Server Plugins Development

Server Plugins Development

Server plug-in basic features

The server plugin is executed by the Pandora FMS Plugin Server, so it must have some special features:

- Each execution of the plugin must return a single value. This must be the case since the Plugin Server performs one execution for each Plugin Module.
- You must be able to access the resources to be monitored remotely.
- It is possible to use any programming language supported by the operating system where the Pandora FMS server is installed.
- All the dependencies or software needed to run the plugin should be available or installed in the same machine that runs the Pandora FMS server.

More information about monitoring with remote server plugins can be found at this link. There you will get simpler examples and how they work with the Modules and the Agents that contain them. This article discusses in detail the creation of server plugins.

Server plugin development example

A possible example of a server plugin for Pandora FMS is described below.

The following plugin returns the sum of the incoming and outgoing traffic of a device interface, the data is obtained by SNMP.

The plugin code would be as follows:

```
#!/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_bandwith.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 function `usage`:

```
sub usage () {
        print "iface_bandwith.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";
}
```

This function describes the version and how to use the plugin, it is very important and should always be shown when executing the plugin without any parameter or with an option like `-h`:

```
--help
```

Regarding the value returned by the plugin, this is printed in the standard output on the penultimate line with the following instruction:

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

The value returned by the plugin is a single data, which then the Pandora FMS Plugin Server will add as data to the associated Module.

In order to execute this server plugin it will be necessary to install the snmpwalk and snmpget commands in the machine that runs the Pandora FMS server.

Console plugin manual registration

- Plugin type

The difference with PFMS is mainly that Nagios plugins *return an error level to indicate whether the test was successful or not*. If you need to get a data, not a status (Good/Bad), you can use a Nagios type plugin in “Standard” mode (for MS Windows® use Nagios wrapper for agent plugin).

- Max. timeout

This is the expiration time of the add-on. If no response is received within this time, the Module will be marked as unknown and its value will not be updated. This is a very important factor when

implementing monitoring with plugins, because if the time it takes to execute the plugin is greater than this number, you will never be able to obtain values with it. This value must always be greater than the time it normally takes for the script or executable used as a plugin to return a value. If nothing is specified, the value specified in the configuration as `plugin_timeout` will be used.

- Plug-in command

It is the path where the plugin command is. By default, if the installation has been standard, they will be in the directory `/usr/share/pandora_server/util/plugin/`, although it can 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 it must have access and execution permissions on it.

- Plug-in parameters

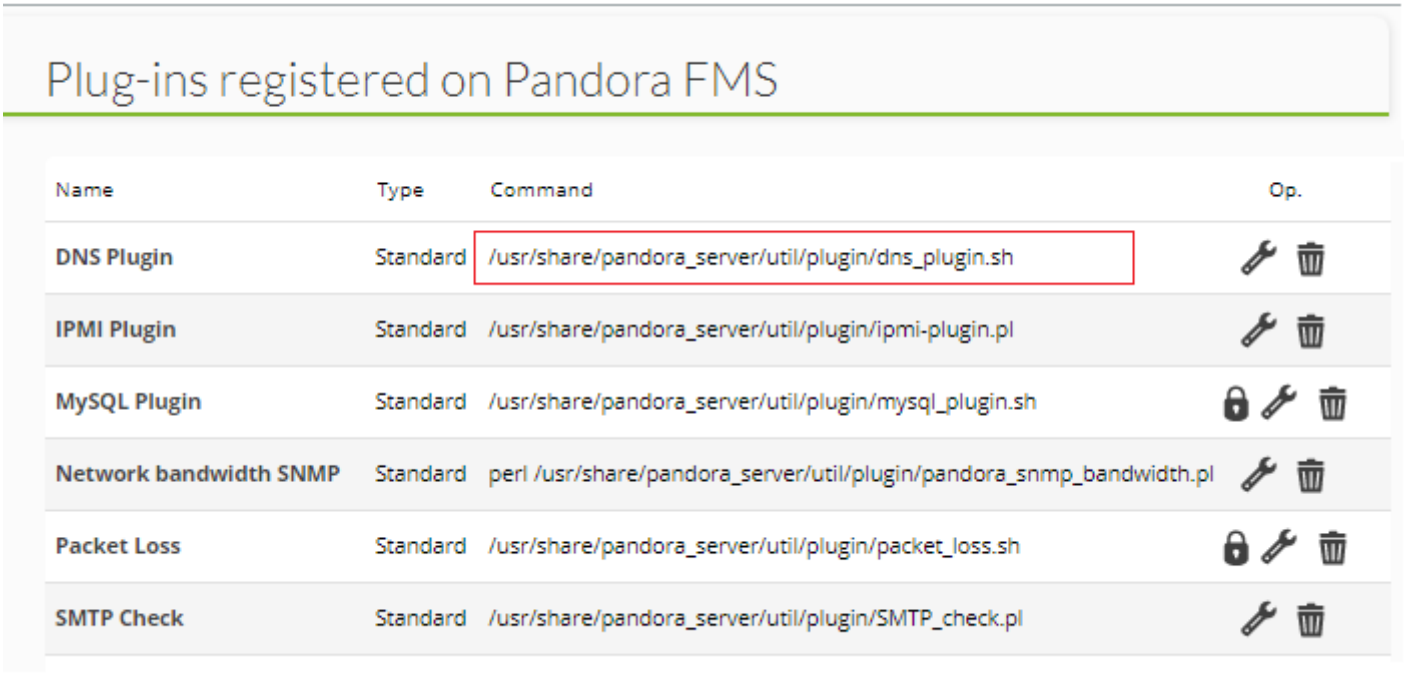
A string with the plugin parameters, which will follow the command and a blank space. This field accepts macros such as `_field1_ _field2_ … _fieldN_`. The following section expands on how macros work.

- Parameters macros

It is possible to add unlimited macros for use in the plugin parameters field. These macros will appear as text fields in the Module configuration. The following section expands on how macros work.

Plugin macros

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



This add-on, among other uses, allows a web domain and its corresponding IP address to both be checked by a domain resolution server (DNS server).

- `-i` : Known IP address of the domain.
- `-d` : Corresponding web domain.
- `-s` : DNS server to query.

Knowing these three parameters, proceed to register manually a new plugin, put in the name "New DNS Plugin" and in the description copy the previous summary and also indicate that it is necessary that the Module collects for the monitoring a false value (zero) or true (one). This data type is also known as *boolean* and in Pandora FMS is called `generic_proc` .

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

For `_field1_` put the description corresponding to the parameter `-i` and so on for the parameters `-d` and `-s` . Leave the default values (default value) empty, write in the help (Help) of each one the text that you consider convenient.

In the Plugin command text box type:

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

In the Plugin parameters text box type:

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

When this plugin is used in a Module, the DNS Plugin will be executed replacing each of

the fields with the parameters that are written in the Module.

```
/usr/share/pandora_server/util/plugin/dns_plugin.sh -i _field1_ -d _field2_ -s
_field3_
```

Operation

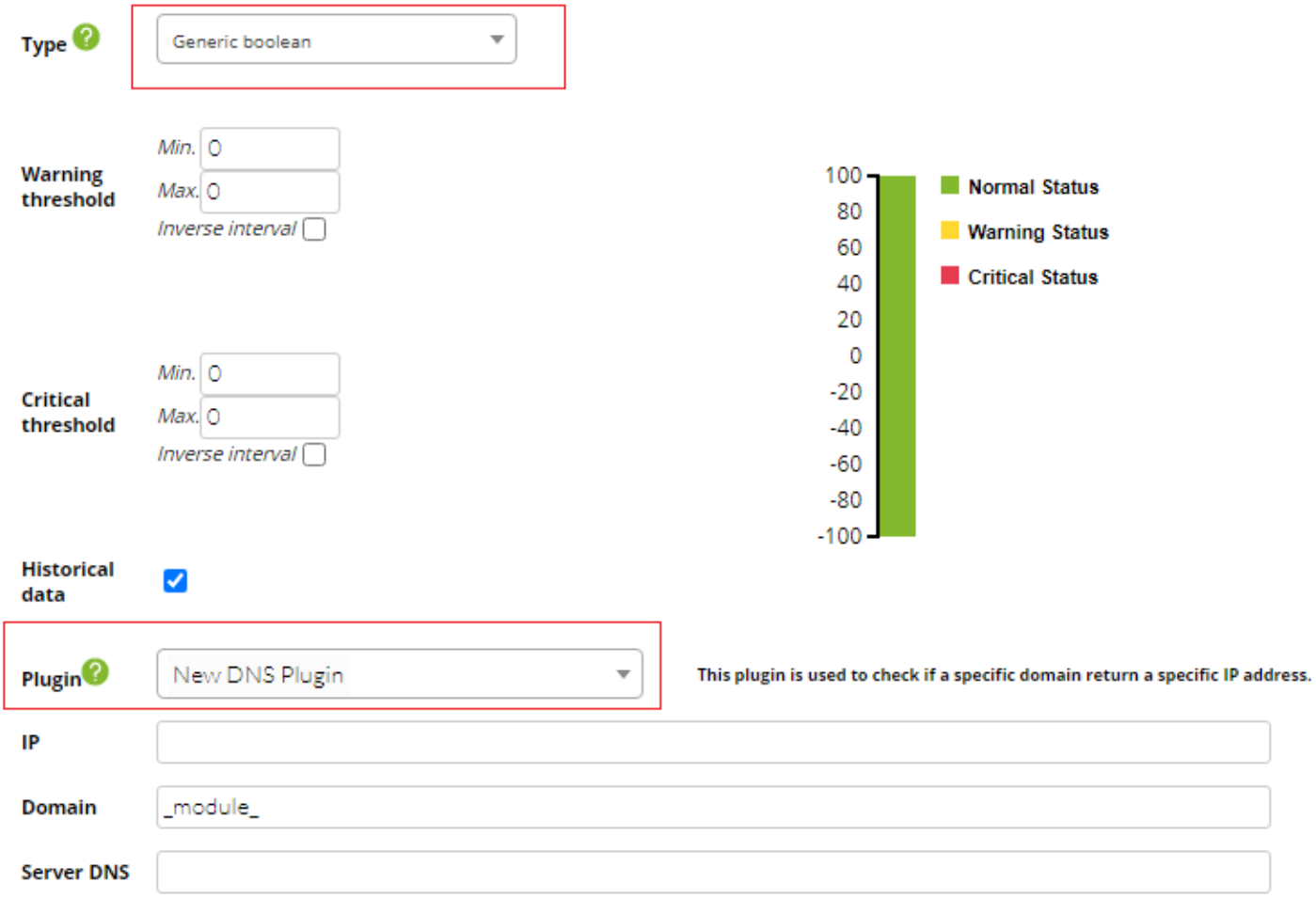
In the same way that _field1_, _field2_, (...), _fieldN_ work, so do the macros, only that they will hold special values of both the Modules and the Agents that contain those Modules.

Return to the example from the previous section where the default values of the _fieldN_ were left empty. Edit it and for the default value of `_field2_` place the macro `_module_`.

If any Module or component is using this server plugin, a
lock icon will appear and cannot be updated.

This macro `_module_` returns the name of the Module used by the plugin and will be placed to the user or policy when creating the Module. To check this, proceed to create a new Agent called “DNS verify” and add a new Module by means of the Create a new plugin server module option.

Once you enter the edit form of the new Module, in Plugin select from the list “New DNS Plugin” and the macro `_module_` will appear as follows:



Remember to set the data type to be collected to “Generic boolean” and for the name of the new Module simply set the web domain to which you are going to check its corresponding IP address against a specified DNS server. The *token* `critical_on_error` of the PFMS server is configured by default so that the modules in unknown status are switched to critical status. Store and check operation.

The advantage of this method is that you will have as many Modules as web domains you need to verify and they will be easily identified in different components (Dashboards, reports, etc.) by their name.

List of macros

- `_agent_`: Alias of the Agent to be used in the macro. If no alias is assigned, the name of the Agent will be used.
- `_agentalias_`: Alias of the Agent 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 use the macro.
- `_agentname_`: Name of the Agent to be used in the macro (see also `_agent_`).
- `_address_`: Address of the Agent to use the macro.
- `_module_`: Name of the Module to use the macro.
- `_modulegroup_`: Name of the Module group to use the macro.
- `_moduledescription_`: Description of the Module to use the macro.
- `_modulestatus_`: Status of the Module to use the macro.

- _moduletags_: URLs associated to the module tags.
- _id_module_: Identifier of the Module to use the macro.
- _id_agente_: Identifier of the Agent to use the macro, useful to build access URL to the Pandora FMS Console.
- _id_group: Identifier of the Agent group to be used by the macro.
- _interval_: Interval of the execution of the Module to use the macro.
- _target_ip_: IP address of the target of the Module to be used by the macro.
- _target_port_: Module target port to use the macro.
- _policy_: Name of the policy to which the Module belongs (if applicable).
- _plugin_parameters_: Parameters of the Module plugin to be used by the macro.
- _email_tag_: Mailboxes associated to the Module tags.
- _phone_tag_: Phones associated to the module tags.
- _name_tag_: Name of the tags associated to the Module to be used by the macro.

Packaged in PSPZ

Pandora Server Zipfile Plugin (.pspz)

There is a way to register plugins and Modules that use the new plugin (such as the module library that depends on the plugin). It is basically an administrator extension to upload a file in `.pspz` format, described in detail in the sections following this one. The system reads the file, unpacks and installs the binaries and/or scripts on the system. In addition, it registers the plugin and creates all the modules defined in the `.pspz` in the Pandora FMS module library (network components).

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

Package File

A `.pspz`' is a file copied in zip format with two lines:

plugin_definition.ini: contains the specification of the plugin and modules. It must have exactly this name (it is *case sensitive*).

<script_file>: is the plugin script binary (command script or executable binary code) itself. It can have any valid name. An example of a `.pspz` file (itself compressed as `.zip` to be able to include its documentation) can be downloadedthis link.

plugin_definition.ini structure

Header/Definition

This is a classic INI file with optional sections. The first section, which is the most important, is a

fixed name section called `plugin_definition`. Here is an example:

```
[plugin_definition]
name = Remote SSH exec
filename = ssh_pandoraplugin.sh
description = This plugin execute remotely any command provided
timeout = 20
execution_command =
execution_postcommand =
ip_opt = -h
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, named before as < script_file >. In this example it is a shell script (format `.sh`) named `ssh_pandoraplugin.sh`.

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

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

total_modules_provided: Specifies how many modules are defined in the following sections of the `.ini` file. You must set at least one (to use in an example at least).

execution_command: If it is used, it must be placed in front of the script. It could be an interpreter, such as `java -jar`. So, the plugin will be called to execution, from the Pandora FMS Plugin Server, with the following code: `java -jar <plugin_path>/<plugin_filename>`.

execution_postcommand: If used, defines additional parameters passed to the plugin after the <plugin_filename>, which is invisible to the user.

Module Definition / Network Components

You must define here the same number of modules as those defined in `total_modules_provided` in the previous section.

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

Here 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 keep in mind:

- All fields *must* be defined. If you have no data, leave it blank; see the `plugin_pass` field in the example above.
- Use double quotes in pairs **"…"** to define values containing special characters or spaces, such as the `plugin_parameter` field in the example above. INI files containing characters such as `' "/ -_ ( ) [ ]` and others, *must* have double quotes. Try to 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 these fields, you can check `tnetwork_component` in the Pandora FMS database, because 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 your plugin and analyze the input record in that table to understand all the values.
- id_module: It should always be 4 (this means that it is a Module plugin).
- type: Define what type of Module it is: generic_data (1), generic_proc (2), generic_data_string (3) or generic_data_inc (4) as defined in `ttipo_modulo`.
- id_group: It is the PK (primary key) of the group table containing group definitions. Group 1 is "all groups" ("*All*"), and acts as a special group.
- id_module_group: Proceeds from table `tmodule_group`. It is a module association by functionality, purely descriptive. You can use 1 for the General module group.

Version 2

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

These plugins will be differentiated by the extension of the pspz2 file. If it does not have this extension, a type 1 PSPZ (without dynamic macros in the remote plugin extension) will be assumed.

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

In the section `plugin_definition`:

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

In the section `module<N>`:

- `macro_<N>_value` that defines the value for that Module using that dynamic macro, if it does not exist the default value is taken.

And you must create a section for each dynamic macro, for example:

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

- Macros must be called in the `execution_postcommand` section to perform the substitution (see example).
- The previous version is still supported. If the version parameter is not defined, the version is assumed to be 1.

Example

Definition of a v2 plugin (`.pspz2`) for didactic 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 adddress
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 = %
```

Back to Pandora FMS documentation index