



Server Plugins Development



<https://pandorafms.com/manual/!775/>

Permanent link:

https://pandorafms.com/manual/!775/en/documentation/pandorafms/technical_reference/05_anexo_server_plugins_development

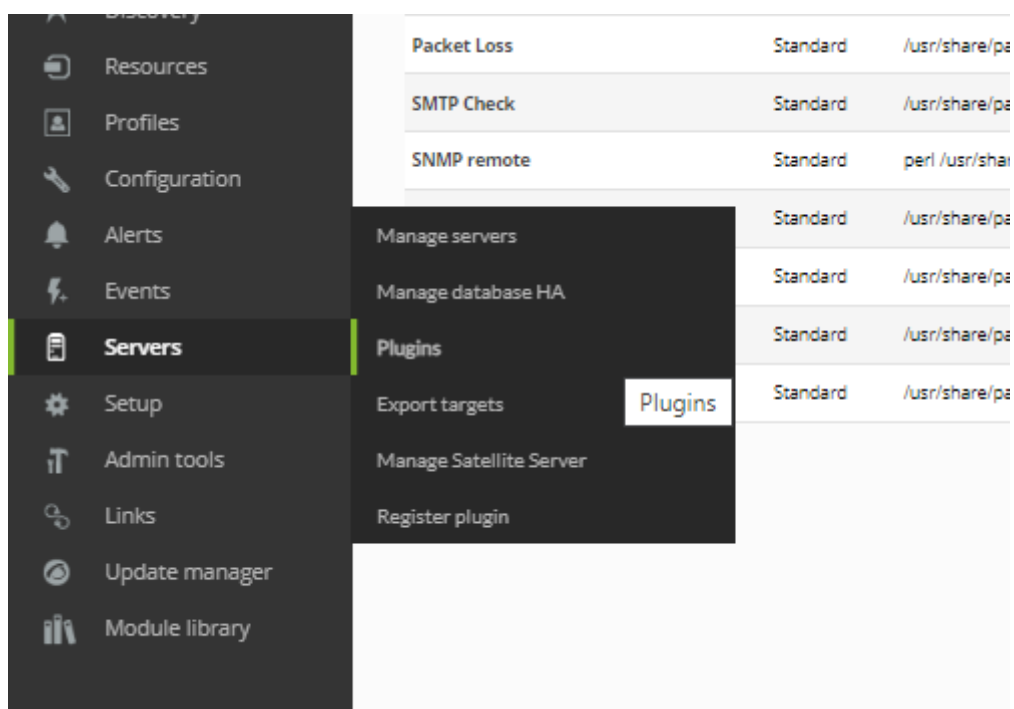
2022/03/18 21:03



Server Plugins Development

Servers Plugin Development

Basic Features of the Server Plugin



The plugin server is executed by Pandora FMS Server Plugin, so it should have very specific features:

- Every execution of the plugin should return a single value. This should be like this, because the Server Plugin makes an execution by each module type plugin.
- It should have access to the resources to monitor in a remote way.
- It is possible to use any programming language that supports the operative system where the Pandora server is installed
- All dependencies or necessary software to execute the plugin should be available or be installed in the same machine that executes the Pandora server.

You may find more information about monitoring with remote server plugins [here](#). There you will find simple examples and how the modules and agents they contain work. In this article server plugin creation is analyzed in depth.

Example of Server Plugin Development

Next we are going to describe a possible example of server plugin for Pandora FMS.

The following plugin returns the sum of the entry and exit traffic of a device interface. Data are got through SNMP.

The plugin code would be this:

```
#!/usr/bin/perl -w

use strict;
use warnings;

sub get_param($) {
    my $param = shift;
    my $value = undef;

    $param = "-" . $param;

    for(my $i=0; $i< $#ARGV; $i++) {

        if ($ARGV[$i] eq $param) {
            $value = $ARGV[$i+1];
            last;
        }

    }
    return $value;
}

sub usage () {
    print "iface_bandwidth.pl version v1r1\n";
    print "\nusage: $0 -ip <device_ip> -community <community> -ifname
<iface_name>\n";
    print "\nIMPORTANT: This plugin uses SNMP v1\n\n";
}

#Global variables
my $ip = get_param("ip");
my $community = get_param("community");
my $ifname = get_param("ifname");

if (!defined($ip) ||
    !defined($community) ||
    !defined($ifname) ) {
    usage();
    exit;
}

#Browse interface name
my $res = `snmpwalk -c $community -v1 $ip .1.3.6.1.2.1.2.2.1.2 -On`;

my $suffix = undef;
```

```

my @iface_list = split(/\n/, $res);

foreach my $line (@iface_list) {

    #Parse snmpwalk line
    if ($line =~ m/^( [\d|\.] + ) = STRING: (.*)$/) {
        my $aux = $1;

        #Chec if this is the interface requested
        if ($2 eq $ifname) {

            my @suffix_array = split(/\./, $aux);

            #Get last number of OID
            $suffix = $suffix_array[$#suffix_array];
        }
    }
}

#Check if iface name was found
if (defined($suffix)) {
    #Get octets stats
    my $inoctets = `snmpget $ip -c $community -v1
.1.3.6.1.2.1.2.2.1.10.$suffix -OUevqt`;
    my $outoctets = `snmpget $ip -c $community -v1
.1.3.6.1.2.1.2.2.1.16.$suffix -OUevqt`;

    print $inoctets+$outoctets;
}

```

An important part of the code is the usage function:

```

sub usage () {
    print "iface_bandwidth.pl version v1r1\n";
    print "\nusage: $0 -ip <device_ip> -community <community> -ifname
<iface_name>\n";
    print "\nIMPORTANT: This plugin uses SNMP v1\n\n";
}

```

In this function, it describes the version and how to use the plugin. It is very important and always should be shown when executing the plugin without any type of parameter or also with an option type `-h` or `-help`.

Concerning to the value that the plugin has returned, this is printed in the standard output of the second to last line with the following instruction:

```
print $inoctets+$outoctets;
```

As you can see the value returned by the plugin is a single data, that after the Pandora Server

Plugin will add as data to the associated module.

To could execute this server plugin, you should install the commands snmpwalk and snmpget in the machine that the Pandora FMS server executes.



Plugin manual registration

PLUGIN REGISTRATION

General

Name

Plugin type

Max. timeout  

Description

- Plugin type

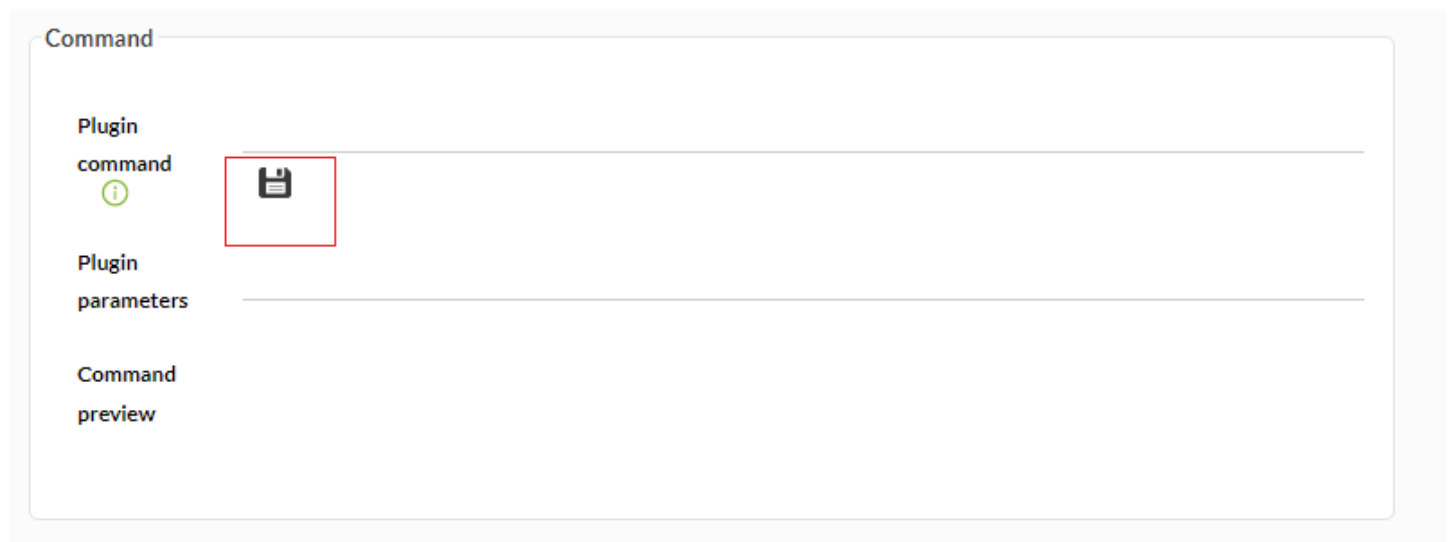
There are two kinds of plugins, the standard ones and the kind Nagios. The standard plugins are scripts that execute actions and accept parameters. The Nagios plugins are, as their name shows, Nagios plugins that could be being used in Thor. The difference is mainly on that the Nagios plugins return an error level to show if the test has been successful or not.

If you want to use a plugin kind Nagios and you want to get a data, not an state (Good/Bad), then you can use a plugin kind Nagios is the "Standard" mode.


- Max. timeout

It is the time of expiration of the plugin. If you do not receive a response in this time, you should select the module as unknown, and its value will be not updated. It is a very important factor when implementing monitoring with plugins, so if the time it takes at executing the plugin is bigger than this number, we never could obtain values with it. This value should always be bigger than the

time it takes usually to return a value the script/executable that is used as plugin. In there is nothing said, then you should used the value that in the configuration is named `plugin_timeout`.



Command

Plugin command 

Plugin parameters

Command preview

- Plug-in command


It is the path where the plugin command is. In a default way, if the installation has been an standard one, there will be in the directory `/usr/share/pandora_server/util/plugin/` . Though it could be any path of the system. For this case, write `/usr/share/pandora_server/util/plugin/udp_nmap_plugin.sh` in the field.


The server will execute this script, so this should have permissions of access and execution on it.

- Plug-in parameters

A string with the parameters of the command that will be after command and a blank space. This parameters field accepts macros as `_field1_ _field2_ ... _fieldN_`.

Macro parameters

Description (_field1_)	Default value (_field1_)
Hide value 	<input type="checkbox"/>
Help (_field1_)	<div style="border: 1px solid #ccc; height: 100px; width: 100%;"></div>

Add macro 

[Create !\[\]\(47a3b08df1ba32720cad69432dbd8375_img.jpg\)](#)















- Parameters macros

Is possible to add unlimited macros to use it in Plug-in parameters field. This macros will appear as normal text fields in the module configuration. The following section explains how macros work.

Plugin macros

Consider the DNS Plugin installed by default in Pandora FMS server.

Plug-ins registered on Pandora FMS

Name	Type	Command	Op.
DNS Plugin	Standard	<code>/usr/share/pandora_server/util/plugin/dns_plugin.sh</code>	 
IPMI Plugin	Standard	<code>/usr/share/pandora_server/util/plugin/ipmi-plugin.pl</code>	 
MySQL Plugin	Standard	<code>/usr/share/pandora_server/util/plugin/mysql_plugin.sh</code>	  
Network bandwidth SNMP	Standard	<code>perl /usr/share/pandora_server/util/plugin/pandora_snmp_bandwidth.pl</code>	 
Packet Loss	Standard	<code>/usr/share/pandora_server/util/plugin/packet_loss.sh</code>	  
SMTP Check	Standard	<code>/usr/share/pandora_server/util/plugin/SMTP_check.pl</code>	 

Said plugin, **among other uses**, allows for a web domain and its corresponding IP to be checked by a DNS server.

- `-i` : Known domain IP address.
- `-d` : Corresponding web domain.
- `-s` : DNS server for checking. Knowing these three parameters, proceed to **manually register** a new plugin, add the name "New DNS Plugin" and in the description copy the previous summary and in addition indicate that it is necessary for the Module to retrieve a false value (0) or true (1) for monitoring. This type of data is also known as boolean and in Pandora FMS it is called `generic_proc`.

In the section Macro parameters add three macro fields `field1`, `field2` and `field3`.

For `_field1_` add the description according to the `-i` parameter and so on for parameters `-d` and `-s`. Leave the values by default empty, type in in the Help of each one of them the text you wish.

In the text box Plugin command type in:

```
/usr/share/pandora_server/util/plugin/dns_plugin.sh
```

In the Plugin parameters text box, type in:

```
-i _field1_ -d _field2_ -s _field3_
```



When this plugin is used within the module, the DNS Plugin will be executed replacing each of the fields by the

parameters written on the Module.

```
/usr/share/pandora_server/util/plugin/dns_plugin.sh -i _field1_ -d _field2_ -s  
_fiel
```

Performance

Macros work like `_field1_`, `_field2_`, (...), `_fieldN_`, but they host special values both from the Modules and the Agents those Modules contain.

Going back to the example of the previous section, where the default values of the `_fieldN_` were left blank. Edit it for the default value of `_field2_` and add the macro `_module_`.

If any module or component is using that server plugin, a lock icon that cannot be updated will appear.

The `_module_` macro returns the Module the plugin uses and will be added to the user or policy when said Module is created. To verify this, create a new Agent called “DNS verify” and a new Module through the option Create a new plugin server module:



Once you are again in the new Module editing form, in Plugin select the “New DNS Plugin” list and the `_module_` macro will appear like this:

Type ?

Warning threshold
 Min.
 Max.
 Inverse interval

Critical threshold
 Min.
 Max.
 Inverse interval

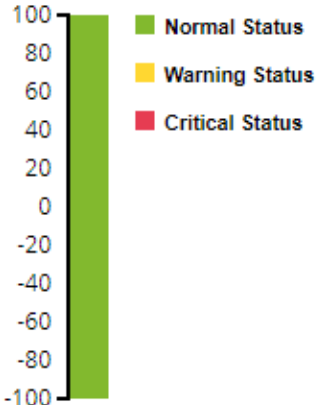
Historical data

Plugin ? This plugin is used to check if a specific domain return a specific IP address.

IP

Domain

Server DNS



Remember to add that the type of data to be collected must be “Generic boolean” and for the name of the new Module, just add the web domain whose IP address you will verify with a specified DNS server. The PFMS server's `critical_on_error` token is configured by default to make **modules in unknown state go to critical state**. Save it and check it works.

The advantage of this method is that it will have both Modules and Web domains you need to check and they are easily identified in different components (Dashboards, reports, etc.) through its name.

Macro list

- `_agent_` : Agent alias to be used in the macro. If no alias is assigned, the name of the agent will be used.
- `_agentalias_` : Agent alias to be used in the macro.
- `_agentdescription_` : Description of the agent to be used in the macro.
- `_agentstatus_` : Current status of the Agent when used in the macro.
- `_agentgroup_` : Name of the Agent group to be used in the macro.
- `_agentname_` : Name of the Agent to be used in the macro (see also `_agente_ >`).
- `_address_` : Agent adress to be used in the macro.
- `_module_` : Name of the Module to be used in the macro.
- `_modulegroup_` : Name of the Module group to be used in the macro.
- `_moduledescription_` : Description of the module to be used in the macro.
- `_modulestatus_` : Status of the module to be used in the macro.
- `_moduletags_` : URL associated to the module tags.

- `_id_module_` : ID of the module to be used in the macro.
- `_id_agente_` : ID of the agent to be used in the macro, useful for building the URL of access to Pandora FMS console.
- `_id_group_` : ID of the agent group to be used for the macro.
- `_interval_` : Interval of the modules execution to be used for the macro.
- `_target_ip_` : Target IP address of the module to be used for the macro.
- `_target_port_` : Target port of the module to be used for the macro.
- `_policy_` : Name of the policy the module belongs to (if it applies).
- `_plugin_parameters_` : Plugin parameters of the module to be used in the macro.
- `_email_tag_` : Email inboxes associated to the module tags.
- `_phone_tag_` : Phone numbers associated to the module tags.
- `_name_tag_` : Name of the tags associated to the module for the macro.

PSPZ Packaging

You may find more information in our tutorial «[Monitoring in Pandora FMS with server plugins](#)».

Pandora FMS server Zipfile plugin (.pspz)

From Pandora FMS 3.0 there is a new way to register plugins and modules that use the new plugin (for example the module library that depends on the plugin). It is basically an extension of the administrator to upload a file in `.pspz` format, described in detail in the next sections. The system reads the file, unpackages and installs the binaries and/or scripts in the system. In addition, it registers the plugin and creates all the modules defined in the `.pspz` in Pandora FMS module library (network components).

This section describes how to create a `.pspz` folder.

Package File

A `.pspz` is a compressed file in zip format with two rows:

`plugin_definition.ini`: It contains the plugin and module specification. It must have exactly that name (it is *case sensitive*, it differences uppercase and lowercase).

`<script_file>`: It is the *plugin script/binary* itself. It may have any valid name. An example of a `.pspz` file (in turn compressed as `.zip` to include its documentation) can be downloaded through [this link](#).

Structure of `plugin_definition.ini`

Header/Definition

This is a **classic INI file** with optional sections. The first section, the most important, is a fixed name section called `plugin_definition`, and this is an example:

```
[plugin_definition]
name = Remote SSH exec
filename = ssh_pandoraplugin.sh
description = This plugin execute remotely any command provided
timeout = 20
ip_opt = -h
execution_command =
execution_postcommand =
user_opt = -u
port_opt =
pass_opt =
plugin_type = 0
total_modules_provided = 1
```

`filename`: It should have the same name as the script included in the `.pspz` file, referenced before as `<script_file>`. In this sample is a `.sh` shell script called `ssh_pandoraplugin.sh`.

`*_opt`: Here are the registration options for the plugin, shown in the form to “manually” register the plugin in Pandora FMS console.

`plugin_type`: 0 for a standard Pandora FMS plugin, and 1 for a Nagios-type plugin.

`total_modules_provided`: It specifies how many modules are defined in the `.ini` file. At least one must be set (to use at least in an example).

`execution_command`: If used, add it before the script. It could be an interpreter, like for example `java -jar`. So the plugin will be called from Pandora FMS Plugin Server as `java -jar <plugin_path>/<plugin_filename>`.

`execution_postcommand`: If used, defines additional parameters added to the plugin after the `<plugin_filename>`, invisible for the user.

Module definition / Network components

Define the same number of modules as those defined in `total_modules_provided` from the previous section.

If you have for example four modules, the names of those sections must be: `module1`, `module2`, `module3` and `module4`.

This is an example of a module definition:

```
[module1]
name = Load Average 1Min
description = Get load average from command uptime
id_group = 12
type = 1
max = 0
min = 0
module_interval = 300
id_module_group = 4
id_modulo = 4
plugin_user = root
plugin_pass =
plugin_parameter = "uptime | awk '{ print $10 }' | tr -d ','"
max_timeout = 20
history_data = 1
min_warning = 2
min_critical = 5
str_warning = "danger"
min_critical = "alert"
min_ff_event = 0
tcp_port = 0
critical_inverse = 0
warning_inverse = 0
critical_instructions = "Call the department head"
warning_instructions = "Call the server manager to reduce the load."
unknown_instructions = "Verify that the Pandora FMS agent is running"
```

Some things to bear in mind:

- All fields *must* be defined. If you have no data, leave them blank; focus on fields `plugin_pass` from the previous example.
- Use double quotation marks by pairs “...” to define values that contain special characters or spaces such as the field `plugin_parameter` from the example above. INI files that contain characters like ‘ / - _ () [] and others, *must* have double quotation marks, Avoid using the ” character for data. If you must use it, *escape* with the combination \” .
- If you have doubts about the purpose or meaning of any fields, check out `tnetwork_component` in Pandora FMS database, since it has almost the same fields. When a new network component is created, it is stored in this database, try to create a network component that uses its plugin and analyze the entry log in that table to understand all values.
- `id_module`: It must always be 4 (This means it is a plugin module).
- `type`: It defines the type of module it is: `generic_data` (1), `generic_proc` (2), `generic_data_string` (3) or `generic_data_inc` (4) as defined in `ttype_module`.
- `id_group`: It is the Primary Key of the group table that contains group definitions. Group 1 is “all groups” (“All”), and works as a special group.
- `id_module_group`: It comes from the table `tmodule_group`. It is a purely descriptive module-feature association. You may use 1 for the General module group.

Version 2

From Pandora FMS v5.1.SP1, server plugins use macros.

This plugins are differentiated by the file extension pspz2. If it does not have this extension, PSPZ 1 will be taken by default (with no dynamic macros in the remote plugin extension).

In addition `plugin_definition.ini` has changed. The following fields have been added:

In section `plugin_definition`:

- `total_macros_provided` that defines the number of dynamic macros the plugin has.

In section `module<N>`:

- `macro_<N>_value` that defined the value for that module using that dynamic macro, if it does not exist it takes the default value.

A section must be created for each dynamic macro, for instance:

```
[macro_<N>]
hide = 0
description = <your_description>
help = <text_help>
value = <your_value>
```

It must call in section `execution_postcommand` macros so that the replacing takes place ([see example](#))

The previous version is still supported. If the version parameter is not defined, version 1 is taken by default.

Example

Definition of a plugin v2 (`.pspz2`) for didactical purposes:

```
[plugin_definition]
name = PacketLoss
filename = packet_loss.sh
description = "Measure packet loss in the network in %"
timeout = 20
```

```
ip_opt =
execution_command =
execution_postcommand =
parameters = _field1_ _field2_
user_opt =
port_opt =
pass_opt =
plugin_type = 0
total_modules_provided = 1
total_macros_provided = 2

[macro_1]
hide = 0
description = Timeout
help = Timeout in seconds
value = 5

[macro_2]
hide = 0
description = Target IP
help = IP address
value = 127.0.0.1

[module1]
name = Packet loss
description = "Measure target packet loss in % "
id_group = 10
type = 1
max = 0
min = 0
module_interval = 300
id_module_group = 2
id_modulo = 4
max_timeout = 20
history_data = 1
min_warning = 30
min_critical = 40
min_ff_event = 0
tcp_port = 0
macro_1_value = 5
macro_2_value = localhost
unit = %
```

Upgrade a old PSPZ

From Pandora FMS version 4, some PSPZ prior to parameter dynamic field creation for plugins and had fixed parameters, will not work in new versions.

To migrate them, execute them with the appropriate credentials and after carrying out the

upgrading process, the following:

```
/usr/share/pandora_server/util/pandora_migrate_plugins.pl <dbname> <dbhost>  
<dbuser> <dbpass>
```

Find more information about major and minor version update in [this link](#).

[Go back to Pandora FMS documentation index](#)