Introduction to Dradis plugin programming

22 February 2008

by Siebert Lubbe

Overview

- General client side structure
- Dispatcher
- Plugin framework
- Small example
- Nmap plugin

General client side structure

Have a look at you Dradis client side directory structure and you will find the following directories.

- Conf – XML configuration. Yip that's the configuration used to specify the connection to the server.
- Core – Core processing files. Won't go into depth here but as you guessed it's the core of the program
- Ui – Handling the graphical user interface. Contains the QT and WX graphical interface files.
- Commands

  + Dispatchers – Our plugins inherit from these: Simple or Namespace (We'll look at this in a second)

  + Modules – This is our plugins

Dispatcher

As earlier mentioned, you plugins inherit either from a simple or a namespace dispatcher.

- Simple dispatcher – It is for command lines with a simple command and a bunch of options.
  Format: <command> <option list>
  Example: dir c:\windows

- Namespace dispatcher – It is for the case where you want to group a logical set of commands in a namespace.
  Format: <namespace> <command> <option list>
  Example: add host 10.0.0.1

Plugin framework (1)

The code framework below gives you an idea of how the basic structure of a Dradis plugin looks. After you have seen this, have a look at some of the plugin files in the modules directory.

```
Module Commands
  class <PluginName> < Dispatcher::<Namespace/Simple>
    INFO = {
      <:namespace => 'export',>
      :commands => {
        ‘dir' => {
          :desc => 'shows a random Chuck Norris fact',
          :syntax => [:required => true, :label => path', :regexp => /\w+/]
        }     }   }

         def dir(*args)
           <implementation of command>
       end
    end # class
end # module
```

Plugin framework (2)

- Your plugin must be part of the Command s module thus – *Module Commands.*
- Your plugin class must inherit either from the Simple or the Namespace dispatcher as explained earlier.
- The INFO constant is a hash that defines the structure of the commands that you will receive from the command line. You define the namespace, the commands, the option list and descriptions in this hash. Best is to go through a few code examples to get a feel for this.
- After this you start implementing methods that relates to the commands as defined in you INFO constant. Your method receives *args as a parameter. args[1] is the first option after you command as received from the command line.

Personal details

Position: Software Developer for MWR InfoSecurity (http://www.mwrinfosecurity.com/). Specifically interested in Ruby and the Rails framework.

Email: siebert.lubbe@mwrinfosecurity.com