Agent Plugins Development

Agent Plugins Development

We are working on the translation of the Pandora FMS documentation. Sorry for any inconvenience.

Agent Plugins Development

Basic Features of the Agent Plugin

The agent plugin is executed by the Pandora FMS Software Agent so it should have some special features:

- Each execution of the plugin could return one or several modules with their correspondent values. The exit should have a XML format as we will explain later:
- It could have access both local resources to the machine or a resources from other machine in a remote way.
- It is possible to use any kind of programming language supported by the operative system where the Pandora software agent would be installed.
- All dependencies or necessary software to execute the plugin should be available and be installed in the same machine that executes the Pandora software.

The agent plugins could do a kind of “recon task” so the plugin could return several modules in one execution and the number could change between different executions.

In UNIX and Linux the exit status value of the plugin must
be 0, otherwise the plugin output will be ignored

Example of Agent Plugin Development

We are going to explain now an example of a simple plugin. This agent plugin returns the percentage of use of the system filesystems. The code is the following one:

```
#!/usr/bin/perl

use strict;

sub usage() {
        print "\npandora_df.pl v1r1\n\n";
        print "usage: ./pandora_df\n";
        print "usage: ./pandora_df tmpfs /dev/sda1\n\n";
}
```

```
# Retrieve information from all filesystems
my $all_filesystems = 0;

# Check command line parameters
if ($#ARGV < 0) {
        $all_filesystems = 1;
}

if ($ARGV[0] eq "-h") {
        usage();
        exit(0);
}

# Parse command line parameters
my %filesystems;
foreach my $fs (@ARGV) {
        $filesystems{$fs} = '-1%';
}

# Retrieve filesystem information
# -P use the POSIX output format for portability
my @df = `df -P`;
shift (@df);

# No filesystems? Something went wrong.
if ($#df < 0) {
        exit 1;
}

# Parse filesystem usage
foreach my $row (@df) {
        my @columns = split (' ', $row);
        exit 1 if ($#columns < 4);
        $filesystems{$columns[0]} = $columns[4] if (defined
($filesystems{$columns[0]}) || $all_filesystems == 1);
}

while (my ($filesystem, $use) = each (%filesystems)) {

        # Remove the trailing %
        chop ($use);

        # Print module output
        print "<module>\n";
        print "<name><![CDATA[" . $filesystem . "]]></name>\n";
        print "<type><![CDATA[generic_data]]></type>\n";
        print "<data><![CDATA[" . $use . "]]></data>\n";
        print "<description>% of usage in this volume</description>\n";
        print "</module>\n";
}

exit 0;
```

An important part of the code is the usage function:

```
sub usage() {
        print "\npandora_df.pl v1r1\n\n";
        print "usage: ./pandora_df\n";
        print "usage: ./pandora_df tmpfs /dev/sda1\n\n";
}
```

In this function it describes the version and how to use the plugin. It is very important and it should be shown when executing the plugin without any kind of parameter or with an action type -h or –help. In this example is executed when the parameter -h is executed, the following lines verify it:

```
if ($ARGV[0] eq "-h") {
        usage();
        exit(0);
}
```

Regarding the values returned by the plugin, you can notice that onece the data has been collected from the following file systems, an XML part is created and printe by the standard exit for any one of them. This task is done in the following lines:

```
while (my ($filesystem, $use) = each (%filesystems)) {

        # Remove the trailing %
        chop ($use);

        # Print module output
        print "<module>\n";
        print "<name><![CDATA[" . $filesystem . "]]></name>\n";
        print "<type><![CDATA[generic_data]]></type>\n";
        print "<data><![CDATA[" . $use . "]]></data>\n";
        print "<description>% of usage in this volume</description>\n";
        print "</module>\n";
}
```

An example of the result that this plugin returns could be:

```
<module>
<name><![CDATA[tmpfs]]></name>
<type><![CDATA[generic_data]]></type>
<data><![CDATA[0]]></data>
<description>% of usage in this volume</description>
</module>
<module>
<name><![CDATA[/dev/mapper/VolGroup-lv_home]]></name>
<type><![CDATA[generic_data]]></type>
<data><![CDATA[26]]></data>
<description>% of usage in this volume</description>
```

```
</module>
<module>
<name><![CDATA[/dev/sda9]]></name>
<type><![CDATA[generic_data]]></type>
<data><![CDATA[34]]></data>
<description>% of usage in this volume</description>
</module>
```

The number of returned modules by this plugin will depend on the number of configured filesystems and it could change between different executions.

The XML piece is added to the general XML that the software agent generates and it is sent to the Pandora server to be processed by the Data Server

Troubleshooting

If Pandora FMS does not recognize your agent plugin, you don't get the information you expect or the agent just doesn't want to work, there are a few things which you have to keep in mind:

Check the pandora_agent.conf document

The Software Agent needs a line in this file with the correct path of the plugin.

For example:

```
module_plugin /etc/pandora/plugins/MyMonitor.pl
/etc/pandora/plugins/MyMonitor.conf 2> /etc/pandora/plugins/MyMonitor.err
```

MyMonitor.pl is the agent plugin, MyMonitor.conf is the configuration file passed as an argument, and MyMonitor.err is a file that will receive the possible errors of the plugin execution and will keep clean the standard output.

Reboot the pandora_agent_daemon

The Software Agent will run the plugins every five minutes. For those people who can not wait, it is possible to restart the Software Agent from the command line.

For example:

```
/etc/init.d/pandora_agent_daemon restart
```

Check the plugin permissions

The plugin, and the files which are going to be used for it, must have the correct read, write and execute permissions. In Unix this should be enough:

```
chmod 755 <plugin_path>
```

Validate the output

An easy way to find the errors is run the plugin manually in the command line. Sit down and check the output carefully, for example:

```
popeye:/etc/pandora/plugins # ./pandora_df
<module>
<name><![CDATA[/dev/sda2]]></name>
<type><![CDATA[generic_data]]></type>
<data><![CDATA[19]]></data>
<description>% of usage in this volume</description>
</module>
<module>
<name><![CDATA[udev]]></name>
<type><![CDATA[generic_data]]></type>
<data><![CDATA[1]]></data>
<description>% of usage in this volume</description>
</module>
```

Validate the resulting XML

The XML that prints the plugin must have valid XML syntax. The XML also needs to be well formed. To check if it is, you could follow this steps from the command line:

1. Create an XML document with the plugin output: *./Plugin.pl > Plugin.xml*
2. Check the XML document: *xmllint Plugin.xml*

Debug mode

You can activate the debug mode by changing the value of the label *debug* in your *pandora_agent.conf* file from 0 to 1. Once you do this, when the Software Agent run the plugin, the results will be saved in an XML document with all the agent information.

The name of the document will be the agent name with *.data*, and will be located in */tmp* directory (checkout the pandora agent log at /var/log/pandora/pandora_agent.log). By checking the document, you can see if the data of your plugin are being collected and if it what you expect.

When you enable Debug mode, the agent executes only once and exits

Forum

If the error remains after all, fell free to ask in our forum.

Go back to Pandora FMS documentation index