# Real-World WiFi Security Inadequacies

Prepared for: Dr. Michel Barbeau
Prepared by: Nicholas Osborne, Michael Nemat

April 6, 2010

# Table of Contents

This page has been intentionally left blank.

# Introduction

## Background

In the last decade there has been wide adoption of wireless access points in the both residential and commercial environments. You can find wireless access points in just about any location where people congregate and spend time. Homes, schools, airports, hotels, offices; wireless access points are everywhere.

Security has always been a very serious issue with wireless networks, in that physical access is no longer required to access the network, if you are within range of the radio signal you can interact with the network.

Since 1997, 3 popular encryption formats have been introduced to tackle this problem, each with their own limitations. The first protocol introduced was WEP, in 1997, which used a 64 or 128bit HEX value to act as an encryption key. Unfortunately the limitations of WEP were quickly discovered and it was depreciated and replaced in 2003 with WPA and later 2004 with WPA2. Unlike WEP, WPA uses a 256bit key entered either as a string of 64 hexadecimal digits, or as a passphrase of 8 to 63 printable ASCII characters. If ASCII characters are used, the 256 bit key is calculated by applying the PBKDF2 key derivation function to the passphrase, using the SSID as the salt.

While WPA technologies are far more secure, hardware support is required and legacy hardware is often not compatible. Combined with the risk-understanding of the masses means that many wireless networks are still unencrypted or are still configured to use WEP encryption. Though WPA is more secure, the common use of a dictionary word as passphrase means that the encryption can be brute forced using dictionary attacks in many cases, which makes WPA/WPA2 similarly trivial as WEP when it comes to defeating it.

## Definition of the problem

Determine a wireless security breakdown of networks in both commercial and residential areas of Ottawa. Discern the liability of each encryption type.

## Summary of the result

From our wireless site surveys, we found that in both residential and commercial areas, roughly half of the networks are using strong encryption (WPA/WPA2), the other half are using weak and no encryption. Our simulations showed that WEP encryption does not offer any additional security over no encryption and we were able to break into a WEP network in less than 5 minutes with little effort. Additionally, while WPA/WPA2 is still impossible to crack (cryptographically secure), weak pass-phrases such as dictionary words or short string lengths (less than 9 characters) can be brute forced in a reasonable amount of time (especially when aided by a GPU).

## Outline of the report

1. Collect information on what type of wireless networks are being used in both business and residential areas and their level of security.

2. Demonstrate how easily WEP networks can be compromised.

3. Demonstrate how WPA/WPA2 networks  with a weak pass-phase can be compromised.

4. Discuss examples of how a compromised network can be a severe security risk.

5. Practicality of compromising a WPA/WPA2 network with a strong pass-phrase.

# Wireless Network Site Surveys

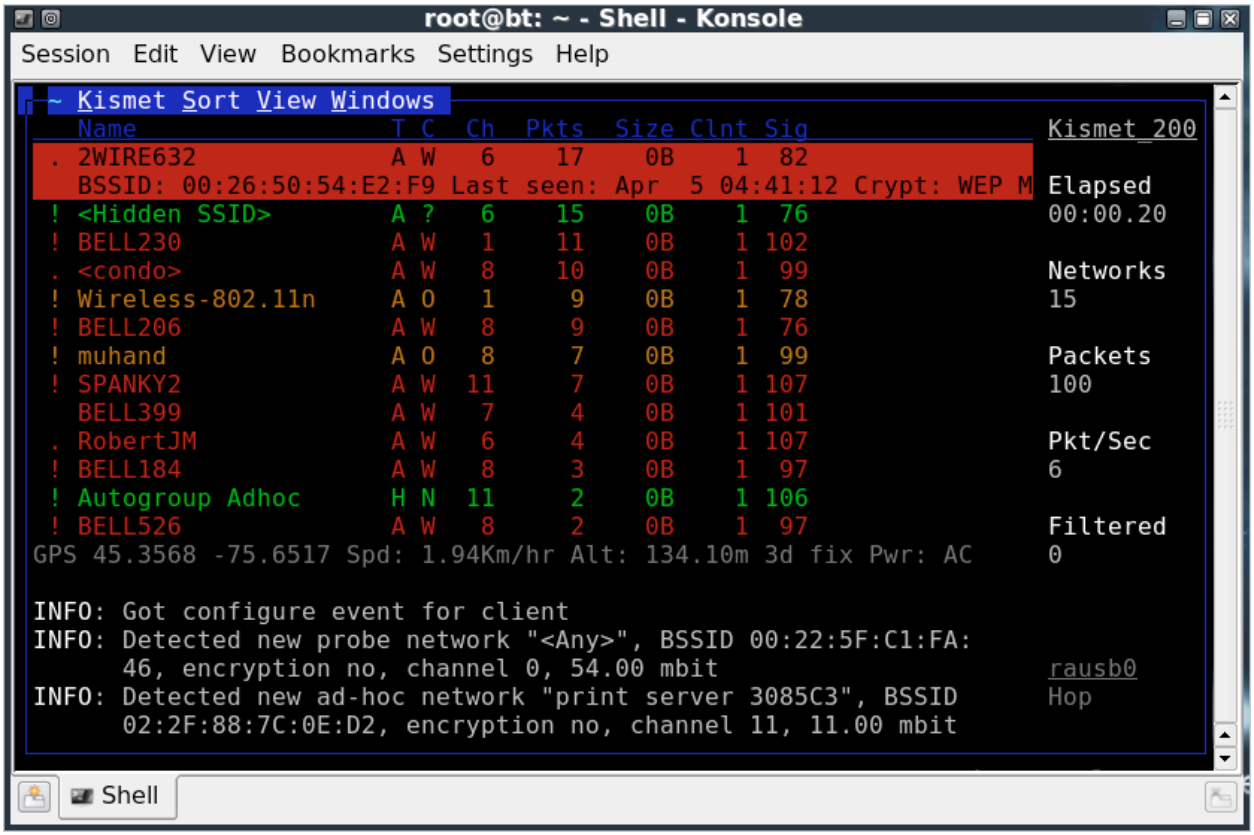
## Detailed Context and Background Information

Anyone with a laptop or a modern smartphone has enjoyed the benefits of insecure wireless networks. However, the prevalence of unencrypted (or poorly encrypted) wireless networks in North American residential and commercial areas is shocking. Someone with malicious intent can intercept privileged information from a wireless network even from a few hundred metres away with the right hardware. Whether this information is personal information or privileged commercial information (trade secrets etc...), it should still be kept secure. Furthermore, even if the information itself isn't an issue, the network is. An insecure network opens a door for an intruder to gain access to the network. Once they are on the network, they may choose to attack internal resources or further compromise infrastructure unrelated to the wireless network itself. Regardless of the methodology, an unencrypted network is a free pass for a malicious person to gain virtually unrestricted access to resources or information, and a network encrypted with WEP or a weak WPA passphrase will only stand in their way for a matter of minutes. There is also a myth that SSL is enough to keep an unencrypted network safe. This could not be further from the truth. Firstly, SSL does absolutely nothing to provide link-layer security to the network itself. A malicious user can still access the network and map it out, or attempt to compromise internal infrastructure regardless of whether SSL is being used to protect information. Secondly, many users are unaware of how much information is sent across the internet in an unencrypted fashion without SSL, and would likely not feel comfortable if this information was intercepted. Thirdly, SSL (HTTPS specifically) can be easily compromised through the use of some relatively new and clever man-in-the-middle attacks, namely SSLStrip by Moxie Marlinspike [1]. Given these realizations, we feel it is important to perform site surveys and generate statistics to look at real-world wireless security implementations and how they are lacking.

## Methodology

### Wireless site surveys of commercial and residential areas

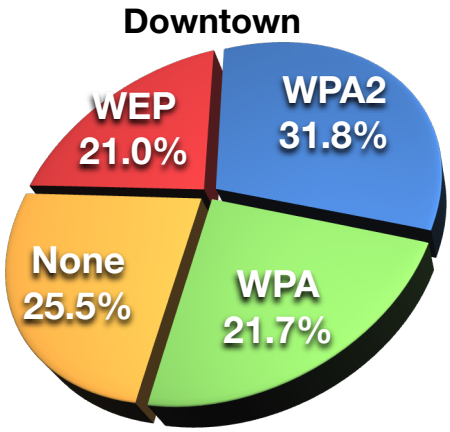
We will perform site surveys of downtown Ottawa and Old Ottawa South (the residential area near Carleton University) to generate maps and statistical information about the real-world implementation of wireless security. We will use the popular Linux survey tool Kismet, along with a bluetooth GPS and a USB wireless card to facilitate data collection. Once we have the data, we will run it through some open source tools to generate maps in Google Earth. We will also write an application to parse the data and generate statistical metrics, which surprisingly are not already provided by the available tools. Two modifications have been made to the open-source perl tool “kisgearth”. First, we modified it to use the color scheme described in our results. Second, we added exception handling so that malformed data (corrupt data, strange characters in SSIDs, etc) would not break the application and prevent output for good data from being generated. The screenshot below is of Kismet when performing site surveys with a GPS. Note the text below the list of networks which starts with GPS and has the current GPS co-ordinates as well as the current speed, altitude, and GPS fix status.



# Results

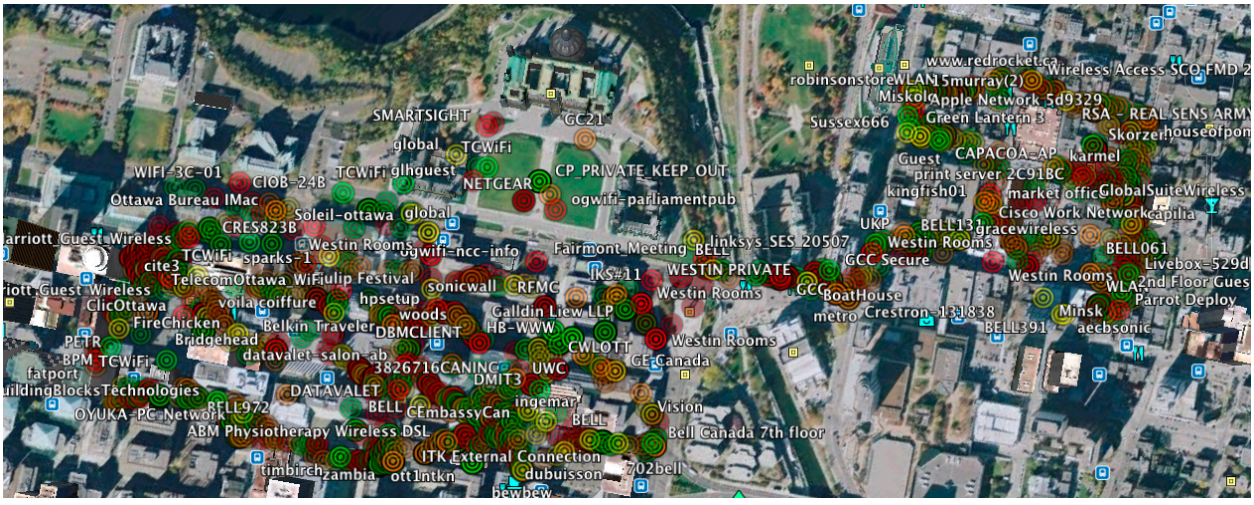
## Downtown

We conducted a site survey of the main downtown area, covering Kent (west) to Elgin (east) and Laurier (south) to Wellington (north). Additionally we mapped Parliament Hill and the Byward Market. This area was then plotted into Google Earth for data verification and to ensure that we had full coverage of our outlined search area. In total 1383 wireless networks were found and logged with GPS data. The data was then run through a small java application we wrote to generate statistical data on which type of network encryption was used.



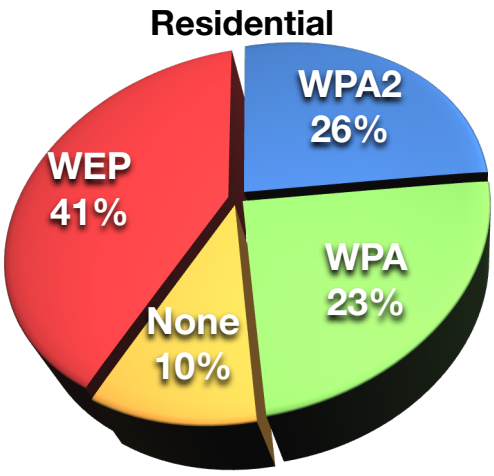

| Type | Number |
|---|---|
| WPA2 | 440 |
| WPA | 300 |
| None | 353 |
| WEP | 290 |

Only 54% of the surveyed networks were found to be using strong encryption (WPA/WPA2), while an alarming 47% had weak (WEP) or no encryption whatsoever.



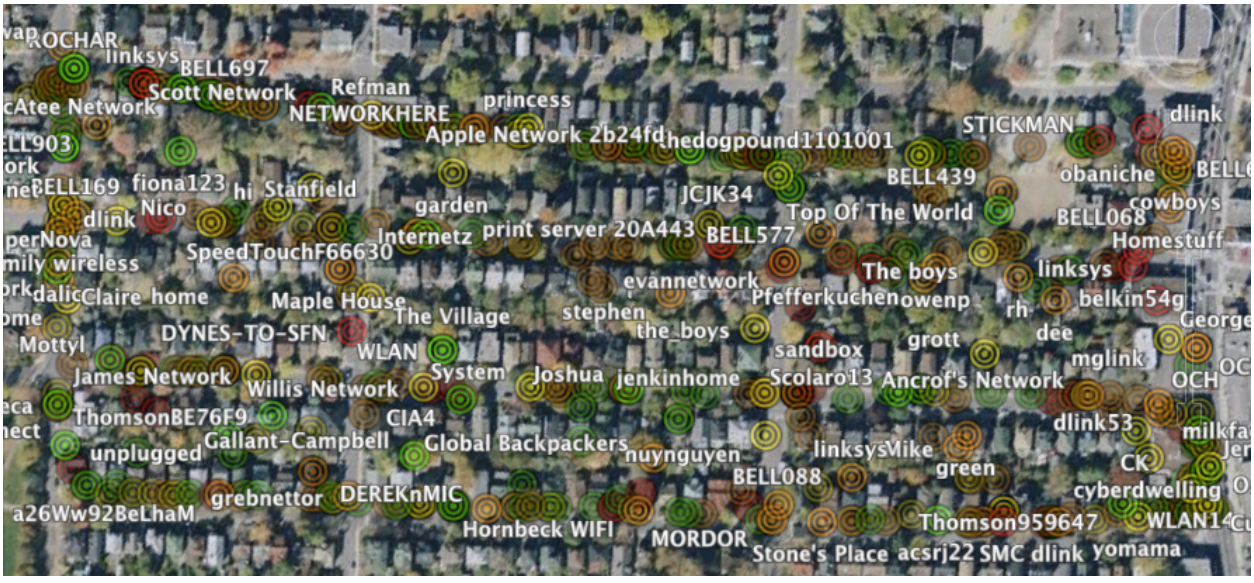
## Residential

We also surveyed a portion of Old Ottawa South, near Carleton, defined by Seneca (West) to Bank (East) and Cameron (South) to Sunnyside (North). This area was then plotted into Google Earth for verification. A total of 448 wireless networks were detected, scanned, and plotted with GPS data. Again the small java application was used to generate statistical data.




| | |
|---|---|
| WPA2 | 116 |
| WPA | 102 |
| None | 46 |
| WEP | 184 |

In the residential area surveyed, 90% of networks were using some form of encryption. The weaker WEP accounted for almost half of access points, many of which being Bell ADSL internet/router gateways which have WEP enabled by default.



# WEP Encryption

## Detailed Context and Background Information

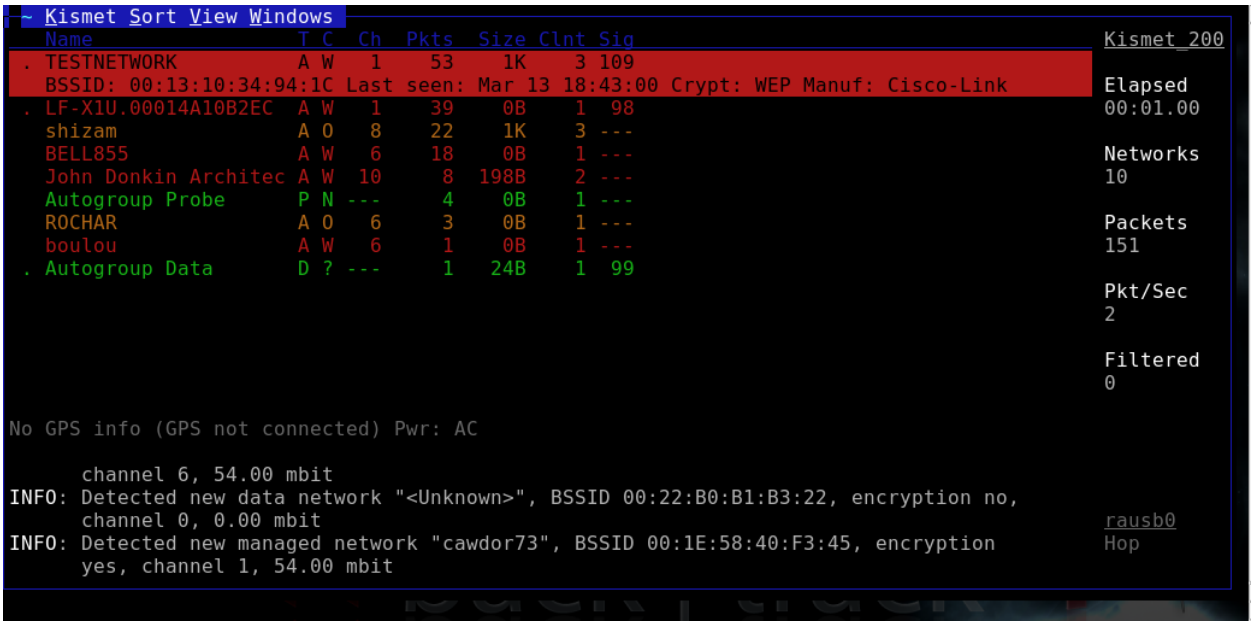
There are three commonly used security implementations for WiFi: WEP, WPA, and WPA2. WEP was introduced in 1997 and made use of the RC4 encryption algorithm. By 2001, it was discovered that the Initialization Vector (IV) of a WEP frame, designed to prevent repetition, has a 50% likelihood of repeating after 5000 packets. It was not long enough to sufficiently prevent repetition. With enough packets, the probability of having repeated IVs approaches 100%. This allows a related key attack to be performed in order to defeat WEP regardless of the strength of the key/passphrase.

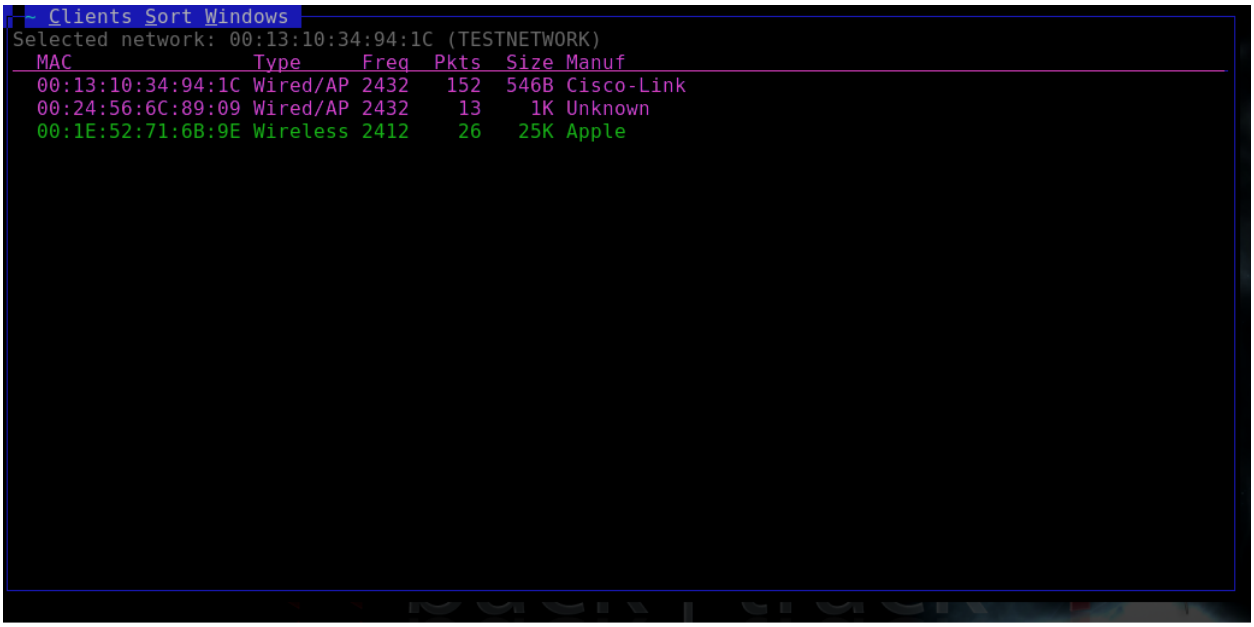
## Methodology

WEP can be broken with relative simplicity. We will make use of several tools which will be discussed in depth further on. We need to survey the networks to gather important information such as hardware addresses of the infrastructure as well as associated clients. While there are attacks on WEP that can succeed with no associated clients on the network, they tend to be unreliable and thus we consider breaking WEP to be impractical if there are no clients on the network. Keeping the real-world usage of wireless networks in mind, having the restriction of requiring clients on the network is completely reasonable in real-world scenarios. Once we have information on the associated clients, we can begin logging traffic to/from the network. If this traffic is sufficient to generate ~100,000 packets in a reasonable amount of time, we will simply wait for this to occur. If there is not sufficient traffic on the network, we will try to capture a known packet like ARP, and replay it hundreds of times a second back to the access point in order to generate data, and by doing so, generating duplicate IVs. In order to get an ARP packet to replay, we can either wait for one to occur through normal usage (which may or may not be very quick depending on the network), or we can use a DeAuth attack to de-authorize one or more clients on the network, who are likely going to send ARP traffic as soon as they reconnect. Once we have a sufficient amount of packets, we will perform a PTW attack against the data, which has a 100% chance of success given sufficient packets (~100,000) [2].

## Results

We attempted to break the WEP encryption on a network named TESTNETWORK. From the router configuration, the WEP key was a 128-bit (104-bit) key: 65:E4:51:DB: 08:F2:C1:84:0C:DA:64:22:00. We started by performing a site survey with Kismet to get information such as the BSSID (MAC address of the access point), the SSID (also known as the ESSID of the access point), and the MAC address of at least one associated client. This information is required to perform various attacks and captures to achieve our end result. In order to launch Kismet, we first inserted our USB wireless card, which uses the RT2573USB chipset. This card can either use the rt73 or rt73usb driver. We chose to use rt73 because it tends to be more stable in monitor mode operations (and rt73usb tends to work better for normal day to day use). Once the card was operational, we used airmon-ng to put the card into monitor mode. This is a handy script included with the aircrack suite and saved us from having to know the exact syntax for enabling monitor mode for the specific brand of wireless card. The syntax we used to execute airmon-ng was: airmon-ng start rausb0. Next, we modified our kismet config to use the device type rt73 and the interface rausb0 [3]. Once the kismet configuration was prepared, we launched kismet.

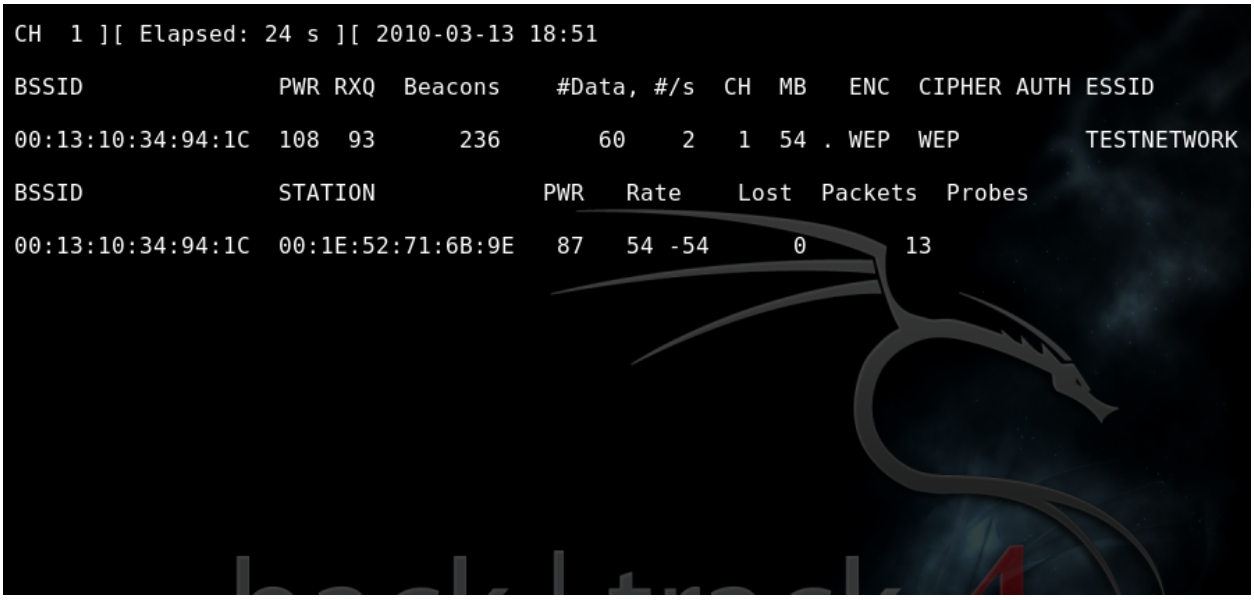


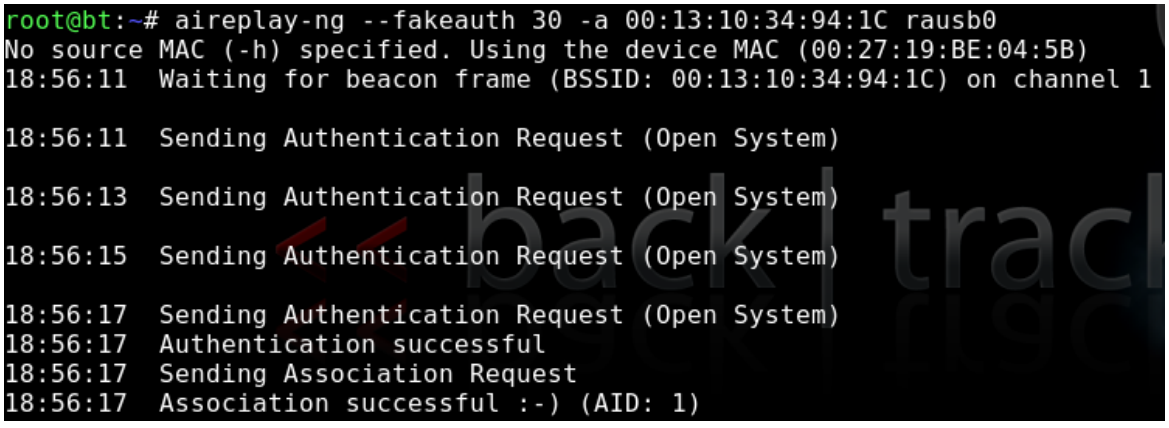

From these screenshots, we have identified that our network with the ESSID "TESTNETWORK" also has the BSSID 00:13:10:34:94:1C. It was also running on channel 1. We can also see that there is a wireless client associated to the access point with the MAC address 00:1E:52:71:6B:9E. Wireless clients are much preferred over wired clients since we can perform DeAuth attacks on wireless clients. We will see why this is useful further on.

Finished with Kismet, we then moved on to the Aircrack-NG suite.

First we have started airodump-ng to capture all of the packets going to and from our wireless interface. The following syntax tells airodump-ng to listen for all packets to/from the access point with the given BSSID on channel 1 on the wireless interface rausb0 and to log them to the file “testnetworkcapture”: airodump-ng --channel 1 --bssid 00:13:10:34:94:1C -w testnetworkcapture rausb0

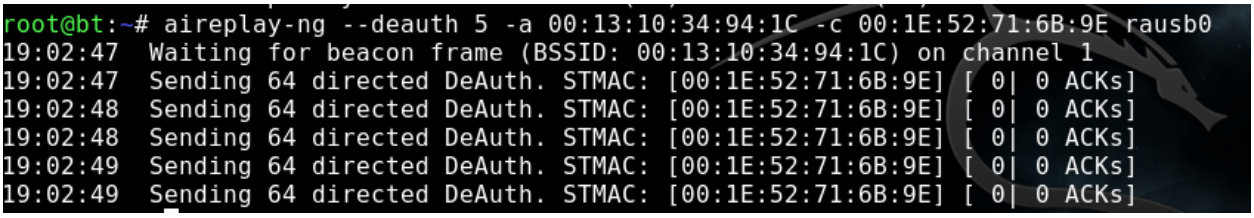



We can see that there are only 2 packets per second on the network. This is not nearly enough for us to crack WEP in any timely manner, thus we must generate traffic using ARP injection. We first associated our MAC address with the access point so that we could inject once we were ready. We used the tool aireplay-ng to do this. The following syntax tells aireplay-ng to perform an association and send keep-alive packets every 30 seconds to the access point given by the MAC address to the -a parameter on the interface rausb0: aireplay-ng --fakeauth 30 -a 00:13:10:34:94:1C rausb0

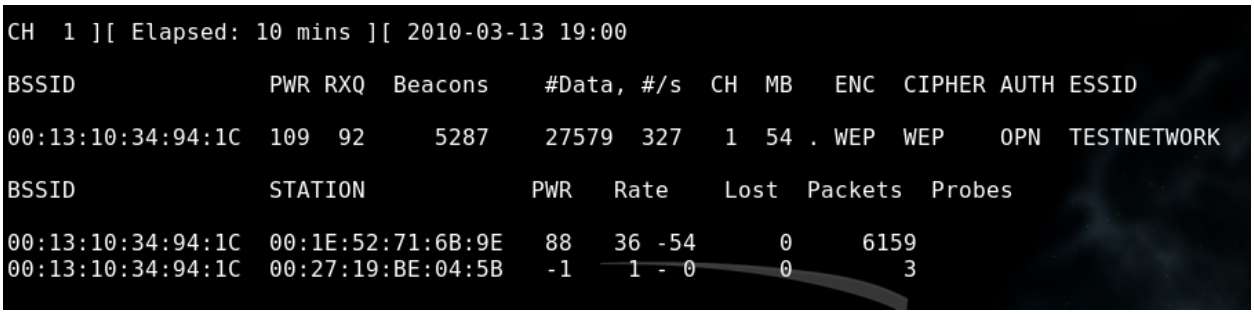


We were then ready to begin an ARP injection attack. The goal of this attack was to capture an ARP packet and replay it to the access point, generating more IVs as they are acknowledged in the hope of getting some duplicate IVs. We used aireplay-ng to do this. The following syntax tells aireplay-ng to begin listening for and if found, replaying ARP packets to the access point given by the -b parameter, from the source MAC address given by the -h parameter on the interface rausb0. Note that the source MAC address should be whatever we had associated with the access point in the previous step. In the event that the previous step failed, we can clone our MAC address to that of an associated client and use that address here: aireplay-ng --arpreplay -b 00:13:10:34:94:1C -h 00:27:19:BE:04:5B rausb0. Note that the following screenshot shows the ARP attack immediately being successful. We must have been lucky and came across an ARP packet by coincidence (they occur periodically on the network). We will still cover how to do a DeAuth attack to ensure an ARP packet is found immediately.

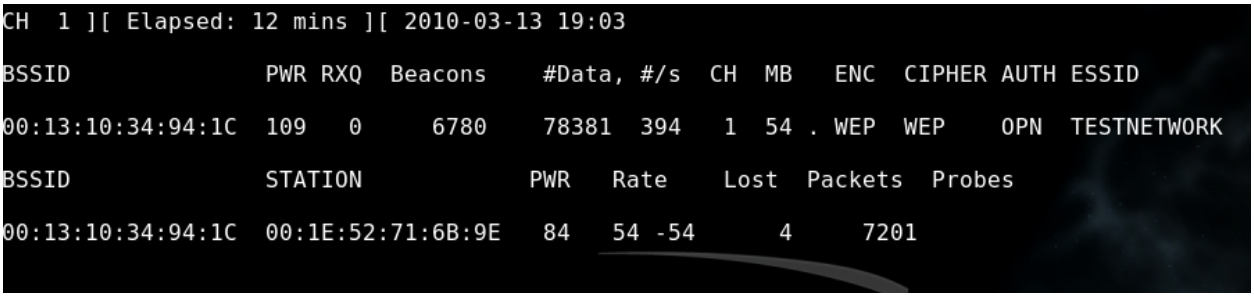

In order to increase the likelihood of success in our ARP replay attack in a timely manner, we tricked a client into thinking it had received a de-authorization request from the access point. Once it disconnects and reconnects, most operating systems will send some ARP traffic immediately. The following syntax tells aireplay-ng to initiate 5 consecutive 1-second deauth attacks between the access point given by -a and the client given by -c on the interface rausb0. Note that we do not want to send too few deauth requests that the chance of dropping them would be high, nor do we want to send too many as to noticeably DoS (denial of service) a client. aireplay-ng --deauth 5 -a 00:13:10:34:94:1C -c 00:1E:52:71:6B:9E rausb0



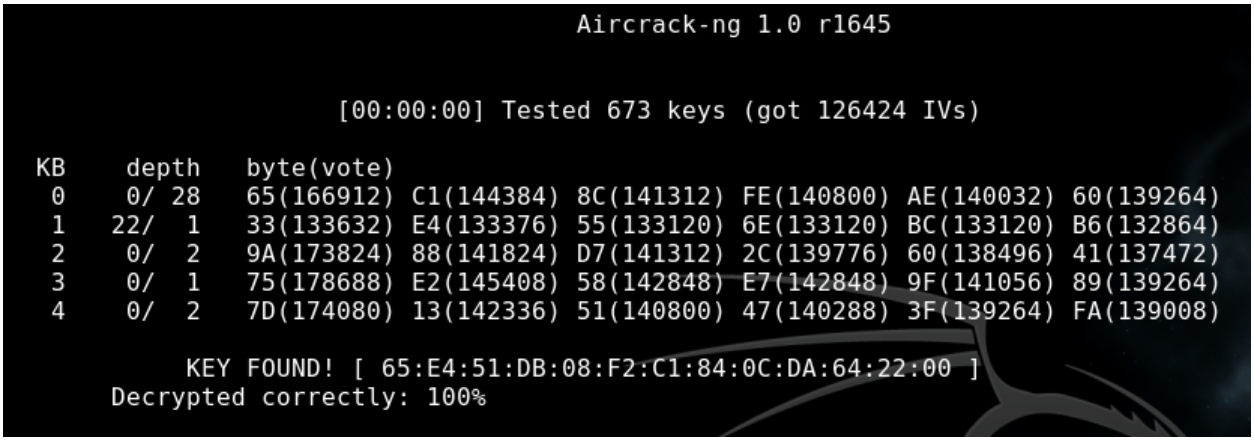
Going back to the to our window or tab where airodump-ng was running, we can see that the number of packets per second on our network has increased dramatically. Our goal is to hit ~100,000 data packets before we attempt to crack the key.




and after another 3 minutes:



By the time that we had aircrack-ng set up, the number of data packets was well over 100,000. We used aircrack-ng to perform the cracking of the key based on the IVs (data packets) that had already been logged. The following syntax will tell aircrack-ng that the encryption algorithm we are breaking is WEP ( -a 1 ), the access point we are breaking into is given by -b (incase our packet capture has more than one access point), and finally, our capture file from airodump, which in this case will be testnetworkcapture-01.cap.




As we can see from the screenshot, there were 126,424 packets containing IVs when we ran this attack. In less than a second, aircrack-ng was able to find the WEP key with a 100% certainty. This key matches the key we entered into our test router's configuration page, and we have proven that 128-bit WEP can be broken with minimal effort and minimal time.

# WPA/WPA2 Encryption

## Detailed Context and Background Information

In 2003, WPA was introduced by the Wi-Fi alliance. It uses an algorithm called TKIP, which borrows many concepts from WEP, but secures the IV through the process of key-mixing and thus renders related-key attacks useless. It still uses RC4 as its encryption algorithm in the interest of backwards-compatibility with old WEP hardware, though backwards-compatibility was rarely achieved as many manufacturers did not release updated firmwares for their devices. WPA, with a strong passphrase, is practically impossible to break. There are some issues with TKIP (similarly to WEP) that allow small packets like ARP to be decrypted, but there is no way to completely compromise a secure WPA key. If a weak passphrase is used, such as a word found in a dictionary, it becomes trivial to break a WPA network as the possible number of keys is no longer exponential but has been reduced to a very, very small linear subset.

In 2005, WPA2 was introduced, which uses AES instead of RC4. It is considered fully secure, and support for it is mandated by the WiFi alliance in order for a device to be WiFi certified. The main advantage to WPA2 (with CCMP, an AES based protocol), is that message integrity is also handled by the block cipher, therefore making the kind of compromises of known or small packets in previous algorithms impossible [4].

## Methodology

### Breaking WPA/WPA2 with a weak passphrase

WPA/WPA2 networks with weak passphrases are trivial to compromise. We started by logging traffic to/from the network with the hope of catching an authentication handshake. Once captured, we began brute-forcing the encryption key in the handshake from a list of known passwords (a wordlist or a dictionary). A dictionary brute-force attack requires immensely less time than a brute-force attack for all possible combinations of the key space. A dictionary attack with a reasonably sized wordlist can take anywhere from 0 to 30 minutes given the hardware and the location of the correct passphrase in the list. Randomization of the list can offer real-world performance advantages. In order to capture the handshake, we could have either waited for an indeterminate amount of time for a new client to connect or we could have forced an associated client to reconnect using a DeAuth attack. It is important to note that DeAuth attacks are not

possible with WPA2 because of the improvements regarding message integrity offered in CCMP vs TKIP. However, in a reasonable real-world scenario, waiting for a new client to associate to a network is not a significant issue.

If a network has a strong passphrase, there is no way to estimate how long it will take to break it. Without pre-determined knowledge about the key space (the characters that make up the passphrase), or the length of the passphrase, we must assume the worst-case scenario in brute forcing. Keeping real world usage scenarios in mind, for the purpose of estimation, we assumed that people will only use upper and lower case characters as well as numbers. Even if special characters are added to the key space, the key space is already large enough that there will be a minimal impact [4].

### Brute-forcing WPA/WPA2 with and without the assistance of GPUs

Most modern operating systems will refuse to connect to a WPA network with a passphrase of less than 8 characters. If we have 26 characters in the english language, giving 52 for both cases, and 10 digits, we have a key space of 62. Immediately, this is a lower bound of $62^8$, or ~218 trillion. A reasonably fast modern CPU can generate ~1000 keys per second per core, or for an average quad-core CPU, ~4000 keys per second. A modern GPU can generate ~13000 keys per second. Assuming we have a single computer with 4 GPUs and a fast CPU, we can brute force ~50000 keys per second. Without a cluster, this puts our lower bound for brute forcing an 8 character WPA/WPA2 key at ~138 years. If we make assumptions about the key space (lower case characters only), we have a much more reasonable ~47 days. Given the rapid advances in CPU and GPU technology, lookup tables, and clusters of said technologies, it becomes realistic to break the maximum key space for 8, 9, or 10 characters. Once we begin reaching lengths of 11 or more at the maximum key space, it becomes practically impossible to brute force a WPA/WPA2 network. It is reasonable to assume that there are only a handful of companies and agencies in the world that possess the computing power and resources to break $62^{10}$, and any information as to the details or infrastructure behind this would most certainly be classified. $62^8$ and $62^9$ may be within the grasp of some wealthy academic institutions or commercial entities, but not within reach for the average person.

# Results

## Breaking WPA/WPA2 with a weak passphrase

We used the same access point (BSSID and ESSIDs) as we used previously. This section will only cover how to break a weak (dictionary) WPA/WPA2 passphrase once our initial site survey was performed.

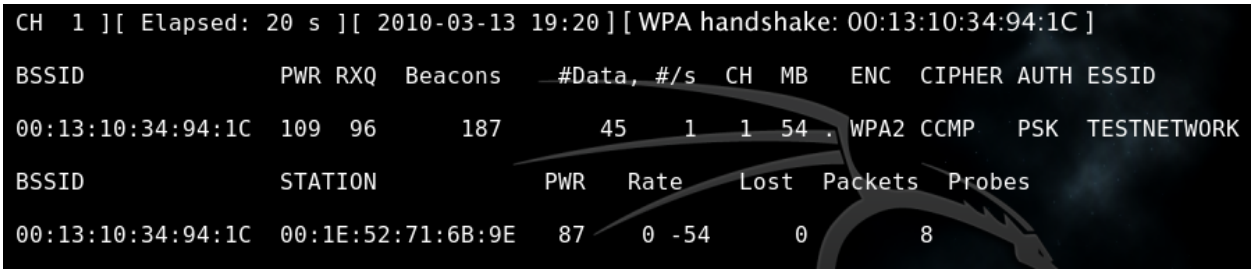
First some preparation to configure the tools. A critical component to breaking WPA/WPA2 is a good wordlist or dictionary. An actual dictionary file is very bad. It will likely not contain many common concatenations, slang terms, or other strange variations used in common passwords including numeric insertions. We required a wordlist specifically designed with passphrase cracking in mind. Such a wordlist happens to be included with the linux distribution we were using, BackTrack 4. This wordlist is called darkc0de and it is located at /pentest/passwords/wordlist/darkc0de.lst in BackTrack 4.

It should also be noted that while having a client on the network was of great assistance when breaking WEP, it is absolutely critical when breaking WPA/WPA2, and there is absolutely no way to break in without clients on the network. In fact with WPA2, the more clients (the busier the network) the better. If there are very few clients with WPA2, it may take a while to successfully capture a WPA handshake. Our goal was to capture a WPA/WPA2 handshake and then attack it.

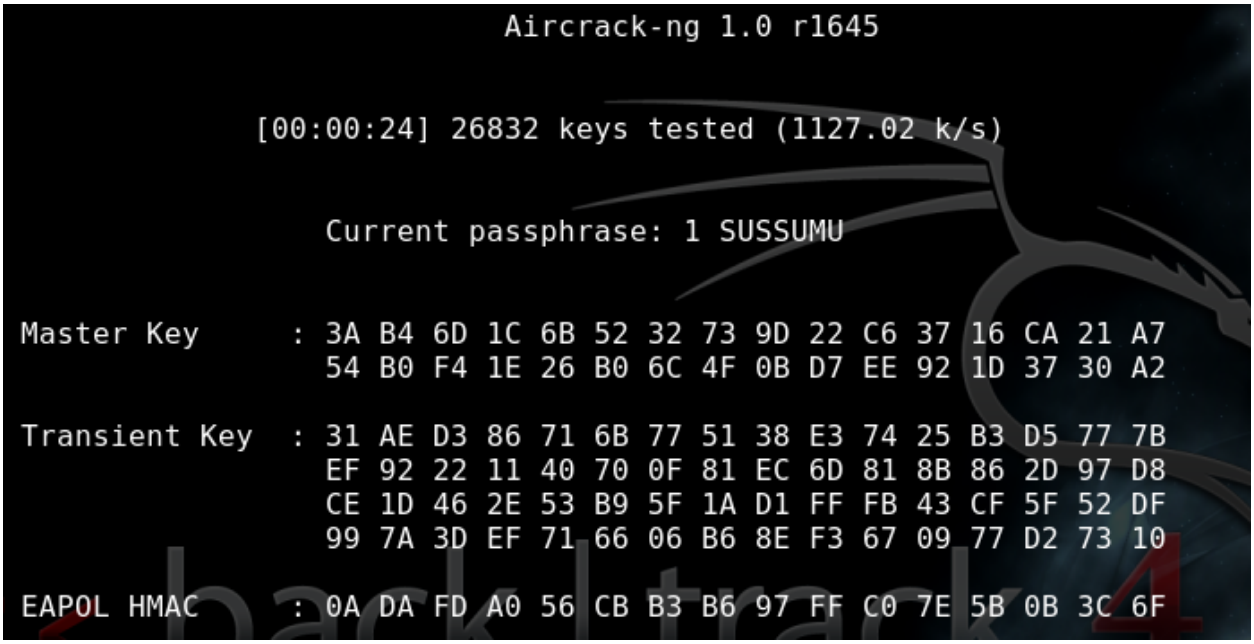
First, we needed to run airodump-ng to log all traffic in and out of the access point. The following syntax will begin logging all traffic on channel 1, to and from the BSSID given by --bssid, and it will output it to testnetworkWPAcapture on rausb0: airodump-ng --channel 1 --bssid 00:13:10:34:94:1C -w testnetworkWPAcapture


Next we had two choices, we could either wait for a client to disconnect and reconnect (or freshly connect) to the network, or we could de-authorize an existing client to force it to reconnect like we did in the WEP section. We knew when we had captured a WPA handshake

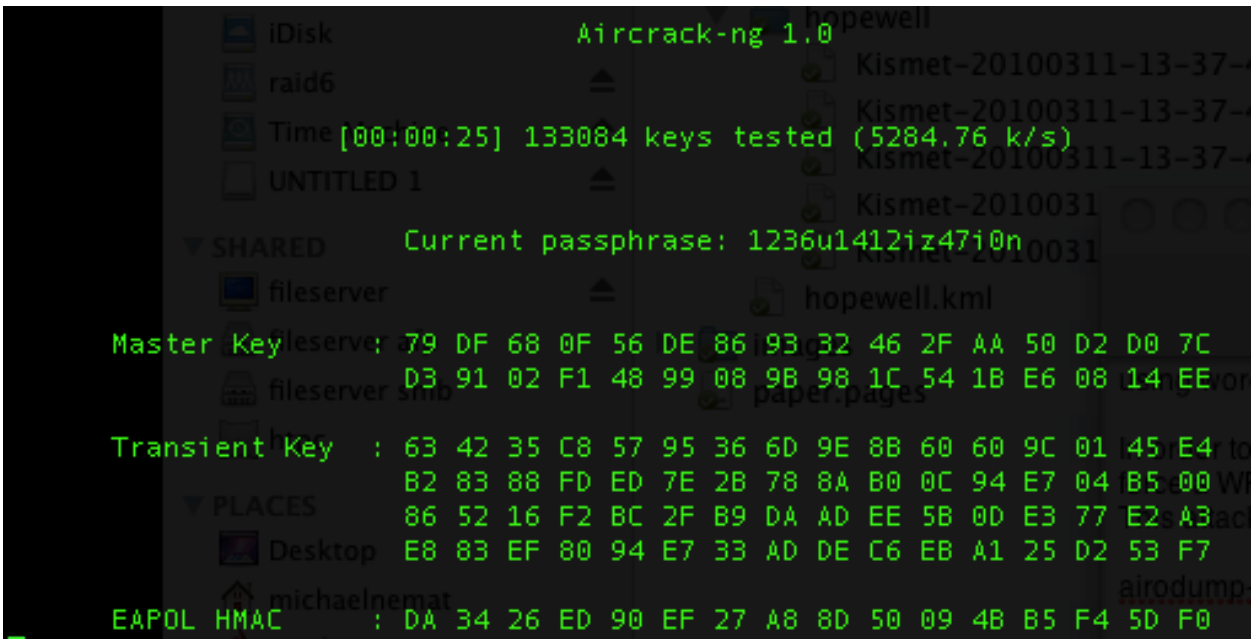
because airodump-ng showed WPA HANDSHAKE in the top right of the screen along with our access point's BSSID indicating it had captured a WPA handshake on our network. Since our test network is a WPA2 network, we could not do a deauth attack so we did a disconnect/reconnect on our laptop to force a WPA handshake. In the real world, with clients coming and going, this will likely happen within a few minutes in a commercial area or in a few hours in a residential area. Note that a better way to do this is to write a script to generate some sort of an alert when a handshake is found, either by trying to run aircrack-ng against the capture file and checking the return code of the application or by monitoring the output of airodump-ng. With an automated notification implementation, we can more efficiently break in to one or more networks simultaneously. The following screenshot shows what happened when we got a WPA handshake. Once we had the WPA handshake, we were able move on to the next (and final) step.




We began the dictionary attack against the WPA key using aircrack-ng, our wordlist, and the WPA handshake capture. The following syntax will tell aircrack-ng to break WPA encryption ( -a 2 ) on the access point with the BSSID given by -b, using the wordlist found at the path given by -w , and finally the capture file containing the WPA handshake. aircrack-ng -a 2 -b 00:13:10:34:94:1C -w /pentest/passwords/wordlist/darkc0de.lst testnetworkWPAcapture-01.cap

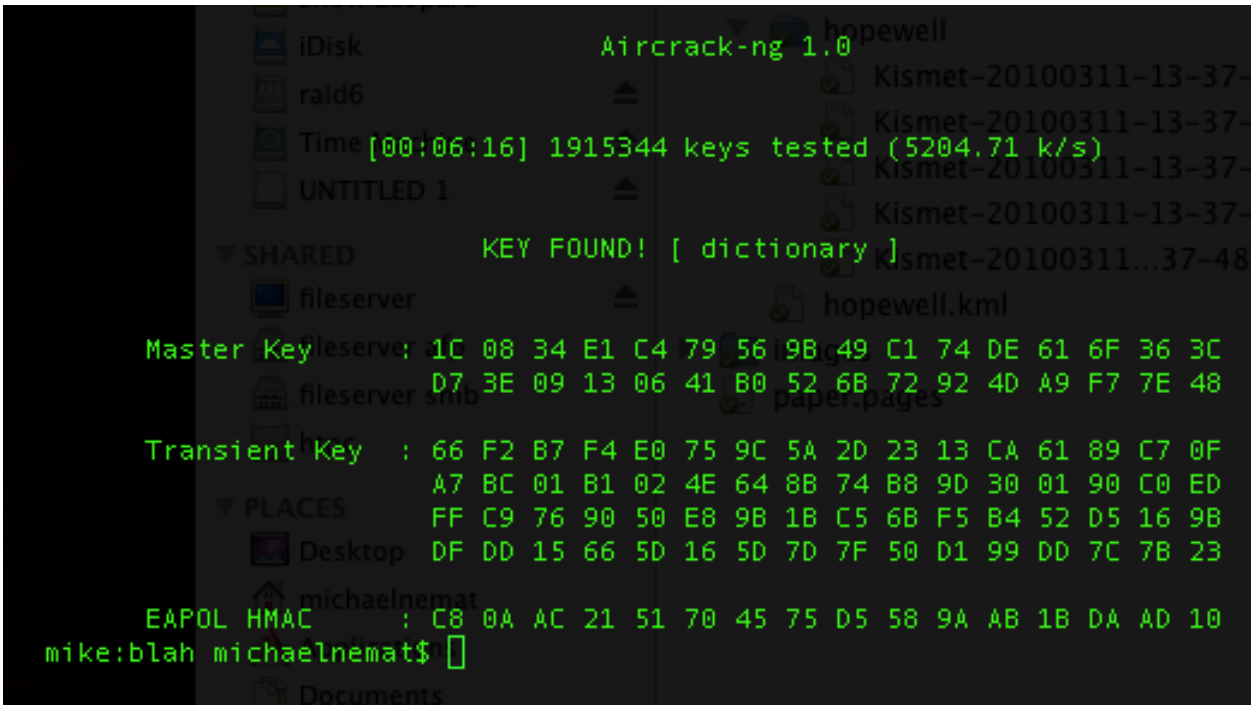



We first tried this on a 2.0ghz dual-core laptop (Intel Core 2 Duo processor). Our results were less than impressive at ~1100 keys per second. In order to speed up the process, we used SSH to send our data back to a 4.0ghz quad-core Intel Core i5 processor, and our results were much more impressive: ~5200 keys per second.




For the record, our passphrase is (ironically) “dictionary”. Since this wordlist is sorted alphabetically, we had to exhaust a large number of words starting with uppercase characters before we got to “dictionary” with a lowercase d. Randomizing our wordlist would have been a

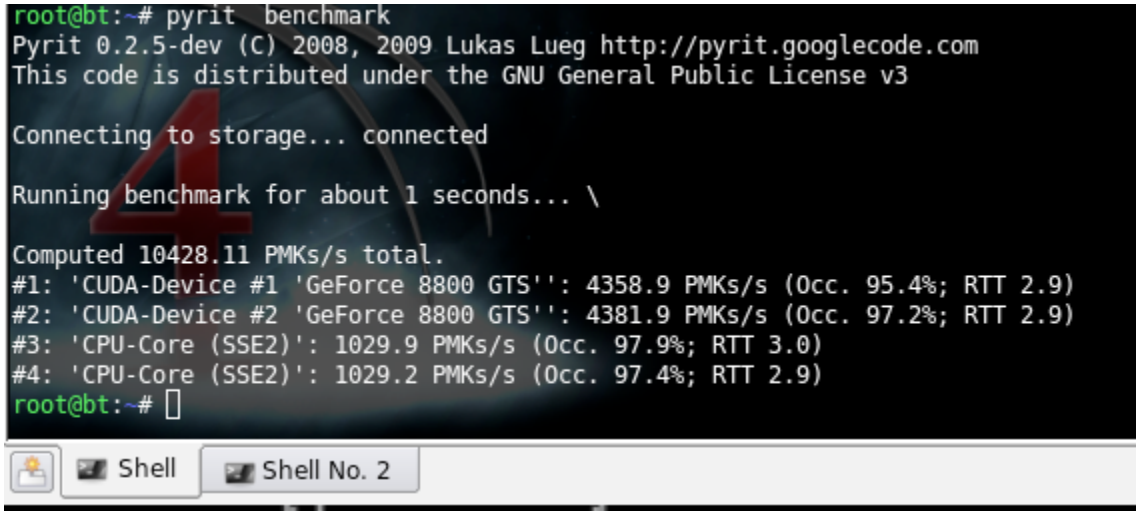
good idea.  It took approximately 6 minutes before aircrack-ng found our passphrase. This was far more reasonable than the ~30 minutes it would have taken on our laptop.



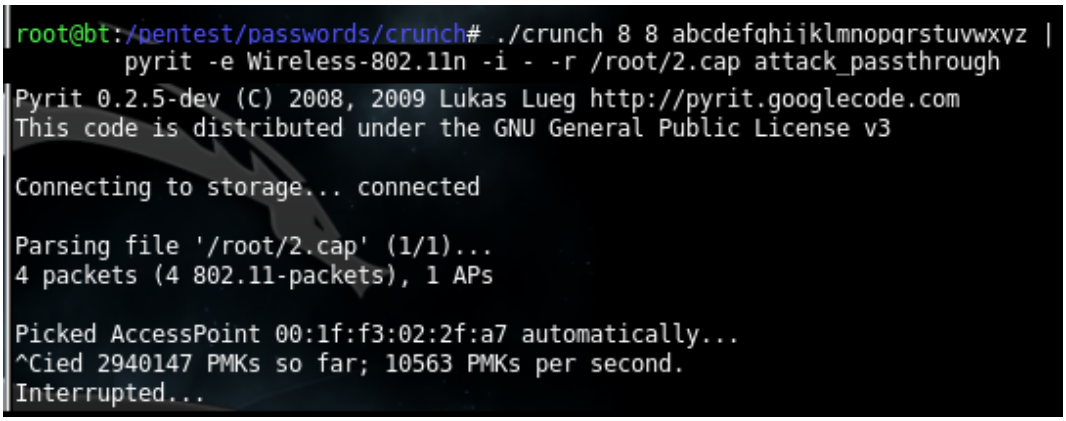

As shown in the screenshot, breaking a WPA key whose passphrase is in a dictionary or wordlist is trivial. In fact, depending on the time it takes to intercept a WPA handshake, breaking a weak WPA key can be much quicker than breaking WEP simply because there is no packet injection or data collection required beyond a single WPA handshake.

## Attempting to Brute-Force WPA/WPA2 with and without GPU Assistance

Assuming we had already captured a WPA or WPA2 handshake with Airodump-ng (covered in the previous section). We attempted to brute force the key using a program called Pyrit along with a key generator called Crunch. We used a workstation with a quad core Intel Core i5 Processor running at 4.0ghz per core, yielding 1030 keys per second per core. The workstation also contains 2 NVIDIA 8800GTS GPUs, yielding 4380 keys per second each. Since pyrit can only use one core per every GPU in GPU cracking mode, our GPU cracking key throughput is 10820 keys per second. Our CPU cracking throughput will be 2060 keys per second. Note that our CPU is very fast, but our graphics cards are older generation technology. Even considering that our graphics cards (GPUs) are 3 generations old, they are still significantly faster than our cutting-edge CPU. A modern graphics card could put out 15,000-20,000 keys per second.



The following command will begin to brute force the WPA handshake found in a file “2.cap” using pyrit and crunch, with a key length minimum and maximum of 8 and the lowercase alphabet key space. Our essid in this test is “Wireless-802.11n”: crunch 8 8 abcdefghijklmnopqrstuvwxyz | pyrit -e “Wireless-802.11n” -i - -r /root/2.cap attack_passthrough.

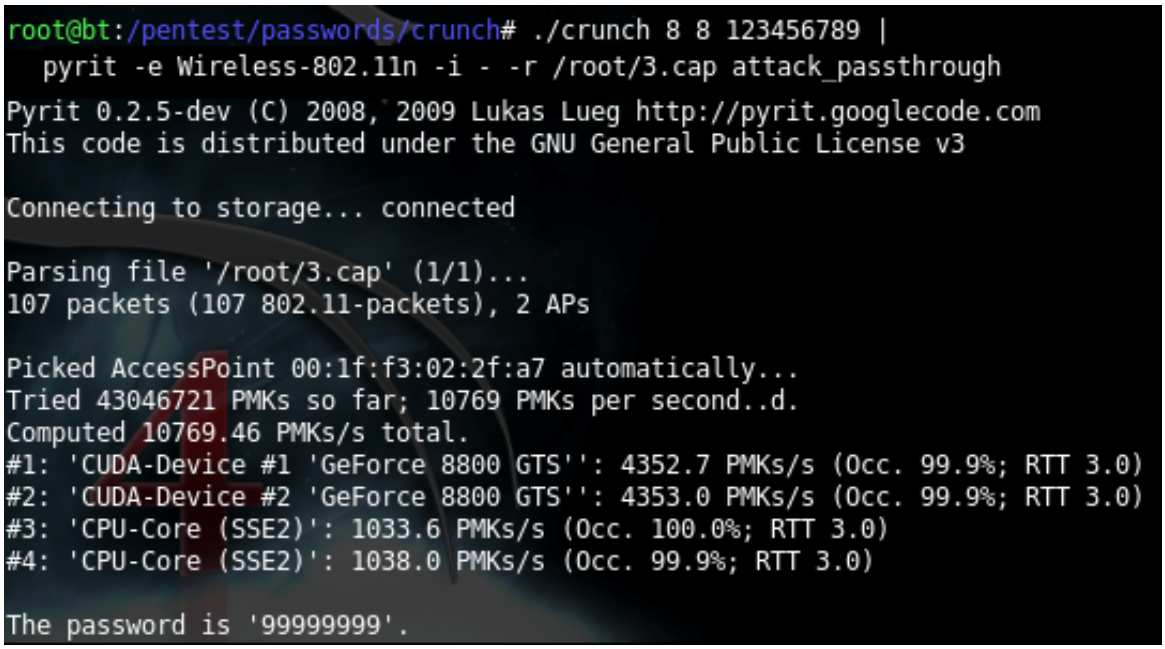



After 5 minutes, only ~3,000,000 keys had been attempted. Our most naive assumptions about the key (8 characters long, composed of only the 26 lowercase alphabet) would require us to attempt 208,827,064,576 keys. On our current hardware, this would take 230 days with our GPUs. If we were to only use our CPU, this would be 604 days or just a little over one and a half years. Without an investment into building a small GPU cluster, an average malicious user has no hope of brute-forcing a WPA/WPA2 key even with the most naive assumptions about the composition of the passphrase [5].

Let’s hypothesize that we have a workstation with an Intel Core i7 980X 6-core CPU, overclocked to 4.2ghz per core. We also have 6 top-of-the-line NVIDIA Geforce GTX480 GPUs installed. The cost of such a workstation would be approximately $8000 due to the immense cooling and power requirements of the hardware in a confined space. Since this hardware was only released at the time of writing this report, we can only estimate the performance. If we generously assume each CPU core can generate 1300 keys per second, and each GPU can generate 20,000 keys per second, we can limit the upper bound of the latest technology available to this day at 127,800 keys per second. If we wanted to break a WPA/WPA2 key with our naive assumptions in half an hour, we would need **908** of these top of the line workstations at a cost of **7.26 million dollars** not including enormous power and supporting infrastructure costs. If we increase our time limit to one day, we would only need **19** of these workstations at a cost of **152**

**thousand dollars**. This cost is perfectly reasonable for a government intelligence agency or even academic institutions and local law enforcement agencies. Lets assume we have a key length of 15, with 62 possible characters, and we want to break the key in 3 days, we would need **2.32 x $10^{16}$** of these workstations at a cost which would be impossible for anyone in the world to afford and support. We can see how breaking a reasonably sophisticated WPA/WPA2 key in a reasonable time period is financially prohibitive for anyone given current technology. Intelligence agencies may be able to get lucky and break somewhat short pass-phrases, but any sophisticated passphrase is unbreakable with current technology [5].

To prove that our brute-forcing solution works, we have modified our router to use a key consisting of only the numbers 0 through 9, with length 8. The key is 99999999 (this is so that we can show the absolute worst case since this key will be generated last). In a little over 2 and a half hours with our hardware, we have our solution:



# Evaluation of the Results

## Results

In both residential and commercial areas, approximately 50% of the networks are using strong encryption, the rest are using WEP, or no encryption at all. For all intents and purposes, a malicious user will likely be skilled enough such that WEP would only slow them down by a matter of minutes rather than stop any malicious activity.

The primary reason that WEP is overwhelming popular in residential networks has little to do with any technical issues like compatibility, performance, or hardware age. The majority of WEP networks are on access points with SSIDs that begin with "BELL" followed by 3 digits. This is the standard configuration for all residential internet connections set up by Bell Canada in the last two years. Bell Canada no longer provides standalone DSL modems, but rather requires you to use their branded version of 2Wire's Wireless Router with a built in ADSL2+ modem. Since Bell is one of the two largest ISPs in Ontario, they are responsible for an overwhelming large size of WEP networks encountered in residential areas. Bell Canada also stresses that the wireless networks deployed by their technicians (in this standard configuration using WEP) are secure. This could not be further from the truth, as demonstrated on our section regarding breaking WEP. The majority of consumer routers from D-Link, Linksys, etc sold today have WPA2 enabled out-of-the-box. We would speculate that the primary reason for open networks in residential areas is old hardware with standard configurations of no encryption, combined with ignorant users.

The primary reason that a lack of encryption is more common in commercial networks than residential networks is likely intentional. Many commercial hotspots run by Rogers, Bell, coffee shops, etc, make use of RADIUS authentication tied to e-commerce solutions in order to sell wireless internet service. These types of services due to their pay-before-access nature means that they cannot use pre-shared key encryption by definition. 802.1x type security measures like WPA2 Enterprise require extensive back-end infrastructure and are very difficult to incorporate into these kinds of pay-per-use systems, which is why open networks remain popular in commercial districts. While this is of serious concern for identity theft and fraud reasons, commercial networks used for internal and private enterprise data are of much more serious concern, as this kind of blatant disregard for information security can lead to identity theft and

fraud en masse, as well as industrial espionage. Pay-per-use hotspots provide the illusion of security to ignorant users, but should not be used as anything more than an access layer for establishing a secure tunnel or VPN since traffic is visible to anyone within range.

We have proven that WEP networks can be broken using open source tools in a matter of minutes given any realistic real-world usage scenario. Given the presence of at least one associated client on the network, which is a legitimate assumption, we can break WEP with 100% certainty in a matter of minutes.

Similarly, we have proven that WPA and WPA2 networks using a dictionary pass-phrase can be broken using open source tools in a matter of minutes in similar usage scenarios to our WEP test (at least one user). Brute-forcing a WPA or WPA2 network on the other hand is only feasible if the key space and key lengths are very low. We have also shown benchmarks for CPU vs GPU brute-forcing of WPA/WPA2 networks, and proven that agencies with significant financial resources (GPU clusters, etc) can break longer keys with large key spaces, but are still limited to ~11-13 characters with a full key space in any reasonable amount of time with statistics in their favor. If we were to linearly brute force and not introduce randomizations in our key stream, we would have to assume the worst case scenario which would likely limit this key length to 10-11 characters. We can conclude that any reasonably sophisticated WPA/WPA2 is impossible to break with current technology.

We can also conclude that the lack of proper wireless network security in residential and commercial areas of Ottawa is alarming. Given the density of wireless networks in these areas and the security statistics provided, a malicious individual would have absolutely no difficulty in using an insecure wireless network to perform a variety of illegal activities.

## Suggestions for future development

### Increase data pool for site surveys

One area of expansion is to increase the number of site surveys. In its current form, only two areas of the city:  Downtown (commercial), and Old Ottawa South (residential) were surveyed. Increasing the number of areas may lead to stronger metrics in the types of Wireless Security used. It would be interesting to see if external demographic properties such as area age and social/economic status have a correlation to both the number of wireless networks present and the type of encryption used. This sort of analysis could be extended to to determine the popular choice of ISP in residential networks (most Cable/ADSL providers offer a wireless router modem option that defaults the SSID of the network). Further, any additional data would create stronger data plots used in the statistical analysis and breakdown of security use.

### Investigate SSLStrip and other man-in-the-middle attacks

If we were to continue our work on this topic, a very interesting and groundbreaking area in internet security is man-in-the-middle attacks focusing on defeating SSL (more specifically, HTTPS). The primary tool for performing this attack is SSLStrip (REFERENCE MOXIE MARLINSPIKE). SSLStrip in combination with an ARP poising attack allows the interception of encrypted information (credit card numbers, account user names and passwords, etc...) being sent to a web server over HTTPS by tricking the browser into sending the data over HTTP instead. SSLStrip rewrites and/or redirects any HTTPS traffic to HTTP through various methods like rewriting URLs in outgoing requests and incoming replies. This is effective because of two key shortcomings in internet browsers and web servers. First, many web servers use the same document root for HTTP and HTTPS. This means that while the web page is designed to send all confidential information over HTTPS, sending it over HTTP will not fail. If one were to somehow (for example, by using SSLStrip) trick the web browser into sending this information over HTTP, it could be readily intercepted. Second, few (if any) web sites make use of HTTPS exclusively. The common practice with web applications is to load a web page initially with HTTP and eventually when a user reaches a stage where confidential information is involved, the desired pages and/or form submits are sent over HTTPS. This is the primary flaw which SSLStrip exploits. SSLStrip will rewrite the pages so that this information is sent over HTTP. The only reason it can rewrite the pages to begin with is because the pages preceding the stages where HTTPS is involved are sent over HTTP and thus can be intercepted and rewritten. If SSLStrip is used when only the second flaw is present, confidential information can be

intercepted however the application functionality will be broken. A malicious user can intercept the information but a security conscientious would notice unusual behaviour. However, if the first flaw of shared document roots is present, then it is possible for information to be intercepted in a completely transparent manner to the user. SSLStrip is capable of using favicons to trick browsers into displaying a lock icon for an unencrypted page. We would have liked to further investigate man-in-the-middle attacks on wireless networks. While we were able to get SSLStrip functioning on a wired network, we encountered difficulties in getting it working on a wireless network and decided to leave it out of our project.

This is related to wireless security because a malicious user who gains access to a wireless network (regardless of whether it was open or if they had to break the encryption) cannot readily intercept confidential information like credit card numbers in most scenarios. SSL still stands in their way. With tools like SSLStrip, it becomes trivial to intercept this information.

### Investigate decrypting packet captures upon discovery of encryption key

If we were to continue our work on this topic, it would have been interesting to use some available open-source tools from the Aircrack suite and other projects to decrypt our packet captures once we have recovered the encryption key. This would be useful to perform analysis of the traffic with tools like Wireshark to gather metrics on the amount of confidential information which could be intercepted once a key is recovered (in terms of how much information is sent over SSL vs unencrypted).

### Explore the legality of breaking into wireless networks

Another area that could be of interest would be to investigate the legalities of breaking into wireless networks. It would be interesting to see where the law stands on breaking an encryption key, illicitly joining an encrypted network and performing malicious activities (breaking into computers, stealing information, monitoring traffic on an illicitly joined network). Additionally it would be interesting to explore the liabilities of a network owner for the actions taken by an illicit agent in terms of hacking, piracy and privacy information.

# References

[1] “SSLStrip”, Moxie Marlinspike http://www.thoughtcrime.org/software/sslstrip/

[2] “Breaking 104 bit WEP in less than 60 seconds”, Eriik Tews, Ralf-Philipp Weinmann, Andrei Pyshkin http://eprint.iacr.org/2007/120

[3] “Kismet Documentation”, Kismet Project http://www.kismetwireless.net/documentation.shtml

[4] “802.11i Amendment 6: Medium Access Control (MAC) Security Enhancements” , IEEE http://standards.ieee.org/getieee802/download/802.11i-2004.pdf

[5] “Pyrit, The Twilight of Wi-Fi Protected Access”, Pyrit http://pyrit.wordpress.com/the-twilight-of-wi-fi-protected-access/