive/pam for ($1) from \S+ port \d+ ssh2
desc2=PAM authentication successful for $1
action2=none
window=30

type=SingleWithThreshold
ptype=RegExp
pattern=PAM_AUTHENTICATION_FAILED_FOR_(\S+)
context=!USER_$1_ALREADY_COUNTED && !COUNTING_OFF
continue=TakeNext
desc=Ten authentication failures for distinct users have been observed
action=pipe '%s' mail -s 'PAM alert' root; create COUNTING_OFF 3600
window=600
thresh=10

type=Single
ptype=RegExp
pattern=PAM_AUTHENTICATION_FAILED_FOR_(\S+)
context=!USER_$1_ALREADY_COUNTED && !COUNTING_OFF
desc=Set up the "count once" context for user $1
action=create USER_$1_ALREADY_COUNTED 600
```

SEC receives its input events from file streams. Regular files, named pipes, and standard input are currently supported as input, allowing one to use SEC as an event log monitoring solution and to integrate it with any application that is able to write its output events to a file stream. Applications that have an event management API can also be integrated through simple plugins that employ API calls to read the application's event stream, and copy it to the standard output or file (a sample plugin for HP OpenView Operations is a part of the SEC package).

SEC can produce output events by executing user-specified shell commands, by writing messages to files or named pipes, by calling precompiled Perl subroutines, etc. Note that output events can be sent over the network to another instance of SEC, allowing one to configure distributed event correlation schemes. Also, although SEC does not have a GUI for viewing and managing output events, it is straightforward to direct output events to a system management application/framework that has such a GUI (e.g. HP OpenView Operations).

SEC configuration is stored in text files which can be created and modified with any text editor. Each configuration file contains one or more rules, and rulesets from different files are applied virtually in

ONLY FRESH IDEAS
TO ORDER: BUYITPRESS.COM



MORE: WWW.SOFTWARE.COM.PL

Software Developer's JOURNAL
new ideas & solutions for professional programmers
Polish, English, Spanish, German and French language versions

MSCoder
Independent magazine for developers using Microsoftplatform
Spanish, French and German language versions

hakin9
Hard Core IT Security Magazine
Polish, French, Spanish, Italian, English, Czech and German language versions

Linux+ DVD
Europe's biggest Linux magazine
Polish, French, Spanish, Czech and German language versions

WE ARE LOOKING FOR LICENSORS AND DISTRIBUTORS WORLDWIDE
CONTACT: MONIKA GODLEWSKA, MONIKAG@SOFTWARE.COM.PL

parallel. SEC reads data from input sources line by line, and each time a new line has been read, it will be matched against rules in configuration file(s).

An important part of the SEC rule is the *event matching pattern*. SEC supports regular expressions, substrings, Perl subroutines, and truth values as patterns. Support for regular expressions eases the configuration of SEC, since many UNIX tools (like *grep*, *sed*, *find*, etc.) rely on regular expressions, and therefore most security, system and network administrators are already familiar with the regular expression language. Also, since majority of event log monitoring tools use regular expression language for matching events, SEC can be deployed as a log monitoring replacement with much less effort. Starting from the 2.3.0 version, events can be passed to precompiled Perl subroutines for recognition which allows the user to configure custom event matching schemes.

In addition to event matching pattern, most rule definitions specify a list of *actions*, and optionally a Boolean expression of *contexts*. The SEC contexts are logical entities created during the event correlation process, with each context having a certain lifetime (either finite or infinite). Contexts can be used for activating and deactivating rules dynamically at runtime, e.g., if a rule definition has (X OR Y) specified for its context expression and neither the context X nor the context Y exist at a given moment, the rule will not be applied. Another important function of the SEC contexts is to act as event stores – events of interest can be associated with a context, and all the collected events supplied for an external processing at a later time (this idea was borrowed from Logsurfer).

Currently, SEC supports nine rule types that implement a number of common event correlation scenarios:

- *Single* – execute an action list when matching event is observed,
- *SingleWithScript* – like *Single*, but also use an external script for matching,
- *SingleWithSuppress* – like *Single*, but ignore following matching events for *t* seconds,
- *Pair* – execute an action list on event A and ignore following instances of A until event B arrives; on the arrival of B execute another action list,
- *PairWithWindow* – after observing event A, wait for *t* seconds for event B to arrive; if B does not arrive on time, execute an action list, otherwise execute another action list,
- *SingleWithThreshold* – count matching input events during *t* seconds and if a given threshold is exceeded, execute an action list,
- *SingleWith2Thresholds* – like *SingleWithThreshold*, but with additional second round of counting with a falling threshold,
- *Suppress* – suppress matching input events,
- *Calendar* – execute an action list at specific times.

Most SEC rule definitions have a parameter called *event description string* that is employed for defining the scope of event correlation (see *SEC rules and event correlation operations* for a detailed discussion). When an event matches the rule, SEC calculates the event correlation key by concatenating the rule file name, rule ID, and event description string. If an event correlation operation with the same key exists, event is correlated by that operation. If there is no such operation and the rule specifies a correlation of events over time, SEC starts a new operation with the calculated key. It should be noted that there is no one-to-one correspondence between rules and event correlation operations – SEC could start several operations for one rule, and rules of type *Single*, *SingleWithScript*, *Suppress*, and *Calendar* will never trigger operations, because they don't define event correlation over a time window.

SEC actions were not only designed for generating output events, but also for making rules to interact, for managing contexts and storing events, for connecting external event analysis modules to SEC, for executing custom Perl code without forking a separate process, etc. By combining several rules with appropriate action lists and context expressions, more complex event correlation schemes can be defined. The following section provides detailed examples and discussion on building SEC rulesets for security event log monitoring.

Security log monitoring with SEC – ruleset examples and discussion

In this section, we will discuss several ruleset examples and event processing capabilities of SEC. The example rulesets have been written for monitoring real-life event logs – the Snort IDS event log, the Solaris */var/adm/messages* system log, and the Apache web server error log. The rulesets have been tested with SEC version 2.3.3.

For experimenting with the rulesets presented in this section, one can download SEC from its home page. For installing SEC from the source package, unpack the distribution (e.g., `tar -xzvf sec-2.3.3.tar.gz`) and copy the *sec.pl* file from the distribution to the appropriate directory (e.g., `cp sec-2.3.3/sec.pl /usr/local/bin`). SEC home page also contains links to binary packages of SEC for several OS platforms.

In order to start SEC in interactive mode for monitoring the */var/log/messages* log file with rules from *my.conf*, use the following command line:

```
sec.pl -conf=my.conf -input=/var/log/messages
```

In order to configure SEC to monitor its standard input (useful for testing purposes), use the following command line:

```
sec.pl -conf=my.conf -input=-
```

Listing 3. *SEC ruleset for consolidating priority 1 alert messages from Snort IDS*

```
# Matching input line:
# Apr  4 10:10:55 snorthost.mydomain [auth.alert] snort[18800]:
# [1:2528:14] SMTP PCT Client_Hello overflow attempt
# [Classification: Attempted Administrator Privilege Gain]
# [Priority: 1]: {TCP} 192.168.5.43:28813 -> 192.168.250.44:25

type=Single
ptype=RegExp
pattern=snort\[\d+\]: \[[\d:]+\].*\[Priority: 1\]: \S+ \
([\d\.]+):?\d* -> [\d\.]+:?\d*
context=!ATTACK_FROM_$1
continue=TakeNext
desc=Priority 1 attack started from $1
action=create ATTACK_FROM_$1; \
        pipe '%s' mail -s 'Snort: priority 1 attack from $1 (alert)' root

type=Single
ptype=RegExp
pattern=snort\[\d+\]: \[[\d:]+\].*\[Priority: 1\]: \S+
([\d\.]+):?\d* -> [\d\.]+:?\d*
context=ATTACK_FROM_$1
desc=Priority 1 incident from $1
action=add ATTACK_FROM_$1 $0; \
        set ATTACK_FROM_$1 300 ( report ATTACK_FROM_$1 \
        mail -s 'Snort: priority 1 attack from $1 (report)' root )
```

Listing 4. *SEC rule for passing lines that come from /var/log/messages only*

```
type=Suppress
ptype=TValue
pattern=TRUE
context=!_FILE_EVENT_/var/log/messages
desc=Pass only those lines that come from /var/log/messages
```

Note that one can specify several `-input` and `-conf` options in the command line. Other commonly used options include `-log` (sets the log file for SEC), `-syslog` (configures SEC to log through *syslog*), `-debug` (sets the logging level for SEC), `-pid` (sets the process ID file for SEC), `-detach` (forces SEC to disassociate itself from the controlling terminal and to become a daemon), and `-testonly` (tests the validity of rules without starting SEC).

SEC rules and event correlation operations

Suppose we have a rule file called *my.conf* containing one rule presented in Listing 1.

The *SingleWithSuppress* rule from Listing 1 has been designed for matching *SNMP public access udp* messages from the Snort IDS log. Each time the Snort daemon observes an SNMP query packet with the *public* community field in the network, it logs such a message – however, since a number of network management tools poll the same host repeatedly during a short time interval, the message could also be logged repeatedly for the same source and destination IP addresses. The rule implements an event correlation scenario called *compression* – repeated occurrences of identical events are reduced into a single event. The *ptype* parameter of the rule definition specifies that the event matching pattern is a regular expression, and the *pattern* parameter specifies the regular expression. The *desc* parameter defines the event description string, the *action* parameter the action list of sending an e-mail alert to the local *root* user, and the *window* parameter the correlation window of 300 seconds.

When the regular expression matches an input line, the special variables $1 and $2 will be set to the source and destination IP address fields of the input line, since the regular expression contains bracketing constructs for these fields. SEC will then calculate the event correlation key by concatenating the rule file name, rule ID and event description string – e.g., if $1 is 192.168.115.34 and $2 is 192.168.52.179, then the resulting key will be `my.conf | 0 | SNMP public access from 192.168.115.34 to 192.168.52.179` (rule IDs start from zero and the bar symbol is used as a separator). If the operation with the key exists, SEC will hand over the input event to the operation. If the operation with the key does not exist, SEC will start a new operation with the lifetime of 300 seconds. The operation immediately sends an e-mail alert to the local *root* user with the *pipe* action – the event description string denoted by %s will be piped to the standard input of the `mail -s 'Snort alert' root` command – and after that, the operation will ignore the following events received from SEC for correlation. In other words, the rule will reduce repeated *'SNMP public access udp'* messages for the same source and destination IP address into a single message (the first one).

The inclusion of the rule file name and rule ID in the event correlation key guarantees that event correlation operations triggered by different rules will never clash. Also, by choosing appropriate value for the *desc* parameter, the end user can change the scope of event correlation. E.g., if the value for the *desc* parameter is `SNMP public access from $1`, SEC will reduce all messages with the same source IP address field into a single message, disregarding destination IP addresses completely.

As a final note, one should be careful when using $1, $2, … special variables as a part of a command

line definition, since the content of the special variables will be interpreted by the shell like the rest of the command line. E.g., if the *pattern* parameter is `sshd\[\d+\]: (.+)` and the *action* parameter is `shellcmd echo $1 >> myfile`, then a malicious user can fork an arbitrary command from SEC by logging a fake line `sshd[0]: `mycommand`` with the *logger* utility. In order to avoid such situations, SEC patterns that set special variables for command lines should be written in a way that shell metacharacters and other unexpected data would not be assigned to the variables.

Building SEC rulesets from individual rules

The ruleset presented in Listing 2 for processing authentication failure and success messages is a more complex example that illustrates how rules can be set to interact through the use of synthetic events and contexts. The purpose of the ruleset is to weed out accidental authentication failures that are shortly followed by success, and then count non-accidental failures, in order to detect attempts to hack a larger number of different accounts in a short time period and to distinguish those attempts from an activity against a single (or a few) account(s).

The first rule of type *PairWithWindow* has been designed for matching *sshd* authentication failure and success messages from the Solaris */var/adm/messages* system log. After the regular expression given with the *pattern* parameter matches an authentication failure message for a user, the $1 variable will be set to the user name. SEC then starts an event correlation operation which will wait for the authentication success message for the same user name during the next 30 seconds. If the authentication success message arrives on time, no action will be taken (because the *action2* parameter is set to `none`). It should be noted that with *Pair** rules one can use $1, $2, … special variables in the *pattern2* parameter, i.e., the pattern for the second half of the *Pair** rule can have a dynamic nature. If the authentication success message does not appear, the operation will generate a synthetic event called `PAM_AUTHENTICATION_FAILED_FOR_<username>` with the *event* action. SEC synthetic events are treated like regular input events read from log files – they are appended to the input queue and matched against all rules.

The second rule of type *SingleWithThreshold* starts an event correlation operation that matches and counts `PAM_AUTHENTICATION_FAILED_FOR_<username>` messages. If 10 messages have been observed in the window of 600 seconds, the operation sends an e-mail alert to the local *root* user, and also, creates the context `COUNTING_OFF` with the lifetime of 1 hour, in order to avoid sending alerts to *root* once per each 10 minute period if the account scan is long-lasting. The expression given with the *context* parameter of the rule definition reads: *the context USER_<username>_ALREADY_COUNTED does not exist and the context COUNTING_OFF does not exist* (in SEC context expressions, `!` means logical negation, `&&` logical AND, and `||` logical OR). Therefore, in the presence of the `COUNTING_OFF` context the expression evaluates false, and the rule will not match any event. After the `PAM_AUTHENTICATION_FAILED_FOR_<username>` event has been counted, it will be passed to the third rule, because the *continue* parameter of the second rule has the value `TakeNext`. The third rule creates the context `USER_<username>_ALREADY_COUNTED`, and since the lifetime of the context and the counting window are equal (600 seconds), this ensures that each distinct user name increases the counter value only once during the counting (after the context has been created for a user name, the context expression of the second rule for the user name will evaluate false). In other words, the interaction between the second and third rule means that e-mail alerts will be sent only for incidents involving ten distinct user accounts.

Using SEC contexts for event consolidation

SEC contexts cannot only be used for rule activation and deactivation, but they can also be employed as event stores. SEC has the *add* action for appending an event to the event store of the context, the *report* action for piping all events from the store to the standard input of an external command, plus a number of actions for other context operations (e.g., moving data between contexts and SEC special variables). In this section, we will look at a simple scenario how to employ contexts for Snort IDS alert message aggregation and reporting.

Alert messages that Snort daemon logs have a priority from 1 to 3 (with 1 being the highest and 3 the lowest), and each message has a source and destination IP address field that reflect the source and destination of the suspicious network traffic. It is quite common that after Snort has observed an event for a certain source IP, the event will be shortly followed by other events for the same IP address (this is particularly true for attacks carried out with a toolkit that attempts to find as many vulnerabilities as possible in the destination network). Therefore, it is often not wise to generate an alert on every event, but to consolidate events into fewer reports.

The ruleset presented in Listing 3 was designed for processing Snort priority 1 alert messages with the same source IP address field (in the rest of this subsection, the network activity triggering such messages is called an *attack*). When the first priority 1 message is observed for a certain source IP address, SEC will send an e-mail alert about the start of an attack. If no priority 1 messages have been seen during 5 minutes for that source IP, SEC considers it to be the end of the attack, and sends an e-mail report containing all log messages relevant to the attack.

For storing log messages for a certain IP address `<ipaddress>`, the ruleset creates the context `ATTACK_FROM_<ipaddress>`. The first rule

detects the first event of an attack – the rule matches a priority 1 event for the source IP address only if the context for that IP address has not been created yet. After matching the event, the rule creates the context and sends an e-mail alert to the local *root* user that an attack has begun. The second rule matches a priority 1 log message and appends it to the event store of the relevant context with the *add* action (the $0 special variable holds the entire matching log message line). After that, the rule uses the *set* action for extending the context lifetime for the next 300 seconds, and for setting the action-on-delete for the context (`report ATTACK_FROM_$1 mail -s 'Snort: priority 1 attack from $1 (report)' root`). The action-on-delete will be executed immediately before the context's lifetime ends and the context is deleted, i.e., when no priority 1 events for a given IP have been observed during the last 300 seconds. The action-on-delete uses the *report* action for piping the event store of the context to the `mail -s 'Snort: priority 1 attack from $1 (report)' root` command which sends collected events to the local *root* user. In that way, attacks that comprise many events will be reported with a single e-mail, and on the other hand, even if the attack is long-lasting, the end user will still get a timely e-mail alert about its start.

Monitoring multiple files

Apart from advanced event correlation and consolidation capabilities, SEC has another important advantage over several other well-known log monitoring solutions – it is the ability to monitor several log files simultaneously which allows SEC to cross-correlate events from different sources. Also, when there are a larger number of log files on the system, they can be monitored by a single SEC process that not only saves space in the process table, but also eases the maintenance of SEC itself (e.g., SEC will have just one process ID file and log file). Configuring SEC to monitor more than one input source is easy – one just has to give more than one `-input` option in the command line or specify a file name containing wildcard(s) for the `-input` option (or both).

However, when there are many rules, having more than one input source could introduce performance and transparency problems. If there are many rules that have been designed for one input source only, the matching of lines from other input sources with such rules could involve a considerable runtime overhead. Also, if input lines coincidentally match the rule they were not supposed to match, unexpected side-effects might make the behavior of the ruleset incomprehensible for the end user.

In order to address these problems, SEC has the `-intcontexts` command line option that tells SEC to create an *internal context* after a line has been read from an input source, and to delete the context after the line has been matched against all rules. E.g., if the name of the input source is */var/log/messages*, the name of the corresponding internal context is `_FILE_EVENT_/var/log/messages`. Since the names of internal contexts can be used in context expressions of rule definitions, the user can write rules that match events from certain input sources only. If the user wishes to have custom names for internal contexts or a single name for multiple input sources, the names can be specified with the `-input` option. E.g., `-input=/var/log/syslog=SYSLOG -input=/var/adm/messages=SYSLOG` options instruct SEC to employ the internal context `SYSLOG` for both */var/log/syslog* and */var/adm/messages*.

As an example of the use of internal contexts, consider the *Suppress* rule from Listing 4 in the beginning of the rule file.

The SEC *Suppress* rule suppresses matching events – it acts as a filter that does not pass the events to later rules in the rule file. In the rule definition from Listing 4, the *ptype* and *pattern* parameters specify that the *pattern* is a truth

Listing 5. *SEC ruleset for monitoring the local Apache web server log with a dynamic list of regular expressions and for forwarding matching lines to the remote syslog server*

```
type=Single
ptype=SubStr
pattern=SEC_STARTUP
context=SEC_INTERNAL_EVENT
continue=TakeNext
desc=Load the Sys::Syslog module
action=assign %a 0; eval %a (require Sys::Syslog); \
eval %a (exit(1) unless %a)

type=Single
ptype=RegExp
pattern=(SEC_STARTUP|SEC_RESTART)
context=SEC_INTERNAL_EVENT
desc=Compile the logging routine and initialize the list of patterns
action=eval %syslog ( sub { Sys::Syslog::syslog('err', $_[0]); } ); \
       eval %a ( @regexp = ('192\.168\.1\.1', 'File does not exist:'); \
                 Sys::Syslog::openlog('SEC', 'cons,pid', 'daemon') )

# Matching input line:
# [Fri Mar 24 09:19:50 2006] [error] [client 192.168.1.1]
# File does not exist: /var/apache/htdocs/robots.txt

type=Single
ptype=PerlFunc
pattern=sub { foreach my $pat (@regexp) {\
 if ($_[0] =~ /$pat/) { return 1; } } return 0; }
desc=Forward the suspicious message line to remote syslog server
action=call %o %syslog $0
```

value TRUE that matches any line. However, the context expression `!_FILE_EVENT_/var/log/messages` evaluates true only for lines not coming from */var/log/messages*. Therefore, the rule can be used in the beginning of the rule file designed for monitoring */var/log/messages*, since it only passes relevant lines.

If the *–intcontexts* command line option has been given, SEC employs the internal context `_INTERNAL_EVENT` for synthetic events generated with the *event* action. However, sometimes the end user would like to have another internal context for a synthetic event. As a workaround, one can create a named pipe with the *mkfifo* tool, let SEC to monitor the named pipe with the `-input` option, and use the *write* action instead of *event* in rule definitions. E.g., if the named pipe */var/log/pipe* has been created with `mkfifo /var/log/pipe` and SEC has been started with the command line option `-input=/var/log/pipe=SYSLOG`, then using `action=write /var/log/pipe MY_SYNTHETIC_EVENT` (it tells SEC to write the line `MY_SYNTHETIC_EVENT` to */var/log/pipe*) makes the `MY_SYNTHETIC_EVENT` event appear with the `SYSLOG` internal context set.

Integrating custom Perl code with SEC rules

Although the features of SEC we have discussed so far allow one to write rulesets for a wide variety of event correlation scenarios, there are still cases that can't be covered by combining these features. E.g., *RegExp* patterns can't be used for specifying a dynamic list of regular expressions. Also, the *pipe* action from previous ruleset examples involves creating a separate process for an external command, but when *pipe* is called hundreds of times per second, considerable amount of CPU time would be spent for forking new processes. Although SEC supports special variables that the user can employ for storing values, these variables are similar to Perl scalars and more complex data structures (like Perl lists and hashes) can't be set up with them. In order to address these problems, SEC supports *PerlFunc* patterns (user-defined Perl functions for matching input lines) and Perl context expressions, but also *eval* and *call* actions for compiling and running custom Perl code from SEC.

The ruleset from Listing 5 illustrates how to employ *eval* and *call* actions and *PerlFunc* patterns, but also how to use Perl modules with SEC and how to set up and access Perl data structures with custom code. The ruleset was designed for monitoring the local Apache web server error log with a dynamic list of regular expressions, and for forwarding matching lines to the remote *syslog* server where they could be correlated by another SEC instance. In order to save CPU time, the ruleset does not call the *logger* utility for forwarding lines as *syslog* messages, but rather relies on the `openlog()` and `syslog()` functions of the Perl *Sys::Syslog* module.

In order to take advantage of the *Sys::Syslog* module, it must be loaded at SEC startup. If SEC has been started with the `-intevents` command line option, it generates a synthetic event called `SEC_STARTUP` as its very first event at startup, sets the internal context `SEC_INTERNAL_EVENT` for the event, and processes it before any other input event. This allows the user to write rules for executing various startup procedures. The first rule is such a rule which attempts to load the *Sys::Syslog* module with the help of *assign* and *eval* actions. It first sets the special variable %a to 0 with the *assign* action, and then evaluates the Perl code `require Sys::Syslog` with the *eval* action (internally, the *eval* action calls the Perl `eval()` function). If *eval* succeeds and the module is loaded, 1 will be assigned to %a (since this value is returned by the successful `require Sys::Syslog`), if *eval* fails, %a will retain its original value (0). Then the *eval* action is used again for checking the value of %a, and if it is 0 (i.e., the module couldn't be loaded), `exit(1)` is called from the Perl code executed by *eval*. Since the execution takes place within the SEC process, `exit(1)` will terminate SEC with the exit code 1.

The second rule has been designed for matching both `SEC_STARTUP` and `SEC_RESTART` internal events (when SEC has been started with the `-intevents` option and it receives the SIGHUP signal – a request for resetting internal state and reloading configuration –, then SEC generates a synthetic event `SEC_RESTART` with the internal context `SEC_INTERNAL_EVENT`). After observing a matching event, the rule first uses the *eval* action for evaluating the Perl code `sub { Sys::Syslog::syslog('err', $_[0]); }`. Since the code is a function definition, *eval* will compile the function and return the pointer to the compiled code that will be saved to the %syslog special variable. The function itself expects one input parameter and employs the `syslog()` function from the *Sys::Syslog* module for sending the input parameter as an *err*-level message to the *syslog* server. The rule will then initialize the *@regexp* list which is a Perl list for holding regular expressions. Since *@regexp* is a global list, it can be accessed and modified with subsequent calls to *eval*. (In order to avoid clashes with variable names in the SEC code, a separate namespace called `main::SEC` is defined in the SEC code, and the *eval* action always evaluates custom Perl code in that namespace.) As a final step, the rule will open the *syslog* connection with the `openlog()` function, setting the program name to `SEC`, the logging facility to `daemon`, and logging options to `cons,pid` (log to console if regular logging fails and include process ID with each message).

The third rule was designed for matching input lines with regular expressions from the *@regexp* list that was initialized by the second rule (and can be changed by other rules at runtime). The rule employs the *PerlFunc* pattern for matching – the value for the *pattern* parameter must be a valid Perl function definition that is compiled when rules are loaded. In the case of the third rule, the func-

THE AFFILIATE PROGRAM
Software-Wydawnictwo

Join and start making money!

The affiliate program offered by Software-Wydawnictwo publishing house is aimed at website owners and administrators.

Anyone who has a website can join the Affiliate Program.

Joining the program is completely free and can bring you significant financial benefits.
To start making money, simply put a link (a banner or button) to our online store on your website!

You will receive 10% of the value of each purchase made in our store by visitors coming from your website.

www.pp.software.com.pl

tion takes the input line (passed to the function as the input parameter $_[0]) and scans the *@regexp* list for a matching regular expression. If such a regular expression is found, the function returns 1 which is an indication that the *PerlFunc* pattern matches the input line, otherwise it returns 0 which indicates no match. In the former case, the rule will call a precompiled Perl function for *syslog* logging with the *call* action. The %o special variable is used for storing the return value from the function call, the %syslog special variable holds a pointer to the function, and $0 (that holds the entire matching input line) is the input parameter for the function.

In that way, the ruleset efficiently implements the dynamic regular expression matching for a web server error log which can't be expressed in terms of *RegExp* patterns, and the forwarding of matching lines to remote *syslog* server without forking a separate process for an external command. Since all Perl code fragments employed by the third rule are compiled at SEC startup, executing them at runtime is as efficient as executing the SEC code itself.

SEC performance and application experience

Although SEC is written in an interpreted language (and is thus not as fast and memory-efficient as a compiled C program), it can handle hundreds of events per second and still have relatively modest resource requirements. In a recently conducted experiment that lasted 49.8 days, two instances of SEC were set to run on a Linux *syslog* server with two 3 GHz Intel P4 Xeon processors. The first instance was monitoring 20 log files simultaneously with a configuration of 243 rules from 22 rule files, while the second instance was reading input from a named pipe with a configuration of 67 rules from 5 rule files. The first instance processed 107,059,511 input lines (24.9 lines per second as an average), and consumed 3.0% of CPU time and 8.1 MB of memory. The second instance

On the Net

- *http://www.bmc.com/* – BMC Patrol,
- *http://www.cisco.com/* – CiscoWorks,
- *http://www.managementsoftware.hp.com/products/ecs/index.html* – HP ECS,
- *http://www.openview.hp.com/* – HP OpenView,
- *http://www.netfilter.org/* – Iptables,
- *http://www.logec.com/* – LOGEC,
- *http://www.cert.dfn.de/eng/logsurf/* – Logsurfer,
- *http://www.mysql.com/* – MySQL,
- *http://www.nagios.org/* – Nagios,
- *http://www.openservice.com/products/nervecenter.jsp* – NerveCenter,
- *http://www.micromuse.com/* – NetCool,
- *http://www.prelude-ids.org/* – Prelude IDS,
- *http://www.rulecore.com/* – RuleCore,
- *http://simple-evcorr.sourceforge.net/* – Simple Event Correlator,
- *http://www.smarts.com/* – SMARTS,
- *http://snmptt.sourceforge.net/* – SNMPTT,
- *http://www.snort.org/* – Snort IDS,
- *http://swatch.sourceforge.net/* – Swatch,
- *http://www.balabit.com/products/syslog_ng/* – Syslog-ng.

References

- Jim Brown. 2003. Working with SEC – the Simple Event Correlator. *http://sixshooter.v6.thrupoint.net/SEC-examples/article.html,*
- P. Froehlich, W. Nejdl, M. Schroeder, C. V. Damasio, L. M. Pereira. 2002. *Using Extended Logic Programming for Alarm-Correlation in Cellular Phone Networks. Applied Intelligence 17(2)*, pp. 187-202,
- Boris Gruschke. 1998. *Integrated Event Management: Event Correlation using Dependency Graphs. Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pp. 130-141,
- G. Jakobson and M. Weissman. 1995. *Real-time telecommunication network management: Extending event correlation with temporal constraints. Proceedings of the 4th International Symposium on Integrated Network Management*, pp. 290-301,
- Dilmar Malheiros Meira. 1997. *A Model For Alarm Correlation in Telecommunication Networks. PhD thesis*, Federal University of Minas Gerais, Brazil,
- Elaine Rich and Kevin Knight. 1991. *Artificial Intelligence, 2nd edition*, McGraw-Hill, ISBN 0-07-052263-4,
- John P. Rouillard. 2004. *Real-time Logfile Analysis Using the Simple Event Correlator (SEC). Proceedings of USENIX 18th System Administration Conference*, pp. 133-149,
- M. Steinder and A. S. Sethi. 2002. *End-to-end Service Failure Diagnosis Using Belief Networks. Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium*, pp. 375-390,
- James Turnbull. 2005. *Hardening Linux*, Apress, ISBN: 1-59059-444-4.
- Risto Vaarandi. 2005. *Tools and Techniques for Event Log Analysis. PhD thesis*, Tallinn University of Technology, Estonia,
- Hermann Wietgrefe. 2002. *Investigation and Practical Assessment of Alarm Correlation Methods for the Use in GSM Access Networks. Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium*, pp. 391-404,
- Hermann Wietgrefe, Klaus-Dieter Tuchs, Klaus Jobmann, Guido Carls, Peter Froehlich, Wolfgang Nejdl, Sebastian Steinfeld. 1997. *Using Neural Networks for Alarm Correlation in Cellular Phone Networks. Proceedings of the International Workshop on Applications of Neural Networks in Telecommunications*, pp. 248-255,
- S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. 1996. *High speed and robust event correlation. IEEE Communications Magazine 34(5)*, pp. 82-90.

processed 364,534,428 input lines (84.7 lines per second as an average), and consumed 8.8% of CPU time and 6.1 MB of memory.

SEC event processing speed depends heavily on how the rules are arranged, and there are several ways for improving the performance. Since input lines are compared with rules in the order they are defined in the rule file, moving most frequently matching rules to the beginning of the file saves CPU time. Also, if many input lines don't match any rules, having a *Suppress* rule for such lines in the beginning of the rule file saves CPU time as well. If SEC has been configured to monitor several input sources, one can employ internal contexts (as described in *Monitoring multiple files*) for increasing SEC event processing speed. Other suggestions for improving SEC performance include writing efficient regular expressions and replacing *RegExp* patterns with *SubStr* patterns where possible (the latter are faster).

Over the past few years, SEC has been adopted by many institutions with various sizes and has been employed in a number of domains, including event log monitoring, firewall management, intrusion detection, and network management (please see [Vaarandi 2005] for some detailed case studies). SEC has been successfully used with Snort IDS, Prelude IDS, the iptables firewall, HP OpenView (both NNM and Operations), Nagios, CiscoWorks, BMC patrol, SNMPTT, etc. SEC has been employed on a wide variety of OS platforms, including Linux, FreeBSD, OpenBSD, Solaris, HP-UX, AIX, Tru64 Unix, Mac OS X, and Windows 2000.

Conclusion

This paper has discussed SEC (Simple Event Correlator) – an open-source tool for lightweight and platform-independent event correlation – and has presented several real-life examples how to employ SEC for real-time monitoring of security event logs. However, due to space limitations, many features of SEC were not mentioned in this paper. For a thorough description, the interested reader is referred to the SEC online documentation. There are also several other sources of information available about SEC. SEC rule repository at BleedingSnort (*http://www.bleedingsnort.com/sec/*) contains a number of example rulesets for various scenarios (e.g. event correlation for Snort and management of the iptables firewall). *Working with SEC – the Simple Event Correlator* [Brown 2003] is an online tutorial that not only provides a good introduction to SEC but also covers a number of advanced issues like integrating SEC with MySQL. Chapter 5 of *Hardening Linux* [Turnbull 2005] discusses how to employ SEC for monitoring *syslog* log files. Also, recently a paper with a useful ruleset library has been published that describes the application of SEC at the University of Massachusetts at Boston [Rouillard 2004]. ●

Acknowledgements

This work is supported by SEB Eesti Ühispank, and also, the work has received financial support from Estonian national grant no. SF0182712s06.

About the author

Risto Vaarandi received his PhD in Computer Engineering from the Tallinn University of Technology, Estonia, in June 2005. For the past eight years, he has been working in SEB Eesti Ühispank as an IT development engineer, and currently he is also a part-time researcher at the Institute of Computer Science, University of Tartu, Estonia. You can contact Risto through his home page at *http://kodu.neti.ee/~risto*.