SIEM monitoring

SIEM monitoring

Introduction

The acronym SIEM stands for Security Information and Event Management. SIEM describes security monitoring features by collecting, filtering, normalizing, correlating and visualizing security events. It combines security information management (SIM) with security event management (SEM).

SIEM manages security events, which may come from an ordinary log (e.g. Apache logs), from the log of a specific security tool (logs of a firewall, IDS, honeypot, EDR, et cetera) or enriched events from another tool (another SIEM).

To take full advantage of all SIEM PFMS features, it is recommended to have at least version 783 installed in the EndPoints.

How it works

Pandora FMS SIEM is responsible for processing log entries obtained through your collection, normalizing the data and generating security events based on these entries.

As with log collection, the generated SIEM events are stored in OpenSearch®, so the first requirement to make this monitoring work is to have an OpenSearch installation.

Pandora FMS SIEM generates security events using two components in the server, so events are also generated in two steps:

- Data standardization: All log collection entries are decoded, generating new normalized log entries that are stored temporarily.
- Event generation: Normalized logs are checked for compliance with a set of rules, in which case a SIEM event is generated with all the rule information and the normalized log that generated the event.

Thus the complete flow for SIEM monitoring is:

- Agents, plugins and other sources of information monitor or generate logs that are sent to the `dataserver` or `syslogserver`.
- `dataserver` and `syslogserver` take care of processing these log entries and store them on the

OpenSearch server configured for log collection.
- `siemserver` decodes all logs obtained through log collection to generate normalized logs on the OpenSearch server configured for SIEM monitoring. These normalized logs are stored temporarily.
- `siemevents` processes each of the normalized logs and if they meet a set of predefined rules, SIEM events are generated and stored on the OpenSearch server configured for SIEM monitoring.

With the events generated, it is possible to check them for their operation from Pandora FMS Console.

Console configuration

To use SIEM event monitoring, it must first be activated from the main configuration.

Go to menu Management → Settings → System Settings → SIEM → Activate SIEM, fill in the form completely and click Update to save the changes.

This will create the necessary templates on the OpenSearch server specified for log normalization and SIEM event generation.

It will also activate this type of monitoring in Pandora FMS servers where `siemserver` and `siemevents` were started.

If Command Center is enabled, Pandora SIEM will only be available in the nodes.

Server configuration

To perform SIEM event monitoring, activate `siemserver` and `siemevents` servers in server configuration.

In order to decode and normalize log collection entries, it will be necessary to have decoding XML files that establish the way to obtain the necessary information in each case (hereinafter, “decoders”). These XML files will be located in the path indicated by the `siem_decoders` parameter of the server configuration, by default in:

```
/usr/share/pandora_server/util/siem/decoders
```

In order to generate SIEM events, it will be necessary to have rules XML files that set the conditions to generate an event based on the information obtained in normalized logs (hereinafter, “rules”). These XML files will be located in the path indicated by the `siem_events_rules`

parameter of the server configuration, by default in:

```
/usr/share/pandora_server/util/siem/rules
```

The XML files of Wazuh® decoders and rules are supported by Pandora FMS SIEM monitoring.

Agent configuration

Most of the log collection is done through Pandora FMS EndPoints. These agents, both in GNU/Linux® and MS Windows® systems, have specific types of modules to perform this task.

To take full advantage of all SIEM PFMS features, it is recommended to have at least version 783 installed in the EndPoints.

SIEM monitoring relies heavily on the type of log collected, so it will be necessary to specify `module_source_type` in log collection modules to indicate that type.

The type is used by decoders and rules, so the active decoders and rules must be checked to know the type to be indicated in each log.

The most commonly used types of logs are:

- syslog
- ids
- web-log
- squid
- windows
- host-information
- ossec

Linux® agents

Log collection on Linux systems is mainly done by reading log files. This can be achieved by using module configurations with this minimal structure:

```
module_begin
module_name <program_name>
module_type log
```

```
module_regexp <path_to_log_file>
module_pattern <capture_regexp>
module_source_type <log_type>
module_end
```

For example, to collect all access log entries from an Apache server:

```
module_begin
module_name apache
module_type log
module_regexp /var/log/httpd/access_log
module_pattern .*
module_source_type web-log
module_end
```

Entries collected from a log like the above would be normalized by, among others:

- decoders such as `web-accesslog`,
- `web-accesslog-ip` or,
- `web-accesslog-domain`.

The decoded logs of such a log could generate events such as (among others):

- `Common web attack`,
- `XSS (Cross Site Scripting) attempt` or,
- `SQL injection attempt`.

MS Windows® agents

Log collection in MS Windows® is mainly done by monitoring system events, although it can also be done by reading log files as in Linux systems.

Using the Windows event system, these logs may be collected by using module configurations with one of these two minimum configurations.

If the events are either Application, System or Security:

```
module_begin
module_name <module_name>
module_type log
module_logchannel
module_source <Application|System|Security>
module_source_type <log_type>
module_end
```

Or if they are events belonging to a different channel:

```
```

```
module_begin
module_name <module_name>
module_type log
module_logchannel
module_source <log_channel_path>
module_source_type <log_type>
module_end
```

For example, to collect all Security and Windows Defender event entries:

```
module_begin
module_name Windows_LogEvents_System
module_type log
module_logchannel
module_source Security
module_source_type ossec
module_end

module_begin
module_name Windows_LogchannelEvents_WindowsDefender
module_type log
module_logchannel
module_source Microsoft-Windows-Windows Defender/Operational
module_source_type ossec
module_end
```

Inputs collected from events such as the above would be normalized by decoders as `windows_eventchannel`.

The decoded logs of events such as the above could generate events such as, among others:

- `Windows error event`,
- `Short-time multiple Windows Defender warning events` or,
- `Multiple Windows Defender error events`.

Using log file monitoring, the configuration is identical to that of Linux systems. A minimal configuration like this is required:

```
module_begin
module_name <program_name>
module_type log
module_regexp <path_to_log_file>
module_pattern <capture_regexp>
module_source_type <log_type>
module_end
```

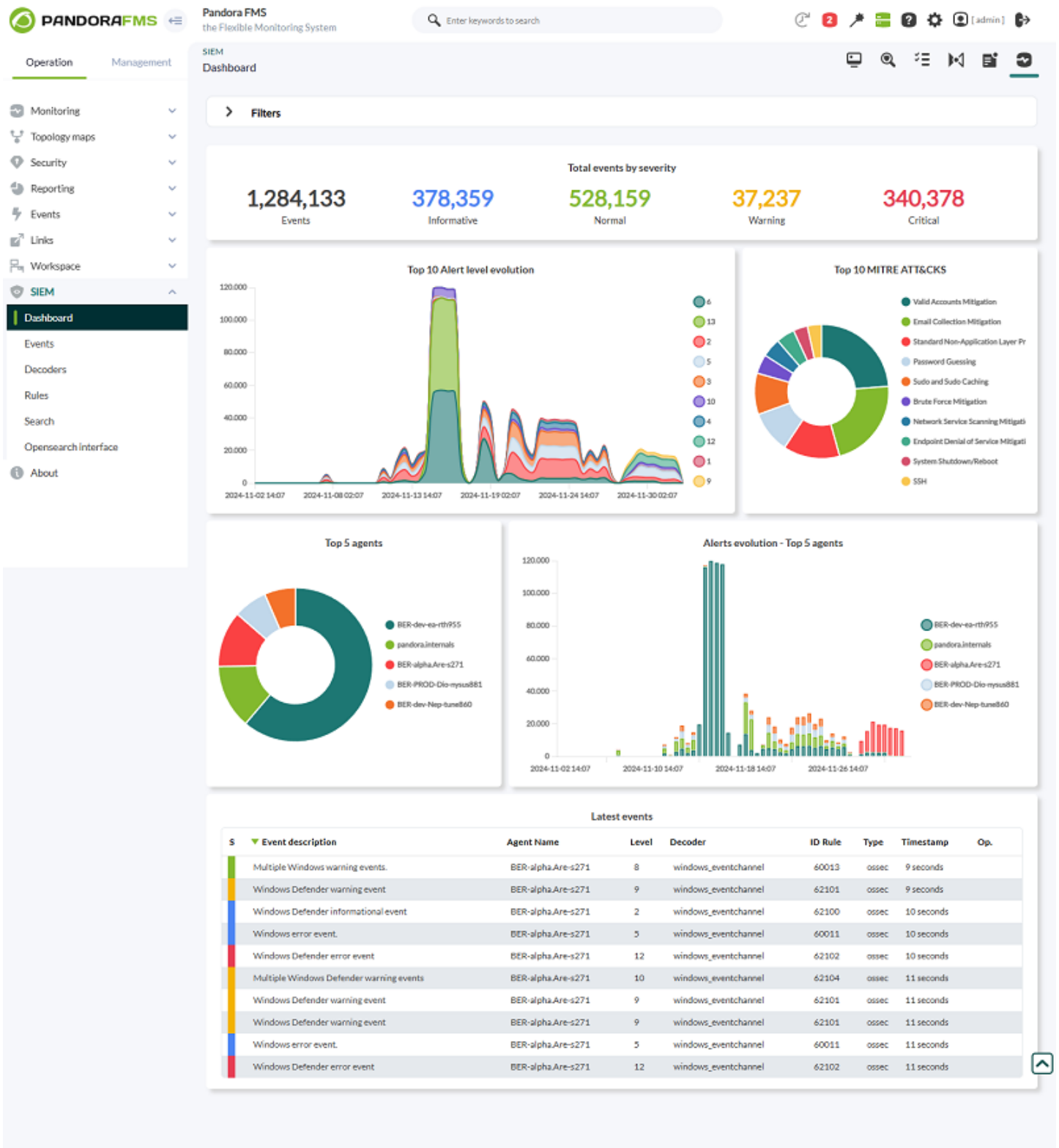
For example, to collect all log entries of an X server:

```
module_begin
module_name xserver
```

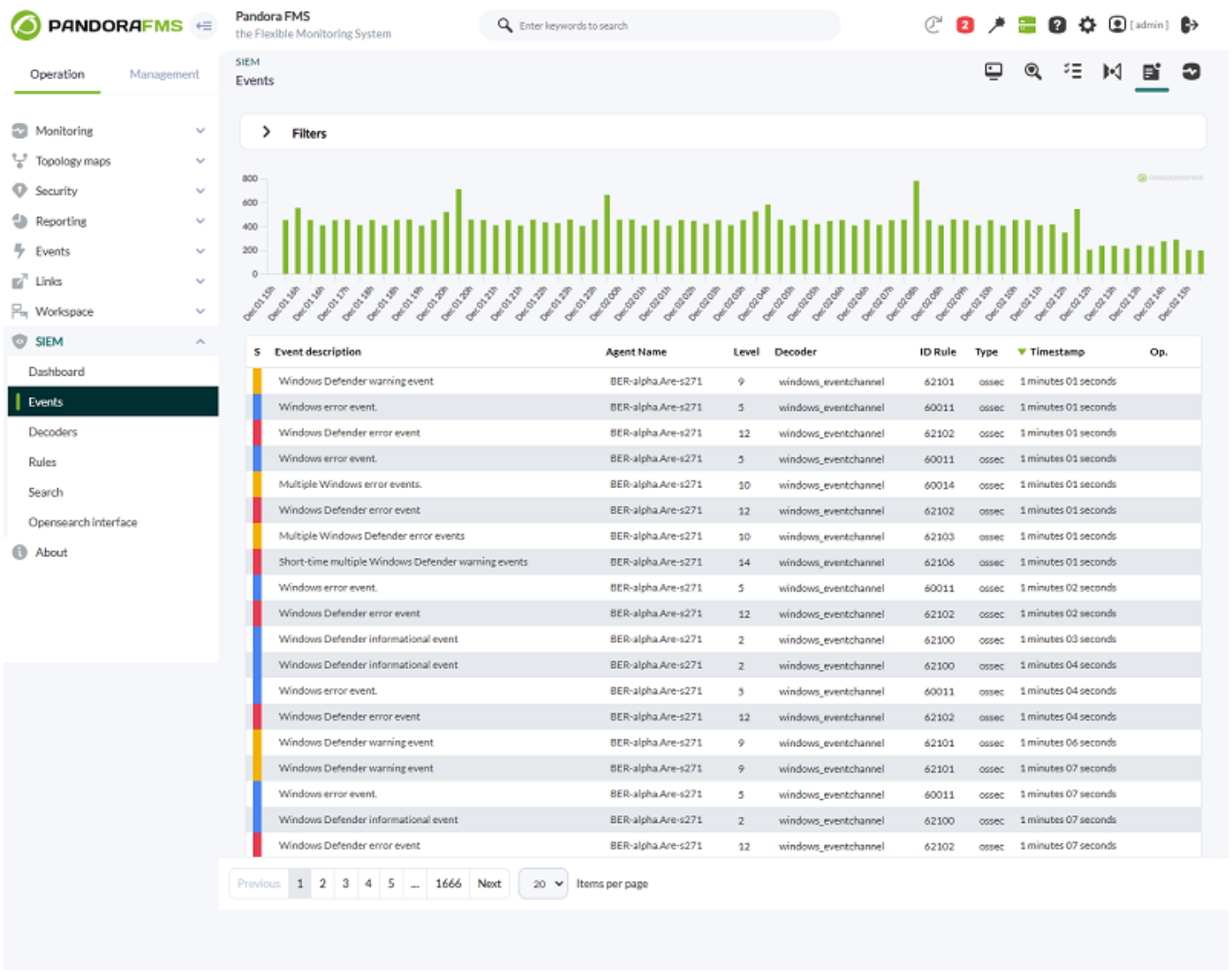
```
module_type log
module_regexp C:\server\logs\xserver.log
module_pattern .*
module_source_type xserver
module_end
```

SIEM events

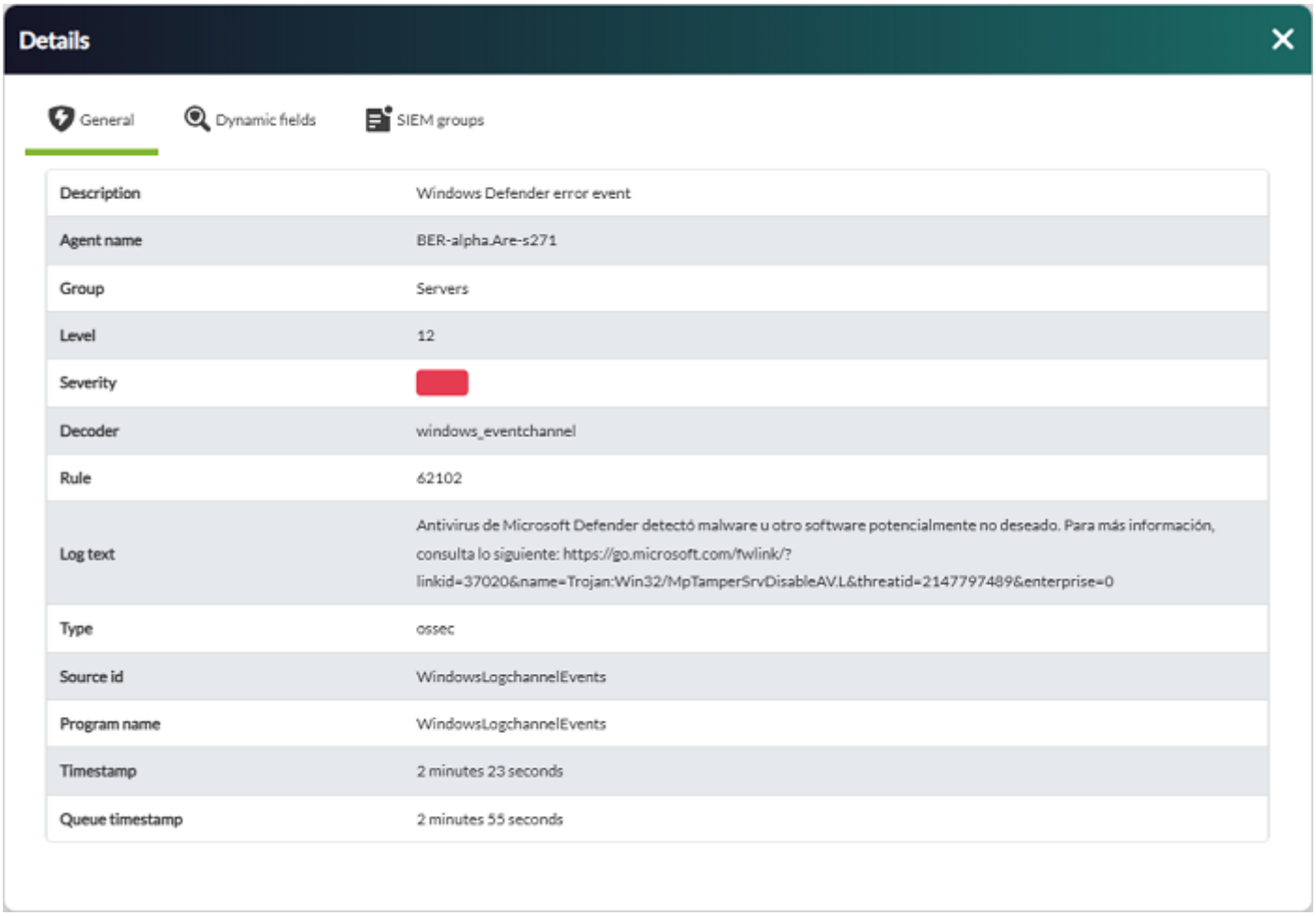
With SIEM monitoring enabled and configured, a preview of the monitoring status may be accessed in the Operation → SIEM → Dashboard menu.



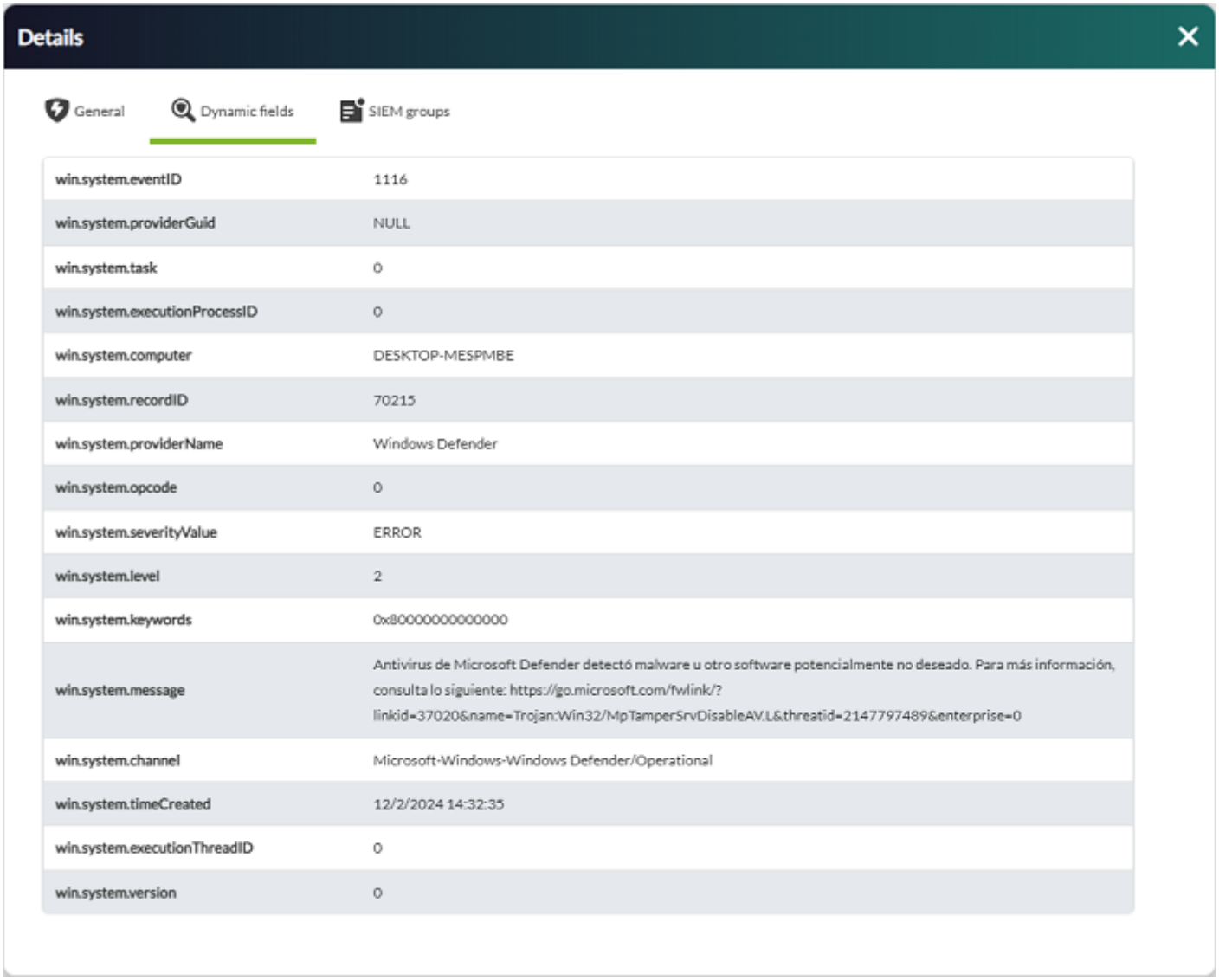
Generated SIEM events can be fuly displayed in menu Operation → SIEM → Events.



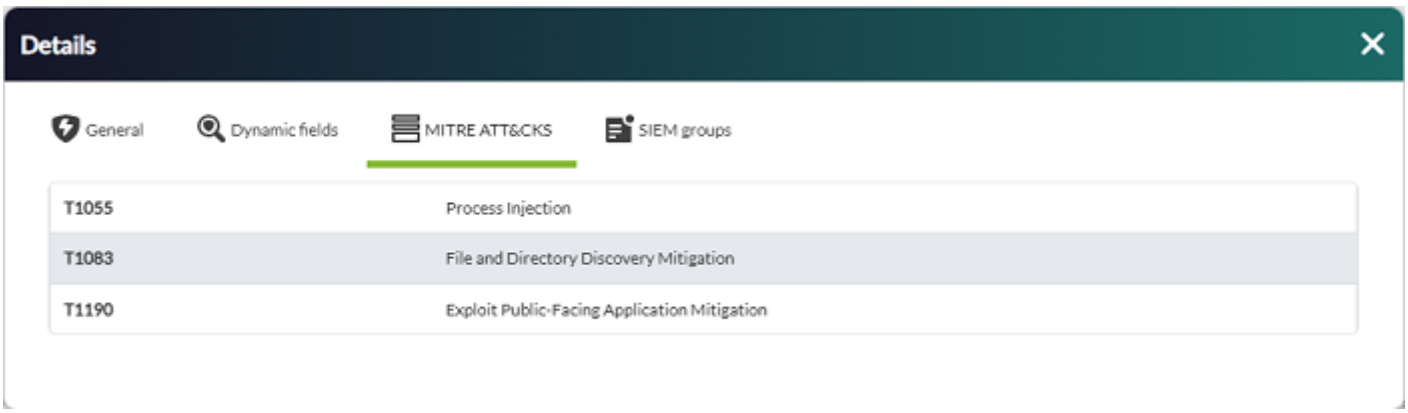
Each SIEM event will have a window with the event details, showing information of the normalized log and the rule that generated the event.

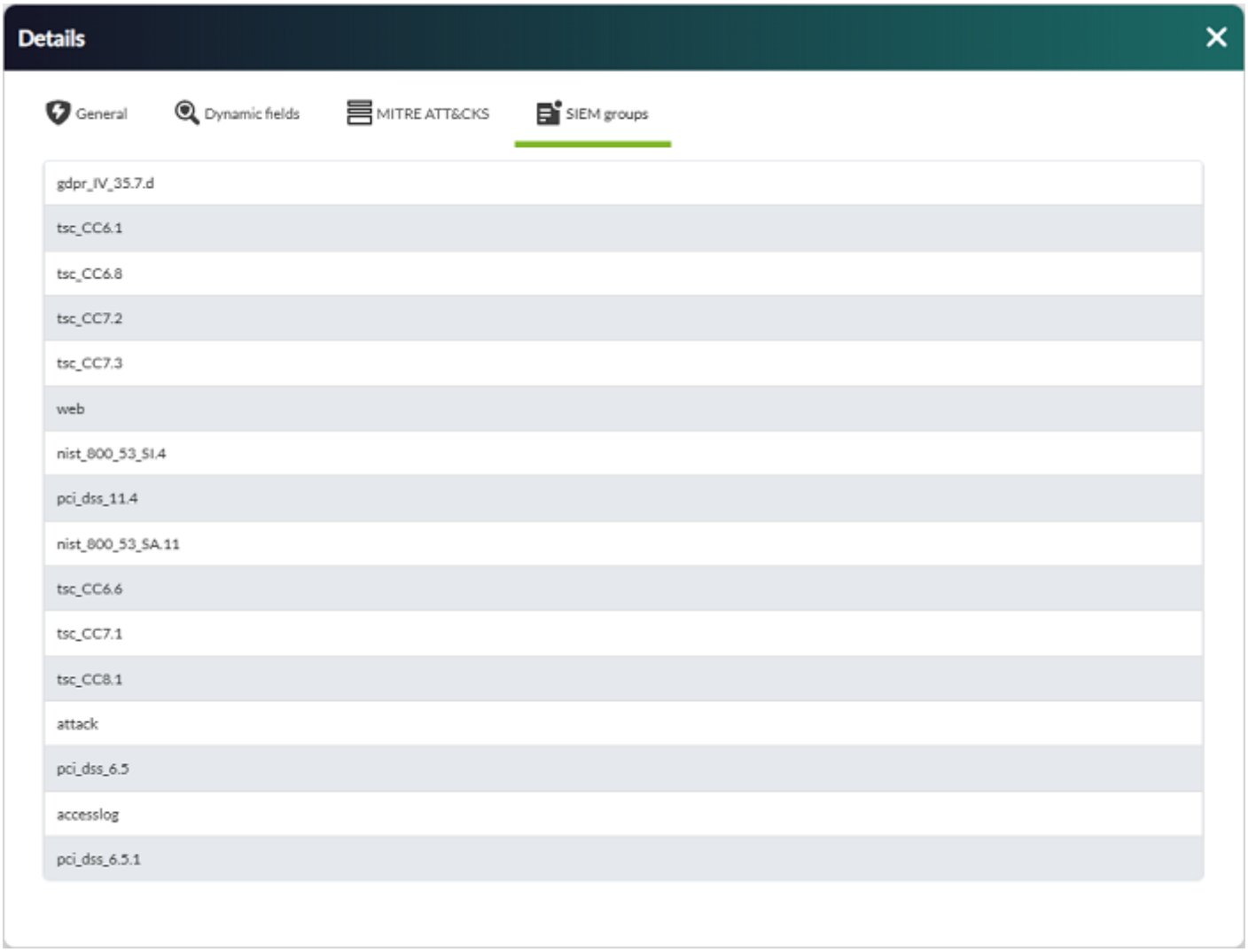


Depending on the decoders that normalized the log, the event will have Dynamic fields tab with useful log information.



Depending on the rule that generated the event, there will be MITRE ATT&CKS® and SIEM groups tabs with useful information on the impact of the event.





The information shown both in the general Dashboard and in the events table may be included in Pandora FMS Dashboards by means of the included SIEM widgets.

Decoders

Pandora FMS includes a series of decoders by default for SIEM monitoring, but any administrator may include their own.

Management

To include new decoders, first add or edit an XML file in the path indicated to Pandora FMS server in its configuration parameter `siem_decoders`.

Decoders are loaded into the environment via XML files that Pandora FMS master server reads at each service startup.

Once decoders are loaded, their configuration is stored in the database, and each `siemserver` processes them for entries obtained through log collection.

From Pandora FMS Console, it will be possible to view the loaded decoders with their complete configuration from menu Operation → SIEM → Decoders.

It is also possible to disable decoders from this view, so that `siemserver` does not take them into account when normalizing log entries.

All decoders are fully loaded at each restart. This means that decoders that could not be read from XML files will not be available (even if they were at some point), and decoders that were disabled from the Console will be re-enabled again (if they exist).

Syntax

Decoders are configured and loaded in Pandora FMS with XML files. These files may have the following valid syntax:

```
<var name="VarName">VarValue</var>

<decoder name="DecoderName" discard="yes|no">
    <parent>DecoderName</parent>
    <program_name>REGEXP</program_name>
    <type>EventType</type>
    <prematch type="pcre2">REGEXP</prematch>
    <prematch offset="after_parent">REGEXP</prematch>
    <prematch offset="after_prematch">REGEXP</prematch>
    <regex type="pcre2">REGEXP</regex>
    <regex offset="after_parent">REGEXP</regex>
    <regex offset="after_regex">REGEXP</regex>
    <order>Field1, Field2.Sub1, Field2.Sub2</order>
    <json_null_field>string|discard</json_null_field>
</decoder>
```

var

↑Full syntax

Used to indicate variables with their values, and use them later in XML (`$VarName`).

```
<var name="VarName">VarValue</var>
```

decoder

↑Full syntax

It is the decoder information, with its name. The XML file itself may contain several.

- `name`: It is the name of the decoder. Several decoders may have the same name, and they are all evaluated.
- `discard`: With a `yes` or `no` value, it allows to discard logs from the decoder evaluation if they match this one. Decoders with `discard="yes"` are evaluated before the rest (as long as they do not have a parent decoder).

```
<decoder name="DecoderName" discard="yes|no">
  …
  …
  …
</decoder>
```

parent

↑Full syntax

It specifies the name of the parent decoder to generate a hierarchical structure.

```
    <parent>DecoderName</parent>
```

program_name

↑Full syntax

The name of the program to be found in the log headers or in the `source_id` of the log.

- `type`: It allows you to specify the type of regular expression. If not specified, OS Regex is used.

```
    <program_name>REGEXP</program_name>
```

As the case may be, it is specified for PCRE2:

```
    <program_name type="pcre2">REGEXP</program_name>
```

type

↑Full syntax

The type of logs for which the decoder will be compared. It must match the log `type`.

```
    <type>EventType</type>
```

prematch

↑Full syntax

If the contents of the log match this, it generates a normalized log.

- `type`: It allows you to specify the type of regular expression. If not specified, OS Regex is used.
- See offset.

```
    <prematch type="pcre2">REGEXP</prematch>
    <prematch offset="after_parent">REGEXP</prematch>
    <prematch offset="after_prematch">REGEXP</prematch>
```

regex

↑Full syntax

Groups captured with this regular expression are normalized information for the log.

- `type`: It alows you to specify the type of regular expression. If not specified, OS Regex is used.
- See offset.

```
    <regex type="pcre2">REGEXP</regex>
    <regex offset="after_parent">REGEXP</regex>
    <regex offset="after_regex">REGEXP</regex>
```

order

↑Full syntax

Values captured by regex are stored in these field names, sorted by capture groups.

```
    <order>Field1, Field2.Sub1, Field2.Sub2</order>
```

json_null_field

↑Full syntax

Null values of the capture groups will be stored as empty strings or discarded.

```
    <json_null_field>string|discard</json_null_field>
```

offset

↑Full syntax

It is used to indicate the order in which checks are performed and to discard from the log content for the following checks the text that matches the regular expression: `after_parent`, `after_regex`, `after_prematch`. For example:

```
<decoder name="my_decoder">
  <prematch type="pcre2">^\d\d\d\d/\d\d/\d\d \d\d:\d\d:\d\d </prematch>
  <regex type="pcre2" offset="after_prematch">(\w):(\d+)</regex>
  <order>srcip,srcport</order>
</decoder>
```

It will check that the text in the log matches the regular expression `^\d\d\d\d/\d\d/\d\d \d\d:\d\d:\d\d` , will discard that content of the text and when evaluating the `regex`, it will capture accordingly. In the example, it would remove a date from the text when evaluating to simplify the capture groups.

Rules

Pandora FMS includes a set of default rules for SIEM monitoring, but any administrator may include their own.

Rule management

To include new rules, first add or edit an XML file in the path indicated to Pandora FMS server in its configuration parameter `siem_rules`.

Rules are loaded into the environment by means of XML files that Pandora FMS master server reads at each service start.

Once rules are loaded, their configuration is stored in the database, and each `siemevents` server processes them for normalized entries in SIEM monitoring.

From Pandora FMS console, it will be possible to see loaded rules with their full configuration from menu Operation → SIEM → Rules.

It is also possible to disable rules from this view, so that `siemevents` does not take them into account when processing normalized logs.

All rules are fully loaded at each restart. This means that the rules that could not be read from the XML files will not be available (even if they were at some point). Unlike decoders, rules have an ID that allows to keep them disabled at each reload.

Rules that could not be read from XML files will be marked in the Console as “not active” and will not be taken into account when generating SIEM events. Rules that were manually disabled by an administrator will not be taken into account either. Therefore, for rules to be evaluated, they must be active and enabled.

Once rules are active and enabled, if one rule needs to depend on the evaluation and result of another, the first rule must have a lower numerical identifier than the second (i.e., they are executed in ascending numerical order, see corresponding case study).

Sorting

Rules are classified in several levels, ranging from the lowest (`0`) to the highest (`15`). The following table describes each level, providing information on the severity of each event generated by SIEM monitoring.

Level	Title	Description
0	Ignored.	No action was taken. Used to avoid false positives. These rules are scanned before all other rules, including events with no security relevance, and do not appear in the security events panel.
2	Notification of low system priority.	System notifications or status messages. They have no security relevance and do not appear in the security event panel.
3	Successful/authorized events.	These include successful login attempts, events allowed by the firewall, and so on.
4	Low system priority error.	Errors related to bad configurations or unused devices/applications. These have no security relevance and are usually caused by default installations or software testing.
5	User-generated error.	These include forgotten passwords, denied actions, and so on. By themselves, they have no security relevance.

Level	Title	Description
6	Low relevance attack.	These indicate a worm or virus that has no effect on the system (such as code red for Apache servers, etc.). This also includes frequent Intrusion Detection System (IDS) events and frequent errors.
7	Coincidence of “bad words”.	These include words such as “bad”, “error”, etc. Most of these events are not classified and may have some relevance from a security point of view.
8	First time seen.	Include events seen for the first time. The first time an IDS event is triggered or the first time a user logs in. It also includes security-relevant actions, such as the activation of a sniffer or similar activities.
9	Invalid source error.	It includes attempts to log in as an unknown user or from an invalid source. It may have security relevance (especially if repeated). This also includes errors related to the “admin” account (root).
10	Multiple user-generated errors.	These include multiple incorrect passwords, multiple failed logins, etc. These may indicate an attack or just signal that a user forgot their credentials.
11	Integrity check warning.	These include messages about binary modification or the presence of rootkits (by Rootcheck). These may indicate a successful attack. They also include IDS events that will be ignored (large number of repetitions).
12	Event of high importance.	These include error or warning messages from the system, kernel, and so on. These may indicate an attack against a specific application.
13	Unusual error (high importance).	It matches a common attack pattern most of the time.
14	Security event of high importance.	It is most often triggered by correlation and indicates an attack.
15	Severe attack.	There is no possibility of false positives. Immediate attention is necessary.

Based on these levels, events will have a specific severity that will be displayed on the console:

- Informational: Levels from 0 to 6.
- Normal: Levels from 7 to 8.
- Warning: Levels from 9 to 11.
- Critical: Levels from 12 to 15.

Syntax

Detailed syntax elements

```
<var name="VarName">VarValue</var>

<group name="GROUP1,GROUP2,">
    <rule id="N" level="N" frequency="N" timeframe="N" ignore="N"
overwrite="yes|no">
        <if_matched_sid>N</if_matched_sid>
        <if_matched_group>GROUP</if_matched_group>
```

```
        <same_id />
        <different_id />
        <same_field>Field1</same_field>
        <same_field>Field2.Sub1</same_field>
        <different_field>Field1</different_field>
        <different_field>Field2.Sub1</different_field>
        <description>TEXT</description>
        <match type="pcre2">RREGEXP</match>
        <match negate="yes|no">RREGEXP</match>
        <regex type="pcre2">RREGEXP</regex>
        <regex negate="yes|no">RREGEXP</regex>
        <decoded_as>DecoderName</decoded_as>
        <category>EventType</category>
        <field name="Field1">REGEXP</field>
        <field name="Field2.Sub1" negate="yes|no">REGEXP</field>
        <program_name negate="yes|no">REGEXP</program_name>
        <time>TIME-RANGE</time>
        <weekday>DAYS</weekday>
        <if_sid>PARENT1, PARENT2</if_sid>
        <if_group>GROUP</if_group>
        <if_level>N</if_level>
        <info type="text|link|cve">TEXT|LINK|CVE</info>
        <group>GROUP1,GROUP2,</group>
        <mitre>
            <id>MITRE_ID</id>
            <id>MITRE_ID</id>
        </mitre>
    </rule>
</group>
```

See also Case study.

var

↑Complete syntax.

```
<var name="VarName">VarValue</var>
```

Used to indicate variables with their values, and use them later in XML (`$VarName`).

group

↑Complete syntax.

```
<group name="GROUP1,GROUP2,">
…
```

```
</group>
```

It allows rule grouping. It is also used for conditions of the same rules.

rule

↑Complete syntax.

```
    <rule id="N" level="N" frequency="N" timeframe="N" ignore="N"
overwrite="yes|no">
    …
    </rule>
```

It is the information in the rule:

1. `id`: Rule identifier. It must be unique (unless another rule is overwritten).
2. `level`: Level of the event when generated (`0` to `15`). Rules with level `0` do not generate an event.
3. `frequency`: Times that a concurrence must take place to generate an event. Rule frequency is checked for logs of the agent itself, not for any log.
4. `timeframe`: Time window in which concurrences must be given (seconds).
5. `ignore`: This rule restarts your `frequency` counters after these seconds.
6. `overwrite`: With `yes` or `no` values, it allows to overwrite the configuration of a rule with the same ID. If together with `overwrite="yes"` the rule has `level="0"`, it is evaluated before any other and in case of a log match, it discards the evaluation of the rest of the rules for that log.

if_matched_sid

↑Complete syntax.

```
        <if_matched_sid>N</if_matched_sid>
```

The rule is satisfied if another rule with the indicated ID triggered an alert the times indicated by `frequency` within the `timeframe` time.

if_matched_group

↑Complete syntax.

```
        <if_matched_group>GROUP</if_matched_group>
```

Like if_matched_sid but for groups.

same_id

↑Complete syntax.

```
        <same_id />
```

Concurrences with the same ID must be given.

different_id

↑Complete syntax.

```
        <different_id />
```

Concurrences with different IDs must be given.

same_field

↑Complete syntax.

```
        <same_field>Field1</same_field>
        <same_field>Field2.Sub1</same_field>
```

Concurrences with the same values in the field must take place.

different_field

↑Complete syntax.

```
        <different_field>Field1</different_field>
        <different_field>Field2.Sub1</different_field>
```

Concurrences with different values in the field must take place.

description

↑Complete syntax.

```
        <description>TEXT</description>
```

The text for the event to be generated.[1)]

match

↑Complete syntax.

```
        <match type="pcre2">RREGEXP</match>
        <match negate="yes|no">RREGEXP</match>
```

If the contents of the log match this, it generates a SIEM event.

1. `type`: It allows you to specify the type of regular expression. If not specified, OS Regex is used.
2. `negate`: With a `yes` value you may deny the match.

regex

↑Complete syntax.

```
        <regex type="pcre2">RREGEXP</regex>
        <regex negate="yes|no">RREGEXP</regex>
```

Same as "match".

1. `type`: It allows you to specify the type of regular expression. If not specified, OS Regex is used.
2. `negate`: With a `yes` value you may deny the regular expression.

decoded_as

↑Complete syntax.

```
        <decoded_as>DecoderName</decoded_as>
```

The rule is satisfied if the log was decoded by the indicated decoder.

category

↑Complete syntax.

```
        <category>EventType</category>
```

The rule is satisfied if the type of the decoder matches.

field

↑Complete syntax.

```
        <field name="Field1">REGEXP</field>
        <field name="Field2.Sub1" negate="yes|no">REGEXP</field>
```

If the indicated field matches the value, it complies with the rule.

1. `negate`: With a `yes` value, it allows to deny the match with the field.

program_name

↑Complete syntax.

```
        <program_name negate="yes|no">REGEXP</program_name>
```

If the source of the log matches, the rule is met.

1. `negate`: With a `yes` value, it allows to deny matches with the source log.

time

↑Complete syntax.

```
        <time>TIME-RANGE</time>
```

If the event is generated in the indicated time range, the rule matches.

weekday

↑Complete syntax.

```
        <weekday>DAYS</weekday>
```

If the event is generated the days the rule complies with (`monday - sunday`, `weekdays`, `weekends`).

if_sid

↑Complete syntax.

```
        <if_sid>PARENT1, PARENT2</if_sid>
```

The rule is met if any of the parent rules are met.

if_group

↑Complete syntax.

```
        <if_group>GROUP</if_group>
```

The rule is met if the log meets any other rule in the specified group.

if_level

↑Complete syntax.

```
        <if_level>N</if_level>
```

The rule is met if another rule with the same level has been met.

info

↑Complete syntax.

```
        <info type="text|link|cve">TEXT|LINK|CVE</info>
```

Additional information for the generated event.[2)]

group

↑Complete syntax.

```
        <group>GROUP1,GROUP2,</group>
```

List of rule groups.

mitre

↑Complete syntax.

```
        <mitre>
            <id>MITRE_ID</id>
            <id>MITRE_ID</id>
        </mitre>
```

List of the MITRE IDs of the rule.

Case study

The following code describes a rule for PHP-related SELinux logs. This rule creates events for any attempt to connect to ports other than the usual ones on a web server.

```
<rule id="100201" level="6">
        <decoded_as>setroubleshoot_program</decoded_as>
        <match>/usr/sbin/php-fpm</match>
        <field name="object_target"
type="pcre2">^(?!.*\b(9200|3306|80|443)$).*$</field>
        <field name="object_target" type="pcre2">^(?!.*(directory|file)
(conf|data_in|cron.lock)).*$</field>
        <description>SELinux prevented /usr/sbin/php-fpm execution on: $(action)
access on the $(object_target)</description>
        <mitre>
            <id>T1071.002</id>
        </mitre>
        <group>exec,threat,PHP</group>
</rule>
```

Thus, connections to ports 9200, 3306, 80 and 443 will not create events. Neither will events of type *directory* or *file* .

General decoder decoder used for SELinux:

```
<decoder name="setroubleshoot">
    <program_name>^setroubleshoot$</program_name>
</decoder>

<decoder name="setroubleshoot_program">
    <parent>setroubleshoot</parent>
    <prematch type="pcre2">(SELinux is preventing|SELinux está negando
a)</prematch>
    <regex type="pcre2">(\S+) (?:from|de) (\S+) (?:access on the|el acceso a)
(.+?)\. (?:For complete SELinux messages run: sealert -l|Si quiere los mensajes
de SELinux completos, ejecute sealert -l) (\S+)$</regex>
    <order>binary,action,object_target,sealert_id</order>
</decoder>

<decoder name="setroubleshoot_program">
    <parent>setroubleshoot</parent>
    <prematch type="pcre2">failed to retrieve rpm info for path</prematch>
    <regex type="pcre2">failed to retrieve rpm info for path (\S+)</regex>
    <order>path</order>
</decoder>
```

Rule evaluation order

The following case study with a negative result illustrates when a rule “queries” other rules to check whether they found a match and then evaluates the match itself to generate an event.

There is a series of rules used to detect failed logins on a hardware firewall, in order to then create the corresponding alerts:

```
<rule id="81641" level="1">
    <decoded_as>fortigate-firewall-v6</decoded_as>
    <description>Fortigate v6 messages grouped.</description>
</rule>
```

- Rule 81641 matches decoded log to start retrieving data.

```
<rule id="81603" level="0">
    <if_sid>81600,81601,81602,81641</if_sid>
    <description>Fortigate messages grouped.</description>
</rule>
```

- Rule ID 81603 in turn depends on several rules, including the previous 81641. Since the latter 81641 is higher than 81603, 81603 is not executed.

```
<rule id="81614" level="4">
```

```
    <if_sid>81603</if_sid>
    <match>ssl-login-fail</match>
    <description>Fortigate: SSL VPN user failed login attempt.</description>
    <group>authentication_failed,
gdpr_IV_32.2,
gdpr_IV_35.7.d,
gpg13_7.1,
hipaa_164.312.b,
invalid_login,
nist_800_53_AC.7,
nist_800_53_AU.14,
pci_dss_10.2.4,
pci_dss_10.2.5,
tsc_CC6.1,
tsc_CC6.8,
tsc_CC7.2,
tsc_CC7.3,</group>
</rule>
```

- A rule with ID 100700 checks for a match in a rule with ID 81641, which in turn depends on rule ID 81603 (81614 is higher than 81603, so it is executed).

The problem in this case study is that 81614 does not return a result to 100700 because 81603 was not executed due to the numerical order by ID.

Regular expressions

Regular expressions are sequences of characters that define a pattern.

There are two types of regular expressions valid for decoders and rules: OS Regex and PCRE2.

OS Regex

They are simple regular expressions based on a library made in C language. It is designed to be simple and at the same time support the most common regular expressions.

Acceptable expressions

Expression	Valid characters
\w	A-Z, a-z, 0-9, '-', '@', '_'
\d	0-9
\s	Spaces “ “

Expression	Valid characters
\t	Tabulation
\p	()*+,-.:;<=>?[]!"'#$%&|{}
\W	Anything other than \w
\D	Anything other than \d
\S	Anything other than \s
\.	Anything other

Modifiers

Expression	Behavior
+	To match one or more times
*	To match zero or more times

Special characters

Expression	Behavior
^	To specify the beginning of the text
$	To specify the end of the text
|	To create a logical pattern “or” among multiple patterns

Escaping characters

To use the following characters you must escape them with \:

$	(	)	\	|	<
\$	\(	\)	\	|	\<

Limitations

- The * and + modifiers may only be applied to backslash expressions, not to single characters (e.g. \d+ is supported, 0+ is not supported).
- Alternation cannot be used in a group, e.g. (foo|bar) is not allowed.
- Complex backtracking is not supported, e.g., \p*\d*\s*\w*:, it does not match only colon because \p* consumes the colon.
- . matches a literal period, while \. matches any character.
- \s matches only an ASCII space (32), not other blanks such as tabs.
- There is no syntax to match a caret, asterisk or more literal (although \p will match an asterisk or more, along with some other characters).

PCRE2

Perl Compatible Regular Expression (PCRE2) provides features such as recursive patterns, look-ahead and look-back assertions, non-capture groups, non-voracious quantifiers, extended syntax for characters and character classes, among others.

For more details, see the PCRE2 syntax documentation.

Acceptable expressions

Expression	Valid characters
`.`	Any character except new line
`\d`	Any decimal digit, equal to `[0-9]`
`\D`	Any character other than a decimal digit, equal to `[^0-9]`
`\h`	Any horizontal white space character
`\H`	Any character that is not a horizontal blank space
`\s`	Any white space character, equal to `[\t\r\n\f]`
`\S`	Any character that is not a blank space, equal to `[^\t\r\n\f]`
`\w`	Any “word” character
`\W`	Any “non-word” character

Modifiers

Expression	Behavior
`?`	0 or 1, greedy
`?+`	0 or 1, possessive
`??`	0 or 1, lazy
`*`	0 or more, greedy
`*+`	0 or more, possessive
`*?`	0 or more, lazy
`+`	1 or more, greedy
`++`	1 or more, possessive
`+?`	1 or more, lazy
`{n}`	Exactly n
`{n,m}`	At least n, no more than m, greedy
`{n,m}+`	At least n, no more than m, possessive
`{n,m}?`	At least n, no more than m, lazy
`{n,}`	n or more, greedy
`{n,}+`	n or more, possessive
`{n,}?`	n or more, lazy

Escape characters

Expression	Behavior
`\f`	Next page (hexadecimal `0C`)
`\n`	New line (hexadecimal `0A`)
`\r`	Carriage return (hexadecimal `0D`)
`\t`	Tabulation (hexadecimal `09`)
`\0dd`	Character with octal code `0dd`
`\o{ddd..}`	Character with octal code `ddd..`
`\xhh`	Character with hexadecimal code `hh`
`v\x{hhh…}`	Character with hexadecimal code `hh..`

SIEM alerts

See the topic SIEM alert system.

SIEM reports

See the topic SIEM events reports.

Back to Pandora FMS documentation index

1) , 2)

Variables can be used in the description and in `info` with the values of the custom fields, e.g.: `$(Field1)`, `$(Field2.Sub1)`, `$(Field2.Sub2)`.