

Extending Apache SpamAssassin Using Plugins

Michael Parker

ApacheCon 2005

[Start Slide]

Welcome, and thank you for coming.

[Introduction Slide]

Today I want to talk about Apache SpamAssassin and how you can extend the basic functionality using plugins. Added in version 3.0, the plugin functionality allows you to do virtually anything you would like to do without having to change the core SpamAssassin code.

There are literally dozens, maybe even hundreds, of plugins written for SpamAssassin performing various functions. The freedom afforded from the plugin architecture allows developers and end users access to multiple parts of SpamAssassin, this means the sky's the limit when it comes to extending SpamAssassin's functionality.

[The Way It Was Slide]

Adding functionality to SpamAssassin wasn't always easy. If you wanted to add a new configuration option you had to add it into the core Conf.pm module. To add a new eval test you would have to maneuver through EvalTests.pm and other modules, and trust me it is not pretty. For some things you might also have needed to modify the top level SpamAssassin module. This tended to be a fairly large barrier to entry and didn't lead to many contributions from the community.

[The Way it is Now Slide]

Now days it is much easier. You do need some perl programming experience but you no longer have to root around in the core SpamAssassin code. To extend the functionality of SpamAssassin now you create a module, create an eval rule or make use of an existing plugin hook, and then instruct SpamAssassin to load that module.

[Advantages of Plugins Slide]

There are many advantages to having code and rules in plugins.

Moving existing code into plugins really allows the core code to be cleaned up, any new code stays out of the core lessening the clutter even more.

One of the larger issues with creating rules for SpamAssassin is that eval rules were tied to a specific release, with plugins you can release a new eval rule outside of the normal development cycle.

Sometimes you'll have a set of functionality that a good portion of users won't use. Some good examples are AutoWhitelist, Bayes, Razor/Pyzor/DCC and others. If folks aren't going to use it, why should they have to load it into memory. Plugins allow you to keep whole sections of code out of memory.

For beginners, plugins give a well documented API and a way to stay out of the core SpamAssassin code. It also allows folks to create their own plugins and distribute them separately from SpamAssassin.

[The Basics Slide]

Creating a SpamAssassin plugin module is pretty simple, there are a few base elements that you must incorporate but beyond that it is up to you.

[Creating a Basic Plugin Module Slide]

The basic layout of all plugins starts from this simple example. It does take some perl knowledge to understand what is going on here.

To start we name our module via the package command. In this case our plugin is called TestPlugin. Then we use the base Mail::SpamAssassin::Plugin module and set it up as our parent. Then we have our own new method that will construct our plugin object. That's all you need, you now have your own basic plugin. It doesn't do anything yet, but it's a start.

[Loading Your Plugin Module Slide]

Once you have a plugin you need to instruct SpamAssassin to load it in order for it to work. You do this via the loadplugin config option. There are a couple of ways to do this. The first, if you've installed your plugin module somewhere in the perl path, is to just loadplugin and then the name of your module. If you haven't installed the module in your perl path then you will need to use the second form and specify the path to the module file.

The loadplugin option can appear in any configuration file, however it is generally placed in a .pre file. .pre files are special, they are loaded before any of the other configuration files, that gives the plugin an opportunity to load itself and be available by the time the configuration files are parsed.

[Configuration Slide]

Not a required element, but most plugins have some sort of associated configuration options. These behave just like other SpamAssassin configuration options. You can and should use the ifplugin option to wrap your plugin's configuration options. This will keep SpamAssassin from complaining about unknown configuration options when that plugin is not loaded.

To keep things in order it's generally a good idea to separate out your plugin module's configuration into a module specific configuration file.

[Parsing Configuration Options Slide]

Now that we've added configuration data for our plugin module we need to be able to parse that configuration and do something useful. For this we use the `parse_config` plugin hook.

Lets take a minute to talk about plugin hooks. To take advantage of a plugin hook you simply need to create a subroutine by the same name for that hook. Different hooks have different input parameters so be sure to consult the documentation on what to expect for each call.

In the `parse_config` case the input parameter is hash structure containing such things as the current configuration line, the parsed key and value, a reference to the configuration object.

For this test plugin module we've created a `fire_test_eval` config option and when we see that option we set the value in the configuration object. The `inhibit_further_callbacks` method signals that we have now parsed this config option and there is no need to send the option to other plugins.

[Creating an Eval Rule Slide]

Eval rules are just subroutines that return either 1, for a hit, or 0, for a non-hit. What they do to determine that return value is up to you.

Here we have a test eval rule. It's pretty simple, if the value for the configuration option `fire_test_rule` is set to 1 then the eval rule will hit, otherwise it will not. Eval rules expect to be passed a `PerMsgStatus` object along with any other arguments specified in the configuration.

[Registering an Eval Rule Slide]

Now that we have created our eval rule subroutine we need to register it so that it is available for calling in our rules. Here we have the new subroutine from our plugin module. To register our eval rule we make a call to `register_eval_rule` passing in the name of the subroutine.

[Beyond the Basics Slide]

[Example Plugin Hooks Slide/ More Plugin Hooks Slide]

[Parsing Configuration Options Version 3.1 Slide]

Starting in version 3.1 there is another method for parsing configuration that you can use. I personally find it easier to use than the `parse_config` hook but your mileage may

vary. It works exactly like the core config parser so it can be easier to setup initially. Here you can see our test plugin with the new configuration method. We create an array with a config block, see the man page for Mail::SpamAssassin::Conf::Parser for more information on config blocks. Once we've added all of our config options to the array we call the register_commands to register this plugins configuration blocks.

[Parsing Configuration Options Version 3.1 Continued Slide]

The final step is to call our newly created set_config method in new.

The issue with this method is that any plugins you write using it will not be backwards compatible with version 3.0.

[Examples Slide]

You can find the full source and documentation for the examples on the SpamAssassin wiki: <http://wiki.apache.org/spamassassin/CustomPlugins> or <http://www.apache.org/~parker/presentations/index.html>

[Future Directions Slide]

[Where to Get More Info Slide]