COMP 4203 Final Project

Title: Lightweight Solutions for Detecting Deauthentication Attacks
Authors: Matthew Nitschke, Steven Rhodes

Brief Description:

Comparing two research papers that explore different methodologies of detecting deauthentication attacks in WLAN/802.11 wireless networks.

Abstract:

As lots of devices communicate over wireless networks, staying connected can be something people take for granted. One way to deny people access to a network is by doing a deauthentication attack. In this paper, we discuss our findings on a lightweight solution for detecting a deauthentication attack, which is where an attacker takes advantage of unencrypted deauthentication frames and spoofs those packets to force a user to disconnect from the network. Our methods are developed from a research paper and then compared with a second paper's methodology to detect/defend against deauthentication attacks to determine if one has a superior solution.

Description of Domain:

Our chosen course topic for this project is Wireless Network Security; in particular security over 802.11/Wi-Fi networks. One security flaw existing within 802.11 networks is the deauthentication attack. This management frame attack takes advantage of the fact that deauthentication frames sent from a device on the network to the Accept Point – normally used to de-authenticate the device from the network – can be spoofed by other devices on the same network, effectively causing a denial-of-service (D.O.S.) attack by disabling the device from connecting to the Access Point. The reason this issue exists is due to the fact that the management frames are neither encrypted nor authenticated.

Description of the Research Question:

Our research question is to find a way to detect deauthentication attacks made towards a connected client and access point. This denial of service is done when an attacker spoofs the deauthentication and disassociation frames acting as if the client has left the server, thereby disconnecting the client from the access point. Since these frames are unencrypted and unauthenticated this can be done pretty easily by an attacker. The goal of this paper is to identify if there is a deauthentication attack on the client, alert the attacked device (e.g., the device running the detection program Script), and then compare this implementation with another paper's solution.

<u>Literature Review:</u>

The following papers were used for our project. Paper 1 is the paper chosen as the basis for our experiment, and paper 2 is used as a comparison:

<u>1)</u> Kumar & Singh (2019). A Lightweight Solution for Detecting Deauthentication Attack
<u>2)</u> Mital, Nguyen, Nguyen, Tran (2008). A Lightweight Solution for Defending Against Deauthentication/Disassociation Attacks on 802.11 Networks:

<u>Paper 1: A Lightweight Solution for Detecting Deauthentication Attack:</u>

Paper 1 explains the importance of wireless security as it is key for the user and network to have confidentiality, authentication and control. These principles are important as they are the security factors which determine who has access to the transmitted information and verify the correct users. It also explains the flaws of wlan. One of the big flaws is the MAC Access Control List, this list can contain both the SSID as well as the client MAC address which can be used in a deauthentication attack. Furthermore, the transfer of data is wireless which allows for attackers to spoof users and cause them harm. An example of a spoofing attack is the case of a deauthentication attack which is a kind of management frame attack involving the deauthentication frame. The way the attack is explained is that an attacker spoofs the MAC address of a client and then continually sends the deauthentication frames to the access point to keep disconnecting the user.

Moving ahead to the 4th part of the paper, this section goes over comparisons of other papers' methods for detecting deauthentication attacks; one such paper is paper 2 seen later in this section. The 5th part of the paper is “ARCHITECTURE” of the attacker, while the 6th part is “RESEARCH METHODOLOGY '' which goes through the process to detect the deauthentication attack. The attack in this paper was used as a model for our attack, as both use Kali Linux and its aircrack-ng tool to disconnect the user from their wireless network. This paper breaks down how to detect a deauthentication attack into 3 steps. The first being the design of the description of the architecture, languages, and any hardware required. The paper suggests the hardware to use is an Intrusion Detection system with embedded hardware, and to use Python as the script for the code. Also it mentions the requirement of having a wireless network adapter be compatible with Monitor mode. Having access to Monitor mode allows the deauthentication attack to occur and also for the Python Script to use PyShark to detect all packets passing through the device. The 2nd step describes how the detection program should work at a high level: by capturing all 802.11 frames and inspecting the packet contents. With the final step being the detection of the attack: by getting deauthentication frames, checking the difference in timestamps between the successive deauthentication frames, and then finally checking the reason code. If these timestamps are close together (under a certain predetermined time threshold), and the reason codes were the same, it would then confirm a deauthentication attack has occured. The paper offers pseudocode to assist reproduction of the paper’s results:

```
DetectDeauth(Scanned_Packets)
//This algorithm detects the De-Authentication attack. Here Scanned Packets is the
packets
//received while monitoring the network.
Output: Alert
check if management frames
check subtype
if(subtype == deauth) then
extractmactime
if(mactime[intervals] is constant) then
        check reason_code
         if(reason_code == same)
                 alert()
         else if ( data_frames_after_deauth )
                alert()
        end if
else if ( data_frames_after_deauth )
        alert()
end if
else
return
end if
```

Figure 1: Pseudocode for Paper 1's Detection Algorithm

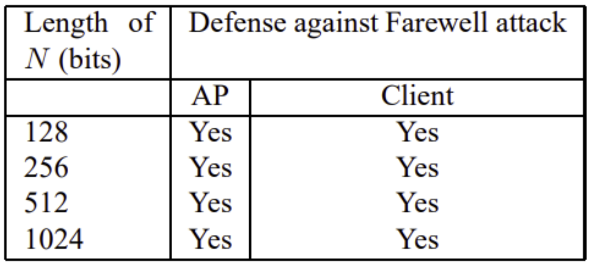
The final part of the paper discusses the "RESULT AND DISCUSSION" and "CONCLUSIONS" made from the paper. In this paper the authors compare MAC timestamps and reason codes to confirm a deauthentication attack. This was done with five clients to confirm the attacks. They also compare this detection method to others to confirm that the method used is for a single client and it uses combinations of parameters to decrease the false positive rate. The conclusion goes over how the authors were able to detect wireless deauthentication attacks with accuracy.

Paper 2: A Lightweight Solution for Defending Against Deauthentication/Disassociation Attacks on 802.11 Networks:

Paper 2 "A Lightweight Solution for Defending Against Deauthentication/Disassociation Attacks on 802.11 Networks" has 5 parts; the intro, a description of how deauthentication attacks (called farewell attack in the article) work, the paper`s solution, results and conclusion. The intro explains the process on how a device connects to an access point and how for the device to disconnect there needs to be a disassociation/deauthentication frame. It goes on further to explain that since the disassociation/deauthentication frame is not encrypted it can easily be spoofed, thereby making a device vulnerable to a deauthentication attack. In this paper, the authors discuss a new way to try and defend against a deauthentication attack.

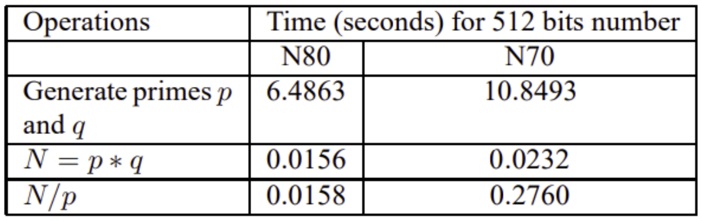
In section 2, it is discussed how a deauthentication attack works and the exploits it uses so an attack is successful. It goes into more detail on how the deauthentication frame is not encrypted, thus making it easy for an attacker to spoof a user. One of the things needed for this attack is to spoof the source MAC address of any device, which can be done with devices such as Airsnarf and MAC Changer. The paper also goes over some solutions to counteract this attack. One of the solutions being the Reverse Address Resolution Protocol (RARP) to detect spoofed frames. The authors`s solution is using a protocol they made called the “Letter-envelop” protocol. This protocol works by using a formula of N = p $*$ q, such that “p” and “q” are two large primes (too big to compute). When a device is being authenticated, N is sent to the server and when the device wants to disconnect it compares the sent N to p (p|N). If p and n correspond then the device is deauthenticated accordingly. The formula of N = p $*$ q makes it more difficult for an attacker to spoof deauthentication frames as now there are keys involved in the deauthentication process.

Section 4 and 5 of the paper explains the results from experiments they conducted and overall conclusions that can be made. The experiments were conducted as the author's first simulation of a deauthentication attack on a device. An issue addressed in the paper is the current device driver that was modified only works on wireless devices with the Atheros chipset running on Linux. This attack would then be monitored by CommView. Once they knew the attack was working, the “Letter-envelop” protocol would be used to defend. The authors then tested different sizes of primes p and q used (64, 128, 256 and 512 bits) and time of deployment.



Length of N (bits)	Defense against Farewell attack	
	AP	Client
128	Yes	Yes
256	Yes	Yes
512	Yes	Yes
1024	Yes	Yes

Table 2. Experimental results



Operations	Time (seconds) for 512 bits number	
	N80	N70
Generate primes p and q	6.4863	10.8493
$N = p * q$	0.0156	0.0232
N/p	0.0158	0.2760

Table 3. Microbenchmarking results

FIgure 2: Comparing Length of N with Defense Against Farewell Attacks

The conclusion that was made is that the “Letter-envelope” protocol can both work and be efficiently implemented to defend against deauthentication attacks, with the main concern being the time it takes to create a 512-bit number.

Technology:

- Language: Python
- O.S.: Linux (Kali)
- Libraries: PyShark
- Other:
    - Wireless USB adapter to enable monitor mode (TP-Link adapter)
    - Aircrack-ng, airmon-ng: for de-authentication attacking
    - Wireshark
    - An access point (AP)
    - A device to perform deauthentication on:
        - PC connected to same wireless network/AP
        - Cell phone

Methodology & Algorithm:

Framework/Background: To reproduce the work of the chosen first paper (Kumar & Singh), a wireless network adapter was needed (see references below for a link to the adapter we used). This device was required as the current wireless interface cards on our desktops/laptops didn't provide monitor mode. Monitor mode is a setting that can be enabled on a wireless network interface that allows all packets from all devices passing through the computer to be monitored, as opposed to just packets relevant to the computer. To enable monitor mode on the wireless adapter, Kali Linux was used; Linux allowed for easier access to enabling monitor mode as well as access to deauthentication attack tools and access to Wireshark in monitor mode. To switch to Monitor mode, the tool "airmon-ng" was used. The process for switching to monitor mode is as follows:

1. Run "sudo airmon-ng check kill" to kill any background processes that may inhibit Monitor mode from being enabled

2. Run "sudo airmon-ng start wlan0" to switch to Monitor mode. Note: replace wlan0 with the name of your wireless interface if it is different.

Deauthentication Attack: To perform the deauthentication attack, first the MAC address of both the Access Point and the device you wish to attack must be found. Both can be found using the "airodump-ng" tool. To get the MAC address of your access point run "airodump-ng wlan0" and pick from the BSSID column the MAC address and channel corresponding to the wireless network name you wish to attack. Next, use this MAC address and run the command "airodump-ng - - bssid <AP MAC address> - - channel <Channel> wlan0." This will display the MAC address of the devices connected to the Access Point. With this information we can perform our deauthentication attack using "aireplay-ng". The command to perform the attack is "aireplay-ng - - deauth 4 -a <AP MAC address> -c <Device to attack MAC address> wlan0"; the

“4” can be changed as it determines how many packets for the attack to send. With this command run, deauthentication frames will be sent to the device. When tested, we were able to disconnect the chosen devices from the network. Using WireShark, deauthentication attack frames were visible.

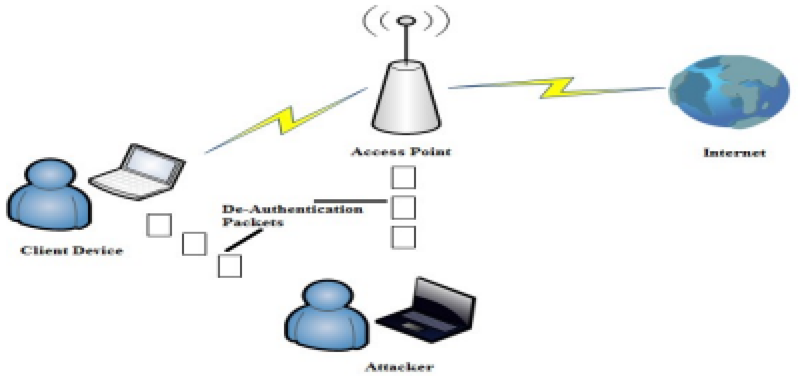


Figure 1 De-Authentication Attack

Figure 3: Architecture Surrounding a Deauthentication Attack

<u>Detection Algorithm:</u> To detect this deauthentication attack we used the algorithm/concepts from our first paper (Kumar & Singh) to write a Python program, with the help of the PyShark library. PyShark allowed our program to access sniffed packets data from Wireshark in Monitor mode. To use our program, we ran the Python script on Linux as well, as we needed access to Monitor mode to detect the correct packets coming in through PyShark, and we only had access to one wireless network adapter. Our algorithm is as follows:

1. Sniff packets continuously on access point while waiting for an attack
2. If a sniffed packet contains a “Type/Subtype” of “0x000c” (i.e., “Deauthentication”) then inspect the packet for more information
    a. If this is the first deauthentication frame, then capture the reason code and timestamp for later comparison
3. Check the timestamp differences between consecutive “Deauthentication” frames/packets.
4. If the timestamp difference is below the threshold, compare the reason code of the current deauth-packet with previous deauth-packet to see if they are the same.
5. If 5 consecutive packets are each a deauthentication packet, have a timestamp below the threshold and have the same reason code, then ***alarm the user of a deauthentication attack***.
6. Otherwise, keep sniffing packets

The values chosen for the timestamp difference threshold and the number for consecutive deauthentication packets to be classified as an attack were somewhat arbitrary. The timestamp threshold was meant to be as small as possible, and from observing the packet contents from the attacks, 0.2 seconds was chosen as a sufficient value. For the number of deauthentication frames required to signal an attack, 5 deauthentication frames was chosen, as this seemed to be a reasonable amount.

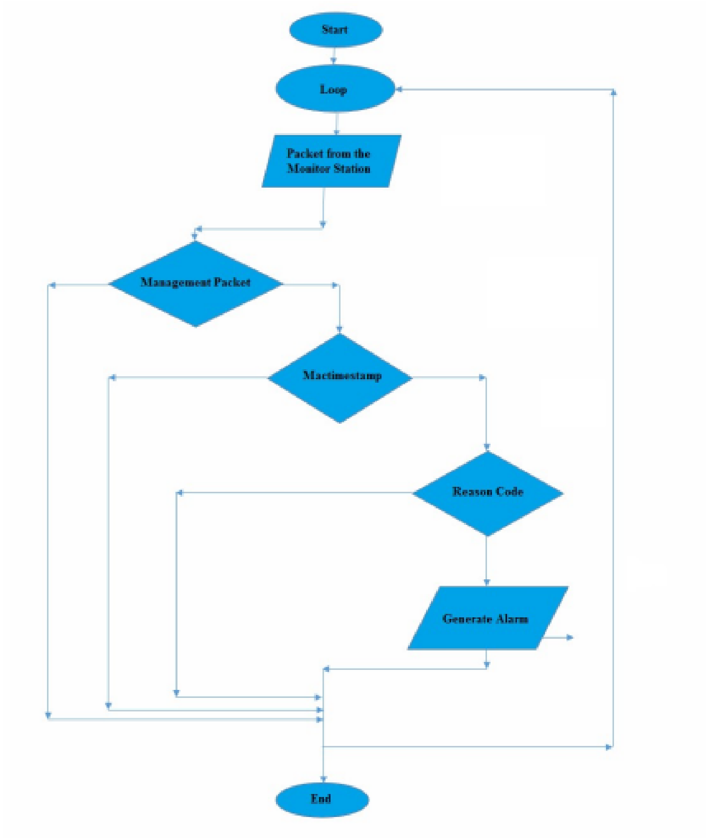


Figure 4: Detection Algorithm

<u>Results & Analysis:</u>
When implemented, the algorithm outlined above was able to wait for an attack and then detect when an attack was occuring.

Figure 5 shows the data from the deauthentication packets, specifically how the timestamp changes with each subsequent deauthentication frame. When the deauthentication attack occurs, the change in the timestamps between each deauthentication frame is small and consistent, as was expected.

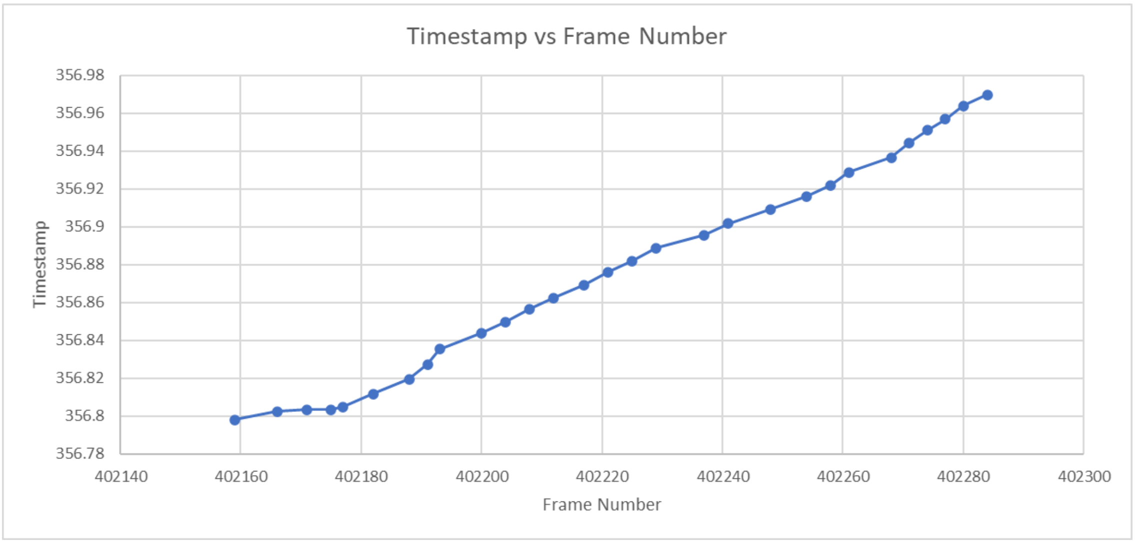
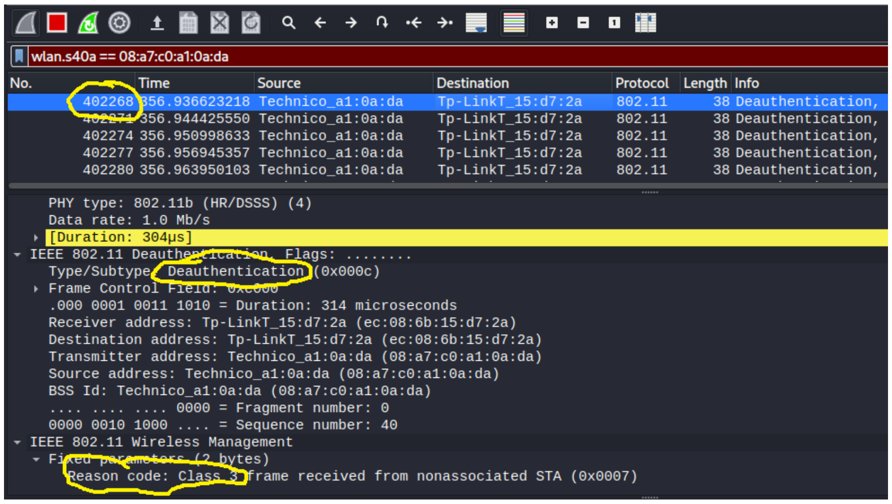


Figure 5: Time Between Different Deauthentication Frames

Figures 6-8 show the WireShark deauthentication frames. The reason code for each is the same: the reason code of 3.

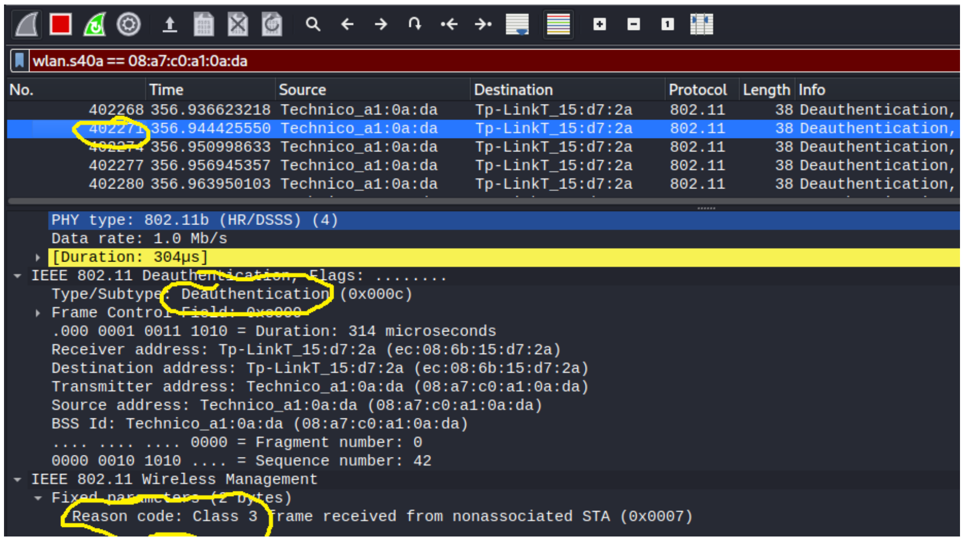


```
wlan.s40a == 08:a7:c0:a1:0a:da

No.     Time          Source               Destination          Protocol Length Info
 402268 356.936623218 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11     38 Deauthentication,
 402271 356.944425550 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11     38 Deauthentication,
 402274 356.950998633 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11     38 Deauthentication,
 402277 356.956945357 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11     38 Deauthentication,
 402280 356.963950103 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11     38 Deauthentication,

    PHY type: 802.11b (HR/DSSS) (4)
    Data rate: 1.0 Mb/s
  ▸ [Duration: 304µs]
▾ IEEE 802.11 Deauthentication, Flags: ........
    Type/Subtype: Deauthentication (0x000c)
  ▸ Frame Control Field: 0xc000
    .000 0001 0011 1010 = Duration: 314 microseconds
    Receiver address: Tp-LinkT_15:d7:2a (ec:08:6b:15:d7:2a)
    Destination address: Tp-LinkT_15:d7:2a (ec:08:6b:15:d7:2a)
    Transmitter address: Technico_a1:0a:da (08:a7:c0:a1:0a:da)
    Source address: Technico_a1:0a:da (08:a7:c0:a1:0a:da)
    BSS Id: Technico_a1:0a:da (08:a7:c0:a1:0a:da)
    .... .... .... 0000 = Fragment number: 0
    0000 0010 1000 .... = Sequence number: 40
▾ IEEE 802.11 Wireless Management
  ▾ Fixed parameters (2 bytes)
      Reason code: Class 3 frame received from nonassociated STA (0x0007)
```

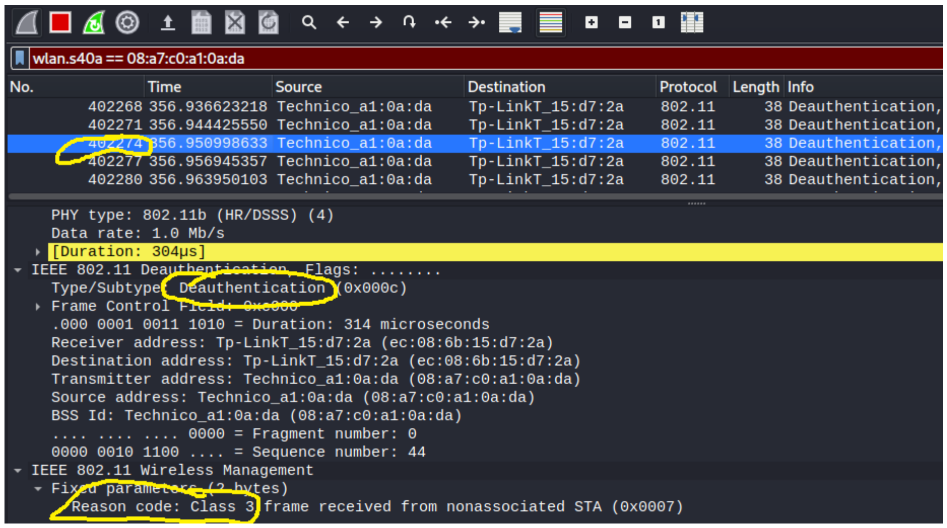
Figure 6: A Deauthentication Frame



```
wlan.s40a == 08:a7:c0:a1:0a:da
No.      Time             Source               Destination          Protocol  Length  Info
  402268 356.936623218 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11        38 Deauthentication,
  402271 356.944425550 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11        38 Deauthentication,
  402274 356.950998633 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11        38 Deauthentication,
  402277 356.956945357 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11        38 Deauthentication,
  402280 356.963950103 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11        38 Deauthentication,

    PHY type: 802.11b (HR/DSSS) (4)
    Data rate: 1.0 Mb/s
  › [Duration: 304µs]
▾ IEEE 802.11 Deauthentication, Flags: ........
    Type/Subtype: Deauthentication (0x000c)
  › Frame Control Field: 0xc000
    .000 0001 0011 1010 = Duration: 314 microseconds
    Receiver address: Tp-LinkT_15:d7:2a (ec:08:6b:15:d7:2a)
    Destination address: Tp-LinkT_15:d7:2a (ec:08:6b:15:d7:2a)
    Transmitter address: Technico_a1:0a:da (08:a7:c0:a1:0a:da)
    Source address: Technico_a1:0a:da (08:a7:c0:a1:0a:da)
    BSS Id: Technico_a1:0a:da (08:a7:c0:a1:0a:da)
    .... .... .... 0000 = Fragment number: 0
    0000 0010 1010 .... = Sequence number: 42
▾ IEEE 802.11 Wireless Management
  ▾ Fixed parameters (2 bytes)
      Reason code: Class 3 frame received from nonassociated STA (0x0007)
```

Figure 7: A Deauthentication Frame

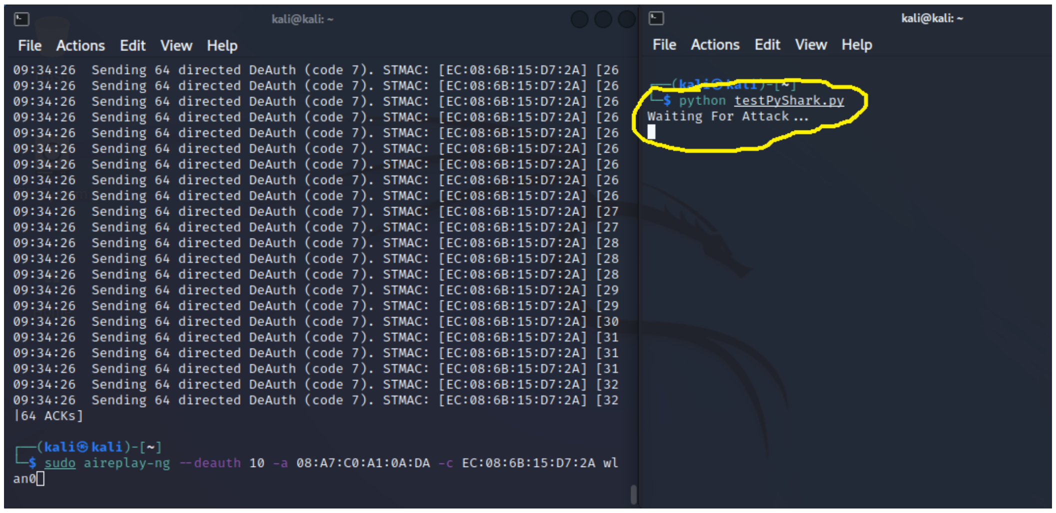


```
wlan.s40a == 08:a7:c0:a1:0a:da
No.      Time             Source               Destination          Protocol  Length  Info
  402268 356.936623218 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11        38 Deauthentication,
  402271 356.944425550 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11        38 Deauthentication,
  402274 356.950998633 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11        38 Deauthentication,
  402277 356.956945357 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11        38 Deauthentication,
  402280 356.963950103 Technico_a1:0a:da    Tp-LinkT_15:d7:2a    802.11        38 Deauthentication,

    PHY type: 802.11b (HR/DSSS) (4)
    Data rate: 1.0 Mb/s
  › [Duration: 304µs]
▾ IEEE 802.11 Deauthentication, Flags: ........
    Type/Subtype: Deauthentication (0x000c)
  › Frame Control Field: 0xc000
    .000 0001 0011 1010 = Duration: 314 microseconds
    Receiver address: Tp-LinkT_15:d7:2a (ec:08:6b:15:d7:2a)
    Destination address: Tp-LinkT_15:d7:2a (ec:08:6b:15:d7:2a)
    Transmitter address: Technico_a1:0a:da (08:a7:c0:a1:0a:da)
    Source address: Technico_a1:0a:da (08:a7:c0:a1:0a:da)
    BSS Id: Technico_a1:0a:da (08:a7:c0:a1:0a:da)
    .... .... .... 0000 = Fragment number: 0
    0000 0010 1100 .... = Sequence number: 44
▾ IEEE 802.11 Wireless Management
  ▾ Fixed parameters (2 bytes)
      Reason code: Class 3 frame received from nonassociated STA (0x0007)
```

Figure 8: A Deauthentication Frame

By checking the deauthentication frames for the timestamps and reason codes, as seen in the figures above, we were able to classify and alert a device whether they were experiencing a deauthentication attack. In our program the user will be presented with a “Waiting for Attack…” message (Figure 9), up until the above conditions are met, resulting in the following alert in Figure 10:

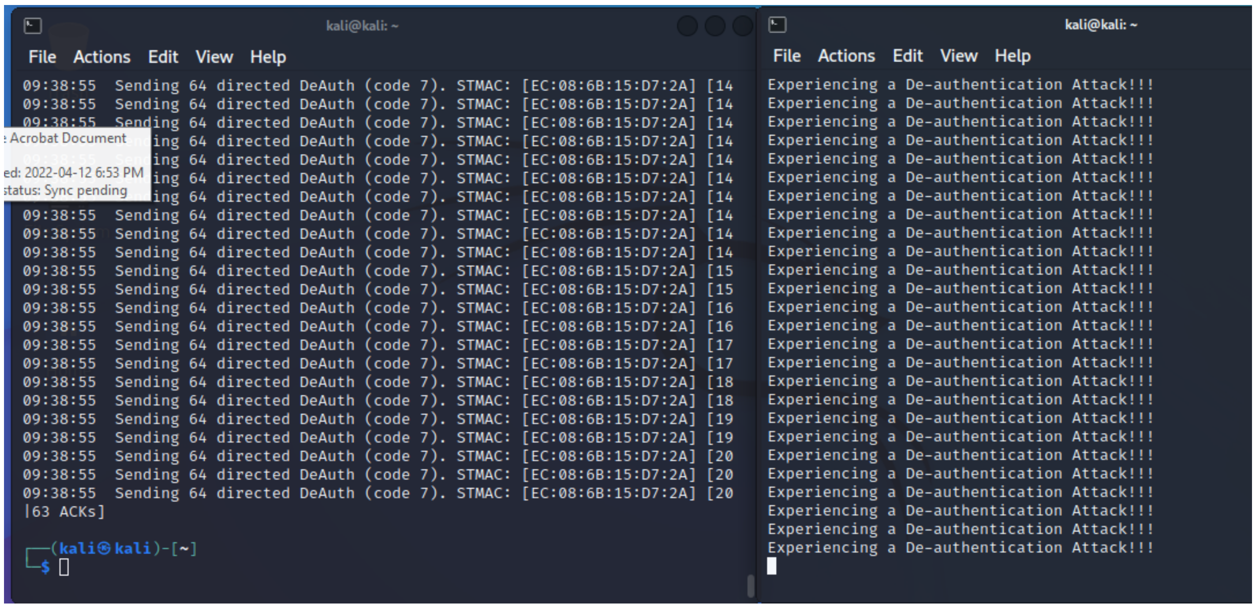


```
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [26
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [26
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [26
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [26
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [26
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [26
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [26
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [26
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [26
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [27
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [27
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [28
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [28
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [28
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [29
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [29
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [30
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [31
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [31
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [31
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [32
09:34:26  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [32
|64 ACKs]

┌──(kali㉿kali)-[~]
└─$ sudo aireplay-ng --deauth 10 -a 08:A7:C0:A1:0A:DA -c EC:08:6B:15:D7:2A wl
an0
```

```
┌──(kali㉿kali)-[~]
└─$ python testPyShark.py
Waiting For Attack ...
```

Figure 9: Detection Program Output Before an Attack



```
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [14
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [14
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [14
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [14
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [14
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [14
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [14
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [14
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [14
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [14
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [15
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [15
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [16
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [16
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [17
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [17
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [18
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [18
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [19
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [19
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [20
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [20
09:38:55  Sending 64 directed DeAuth (code 7). STMAC: [EC:08:6B:15:D7:2A] [20
|63 ACKs]

┌──(kali㉿kali)-[~]
└─$ 
```

```
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
Experiencing a De-authentication Attack!!!
```

Figure 10: Detection Program Alerting of an Attack

<u>Comparison:</u>

<u>Comparing our solution with paper one:</u> "A LIGHT WEIGHT SOLUTION FOR DETECTING DEAUTHENTICATION ATTACK," is what our work is mostly based off of. In this paper we both used Kali Linux and its aircrack-ng tool to simulate a deauthentication attack onto a device, instead of using an Intrusion Detection system; using Kali Linux to perform the attack, and detect was easier and sufficient without purchasing more hardware. For detecting the attack the paper used a python extension called PyShark, which we also decided to use. In paper one, to detect a deauthentication attack they used the reason code and MAC timestamp; the reason code was used for detecting the type of packet and timestamp was to measure time apart from each message. Our program used the same methods as above but there is one big difference: instead of using MAC timestamps we used regular timestamps, as this was sufficient to determine the difference between the times the deauthentication frames were sent. The check for the deauthentication subtype (i.e., checking the management frame) and reason codes remained the same. These are the main similarities and differences between paper one and our project.

Comparing paper one with paper two: Paper one, “A LIGHTWEIGHT SOLUTION FOR DETECTING DEAUTHENTICATION ATTACK" was made for detecting deauthentication attacks. While Paper Two, “A Lightweight Solution for Defending Against Deauthentication/Disassociation Attacks on 802.11 Networks” was made for defending against deauthentication attacks. Although both work covers slightly different topics, there is still some overlap in detection of attacks. This mainly comes into play when executing a deauthentication attack. In paper one, they used kali linux with its aircrack-ng tool and SCAPY to simulate deauthentication attacks. In paper two, they discuss using Airsnarf and other MAC changing devices to simulare their attacks. Furthermore, paper one used PyShark to detect our deauthentication packets while paper two used CommView to detect their deauthentication packets. It is also important to know that both attacks would consist of one client and an attacker with only one access point.

Where the two papers can compare is in the detection of the attack. Paper 1 suggests that paper 2’s solution is insufficient as it requires specific hardware (a wireless device with Atheros Chipset run on Linux) and also lacks access to an open source implementation of its solution. We found that by implementing paper 1’s methodology also required hardware (the proper wireless adapter with monitor mode capabilities). If we were to have used the Intrusion Detection system suggested in paper 1, we would have also required some specialized hardware that used embedded systems. In any case, specificalized hardware would be required, making the detection of such attacks partially inaccessible to users without that hardware. Therefore, in terms of comparing paper 1 and 2 on the basis of hardware requirements, deciding between the two papers is not clear.

The better choice would depend on the hardware a user currently has, and what they can access. Another criticism offered by paper 1 of paper 2’s methodology, is the lack of open source implementation into their solution. Paper 2’s methodology is arguably more advanced, requiring some knowledge of Cryptography. To a user choosing to implement either paper 1 or paper 2’s solution, paper 1 would be easier to implement because the concepts are easier to understand (e.g., comparing reason codes, timestamps, etc.) and also paper 2 offers no clear implementation of the theory; a user would have to take a more complicated theory and develop a program with less help. In comparison, paper 1 does offer an algorithm for their solution; they offer a pseudocode for the reader to use. Paper 1’s criticism of paper 2 seems justified and the authors attempt to ameliorate this issue by giving access to their pseudo-implementation. Since there is pseudocode from a development perspective (and developer’s understanding), paper 1’s solution is likely more accessible. Since both papers offer robust solutions for detecting and, in the case of paper 2, preventing deauthentication attacks, one way to distinguish between the two papers is accessibility. Therefore due to paper 1’s work being more accessible, that would make it the superior solution between the two papers.

Conclusion:

In conclusion, we were able to complete, detect and record the results of a deauthentication attack on a device based on the detection method replicated from the work of paper 1. Although both methods were slightly different – both did detection and another had additional prevention features -- both methods had the same successful end results when it came to detecting a deauthentication attack. Comparing and contrasting both methods to each other we found that, while both were successful and offered robust solutions, paper 1 would be the recommended solution because of its additional access to an open-source implementation which makes it more accessible to developers.

Work Contributed By Each Member:

- Steven:
    - Purchased wireless network adapter and performed deauthentication attacks successfully
    - Wrote Python code file
    - 50% of report
    - Contributed in discussions to get proper Methodology & Algorithm
- Matthew:
    - Purchased wireless network adapter and performed deauthentication attacks successfully
    - Performed, recorded, and edited the demonstration video
    - 50% of report
    - Contributed in discussions to get proper Methodology & Algorithm

References:

Kumar & Singh (2019). A Lightweight Solution for Detecting Deauthentication Attack[sic]. https://aircconline.com/ijnsa/V11N1/11119ijnsa02.pdf

Mital, Nguyen, Nguyen, Tran (2008). A Lightweight Solution for Defending Against Deauthentication/Disassociation Attacks on 802.11 Networks. https://www.researchgate.net/publication/221092095_A_Lightweight_Solution_for_Defending_Against_DeauthenticationDisassociation_Attacks_on_80211_Networks

AC600 High Gain Wireless Dual Band USB Adapter. TP-Link. Accessed March 27, 2022. **https://www.tp-link.com/us/home-networking/usb-adapter/archer-t2u-plus/**

G0t mi1k. Kali Inside VirtualBox (Guest VM). Kali. Updated on February 14, 2022. Accessed on March 27, 2022. https://www.kali.org/docs/virtualization/install-virtualbox-guest-vm/