Sign in

lanmaster53 / recon-ng Public

Notifications Fork 636 Star 4k

Code Issues 14 Pull requests 12 Actions Wiki Security Insights

recon-ng / recon / core / base.py

905 lines (819 loc) · 40.2 KB

Code Blame Raw

```
__author__    = 'Tim Tomes (@lanmaster53)'

from datetime import datetime
from pathlib import Path
from urllib.parse import urljoin
import errno
import imp
import json
import os
import random
import re
import shutil
import sys
import yaml
import builtins

# import framework libs
from recon.core import framework
from recon.core.constants import BANNER, BANNER_SMALL

# set the __version__ variable based on the VERSION file
exec(open(os.path.join(Path(os.path.abspath(__file__)).parents[2], 'VERSION')).read())

# using stdout to spool causes tab complete issues
# therefore, override print function
# use a lock for thread safe console and spool output
```

```python
from threading import Lock
_print_lock = Lock()
# spooling system
def spool_print(*args, **kwargs):
    with _print_lock:
        if framework.Framework._spool:
            framework.Framework._spool.write(f"{args[0]}{os.linesep}")
            framework.Framework._spool.flush()
        # disable terminal output for server jobs
        if framework.Framework._mode == Mode.JOB:
            return
        # new print function must still use the old print function via the backup
        builtins._print(*args, **kwargs)
# make a builtin backup of the original print function
builtins._print = print
# override the builtin print function with the new print function
builtins.print = spool_print

#==================================================
# BASE CLASS
#==================================================

class Recon(framework.Framework):

    repo_url = 'https://raw.githubusercontent.com/lanmaster53/recon-ng-modules/master/'

    def __init__(self, check=True, analytics=True, marketplace=True, accessible=False):
        framework.Framework.__init__(self, 'base')
        self._name = 'recon-ng'
        self._prompt_template = '{}[{}] > '
        self._base_prompt = self._prompt_template.format('', self._name)
        # set toggle flags
        self._check = check
        self._analytics = analytics
        self._marketplace = marketplace
        self._accessible = accessible
        # set path variables
        self.app_path = framework.Framework.app_path = sys.path[0]
        self.core_path = framework.Framework.core_path = os.path.join(self.app_path, 'core')
        self.home_path = framework.Framework.home_path = os.path.join(os.path.expanduser('~'), '.re
        self.mod_path = framework.Framework.mod_path = os.path.join(self.home_path, 'modules')
        self.data_path = framework.Framework.data_path = os.path.join(self.home_path, 'data')
        self.spaces_path = framework.Framework.spaces_path = os.path.join(self.home_path, 'workspac

    def start(self, mode, workspace='default'):
        # initialize framework components
        self._mode = framework.Framework._mode = mode
        self._init_global_options()
        self._init_home()
        self._init_workspace(workspace)
        self._check_version()
        if self._mode == Mode.CONSOLE:
            self._print_banner()
            self.cmdloop()

    #==================================================
    # SUPPORT METHODS
    #==================================================

    def _init_global_options(self):
        self.options = self._global_options
        self.register_option('nameserver', '8.8.8.8', True, 'default nameserver for the resolver mi
        self.register_option('proxy', None, False, 'proxy server (address:port)')
        self.register_option('threads', 10, True, 'number of threads (where applicable)')
        self.register_option('timeout', 10, True, 'socket timeout (seconds)')
        self.register_option('user-agent', f"Recon-ng/v{__version__.split('.')[0]}", True, 'user-ag
        self.register_option('verbosity', 1, True, 'verbosity level (0 = minimal, 1 = verbose, 2 =

    def _init_home(self):
        # initialize home folder
        if not os.path.exists(self.home_path):
            os.makedirs(self.home_path)
        # initialize keys database
        self._query_keys('CREATE TABLE IF NOT EXISTS keys (name TEXT PRIMARY KEY, value TEXT)')
        # initialize module index
        self._fetch_module_index()

    def _check_version(self):
        if self._check:
            pattern = r"'(\d+\.\d+\.\d+[^']*)'"
            remote = 0
            local = 0
            try:
                remote = re.search(pattern, self.request('GET', 'https://raw.githubusercontent.com/
                local = re.search(pattern, open('VERSION').read()).group(1)
            except Exception as e:
                self.error(f"Version check failed ({type(e).__name__}).")
                #self.print_exception()
            if remote != local:
                self.alert('Your version of Recon-ng does not match the latest release.')
                self.alert('Please consider updating before further use.')
                self.output(f"Remote version:  {remote}")
```

```
                        self.output(f'Remote version: {remote}')
```

```

    #==================================================
    # COMPLETE METHODS
    #==================================================

    def complete_index(self, text, line, *ignored):
        if len(line.split(' ')) == 2:
            return [x for x in self._loaded_modules if x.startswith(text)]
        return []

    def complete_marketplace(self, text, line, *ignored):
        arg, params = self._parse_params(line.split(' ', 1)[1])
        subs = self._parse_subcommands('marketplace')
        if arg in subs:
            return getattr(self, '_complete_marketplace_'+arg)(text, params)
        return [sub for sub in subs if sub.startswith(text)]

    def _complete_marketplace_refresh(self, text, *ignored):
        return []
    _complete_marketplace_search = _complete_marketplace_refresh

    def _complete_marketplace_info(self, text, *ignored):
        return [x['path'] for x in self._module_index if x['path'].startswith(text)]
    _complete_marketplace_install = _complete_marketplace_info

    def _complete_marketplace_remove(self, text, *ignored):
        return [x['path'] for x in self._module_index if x['status'] == 'installed' and x['path'].s

    def complete_workspaces(self, text, line, *ignored):
        arg, params = self._parse_params(line.split(' ', 1)[1])
        subs = self._parse_subcommands('workspaces')
        if arg in subs:
            return getattr(self, '_complete_workspaces_'+arg)(text, params)
        return [sub for sub in subs if sub.startswith(text)]

    def _complete_workspaces_list(self, text, *ignored):
        return []
    _complete_workspaces_create = _complete_workspaces_list

    def _complete_workspaces_load(self, text, *ignored):
        return [x for x in self._get_workspaces() if x.startswith(text)]
    _complete_workspaces_remove = _complete_workspaces_load

    def complete_snapshots(self, text, line, *ignored):
        arg, params = self._parse_params(line.split(' ', 1)[1])
        subs = self._parse_subcommands('snapshots')
        if arg in subs:
            return getattr(self, '_complete_snapshots_'+arg)(text, params)
        return [sub for sub in subs if sub.startswith(text)]

    def _complete_snapshots_list(self, text, *ignored):
        return []
    _complete_snapshots_take = _complete_snapshots_list

    def _complete_snapshots_load(self, text, *ignored):
        return [x for x in self._get_snapshots() if x.startswith(text)]
    _complete_snapshots_remove = _complete_snapshots_load

    def _complete_modules_reload(self, text, *ignored):
        return []

#==================================================
# SUPPORT CLASSES
#==================================================

class Mode(object):
   '''Contains constants that represent the state of the interpreter.'''
   CONSOLE = 0
   CLI     = 1
   WEB     = 2
   JOB     = 3

   def __init__(self):
       raise NotImplementedError('This class should never be instantiated.')
```