

# **Backtrack 4**

## **CUDA GUIDE**

**by**  
**Pureh@te**

**Version 2**

# Table of Contents

What is CUDA? .....	3
Supported GPUs .....	3
Why do I care about CUDA? .....	3
Where can I get this CUDA thing? .....	4
What is CUDA not? .....	4
Getting started .....	5
Nvidia-Drivers. ....	5
Overclocking.....	6
Installing the CUDA toolkit and SDK .....	9
CUDA-Multiforcer .....	13
Crack.....	15
Pyrit .....	17
What is pyrit? .....	17
Up and running with pyrit .....	18
Making sure Pyrit is working. ....	19
List cores.....	19
Benchmark.....	20
WPA Wordlist.....	21
Attack_passthrough Mode.....	22
Attack_passthrough with Crunch.....	23
Batch.....	24
Analyze.....	26
Strip.....	27
Building aircrack-ng with CUDA support: .....	28
Useful Links: .....	30

## What is CUDA?

CUDA (an acronym for Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA. CUDA lets programmers utilize a dedicated driver written using C language subroutines to offload data processing to the graphics processing hardware found on Nvidia's latest model GeForce graphics hardware. The software lets programmers use the cards to process data other than just graphics, without having to learn OpenGL or how to talk with the card specifically. Since CUDA tools first emerged in late 2006, Nvidia's seen them used in everything from consumer software to industrial products, and the applications are limitless.

## Supported GPUs

A complete list of supported GPU's can be found at the following link:

[http://en.wikipedia.org/wiki/CUDA#Supported\\_GPUs](http://en.wikipedia.org/wiki/CUDA#Supported_GPUs)

## Why do I care about CUDA?

Hardware acceleration of password recovery is possible with CUDA enabled applications. Many of these applications are already available and there are many more to come. The support of NVIDIA graphic accelerators increases the recovery speed by an average of 10 to 15 times faster than was previously possible.

## Where can I get this CUDA thing?

Backtrack 4 final comes fully ready to execute and build CUDA powered applications. I will review some of the major points involved in setting up the environment and running some of the application.

## What is CUDA not?

CUDA is not a magic bullet that will suddenly make all software on an Nvidia-equipped PC run dramatically faster, in other words -- the programmer needs to figure out where the program can be optimized to process data in parallel. But within that context, programming support for CUDA can make a big difference.

# Getting started

## Nvidia-drivers:

The first thing we need to do is get the nvidia drivers installed. This is done easily with Backtracks package manager apt-get or aptitude. Installing the nvidia drivers is best done while the X server is not running. The X server can be stopped by pressing ctrl - alt -backspace.

```
root@bt:~# aptitude search nvidia-driver
p   nvidia-driver                - Driver for nvidia graphics cards 2.6.30.9
root@bt:~# aptitude install nvidia-driver
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
The following NEW packages will be installed:
  nvidia-driver
0 packages upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 11.4MB of archives. After unpacking 81.4MB will be used.
Writing extended state information... Done
0% [Waiting for headers]_
```

Once you get the drivers installed, a new xorg-config should be generated for you and then you can “startx” and return to the kde desktop environment.

You can make sure the driver was properly installed by greping the xorg.conf for the term nvidia.



```
root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
root@bt:~# cat /etc/X11/xorg.conf | grep nvidia
Driver      "nvidia"
Driver      "nvidia"
root@bt:~#
```

In the event the auto xorg.conf does not work, nvidia provides a utility which may be able to help. To invoke it simply type “nvidia-xconfig” into a terminal and it will try to generate a new xorg config for you.

If you have multiple monitors you may need to use the nvidia-settings tool to configure them. In order to use the settings tool, either launch it from the KDE menu or run the command “nvidiasettings” in a terminal. The actual configuration is beyond the scope of this document however it is fairly easy to understand.

## Overclocking:

There are two ways to overclock your video card in Linux. The first way is to use the nvidiasettings tool which comes with the nvidia-driver. In order to do this you need to edit your xorg.conf in order to unlock the option.

```
nano /etc/X11/xorg.conf
```

and find the section that looks like this:

Section "Device"

Identifier "Videocard1"

Driver "nvidia"

VendorName "NVIDIA Corporation"

BoardName "GeForce 8800 GT"

BusID "PCI:3:0:0"

Screen 1

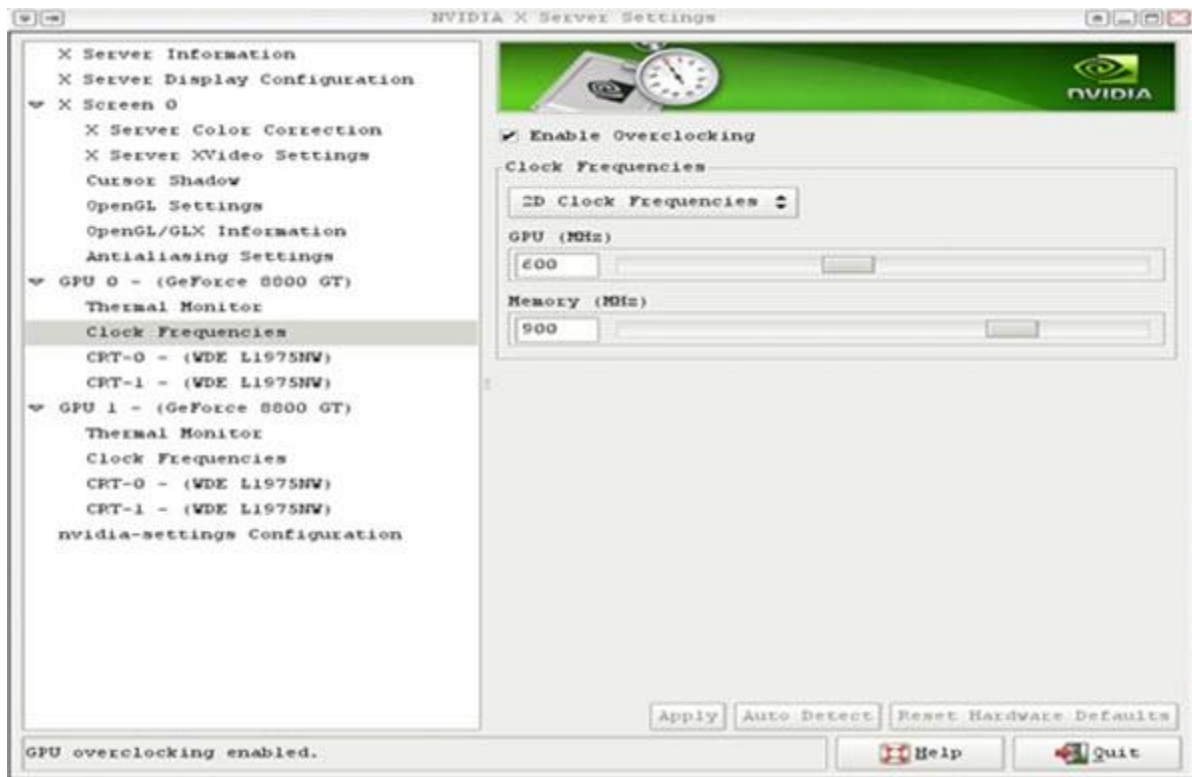
Option "AddARGBGLXVisuals" "true"

Option "Coolbits" "1"

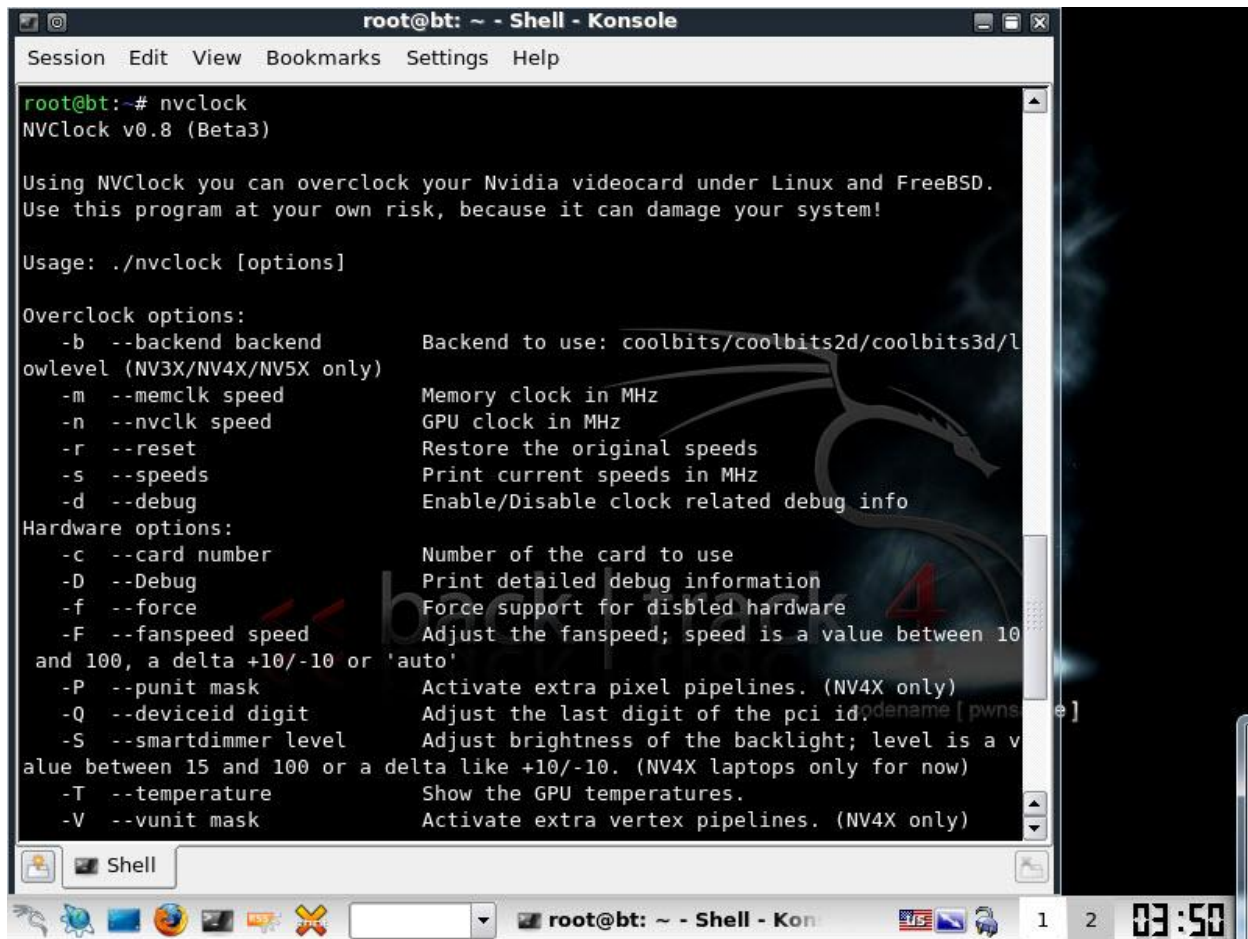
Option "RenderAccel" "true"

EndSection

Add the coolbits option and then restart X and open nvidia-settings and you should have a overclock option like this:



The second way to overclock your card in linux is to use the nvclock command line utility.

A screenshot of a terminal window titled "root@bt: ~ - Shell - Konsole". The window shows the output of the "nvclock" command. The text includes the version "NVClock v0.8 (Beta3)", a warning about overclocking, and a list of options categorized into "Overclock options" and "Hardware options". The options include flags for backend, memory clock, GPU clock, reset, speeds, debug, card number, debug info, force, fanspeed, pixel pipelines, pci id, backlight, temperatures, and vertex pipelines. A large, semi-transparent watermark "b3ak1n9" is visible across the terminal output. The terminal window has a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The bottom of the window shows a taskbar with various icons and a system tray with a clock showing "03:50".

```
root@bt:~# nvclock
NVClock v0.8 (Beta3)

Using NVClock you can overclock your Nvidia videocard under Linux and FreeBSD.
Use this program at your own risk, because it can damage your system!

Usage: ./nvclock [options]

Overclock options:
  -b --backend backend      Backend to use: coolbits/coolbits2d/coolbits3d/lowlevel (NV3X/NV4X/NV5X only)
  -m --memclk speed         Memory clock in MHz
  -n --nvclk speed          GPU clock in MHz
  -r --reset                Restore the original speeds
  -s --speeds               Print current speeds in MHz
  -d --debug                Enable/Disable clock related debug info

Hardware options:
  -c --card number          Number of the card to use
  -D --Debug                Print detailed debug information
  -f --force                 Force support for disabled hardware
  -F --fanspeed speed       Adjust the fanspeed; speed is a value between 10 and 100, a delta +10/-10 or 'auto'
  -P --punit mask           Activate extra pixel pipelines. (NV4X only)
  -Q --deviceid digit       Adjust the last digit of the pci id;dename [pwns e]
  -S --smartdimmer level    Adjust brightness of the backlight; level is a value between 15 and 100 or a delta like +10/-10. (NV4X laptops only for now)
  -T --temperature          Show the GPU temperatures.
  -V --vunit mask           Activate extra vertex pipelines. (NV4X only)
```

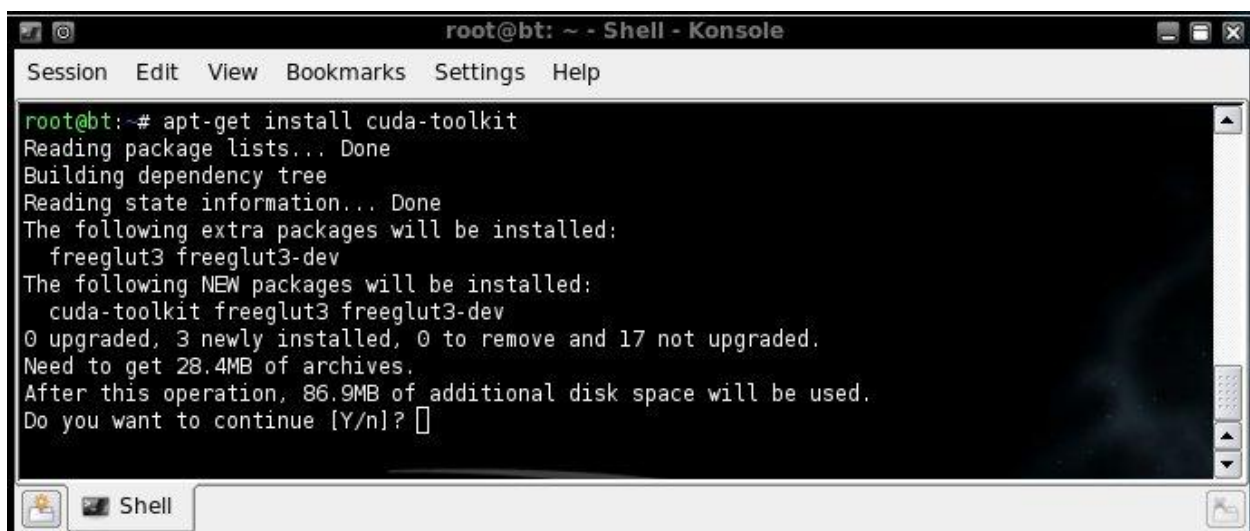
As you can see there are many options with the nvclock utility. Please be sure of your settings before overclocking your card in this manner. I normally use the gamer forums as a good source for finding out optimal clock settings for various cards.



## Installing the CUDA toolkit and SDK :

Now that we have our driver installed and the clock settings to our liking, it is time to get our CUDA development environment set up. This is not necessary if you are only interested in running a tool such as Pyrit however if you want to build any CUDA applications you will need this environment.

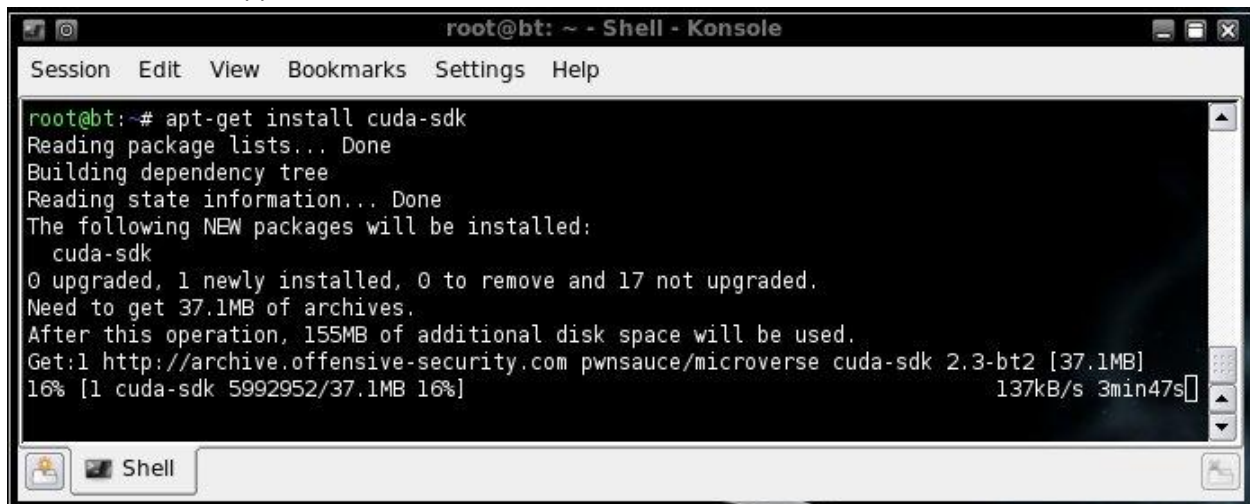
The environment is already built and set up so we simply need to apt-get it. This will require about 250 MB of space so make sure you have the space to set this up. The tool kit should be installed first. This contains all of our tools for working with CUDA applications such as the nvcc compiler and the cuda debugger.



```
root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:~# apt-get install cuda-toolkit
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  freeglut3 freeglut3-dev
The following NEW packages will be installed:
  cuda-toolkit freeglut3 freeglut3-dev
0 upgraded, 3 newly installed, 0 to remove and 17 not upgraded.
Need to get 28.4MB of archives.
After this operation, 86.9MB of additional disk space will be used.
Do you want to continue [Y/n]? 
```

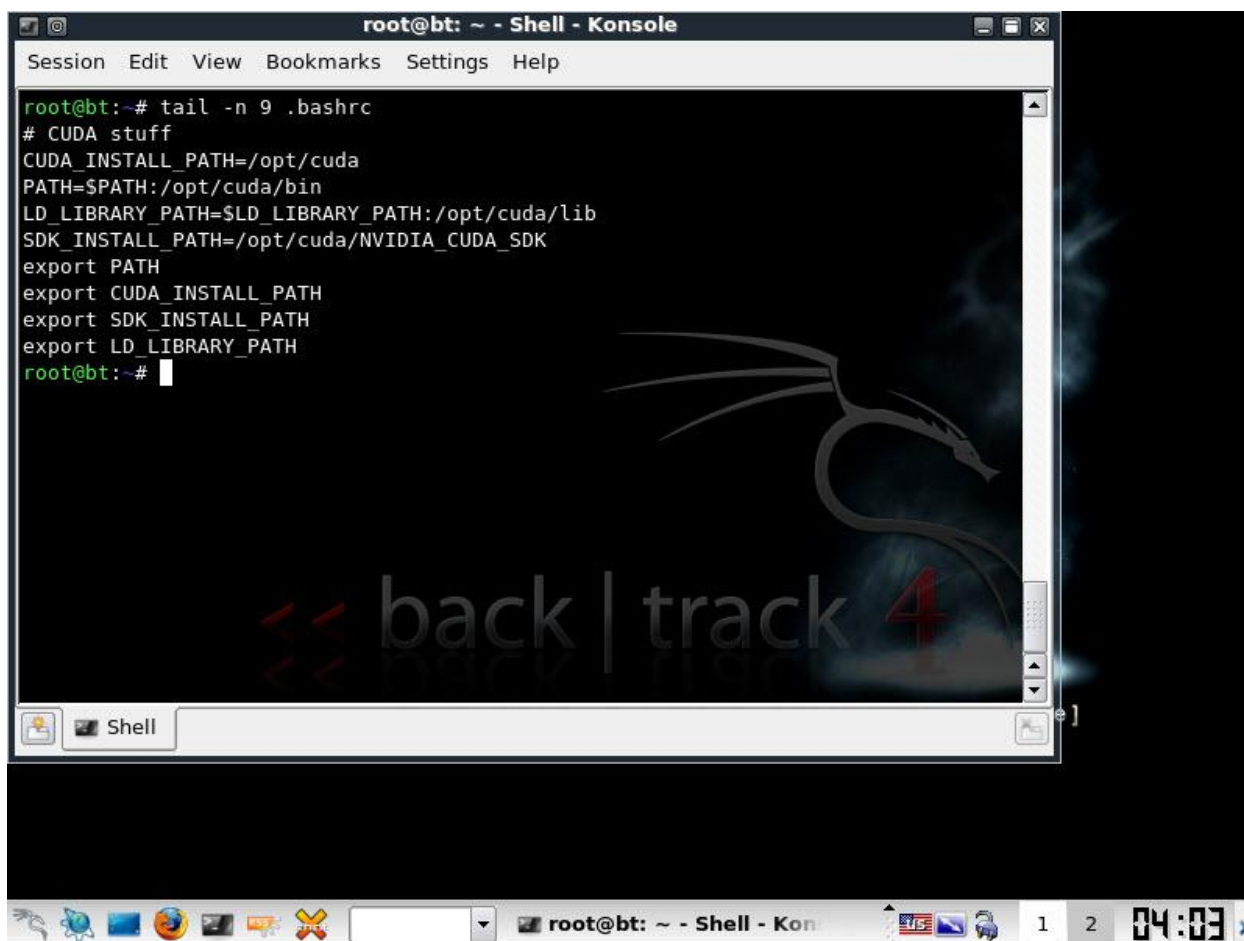
The next package is the CUDA-SDK and contains cuda samples and some of the default make files we can use to build CUDA applications.



```
root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:~# apt-get install cuda-sdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  cuda-sdk
0 upgraded, 1 newly installed, 0 to remove and 17 not upgraded.
Need to get 37.1MB of archives.
After this operation, 155MB of additional disk space will be used.
Get:1 http://archive.offensive-security.com/pwnsaucemicroverse cuda-sdk 2.3-bt2 [37.1MB]
16% [1 cuda-sdk 5992952/37.1MB 16%] 137kB/s 3min47s
```

The CUDA compilers and the libraries should be added to your path by the package manager. The environment variables are, however, set up for root so if you have created another user you will need to move the following piece of roots .bashrc to your new user.

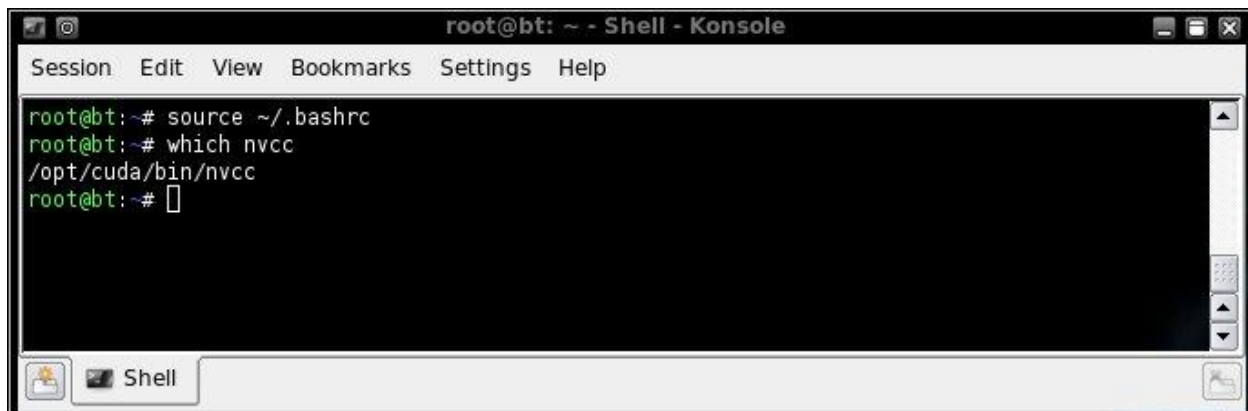


The screenshot shows a terminal window titled "root@bt: ~ - Shell - Konsole". The terminal displays the output of the command `tail -n 9 .bashrc`, which shows the following lines:

```
root@bt:~# tail -n 9 .bashrc
# CUDA stuff
CUDA_INSTALL_PATH=/opt/cuda
PATH=$PATH:/opt/cuda/bin
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/cuda/lib
SDK_INSTALL_PATH=/opt/cuda/NVIDIA_CUDA_SDK
export PATH
export CUDA_INSTALL_PATH
export SDK_INSTALL_PATH
export LD_LIBRARY_PATH
root@bt:~#
```

The terminal window has a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The background of the terminal window features a dark theme with a dragon logo and the text "back | track 4". The bottom of the window shows a taskbar with various icons and a system tray displaying the time "04:03".

Once you have installed the packages you need to update your `$PATH` variable and make sure the CUDA compiler is in the path.

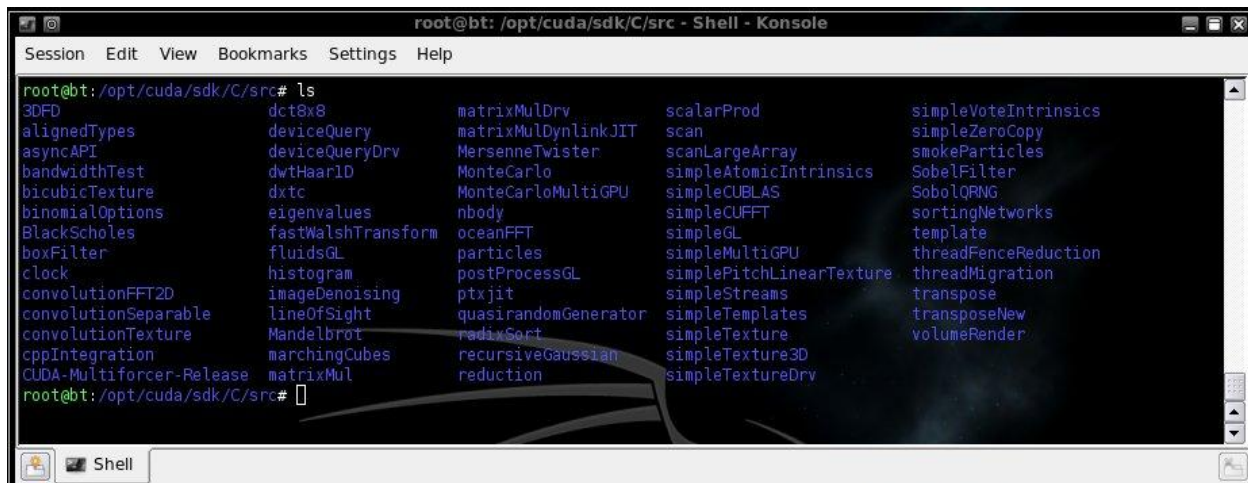


```
root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:~# source ~/.bashrc
root@bt:~# which nvcc
/opt/cuda/bin/nvcc
root@bt:~#
```

If your output looks like mine pictured here then you are good to go.

The code sample projects are located in the `/opt/cuda/sdk/C/src/` directory if you want to play around with them.



```
root@bt: /opt/cuda/sdk/C/src - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:/opt/cuda/sdk/C/src# ls
3DFD                               dct8x8                           matrixMulDrv                     scalarProd                       simpleVoteIntrinsics
alignedTypes                      deviceQuery                      matrixMulDynlinkJIT            scan                             simpleZeroCopy
asyncAPI                         deviceQueryDrv                  MersenneTwister                scanLargeArray                  smokeParticles
bandwidthTest                    dwthaar1D                       MonteCarlo                      simpleAtomicIntrinsics          SobelFilter
bicubicTexture                   dxtc                           MonteCarloMultiGPU             simpleCUBLAS                     SobolQRNG
binomialOptions                  eigenvalues                     nbody                          simpleCUFFT                     sortingNetworks
BlackScholes                     fastWalshTransform              oceanFFT                       simpleGL                         template
boxFilter                        fluidsGL                        particles                       simpleMultiGPU                  threadFenceReduction
clock                            histogram                       postProcessGL                  simplePitchLinearTexture         threadMigration
convolutionFFT2D                 imageDenoising                  ptxjit                         simpleStreams                    transpose
convolutionSeparable             lineOfSight                     quasirandomGenerator           simpleTemplates                  transposeNew
convolutionTexture                Mandelbrot                      radixSort                      simpleTexture                    volumeRender
cppIntegration                   marchingCubes                    recursiveGaussian               simpleTexture3D
CUDA-MultiForcer-Release          matrixMul                       reduction                       simpleTextureDrv
```

Basically you go into the directory of the tool you want to build and issue the make command. The nvcc compiler then builds the tool and outputs the binary into the `/opt/cuda/sdk/C/bin/linux/release` directory and then can be used.

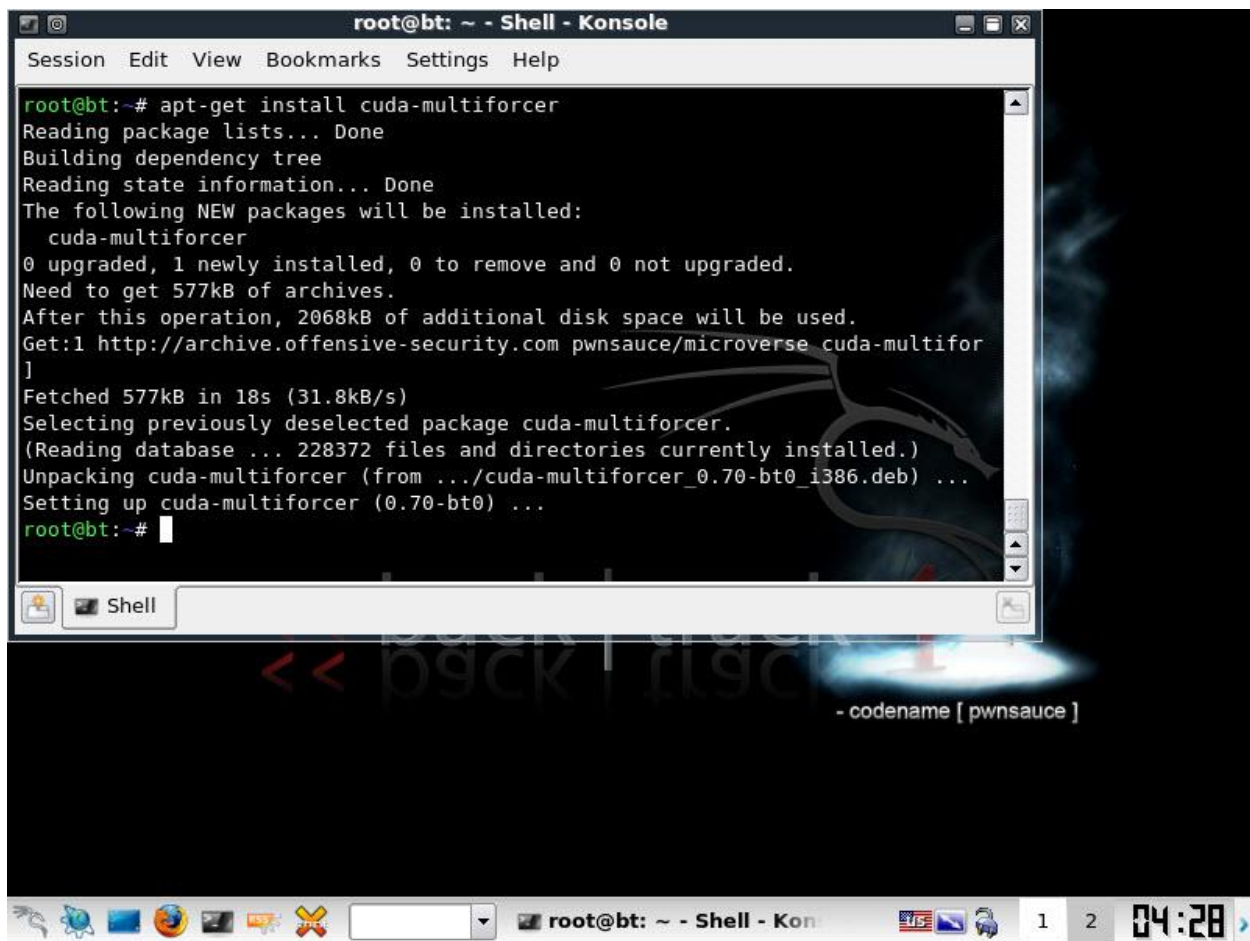
```
root@bt: /opt/cuda/sdk/C/bin/linux/release - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:/opt/cuda/sdk/C/src# cd deviceQuery
root@bt:/opt/cuda/sdk/C/src/deviceQuery# make
root@bt:/opt/cuda/sdk/C/src/deviceQuery# cd /opt/cuda/sdk/C/bin/linux/release/
root@bt:/opt/cuda/sdk/C/bin/linux/release# ls -l
total 116
-rwxr-xr-x 1 root root 110424 2010-01-18 09:22 deviceQuery
root@bt:/opt/cuda/sdk/C/bin/linux/release# ./deviceQuery
CUDA Device Query (Runtime API) version (CUDART static linking)
There are 2 devices supporting CUDA

Device 0: "GeForce 8800 GT"
  CUDA Driver Version:            2.30
  CUDA Runtime Version:          2.30
  CUDA Capability Major revision number:    1
  CUDA Capability Minor revision number:    1
  Total amount of global memory:         536543232 bytes
  Number of multiprocessors:             14
  Number of cores:                      112
  Total amount of constant memory:       65536 bytes
  Total amount of shared memory per block: 16384 bytes
  Total number of registers available per block: 8192
  Warp size:                           32
  Maximum number of threads per block:   512
  Maximum sizes of each dimension of a block: 512 x 512 x 64
  Maximum sizes of each dimension of a grid: 65535 x 65535 x 1
  Maximum memory pitch:                  262144 bytes
  Texture alignment:                     256 bytes
  Clock rate:                            1.51 GHz
```

## CUDA-Multiforcer:

One of the newest tools in Backtrack 4 is the CUDA-Multiforcer. This is a password bruteforcer which supports MD4 / MD5/ NTLM and now SHA1 hash's. It is incredibly fast and can greatly decrease the time it takes to crack password hash's while on a pentest. Installation of the multi-forcer is simple.

A screenshot of a terminal window titled "root@bt: ~ - Shell - Konsole". The terminal shows the command "apt-get install cuda-multiforcer" being executed. The output indicates that the package is being installed from a repository. The terminal also shows the installation progress, including the size of the package and the disk space requirements. The background of the terminal window features a dark, abstract image with a glowing blue light source. The terminal window is part of a desktop environment with a taskbar at the bottom showing various application icons and a system clock displaying "04:28".

```
root@bt:~# apt-get install cuda-multiforcer
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  cuda-multiforcer
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 577kB of archives.
After this operation, 2068kB of additional disk space will be used.
Get:1 http://archive.offensive-security.com pwnsauce/microverse cuda-multifor
]
Fetched 577kB in 18s (31.8kB/s)
Selecting previously deselected package cuda-multiforcer.
(Reading database ... 228372 files and directories currently installed.)
Unpacking cuda-multiforcer (from .../cuda-multiforcer_0.70-bt0_i386.deb) ...
Setting up cuda-multiforcer (0.70-bt0) ...
root@bt:~#
```

Running the multi-forcer with no arguments will show you the required options for running the tool.

```
root@bt:/pentest/passwords/cuda-multiforcer# ./CUDA-Multiforcer
```

Currently supported hash types: MD5 FASTMD5 MD4 FASTMD4 NTLM FASTNTLM SHA1 FASTSHA1

```
./CUDA-Multiforcer: missing option -h|--hashtype={NTLM, MD4, MD5}
```

```
./CUDA-Multiforcer: missing option -f|--hashfile=hashfile
```

```
./CUDA-Multiforcer: missing option --min=<n>
```

```
./CUDA-Multiforcer: missing option --max=<n>
```

Here is a example of running the Cuda-multi-forcer against a NTLM hash



```
root@bt: /pentest/passwords/cuda-multiforcer - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:/pentest/passwords/cuda-multiforcer# ./CUDA-Multiforcer -h NTLM -f ntlm.txt -c charsets/charsetfull --min=5 --max=5
Cryptohaze.com CUDA Multiforcer (multiple hash brute forcer)
by Bitweasil
Version 0.70, length 0-14
Currently supported hash types: MD5 FASTMD5 MD4 FASTMD4 NTLM FASTNTLM SHA1 FASTSHA1
Hash type: NTLM
!!!! cuMemGetInfo failed! (status = 3)CUDA Device Information:
Device 0: "GeForce 8800 GT"
  Number of cores:      112
  Clock rate:          1.51 GHz
  Memory (total):      2936 MB
  Memory (free):       214 MB

Single charset loaded.
Loading & sorting hashes. This may take a while.
Returning from sort.
Hashes loaded (1 hashes)
You may want to consider the FAST[hash] option if it exists.
Done with copy to constant
Launching kernel for password length 5
Done: 4.97% Step rate: 151.8M/s Search rate: 151.8M/sec
NTLM : BBFB842B4BCA06E82F191D87F906BE22: 'ph33r' 0x7068333372
Done: 23.39% Step rate: 153.2M/s Search rate: 153.2M/sec
```

There are different charset files in the charsets directory or you can make a custom one. The syntax of the file is very simple.

```
root@bt:/pentest/passwords/cuda-multiforcer/charsets# cat charsetfull
```

```
!"#$%&'()*+,-
./0123456789;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

The `-f` argument requires a single hash or a file of hash's

The multi-forcer can also handle very large lists of hash's. I loaded 4000 MD5 passwords as a test and the tool easily loaded them all and started cracking.

The only limitation of this tool I can find is that it only supports one GPU card. This example was tested on a 8800 GT card but I have run the tool with a 295 gtx and seen some really amazing speeds. The home page for the cuda-multiforcer is <http://www.cryptohaze.com>. I would like to give a special thanks to Bit Weasil for a great tool and his help with my understanding of CUDA.

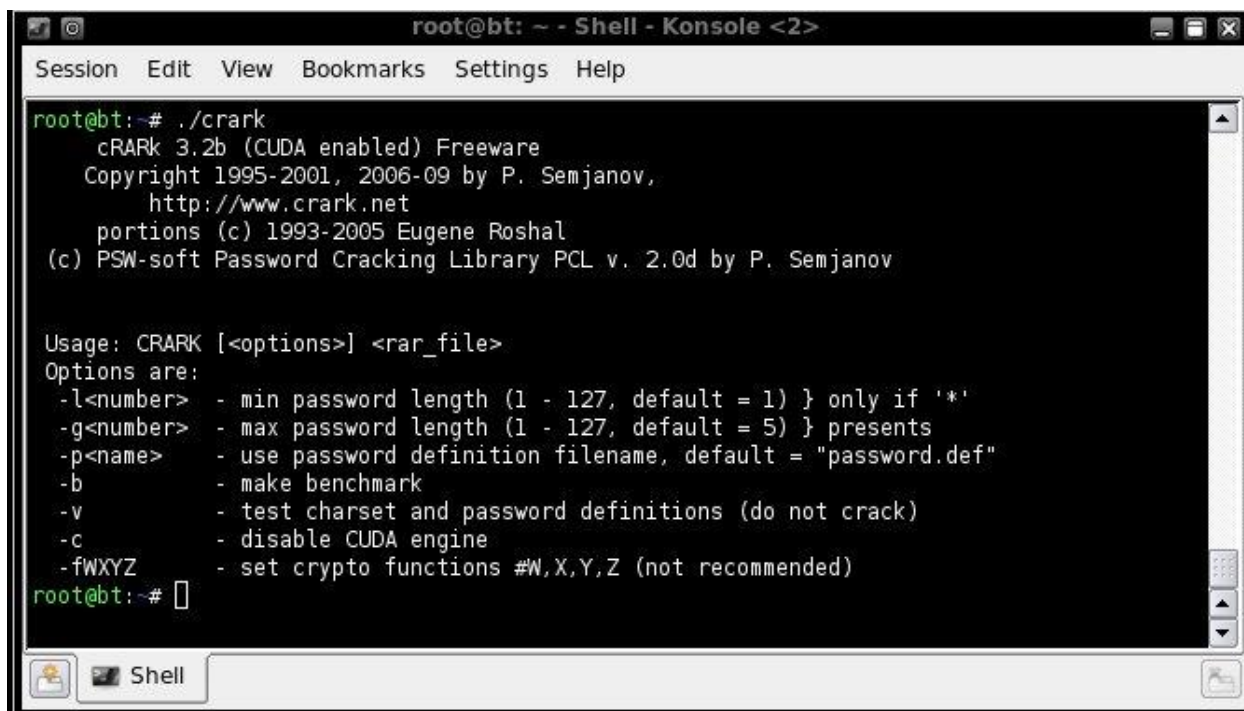
## Crark:

Crark is the newest tool in Backtracks CUDA arsenal. It is a GPU powered rar archive cracker. It works in the same way as many password tools by specifying a charset and then attacking the file with a brute force. The home page for crark is <http://www.crark.net/>

Advantages of cRARK are:

1. One of the fastest RAR password recovery software, uses extremely optimized MMX & SSE cod
2. Support of RAR password recovery on NVIDIA GPU using CUDA technology
3. Multi-volume, self-extracting, encrypted header archives support
4. Unique PCL language which is extremely efficient if user remembers any information about a forgotten password

To install the tool: apt-get install crark



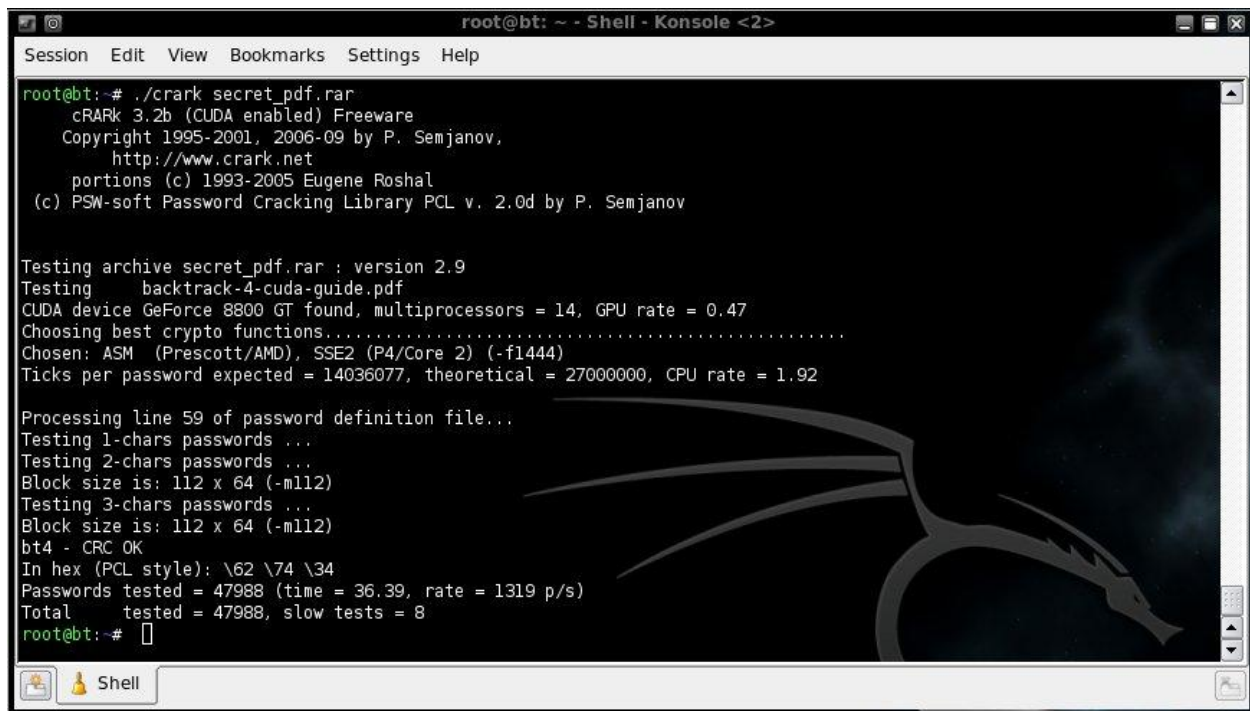
```
root@bt: ~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help

root@bt:~# ./crark
cRARK 3.2b (CUDA enabled) Freeware
Copyright 1995-2001, 2006-09 by P. Semjanov,
http://www.crark.net
portions (c) 1993-2005 Eugene Roshal
(c) PSW-soft Password Cracking Library PCL v. 2.0d by P. Semjanov

Usage: CRARK [<options>] <rar_file>
Options are:
-l<number> - min password length (1 - 127, default = 1) } only if '*'
-g<number> - max password length (1 - 127, default = 5) } presents
-p<name>    - use password definition filename, default = "password.def"
-b         - make benchmark
-v         - test charset and password definitions (do not crack)
-c         - disable CUDA engine
-fWXYZ     - set crypto functions #W,X,Y,Z (not recommended)

root@bt:~#
```

The charsets are defined in password.def. Have a look at the syntax of the file to figure out the charset you need to use. For the sake of this document I created a rar archive with a password of "bt4". Here is a example of an attack.



```
root@bt: ~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help

root@bt: # ./crark secret_pdf.rar
cRARK 3.2b (CUDA enabled) Freeware
Copyright 1995-2001, 2006-09 by P. Semjanov,
http://www.crark.net
portions (c) 1993-2005 Eugene Roshal
(c) PSW-soft Password Cracking Library PCL v. 2.0d by P. Semjanov

Testing archive secret_pdf.rar : version 2.9
Testing backtrack-4-cuda-guide.pdf
CUDA device GeForce 8800 GT found, multiprocessors = 14, GPU rate = 0.47
Choosing best crypto functions.....
Chosen: ASM (Prescott/AMD), SSE2 (P4/Core 2) (-f1444)
Ticks per password expected = 14036077, theoretical = 27000000, CPU rate = 1.92

Processing line 59 of password definition file...
Testing 1-chars passwords ...
Testing 2-chars passwords ...
Block size is: 112 x 64 (-m112)
Testing 3-chars passwords ...
Block size is: 112 x 64 (-m112)
bt4 - CRC OK
In hex (PCL style): \62 \74 \34
Passwords tested = 47988 (time = 36.39, rate = 1319 p/s)
Total tested = 47988, slow tests = 8
root@bt: #
```

The entire manual for this tool is available for viewing online at <http://www.crark.net/cRARK.html>

There is a very complete breakdown of how to use the charset file and how to set the other options to achieve maximum results.



# Pyrit

## What is pyrit?

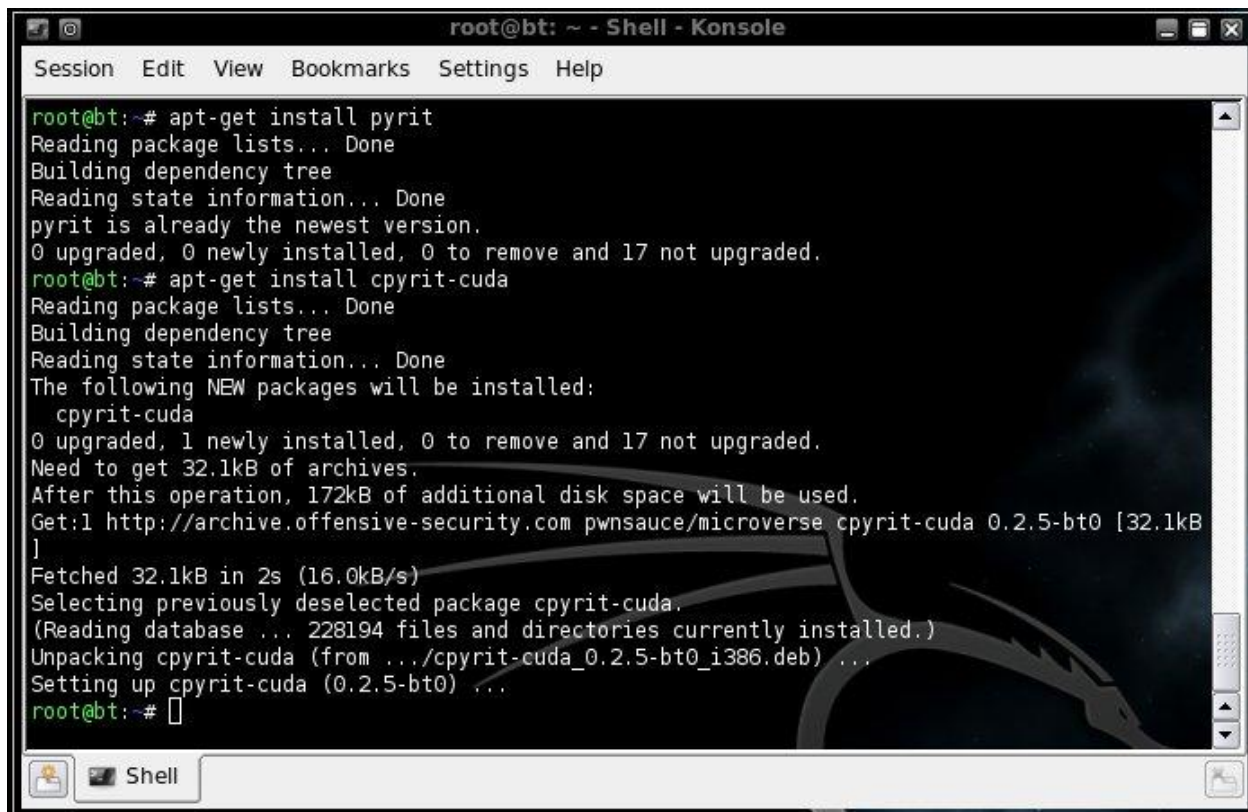
Pyrit takes a step ahead in attacking WPA-PSK and WPA2-PSK, the protocol that today de-facto protects public WIFI-airspace. The project's goal is to estimate the real-world security provided by these protocols. Pyrit does not provide binary files or wordlists and does not encourage anyone to participate or engage in any harmful activity. This is a research project, not a cracking tool.

Pyrit's implementation allows to create massive databases, pre-computing part of the WPA/WPA2-PSK authentication phase in a space-time-tradeoff. The performance gain for real-world-attacks is in the range of three orders of magnitude which urges for re-consideration of the protocol's security.

Exploiting the computational power of GPUs, Pyrit is currently by far the most powerful attack against one of the world's most used security-protocols.

## Up and running with pyrit :

Pyrit is already included in the backtrack .iso however the cuda core is not. In order to make sure we have the most recent version of both we will need to apt-get them.



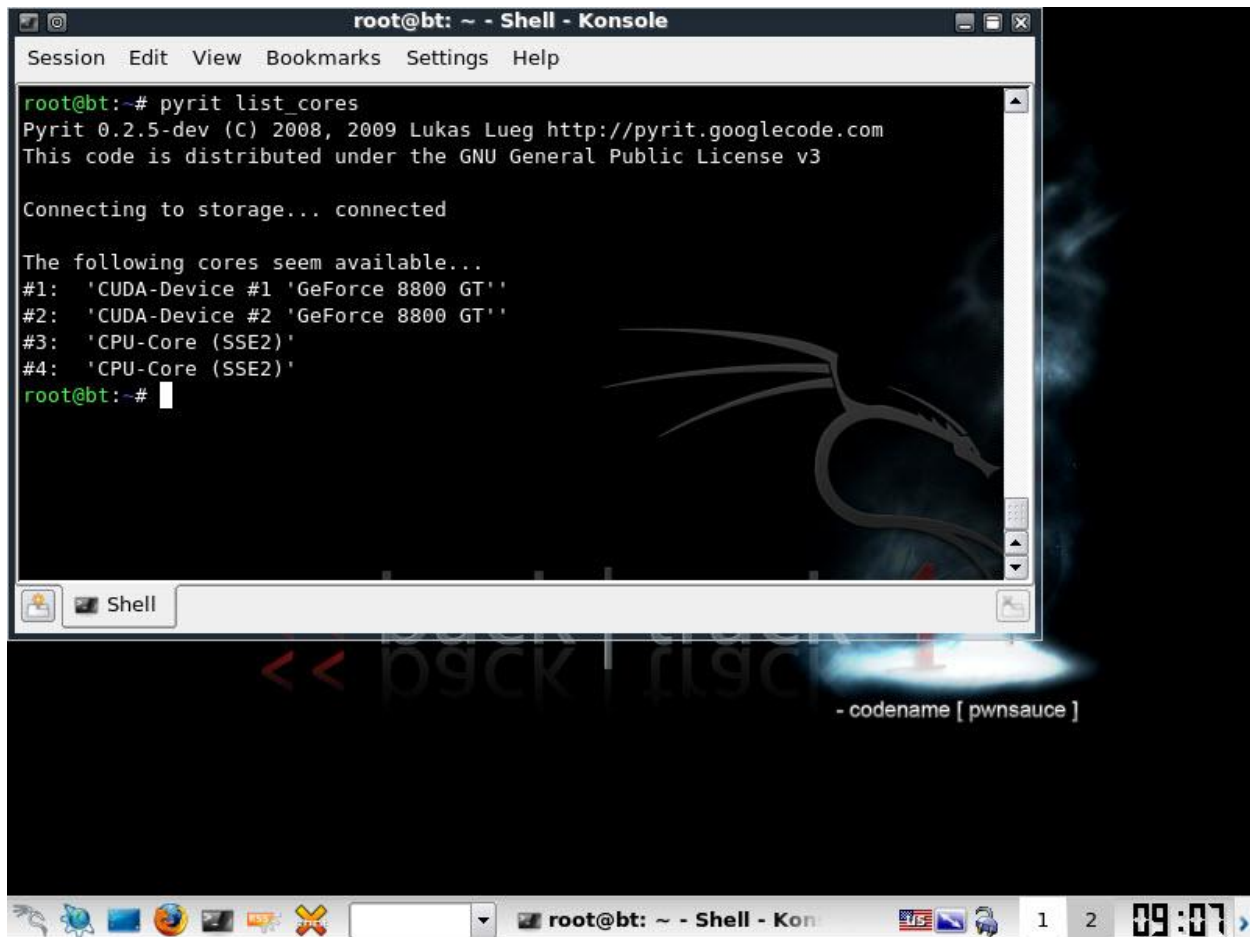
```
root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:~# apt-get install pyrit
Reading package lists... Done
Building dependency tree
Reading state information... Done
pyrit is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 17 not upgraded.
root@bt:~# apt-get install cpyrit-cuda
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  cpyrit-cuda
0 upgraded, 1 newly installed, 0 to remove and 17 not upgraded.
Need to get 32.1kB of archives.
After this operation, 172kB of additional disk space will be used.
Get:1 http://archive.offensive-security.com pwnsauce/microverse cpyrit-cuda 0.2.5-bt0 [32.1kB]
Fetched 32.1kB in 2s (16.0kB/s)
Selecting previously deselected package cpyrit-cuda.
(Reading database ... 228194 files and directories currently installed.)
Unpacking cpyrit-cuda (from .../cpyrit-cuda_0.2.5-bt0_i386.deb) ...
Setting up cpyrit-cuda (0.2.5-bt0) ...
root@bt:~#
```

## Making sure Pyrit is working:

There are a few small tests to run and see if Pyrit is working properly.

### List Cores:



```
root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:~# pyrit list_cores
Pyrit 0.2.5-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Connecting to storage... connected

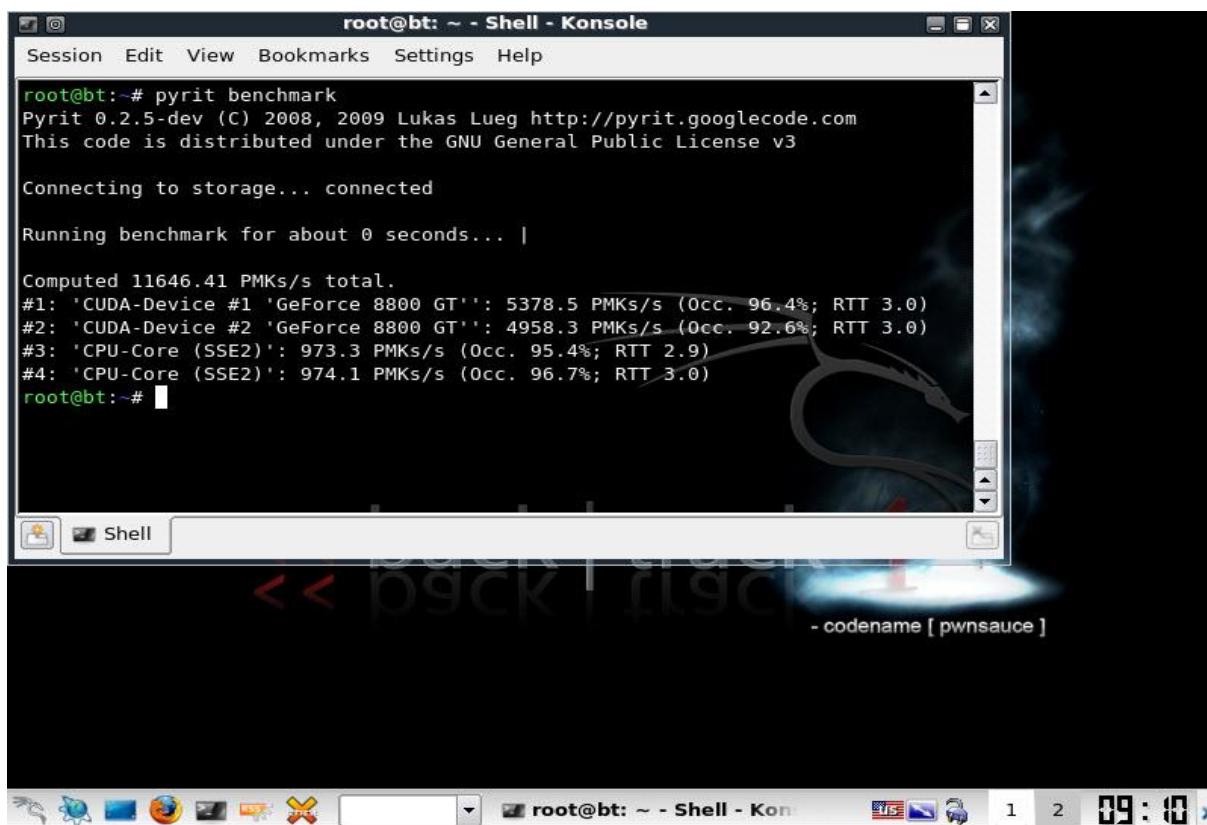
The following cores seem available...
#1: 'CUDA-Device #1 'GeForce 8800 GT''
#2: 'CUDA-Device #2 'GeForce 8800 GT''
#3: 'CPU-Core (SSE2)'
#4: 'CPU-Core (SSE2)'
root@bt:~#
```

This will show us a output of how many video cards and cores are available for pyrit to use. Something interesting to note is that each gpu will use one core of the cpu in order to run at optimal speed. Only the extra cores will be listed. In this example the processor is a intel Q6600 which is a quad core cpu but

since there are two video cards each one of the uses a cpu core therefore only the other two are listed as being available.

## Benchmark:

The next test is the benchmark. This will show use the occupancy of the cores in a percentage value and it will give us a rough estimate of the amount of PMK/second than pyrite will be using. As you can see in the picture the occupancy of the video cards is 96 and 92 percent. The reason for this loss is because the X-server is running. Running pyrit from the framebuffer will allow you to squeeze every ounce of power out possible. Also just as a side note, running pyrit while the X-server is running will render your desktop pretty much useless until its finished so plan on doing something else while the numbers are crunching.



```
root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:~# pyrit benchmark
Pyrit 0.2.5-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Connecting to storage... connected

Running benchmark for about 0 seconds... |

Computed 11646.41 PMKs/s total.
#1: 'CUDA-Device #1 'GeForce 8800 GT': 5378.5 PMKs/s (Occ. 96.4%; RTT 3.0)
#2: 'CUDA-Device #2 'GeForce 8800 GT': 4958.3 PMKs/s (Occ. 92.6%; RTT 3.0)
#3: 'CPU-Core (SSE2)': 973.3 PMKs/s (Occ. 95.4%; RTT 2.9)
#4: 'CPU-Core (SSE2)': 974.1 PMKs/s (Occ. 96.7%; RTT 3.0)
root@bt:~#
```

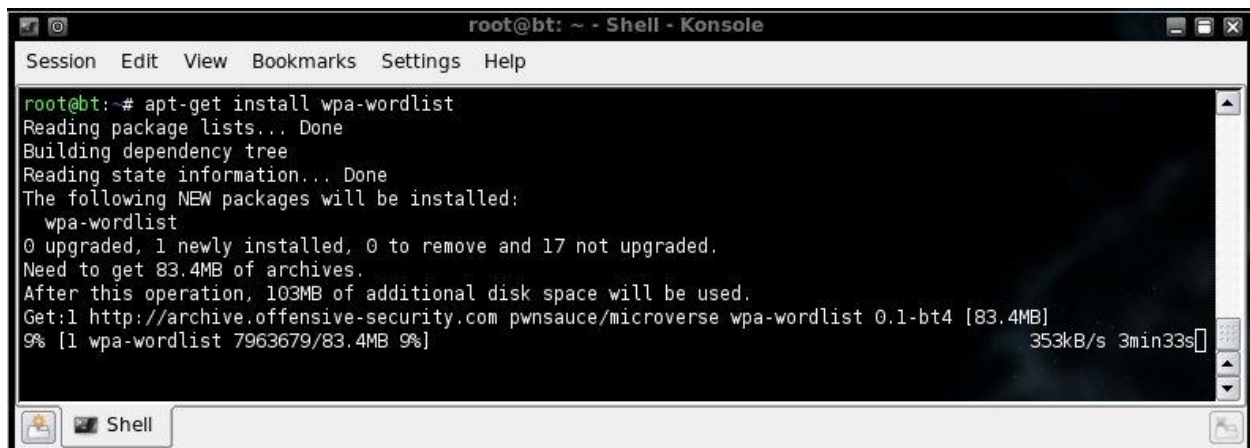
The complete reference guide for all the pyrite commands can be found at

<http://code.google.com/p/pyrit/wiki/ReferenceManual>

I will try to highlight a few of the more popular examples here in this document.

## The WPA Wordlist:

We have created a optimized wpa password list for users to get started with. This was too big for the .iso however we can easily grab it from the repo with apt-get

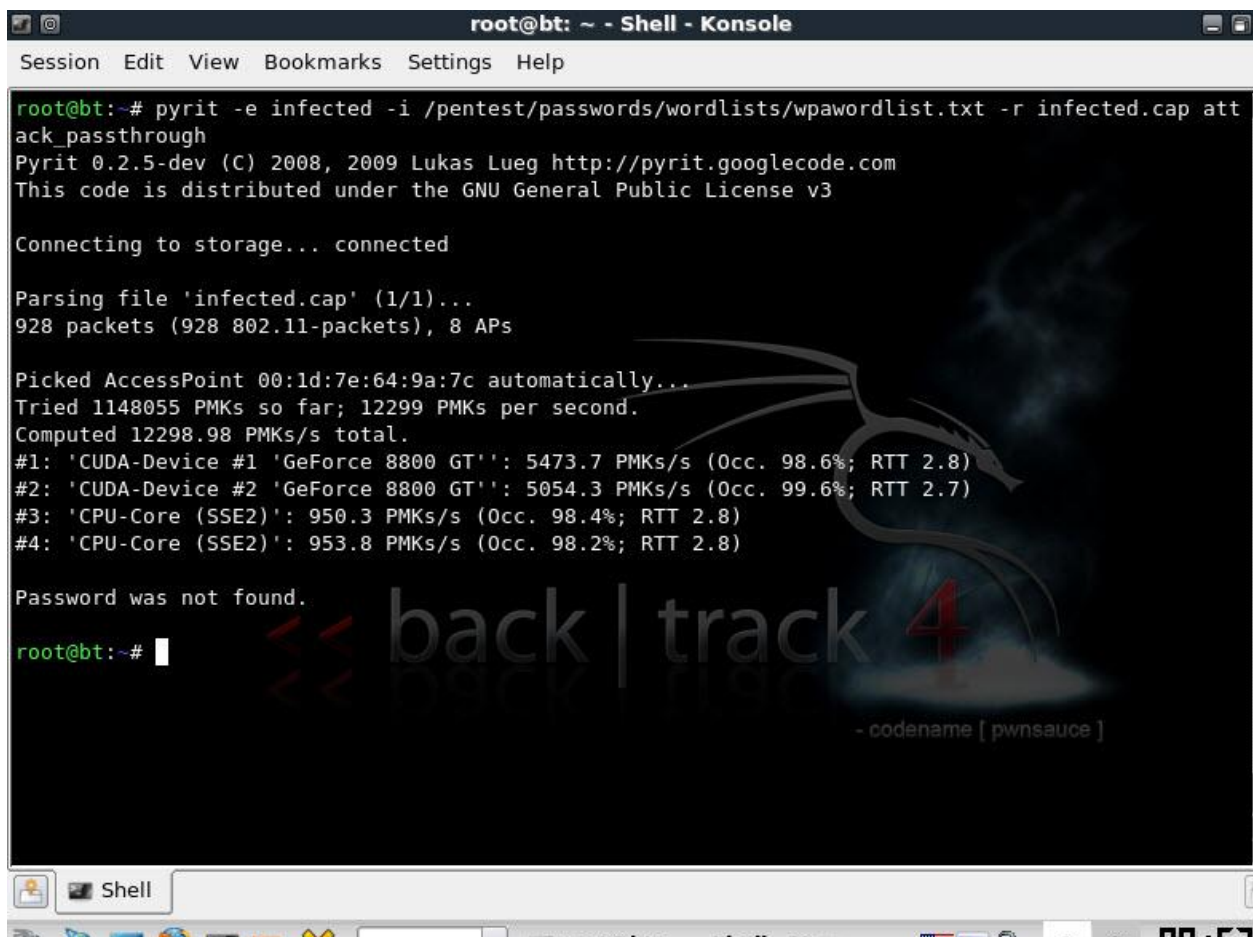
A screenshot of a terminal window titled "root@bt: ~ - Shell - Konsole". The terminal shows the command "apt-get install wpa-wordlist" being executed. The output indicates that the package is being installed from a remote source. The progress bar shows 9% completion. The terminal text is as follows:

```
root@bt:~# apt-get install wpa-wordlist
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
 wpa-wordlist
0 upgraded, 1 newly installed, 0 to remove and 17 not upgraded.
Need to get 83.4MB of archives.
After this operation, 103MB of additional disk space will be used.
Get:1 http://archive.offensive-security.com/pwnsauce/microverse wpa-wordlist 0.1-bt4 [83.4MB]
9% [1 wpa-wordlist 7963679/83.4MB 9%] 353kB/s 3min33s
```

## Attack\_passthrough Mode:

The first way it can be run is in passthrough mode. The reason this mode is nice is because instead of created bulky tables and writing them to hard disk, Pyrit simply computes the hash's and pipes them directly into cowpatty. Aircrack-ng does not currently support this option. In order to use this option simply create a command string with the following syntax:

```
root@bt:~# pyrit -e infected -i /pentest/passwords/wordlists/wpawordlist.txt -r infected.cap  
attack_passthrough
```



```
root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:~# pyrit -e infected -i /pentest/passwords/wordlists/wpawordlist.txt -r infected.cap att
ack_passthrough
Pyrit 0.2.5-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Connecting to storage... connected

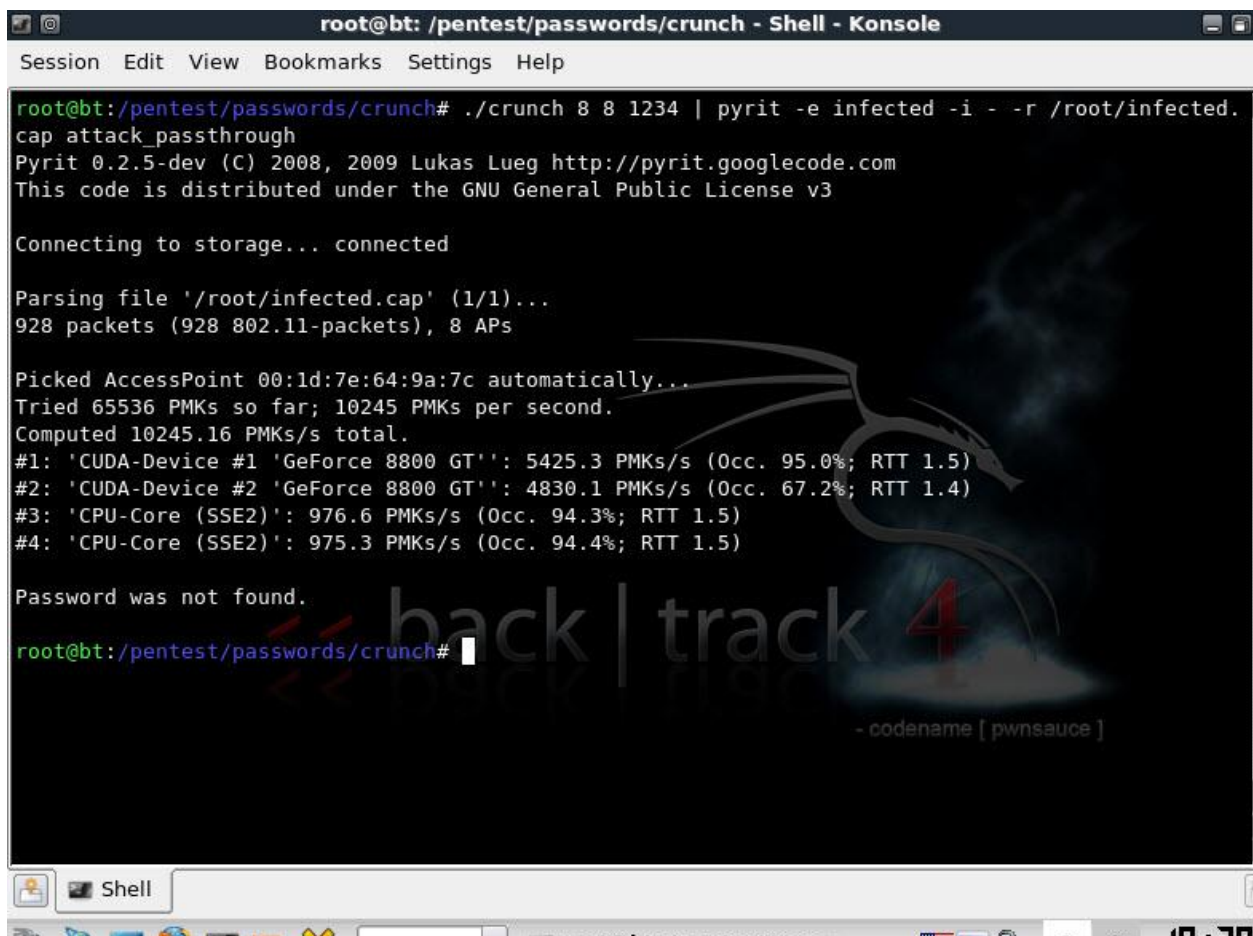
Parsing file 'infected.cap' (1/1)...
928 packets (928 802.11-packets), 8 APs

Picked AccessPoint 00:1d:7e:64:9a:7c automatically...
Tried 1148055 PMKs so far; 12299 PMKs per second.
Computed 12298.98 PMKs/s total.
#1: 'CUDA-Device #1 'GeForce 8800 GT': 5473.7 PMKs/s (Occ. 98.6%; RTT 2.8)
#2: 'CUDA-Device #2 'GeForce 8800 GT': 5054.3 PMKs/s (Occ. 99.6%; RTT 2.7)
#3: 'CPU-Core (SSE2)': 950.3 PMKs/s (Occ. 98.4%; RTT 2.8)
#4: 'CPU-Core (SSE2)': 953.8 PMKs/s (Occ. 98.2%; RTT 2.8)

Password was not found.
root@bt:~#
```

## Attack-passthrough with Crunch:

Although brute forcing WPA is pretty much useless I will show one way it can be done. If the passphrase was all digits or a phone number this would be a viable option. We can use the tool crunch which is located on the backtrack .iso:

A screenshot of a terminal window titled 'root@bt: /pentest/passwords/crunch - Shell - Konsole'. The terminal shows the execution of the command './crunch 8 8 1234 | pyrit -e infected -i - -r /root/infected.cap attack\_passthrough'. The output includes the Pyrit version (0.2.5-dev), a license notice, and performance statistics for different hardware components. The terminal also features a large 'back | track 4' watermark and the text '- codename [ pwnsauce ]'.

```
root@bt: /pentest/passwords/crunch - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:/pentest/passwords/crunch# ./crunch 8 8 1234 | pyrit -e infected -i - -r /root/infected.
cap attack_passthrough
Pyrit 0.2.5-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Connecting to storage... connected

Parsing file '/root/infected.cap' (1/1)...
928 packets (928 802.11-packets), 8 APs

Picked AccessPoint 00:1d:7e:64:9a:7c automatically...
Tried 65536 PMKs so far; 10245 PMKs per second.
Computed 10245.16 PMKs/s total.
#1: 'CUDA-Device #1 'GeForce 8800 GT': 5425.3 PMKs/s (Occ. 95.0%; RTT 1.5)
#2: 'CUDA-Device #2 'GeForce 8800 GT': 4830.1 PMKs/s (Occ. 67.2%; RTT 1.4)
#3: 'CPU-Core (SSE2)': 976.6 PMKs/s (Occ. 94.3%; RTT 1.5)
#4: 'CPU-Core (SSE2)': 975.3 PMKs/s (Occ. 94.4%; RTT 1.5)

Password was not found.
root@bt:/pentest/passwords/crunch#
```

The syntax of the crunch command is:

Crunch <min length> <max length> <charset> -o <output file> or | to stdin to pyrite

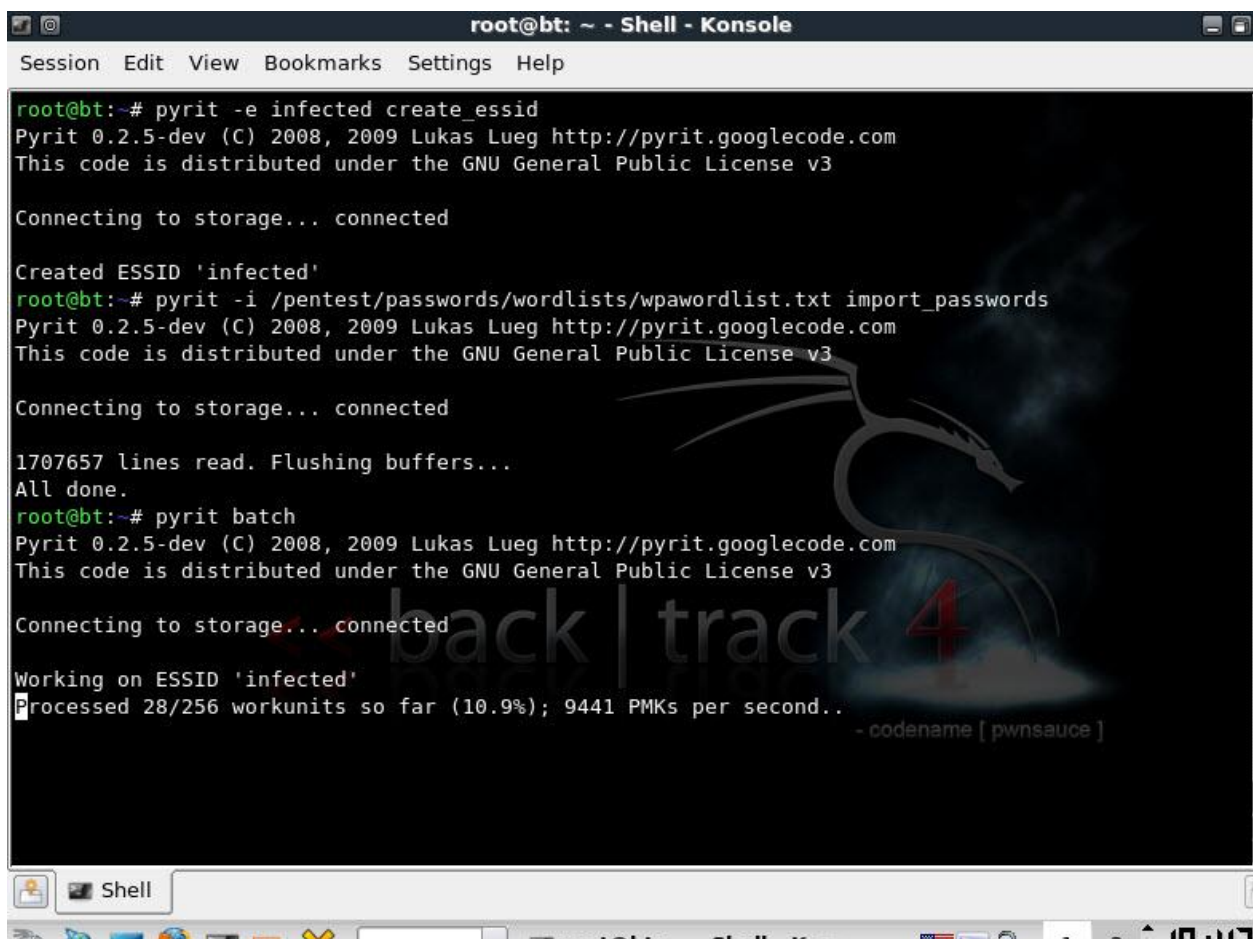
This attack will take a very long time if used with a full charset.



## Batch Mode:

Creating tables with pyrit involves a few extra steps but you will have created a table which can be used over and over as long as the essid of the AP is the same.

The basic procedure for this is to import a essid, import a password list and then run the batch command in order to compute the tables.



```
root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:~# pyrit -e infected create_essid
Pyrit 0.2.5-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Connecting to storage... connected

Created ESSID 'infected'
root@bt:~# pyrit -i /pentest/passwords/wordlists/wpawordlist.txt import_passwords
Pyrit 0.2.5-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Connecting to storage... connected

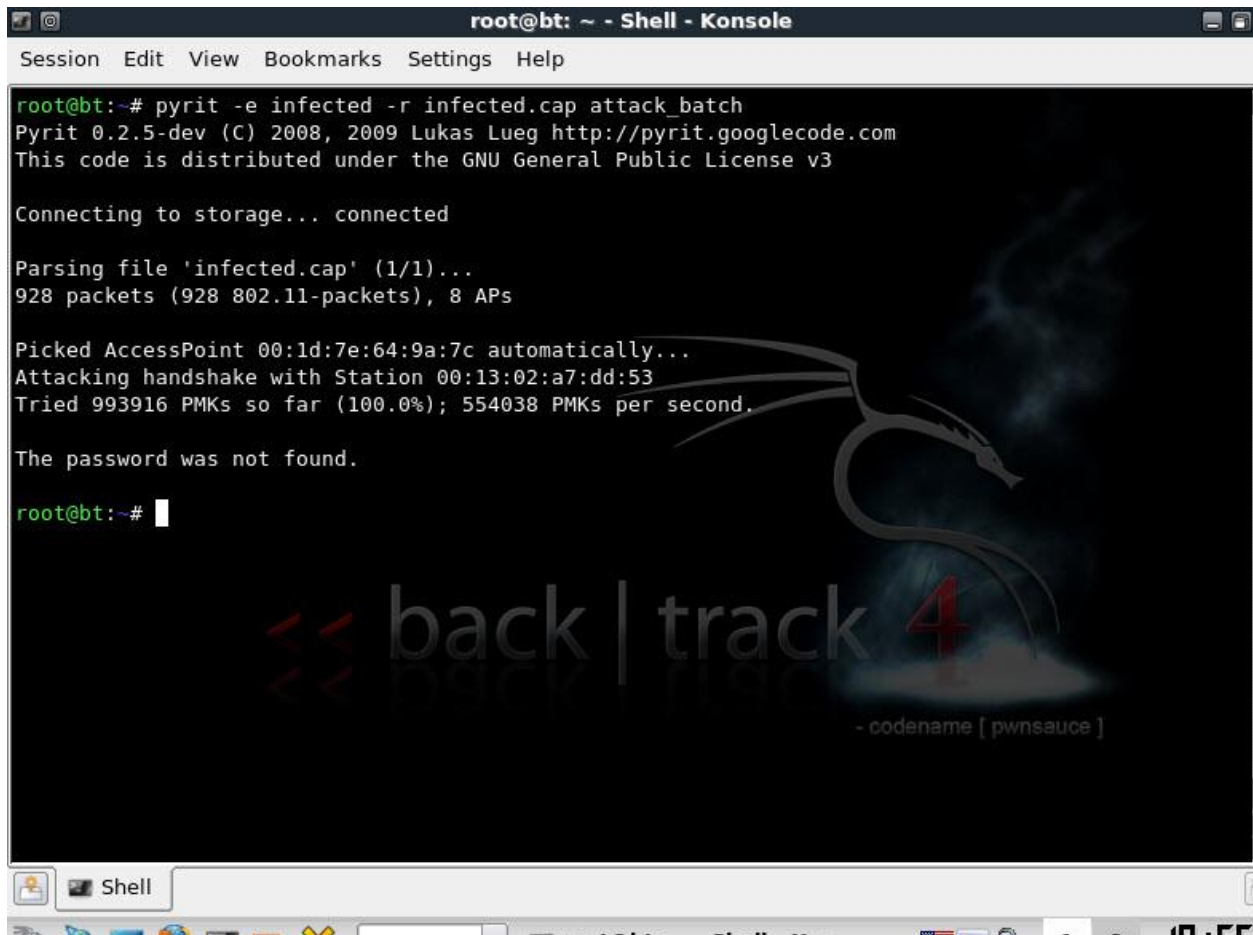
1707657 lines read. Flushing buffers...
All done.
root@bt:~# pyrit batch
Pyrit 0.2.5-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Connecting to storage... connected

Working on ESSID 'infected'
Processed 28/256 workunits so far (10.9%); 9441 PMKs per second..
- codename [ pwnsauce ]
```



Now once you have a database made there are many things you can do. You can still export to cowpatty or airolib-ng format just like always. I am going to skip these two options and highlight the newer ones. Pyrit now has the ability to attack the passwords without any other program. Once you have a database created you can attack the capture directly without any export necessary.



```
root@bt: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

root@bt:~# pyrit -e infected -r infected.cap attack_batch
Pyrit 0.2.5-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Connecting to storage... connected

Parsing file 'infected.cap' (1/1)...
928 packets (928 802.11-packets), 8 APs

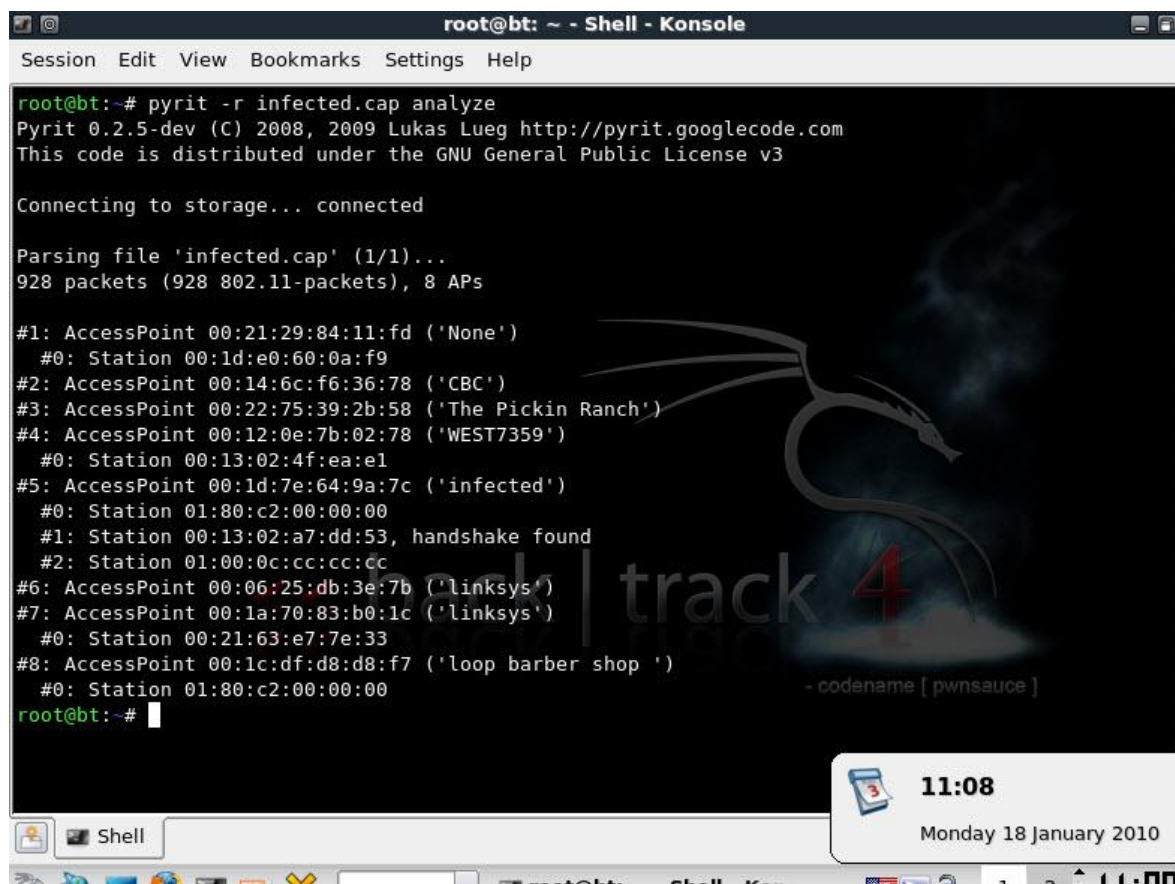
Picked AccessPoint 00:1d:7e:64:9a:7c automatically...
Attacking handshake with Station 00:13:02:a7:dd:53
Tried 993916 PMKs so far (100.0%); 554038 PMKs per second.

The password was not found.

root@bt:~#
```

## Analyze:

Another nice new feature of Pyrit, is the ability to analyze the file by itself.

A screenshot of a terminal window titled "root@bt: ~ - Shell - Konsole". The terminal shows the command "pyrit -r infected.cap analyze" being executed. The output displays the Pyrit version (0.2.5-dev), copyright information (2008, 2009 Lukas Lueg), and the GNU GPL license. It then shows the file "infected.cap" being parsed, resulting in 928 packets and 8 Access Points. A list of detected Access Points and Stations is shown, including their MAC addresses and names. One station is identified as having a valid EAPOL handshake. The terminal also shows a watermark for "track 4" and a system clock in the bottom right corner indicating 11:08 on Monday, 18 January 2010.

```
root@bt:~# pyrit -r infected.cap analyze
Pyrit 0.2.5-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Connecting to storage... connected

Parsing file 'infected.cap' (1/1)...
928 packets (928 802.11-packets), 8 APs

#1: AccessPoint 00:21:29:84:11:fd ('None')
#0: Station 00:1d:e0:60:0a:f9
#2: AccessPoint 00:14:6c:f6:36:78 ('CBC')
#3: AccessPoint 00:22:75:39:2b:58 ('The Pickin Ranch')
#4: AccessPoint 00:12:0e:7b:02:78 ('WEST7359')
#0: Station 00:13:02:4f:ea:e1
#5: AccessPoint 00:1d:7e:64:9a:7c ('infected')
#0: Station 01:80:c2:00:00:00
#1: Station 00:13:02:a7:dd:53, handshake found
#2: Station 01:00:0c:cc:cc:cc
#6: AccessPoint 00:06:25:db:3e:7b ('linksys')
#7: AccessPoint 00:1a:70:83:b0:1c ('linksys')
#0: Station 00:21:63:e7:7e:33
#8: AccessPoint 00:1c:df:d8:d8:f7 ('loop barber shop ')
#0: Station 01:80:c2:00:00:00
root@bt:~#
```

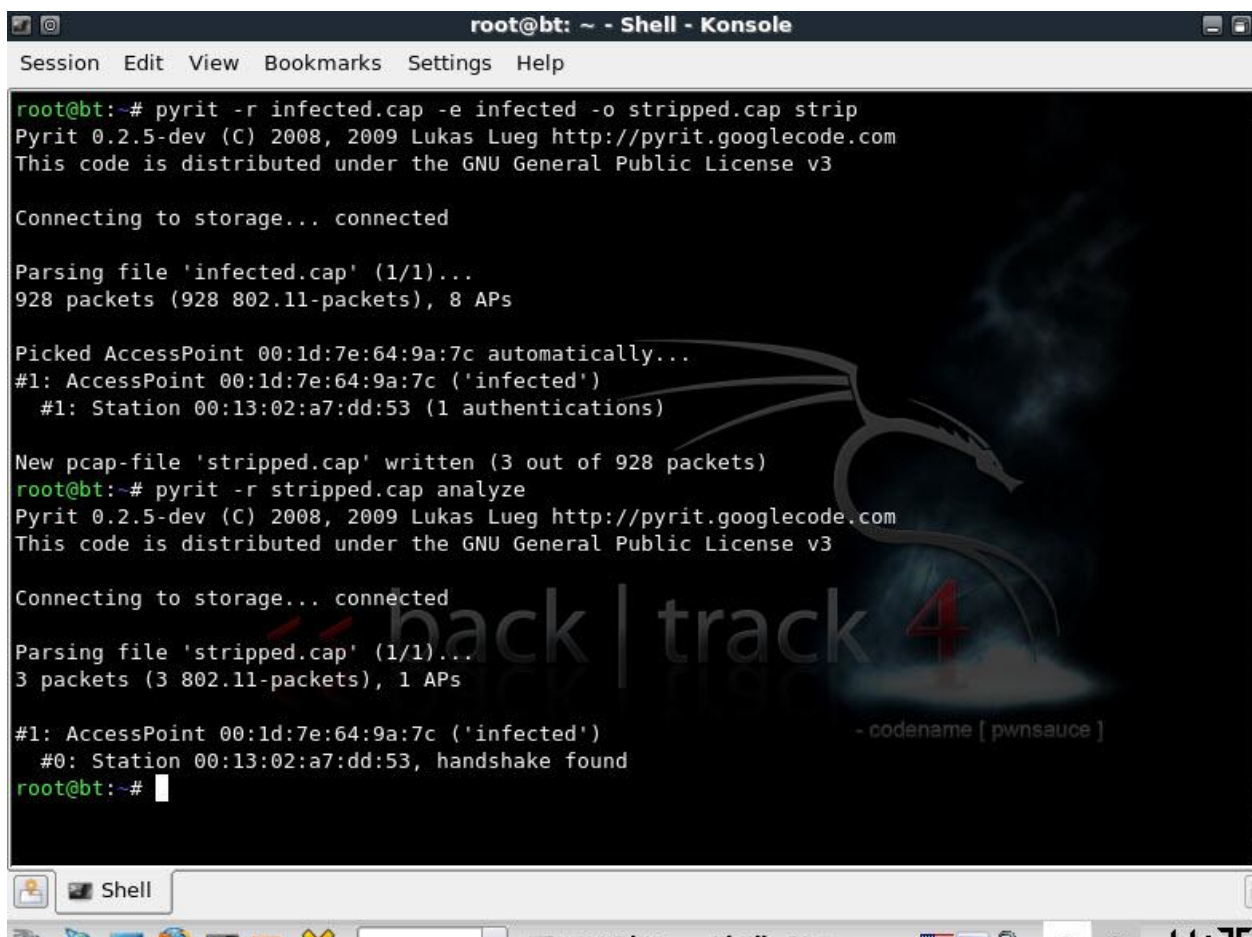
Analyze can parse one or more packet-capture files in cap and pcap-format and try to detect Access-Points, Stations and EAPOL-handshakes. For example:

```
pyrit -r "test*.pcap" analyze
```

The suffix '*handshake found*' is appended to the Station's BSSID if the communication between the Access-Point and the Station seems to include a valid EAPOL-handshake.

## Strip:

The last thing I would like to highlight is the strip feature. I have noticed that the one downfall of Pyrit is that the handshake detection is still a little lacking when compared with aircrack-ng or Cowpatty. The times when I have had problems is when there are multiple handshakes in on file. So there is a new strip feature in Pyrit to strip the file. Look back at the capture from the analyze section. Although there is only one handshake in there are some other access points we could get rid of.

A screenshot of a terminal window titled "root@bt: ~ - Shell - Konsole". The terminal shows the execution of Pyrit commands. First, the command "pyrit -r infected.cap -e infected -o stripped.cap strip" is run, which parses the file and strips unwanted access points. Then, the command "pyrit -r stripped.cap analyze" is run, which analyzes the stripped file and finds a handshake. The terminal output includes details about the number of packets, access points, and the handshake found. A watermark "hack | track 4" and "- codename [ pwnsauce ]" are visible in the background of the terminal window.

```
root@bt:~# pyrit -r infected.cap -e infected -o stripped.cap strip
Pyrit 0.2.5-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Connecting to storage... connected

Parsing file 'infected.cap' (1/1)...
928 packets (928 802.11-packets), 8 APs

Picked AccessPoint 00:1d:7e:64:9a:7c automatically...
#1: AccessPoint 00:1d:7e:64:9a:7c ('infected')
    #1: Station 00:13:02:a7:dd:53 (1 authentications)

New pcap-file 'stripped.cap' written (3 out of 928 packets)
root@bt:~# pyrit -r stripped.cap analyze
Pyrit 0.2.5-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Connecting to storage... connected

Parsing file 'stripped.cap' (1/1)...
3 packets (3 802.11-packets), 1 APs

#1: AccessPoint 00:1d:7e:64:9a:7c ('infected')
    #0: Station 00:13:02:a7:dd:53, handshake found
root@bt:~#
```

## Building aircrack-ng with CUDA support:

This is still under heavy development so it is not yet been added to the backtrack repositories however it deserves mentioning. Aircrack can be built with a switch to add GPU acceleration. In order to do this we need to grab aircrack from svn. You must have the toolkit and the sdk installed to be able to build this.

```
svn co http://trac.aircrack-ng.org/svn/branch/aircrack-ng-cuda aircrack-ng-cuda
```

Next we will build it like normal but it needs a few extra arguments

```
root@bt~# cd aircrack-ng-cuda
```

```
root@bt:~/aircrack-ng-cuda~#CUDA=true make
```

```
root@bt:~/aircrack-ng-cuda~#make CUDA=true sqlite=true unstable=true install
```

Test to ensure everything is working, run aircrack on the test wpa-psk capture file, with the included wordlist :

```
root@bt~# cd src
```

```
root@bt~# ./aircrack-ng -p 1 ../test/wpa.cap -w ../test/password.lst
```

The -p switch is what adds the CUDA function to aircrack-ng. I have tested the tool and it does work but like I said its underdevelopment and could use some optimization. In my testing pyrit was still quite a bit faster however your milage may vary.

Special thanks to Zermelo and fnord0 for testing and posting the results of this tool.

## Outro:

I have only scratched the surface of the power of CUDA in this document. I sincerely hope that programmers will continue to create penetration testing tools which use the awesome power of CUDA. I truly believe that parallel GPU computing is the future for all the repetitious tasks we as penetration testers have to do daily including passwords attacks, brute force attacks and fuzzing. I hope to develop some applications of my own as I become more and more adept at CUDA programming.

## Useful Links:

[http://www.nvidia.com/object/cuda\\_home.html](http://www.nvidia.com/object/cuda_home.html)

<http://forums.nvidia.com>

<http://impact.crhc.illinois.edu/ftp/report/impact-08-01-mcuda.pdf>

<https://visualization.hpc.mil/wiki/GPGPU>

[http://developer.download.nvidia.com/compute/cuda/1\\_0/NVIDIA\\_CUDA\\_Programming\\_Guide\\_1.0.pdf](http://developer.download.nvidia.com/compute/cuda/1_0/NVIDIA_CUDA_Programming_Guide_1.0.pdf)

<http://pyrit.wordpress.com/>

<http://www.cryptohaze.com/>

<http://backtrack-linux.org/forums/>

<http://www.offensive-security.com/blog/>

## Special Thanks:

There is no way to thank everyone who has helped out with this stuff but I will try to name a few:

The entire Backtrack-dev team, thebaron , ebfe, Synok, Gommet and the rest of the guys in #cuda IRC Channel. Zero\_Chaos and Grimmlin from the Pentoo team and the guys from the Net-Sploit Team. I would also like to give a special thanks to Edgan for the long tireless hours we worked together to understand this stuff and anyone else who I have forgotten because I have ADD.

Anyone who wishes to contact me may do so on our IRC channel #backtrack-linux on the freenode network. Please do not contact me via email with silly questions.

Thanks <3 Pureh@te