Airoscript-ng Documentation

Release 0.0.4

David Francos Cuartero

January 08, 2015

Contents

Contents:

CHAPTER 1

Airoscript-ng

Airoscript-ng python complete implementation

- Free software: GNU GENERAL PUBLIC LICENSE 2
- Documentation: http://airoscript-ng.readthedocs.org/en/master

1.1 Features

- Dynamic aircrack-ng API generation (under airoscriptng.aircrack)
- Threaded execution
- Hackability assesment
- Scanning provides a list of best wireless hacking techniques
- Session control (for better process control)
- Wireless monitor interfaces are nicely handled and reused if neccesary
- XMLRPC server implementation

1.2 TODO

- Better parameter parsing & format for aircrack-ng parameters file, read them from manpages
- Implement attacks on airoscript-ng class
- Implement cracking (and control of it, once cracked stop all attacks against that network)
- Build a user interface (probably more than one)

CHAPTER 2

Installation

At the command line:

```
$ easy_install airoscriptng
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv airoscriptng
$ pip install airoscriptng
```

CHAPTER 3

Usage

To use Airoscript-ng in a project:

```
import airoscriptng
```

From there, you'll have access to session managers and airoscript basic object.

You can also launch main airoscript-ng XMLRPC server just by executing airoscriptng script directly or calling the binary:

```
airoscriptngxmlrpc
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/XayOn/airoscriptng/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

Airoscript-ng could always use more documentation, whether as part of the official Airoscript-ng docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/XayOn/airoscriptng/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *airoscriptng* for local development.

1. Fork the *airoscriptng* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/airoscriptng.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv airoscriptng
   $ cd airoscriptng/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 airoscriptng tests
   $ python setup.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7. Check https://travis-ci.org/XayOn/airoscriptng/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_airoscriptng
```

CHAPTER 5

Credits

5.1 Development Lead

- David Francos Cuartero <me@davidfrancos.net>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

CHAPTER 7

0.0.4 (2015-09-01)

- Bugfix on target handling
- Bugfix on basic cracking
- First attack (deauth) implemented
- Added lots of parameters to parameter json file

CHAPTER 8

0.0.3 (2015-09-01)

- Improved parameter handling
- Implemented basic cracking
- First attack scheduled for v0.4

CHAPTER 9

0.0.2 (2015-08-01)

- First usable thing, still no attacks
- “Hackability” property for aps
- Integrated clients on AP object
- External plugin support
- Reaver support
- XMLRPC Working
- General cleanup

CHAPTER 10

0.0.1 (2014-12-26)

- First release, monitor mode and scanning working

CHAPTER 11

airoscriptng

11.1 airoscriptng package

11.1.1 Subpackages

airoscriptng.capabilities package

Submodules

airoscriptng.capabilities.reaver module

class airoscriptng.capabilities.reaver.**main**(*parent*)
Bases: object

scan(*file_*)

Module contents

11.1.2 Submodules

11.1.3 airoscriptng.aircrack module

class airoscriptng.aircrack.**Aircrack**(*attributes={}*)
Bases: object

Exports a method foreach aircrack-ng binary that gets an OrderedDict with the desired parameters as argument.

Those parameters must match the ones specified in the json parameter file

callback(*result*)
Plain aircrack-ng just logs the output and calls custom user callback (hence this callback).

An aircrack-ng session handler can be used to extend this class to avoid having multiple aircrack-ng processes loose

launch(**args*, ***kwargs*)
Launch a new pool with one thread for that one process. Then we call the callback.

exception airoscriptng.aircrack.**AircrackError**
Bases: exceptions.Exception

Base aircrack exception

class airoscriptng.aircrack.**AircrackSession**(*attributes={}*)
Bases: airoscriptng.aircrack.Aircrack

Each session should have one or many aircrack-ng objects. An aircrack-ng object should be able to execute ONE of EACH aircrack-ng suite processes.

TODO

- **Add some exceptions, there are processes of what we might** want to have multiple instances running

callback(*result*)

•Remove the finished process from self.executing.

•Then execute the user-defined callback if exists.

•Otherwise just log output

execute(**args*, ***kwargs*)
Execute aircrack-ng command.

•Parse parameters

•Set up default callback

•Check that proccess is not executing already

•Then execute launch method.

executing = {}

class airoscriptng.aircrack.**Executor**(*command='airmon-ng'*, *_parameters={}*, *callback=False*, *shell=False*, *direct=False*)
Bases: object

Executor objects gets created to manage aircrack-ng execution.

It's called from a threadpoolexecutor, this way the future returns this as result, having control of the commands called and callback.

airoscriptng.aircrack.**parse_parameters**(*attributes*, *_parameters={}*, *command='airodump-ng'*)
Main aircrack-ng parameter parsing from the json file is done here.

TODO

- Automatically generate the json this feeds on

Parameter format is as follows:

```
{
    'main_command': {
        'name_in_airoscript' : ['--flag_in_aircrack', "default_value"]
    }
}
```

If default_value is True, it'll be assumed that is a flag that dont require an argument, and we want it by default enabled.

If it's false, the same will be assumed, but will by default disabled.

All parameters can be overriden

11.1.4 airoscriptng.airoscriptng module

class airoscriptng.airoscriptng.**Airoscript**
Bases: object

Main airoscript object. Contains not-that-direct aircrack-ng interaction methods and attacks

crack()
Launches aircrack-ng in infinite mode against current target. :TODO: This should probably be better managed by set_target.

end_scan()
We send a kill signal to airodump-ng As aircrack object is not aware of this, we must manually change the status

generic_dissasociation()
This does a generic dissasociation attack. Meaning that this attack is both useful on WEP and WPA.

This can be used both to get ARP replays and WPA handshake.

See : http://www.aircrack-ng.org/doku.php?id=deauthentication

on_scan_bumped(*pid*)
Callback to execute each time we rebump on scan(). This schedulles a re-rebump so it's made periodically. That's how we'll get periodically (each second) updated csv files

This implements the plugin system, calling on_after_scan

pids = {}

rebump(*pid*)
Launches sigint to a process. In airodump-ng this means updating the csv contents

scan(*options=OrderedDict()*)
Main scanning function. Launches airodump-ng, saves its pid, and makes itself xmlrpc representable.

This implements the plugin system, calling on_before_scan

class airoscriptng.airoscriptng.**AiroscriptSession**(*config={}*)
Bases: airoscriptng.airoscriptng.Airoscript

Basic airoscriptng session object. This is the basic airoscriptng object. Handles network interfaces. Main interaction with outer world will be here.

current_targets
Returns current targets (getter)

del_mon_iface()
Deletes own monitor interface from system (cleanup)

get_current_targets()
Parses airodump-ng's output file, creating a Target object for each AP found, with its currently detected clients.

get_mac_addr()
Return mac address of the interface

get_target()
Returns currently selected target, clean to send it via xmlrpc

list_wifi()
Returns a list of all the available wireless networks

mon_iface
Return current monitor interface name

set_mon_iface(*result*)
Sets monitor interface. Checks that final monitor interface is really what it should be.

set_target(*target*)
This way we only have to do something like self.target = current_targets[10] and it'll automatically make an object from it.

setup_wifi(*iface*)
Starts monitor mode interface and checks it's ok.

TODO

- Injection test.

target
Returns currently selected target (getter)

class airoscriptng.airoscriptng.**AiroscriptSessionManager**(*wifi_iface*)
Bases: object

Airoscript-ng session manager.

We can have multiple airoscript-ng sessions running on different interfaces

Each one will have an aircrack-ng session so we can execute only one process of each kind in each interface, to avoid collisions.

A session manager object should be used on xmlrpc sessions (but a airoscript session alone can be used too)

create_session(*name=False*, *sleep_time=2*, *scan_time=2*)
Create a AiroscriptSession object and assigns it to session_list

If no name provided it will take current time (used to create monitor wireless interface)

Note that *this does not setup monitor mode* so we have to call the session's setup_wifi method each time.

get_session(*session_name*)
XMLRPC method returning a specific session object.

class airoscriptng.airoscriptng.**Target**(*parent=False*)
Bases: object

Target object, this represents an access point

cleanup()
kill all related PIDS and clean them.

from_dict(*dict_*, *clients=[]*)
Do some magic, get only its clients from the client list, strip extra whitespace in properties, and get its hackability

get_hackability()
This assets a network hackability based on:

•Network power

•Network encryption

•WPS availability

•Dictionary availability

get_key()
XMLRPC function for key()

is_attack_finished()
Return if the attack is finished. This is a more complete check that is_attack_running, it has in account if the network has been cracked and if the attack has actually started.

is_attack_running()
Returns True if the ALL the attack processes are still executing This means that:

If part of the attack (I.E replaying) has stopped, will consider the attack finished.

If the processes don't die after the attack is successful, it wont consider the attack finished.

That's why we'll combine it with is_cracked and key

is_network_cracked()
If the network has been cracked we'll save the key to a key file for that specific target. This function just asks if the network has been cracked. Possibly not going to be used as network_key returns False if not cracked.

key
Return network key or False if network has not been cracked.

key_file

airoscriptng.airoscriptng.**airoscriptxmlrpc**()
Simple xmlrpc server for airoscriptsession. :TODO: - Make it multisession

airoscriptng.airoscriptng.**clean_to_xmlrpc**(*element*, *to_clean*)
Cleans certain properties from dict representation of given object

11.1.5 airoscriptng.broken module

airoscriptng.broken.**get_hackability_name**(*point*)
Return a more human-understandable name for a hackability statistic

11.1.6 airoscriptng.pluginmanager module

airoscriptng.pluginmanager.**load_plugins**(*config_file*)
This reads a config file of a list of plugins to load. It ignores empty lines or lines beginning with a hash mark (#). It is so plugin imports are more dynamic and you don't need to continue appending import statements to the top of a file.

airoscriptng.pluginmanager.**register**(**events*)
This decorator is to be used for registering a function as a plugin for a specific event or list of events.

airoscriptng.pluginmanager.**trigger_event**(*event*, **args*, ***kwargs*)
Call this function to trigger an event. It will run any plugins that have registered themselves to the event. Any additional arguments or keyword arguments you pass in will be passed to the plugins.

11.1.7 Module contents

CHAPTER 12

Indices and tables

- *genindex*
- *modindex*
- *search*

a

A

C

D

E

F

G

I

K

L

M

O

P

R

S

T