

# BackTrack 4 CUDA Guide

Written by

Pureh@te

# Table of Contents

What is CUDA?.....	3
Supported GPUs .....	3
Why do I care about CUDA?.....	3
Where can I get this CUDA thing? .....	3
What is CUDA not? .....	4
Getting started.....	4
Nvidia-drivers: .....	4
Overclocking: .....	5
Installing the CUDA toolkit and SDK : .....	8
CUDA Tools.....	12
CUDA-multiforcerc:.....	12
Pyrit .....	14
What is pyrit?.....	14
Up and running with pyrit.....	14
Making sure Pyrit is working:.....	15
Passthrough Mode: .....	16
Passthrough with Crunch:.....	17
Server / Client Mode: .....	21
Building aircrack-ng with CUDA support: .....	23
Cuda Debugger:.....	24
Useful Links: .....	25
Special Thanks:.....	25

## 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 late-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 pre 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 manger apt-get. 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@backtrack:~# apt-get install nvidia-driver
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  nvidia-driver
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 0B/13.0MB of archives.
After this operation, 81.4MB of additional disk space will be used.
Selecting previously deselected package nvidia-driver.
(Reading database ... 184628 files and directories currently installed.)
Unpacking nvidia-driver (from .../nvidia-driver_185.18.08-bt5_i386.deb) ...
Processing triggers for man-db ...
Setting up nvidia-driver (185.18.08-bt5) ...
-
```

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.

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 “nvidia-settings” in a terminal. The actual configuration is beyond the scope of this document however its fairly easy to understand.

## Overclocking:

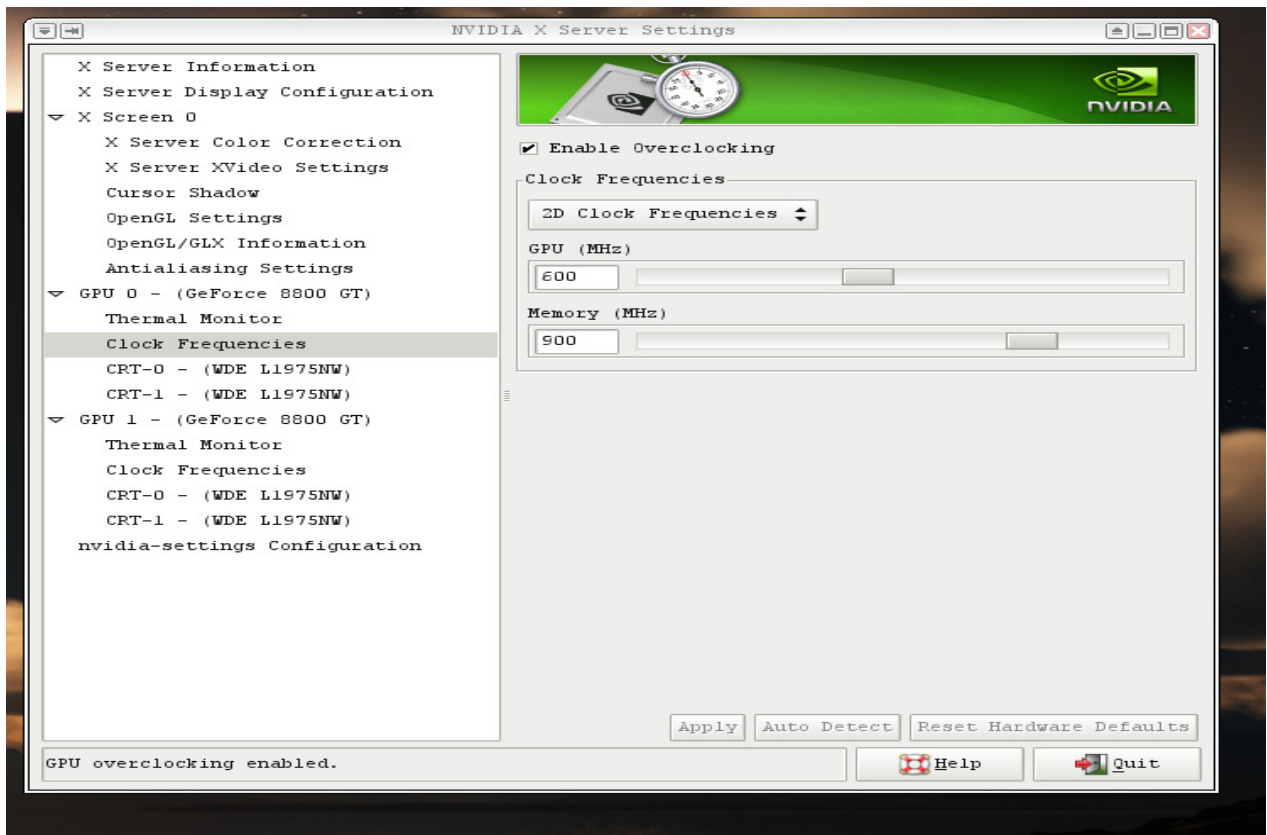
There are two ways to overclock your video card in Linux. The first way is to use the nvidia-settings tool which comes with the nvidia-driver. In order to do this you need to edit your xorg.conf in order to unlock the option.

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

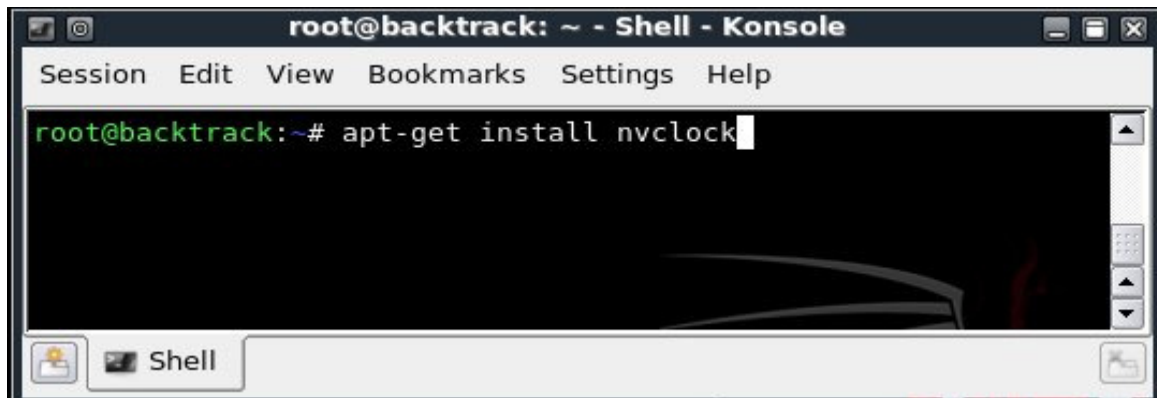
and find the section that looks like this:

```
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.



Then just run nvclock in a terminal to view the command line options:

```
root@bt ~ $ nvclock
```

```
NVClock v0.7
```

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:

-c	--card number	Number of the card to overclock
-m	--memclk speed	Memory speed in MHz
-n	--nvclk speed	Core speed in MHz
-r	--reset	Restore the original speeds

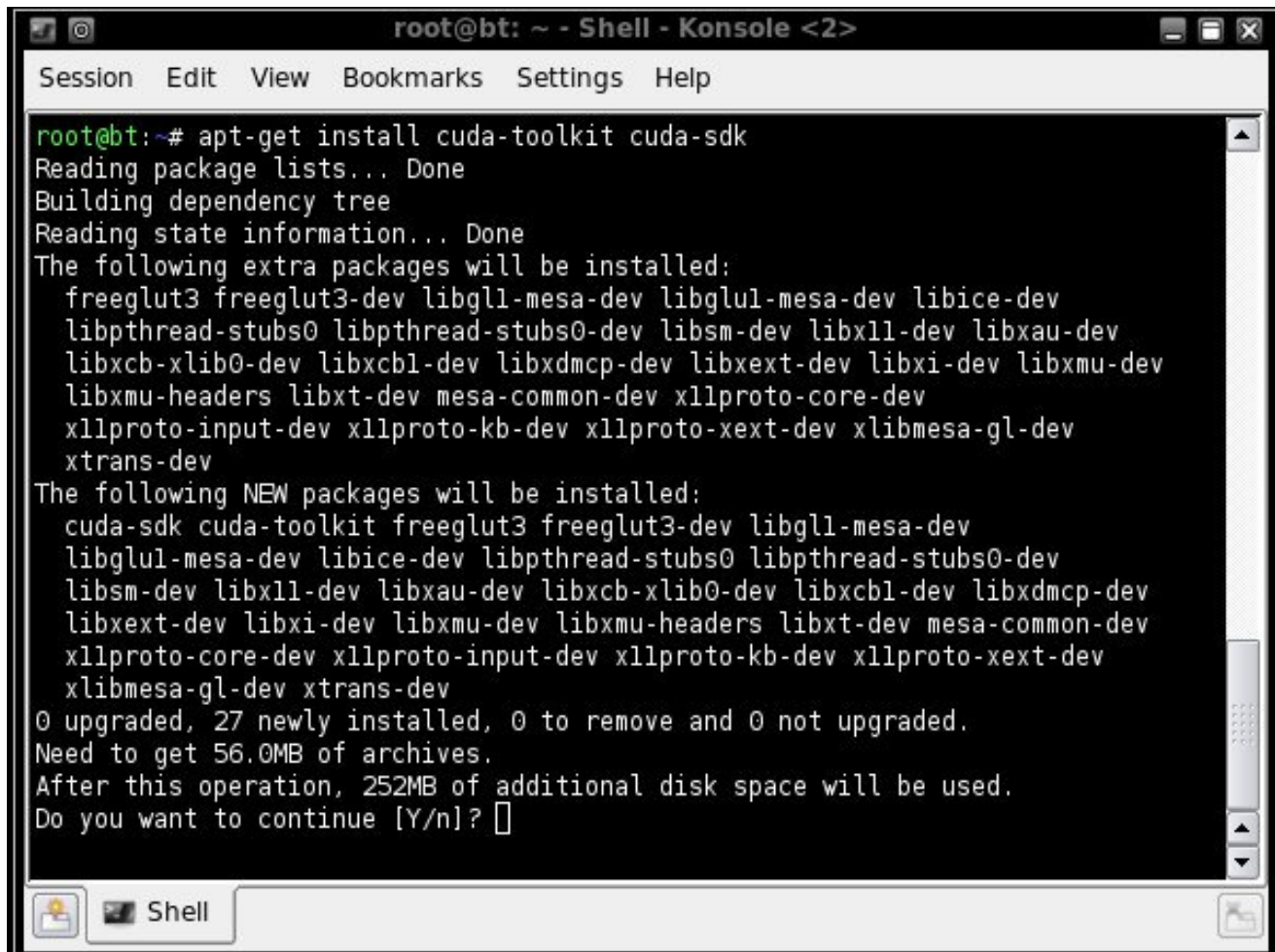
Other options:

-d	--debug	Enable/Disable debug info
-f	--force speeds	Force a speed, NVClock won't check min/max speeds
-h	--help	Show this help info
-i	--info	Print detailed card info
-s	--speeds	Print current speeds in MHz

## Installing the CUDA toolkit and SDK :

Now that we have our driver installed and the clock settings to our liking, its 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.

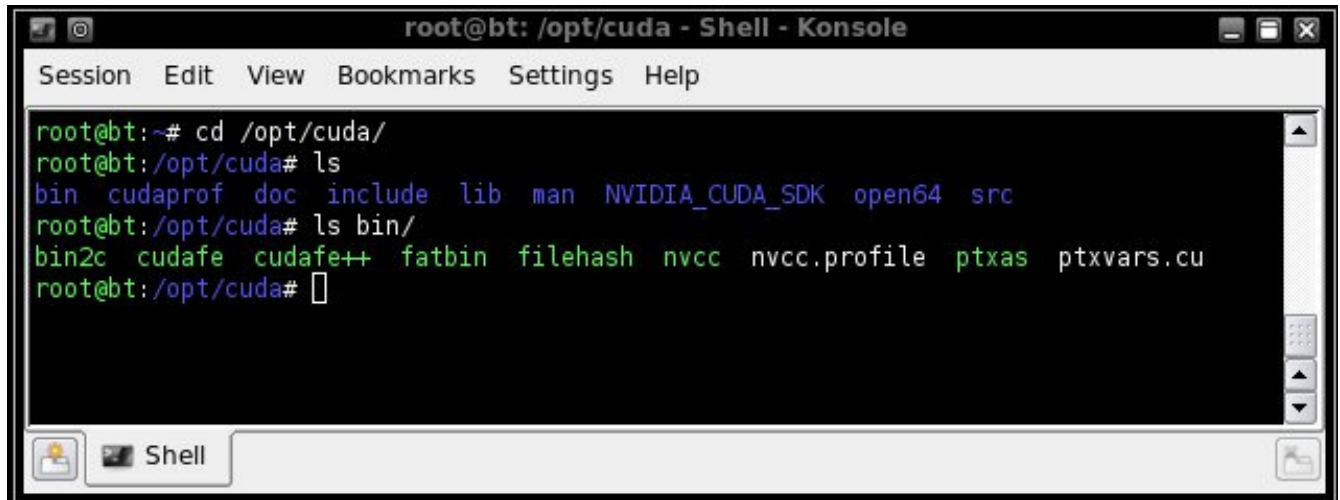
A terminal window titled "root@bt: ~ - Shell - Konsole <2>" showing the command "apt-get install cuda-toolkit cuda-sdk" and its output. The output lists extra packages to be installed, new packages to be installed, and disk space requirements. The prompt "Do you want to continue [Y/n]?" is visible at the end of the output.

```
root@bt:~# apt-get install cuda-toolkit cuda-sdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  freeglut3 freeglut3-dev libgl1-mesa-dev libglul-mesa-dev libice-dev
  libpthread-stubs0 libpthread-stubs0-dev libsm-dev libx11-dev libxau-dev
  libxcb-xlib0-dev libxcb1-dev libxdmcp-dev libxext-dev libxi-dev libxmu-dev
  libxmu-headers libxt-dev mesa-common-dev x11proto-core-dev
  x11proto-input-dev x11proto-kb-dev x11proto-xext-dev xlibmesa-gl-dev
  xtrans-dev
The following NEW packages will be installed:
  cuda-sdk cuda-toolkit freeglut3 freeglut3-dev libgl1-mesa-dev
  libglul-mesa-dev libice-dev libpthread-stubs0 libpthread-stubs0-dev
  libsm-dev libx11-dev libxau-dev libxcb-xlib0-dev libxcb1-dev libxdmcp-dev
  libxext-dev libxi-dev libxmu-dev libxmu-headers libxt-dev mesa-common-dev
  x11proto-core-dev x11proto-input-dev x11proto-kb-dev x11proto-xext-dev
  xlibmesa-gl-dev xtrans-dev
0 upgraded, 27 newly installed, 0 to remove and 0 not upgraded.
Need to get 56.0MB of archives.
After this operation, 252MB of additional disk space will be used.
Do you want to continue [Y/n]?
```

Once this is finished installing you will have every thing you need to build or program your own CUDA applications. I will provide some helpful programming links at the end of this document because how to program in CUDA is beyond the scope of this document. I will show some basic navigation.

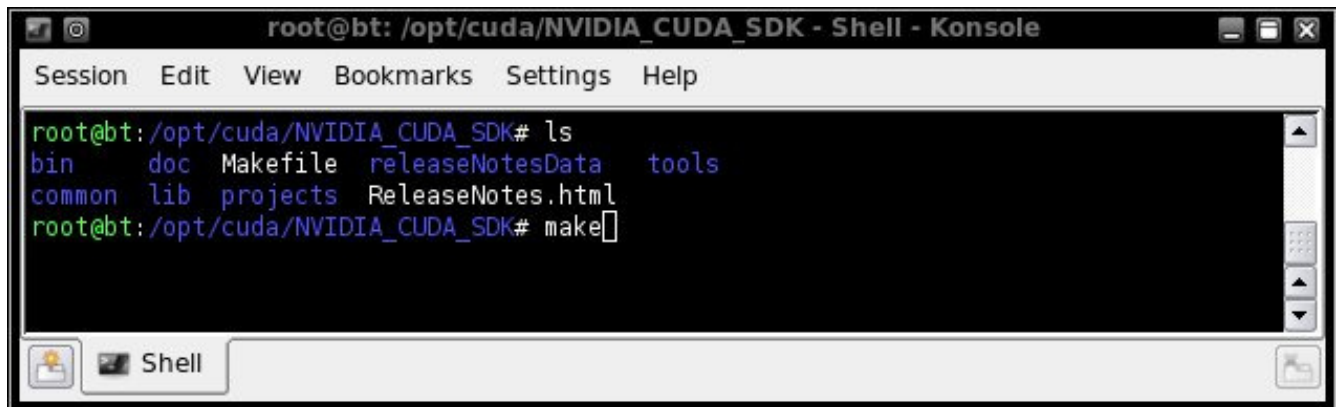


The initial environment is in `/opt/cuda` and in `/opt/cuda/bin` are the build tools for CUDA. The binary `nvcc` is the Nvidia compiler which is used to build applications.



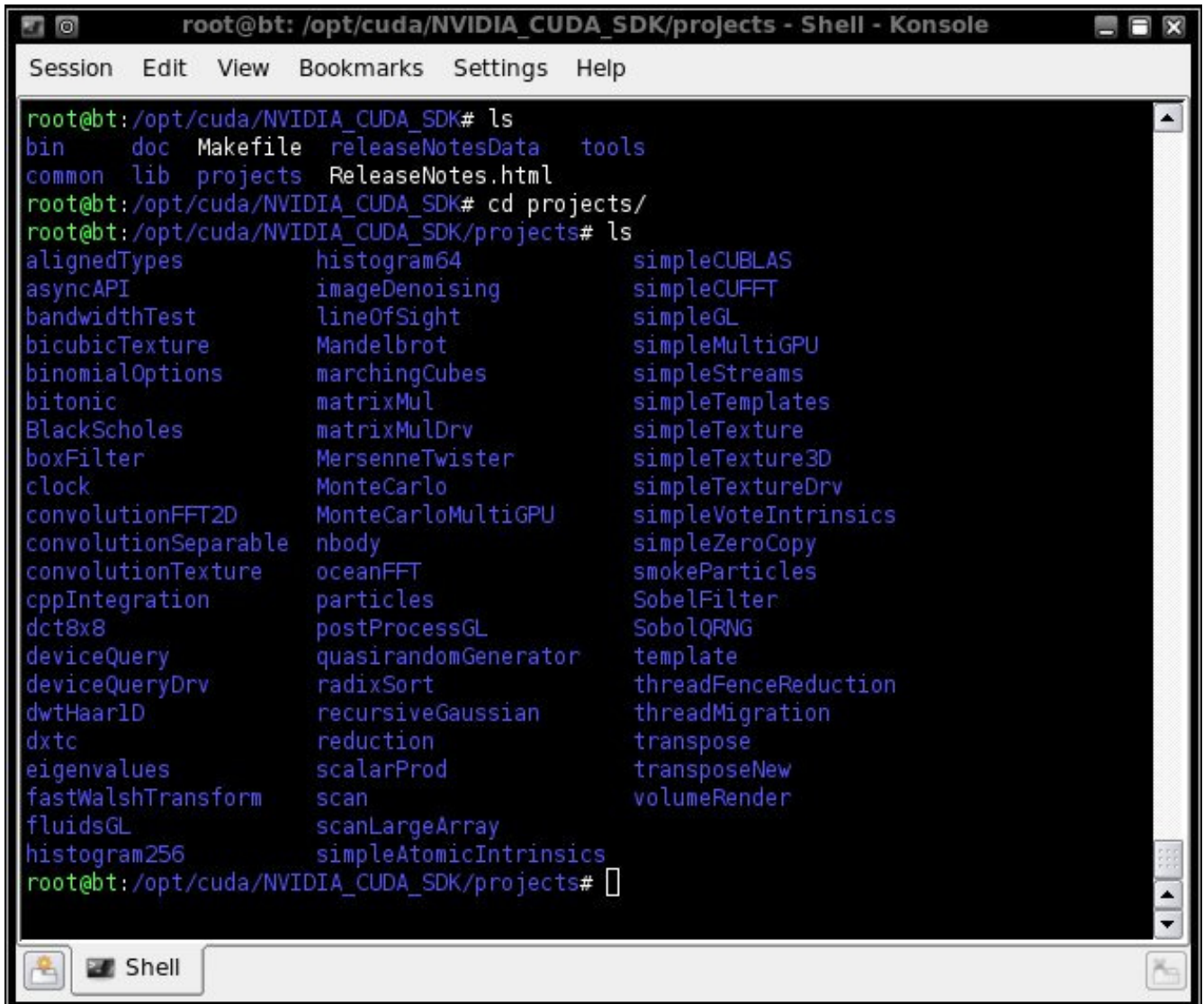
```
root@bt: /opt/cuda - Shell - Konsole
Session Edit View Bookmarks Settings Help
root@bt:~# cd /opt/cuda/
root@bt:/opt/cuda# ls
bin cudaprof doc include lib man NVIDIA_CUDA_SDK open64 src
root@bt:/opt/cuda# ls bin/
bin2c cudafe cudafe++ fatbin filehash nvcc nvcc.profile ptxas ptxvars.cu
root@bt:/opt/cuda#
```

The `cuda-sdk` package contains code samples to help you get started programming in CUDA. This environment is located in `/opt/cuda/NVIDIA_CUDA_SDK`. I have already built all the tools in this folder for you however if you would like to build them yourself simply navigate to the SDK folder and issue the command “`make clean`”. This will wipe out what I have built.



```
root@bt: /opt/cuda/NVIDIA_CUDA_SDK - Shell - Konsole
Session Edit View Bookmarks Settings Help
root@bt:/opt/cuda/NVIDIA_CUDA_SDK# ls
bin doc Makefile releaseNotesData tools
common lib projects ReleaseNotes.html
root@bt:/opt/cuda/NVIDIA_CUDA_SDK# make
```

If you issue the “make” command inside the main SDK folder it will build every tool it finds in the projects folder. If you prefer to build each sample one at a time, simply navigate to the projects folder and choose the tool you want.



```
root@bt: /opt/cuda/NVIDIA_CUDA_SDK/projects - Shell - Konsole
Session Edit View Bookmarks Settings Help
root@bt: /opt/cuda/NVIDIA_CUDA_SDK# ls
bin doc Makefile releaseNotesData tools
common lib projects ReleaseNotes.html
root@bt: /opt/cuda/NVIDIA_CUDA_SDK# cd projects/
root@bt: /opt/cuda/NVIDIA_CUDA_SDK/projects# ls
alignedTypes histogram64 simpleCUBLAS
asyncAPI imageDenoising simpleCUFFT
bandwidthTest lineOfSight simpleGL
bicubicTexture Mandelbrot simpleMultiGPU
binomialOptions marchingCubes simpleStreams
bitonic matrixMul simpleTemplates
BlackScholes matrixMulDrv simpleTexture
boxFilter MersenneTwister simpleTexture3D
clock MonteCarlo simpleTextureDrv
convolutionFFT2D MonteCarloMultiGPU simpleVoteIntrinsics
convolutionSeparable nbody simpleZeroCopy
convolutionTexture oceanFFT smokeParticles
cppIntegration particles SobelFilter
dct8x8 postProcessGL SobolQRNG
deviceQuery quasirandomGenerator template
deviceQueryDrv radixSort threadFenceReduction
dwtHaar1D recursiveGaussian threadMigration
dxtc reduction transpose
eigenvalues scalarProd transposeNew
fastWalshTransform scan volumeRender
fluidsGL scanLargeArray
histogram256 simpleAtomicIntrinsics
root@bt: /opt/cuda/NVIDIA_CUDA_SDK/projects#
```

For this example we will use DeviceQuery. In order to build this, cd in the DeviceQuery folder and issue the “make” command. This will build the tool. The result is then placed in /opt/cuda/NVIDIA\_CUDA\_SDK/bin/linux/release. To run our newly built tool we just navigate to that folder and run the binary just like normal.

```
root@bt: /opt/cuda/NVIDIA_CUDA_SDK/bin/linux/release - Shell - Konsole
Session Edit View Bookmarks Settings Help
root@bt: /opt/cuda/NVIDIA_CUDA_SDK/bin/linux/release# ./deviceQuery
CUDA Device Query (Runtime API) version (CUDA static linking)
There are 2 devices supporting CUDA

Device 0: "GeForce 8800 GT"
  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
  Concurrent copy and execution:              Yes
  Run time limit on kernels:                  Yes
  Integrated:                                  No
  Support host page-locked memory mapping:    No
  Compute mode:                               Default (multiple host threads can use this device simultaneously)

Device 1: "GeForce 8800 GT"
  CUDA Capability Major revision number:      1
  CUDA Capability Minor revision number:      1
  Total amount of global memory:              536608768 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
  Concurrent copy and execution:              Yes
  Run time limit on kernels:                  Yes
  Integrated:                                  No
  Support host page-locked memory mapping:    No
  Compute mode:                               Default (multiple host threads can use this device simultaneously)

Test PASSED
```

Here

is an example of the DeviceQuery running on 2 Nvidia 8800 GT cards

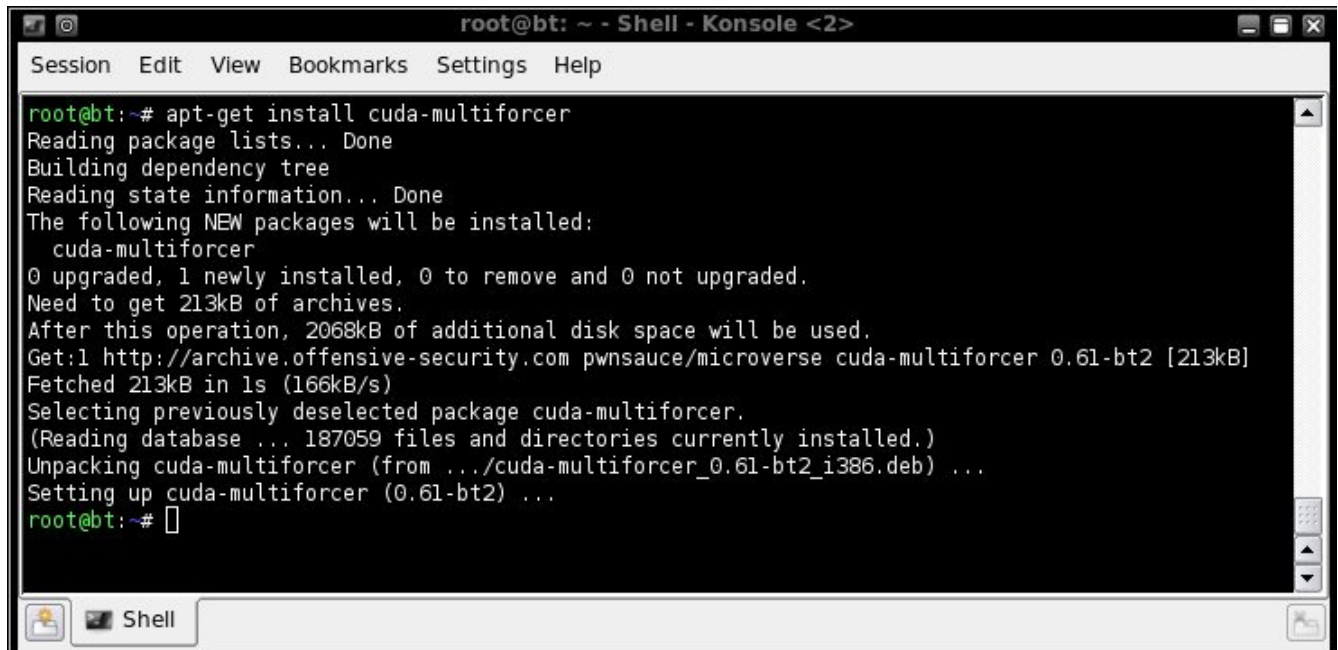
As I said before, issuing the “make” command in the root sdk directory will build all the sample tools. Anything built will appear in the release directory.

There are lots of things which can be done with CUDA parallel computing. The tools include here are only the beginning.

# CUDA Tools

## CUDA-multiforcer:

One of the newest tools in Backtrack 4 is the CUDA-Multiforcer. This is a password bruteforcer which supports MD4 / MD5 and NTLM 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.



```
root@bt: ~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
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 213kB of archives.
After this operation, 2068kB of additional disk space will be used.
Get:1 http://archive.offensive-security.com pwnsauce/microverse cuda-multiforcer 0.61-bt2 [213kB]
Fetched 213kB in 1s (166kB/s)
Selecting previously deselected package cuda-multiforcer.
(Reading database ... 187059 files and directories currently installed.)
Unpacking cuda-multiforcer (from ../cuda-multiforcer_0.61-bt2_i386.deb) ...
Setting up cuda-multiforcer (0.61-bt2) ...
root@bt:~#
```

You can either launch the tool from the KDE menu or you can directly navigate to the folder in the /pentest directory.

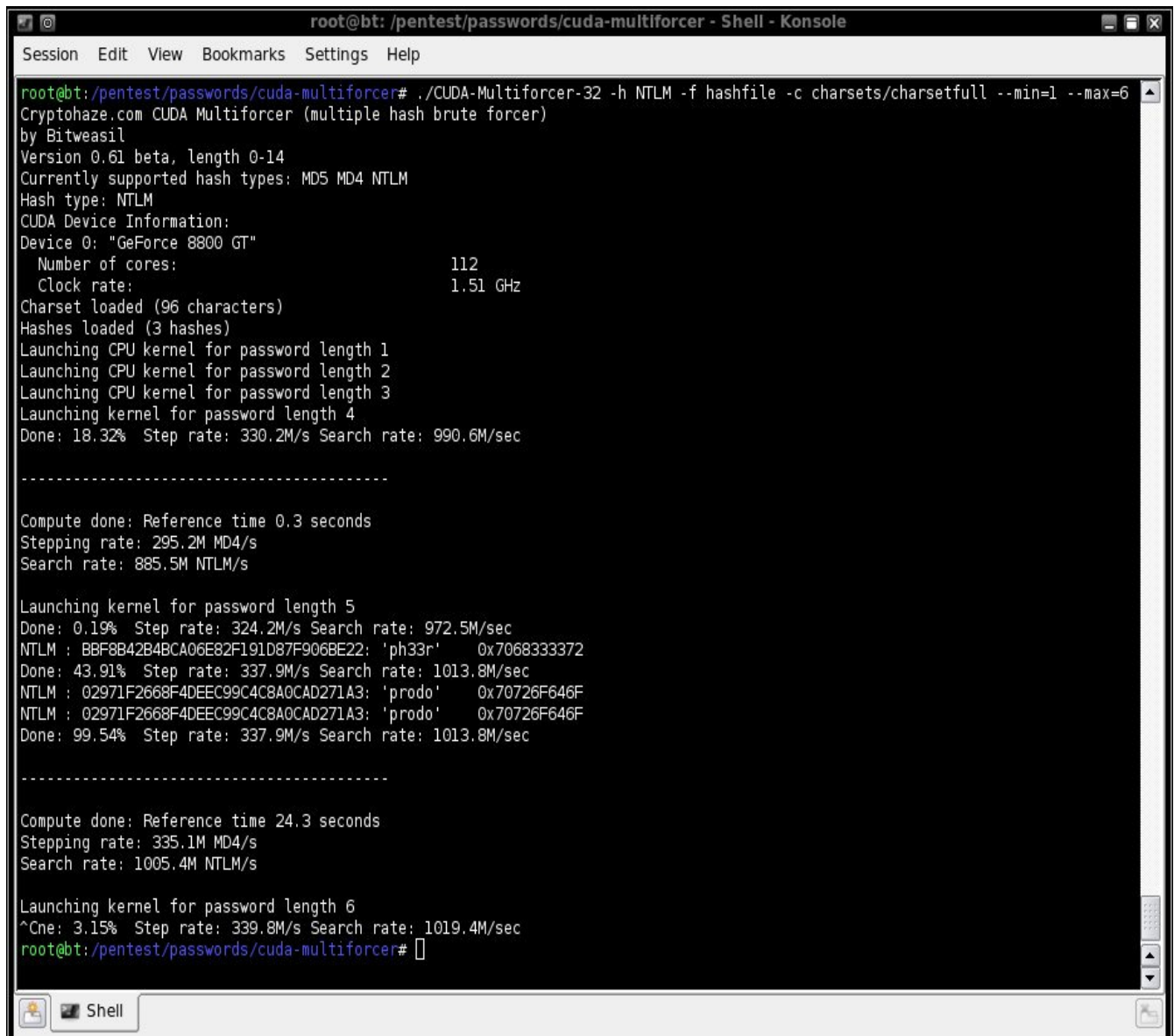


```
root@bt: /pentest/passwords/cuda-multiforcer - Shell - Konsole
Session Edit View Bookmarks Settings Help
root@bt:~# cd /pentest/passwords/cuda-multiforcer/
root@bt:/pentest/passwords/cuda-multiforcer# ls
Charsets  CUDA-Multiforcer  CUDA-Multiforcer-32  lib32  readme.txt  test_hash_files
root@bt:/pentest/passwords/cuda-multiforcer#
```

In order to start cracking you need to create a file in this directory which contains your hash or a list of hash's. The tool is capable of working on lots of hash's at one time. I have added two NTLM hash's to a file called hashfile. A charset file must also be selected.



The charset files are located in the charsets directory. For this example I have selected a “full” 96 char charset. Simply launch the tool with the proper parameters and be amazed at the sheer speed.



```
root@bt: /pentest/passwords/cuda-multiforcer - Shell - Konsole
Session Edit View Bookmarks Settings Help
root@bt: /pentest/passwords/cuda-multiforcer# ./CUDA-Multiforcer-32 -h NTLM -f hashfile -c charsets/charsetfull --min=1 --max=6
Cryptohaze.com CUDA Multiforcer (multiple hash brute forcer)
by Bitweasil
Version 0.61 beta, length 0-14
Currently supported hash types: MD5 MD4 NTLM
Hash type: NTLM
CUDA Device Information:
Device 0: "GeForce 8800 GT"
  Number of cores:          112
  Clock rate:              1.51 GHz
Charset loaded (96 characters)
Hashes loaded (3 hashes)
Launching CPU kernel for password length 1
Launching CPU kernel for password length 2
Launching CPU kernel for password length 3
Launching kernel for password length 4
Done: 18.32% Step rate: 330.2M/s Search rate: 990.6M/sec

-----

Compute done: Reference time 0.3 seconds
Stepping rate: 295.2M MD4/s
Search rate: 885.5M NTLM/s

Launching kernel for password length 5
Done: 0.19% Step rate: 324.2M/s Search rate: 972.5M/sec
NTLM : BBF8B42B4BCA06E82F191D87F906BE22: 'ph33r' 0x7068333372
Done: 43.91% Step rate: 337.9M/s Search rate: 1013.8M/sec
NTLM : 02971F2668F4DEEC99C4C8A0CAD271A3: 'prodo' 0x70726F646F
NTLM : 02971F2668F4DEEC99C4C8A0CAD271A3: 'prodo' 0x70726F646F
Done: 99.54% Step rate: 337.9M/s Search rate: 1013.8M/sec

-----

Compute done: Reference time 24.3 seconds
Stepping rate: 335.1M MD4/s
Search rate: 1005.4M NTLM/s

Launching kernel for password length 6
^Cne: 3.15% Step rate: 339.8M/s Search rate: 1019.4M/sec
root@bt: /pentest/passwords/cuda-multiforcer#
```

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.

# Pyrit

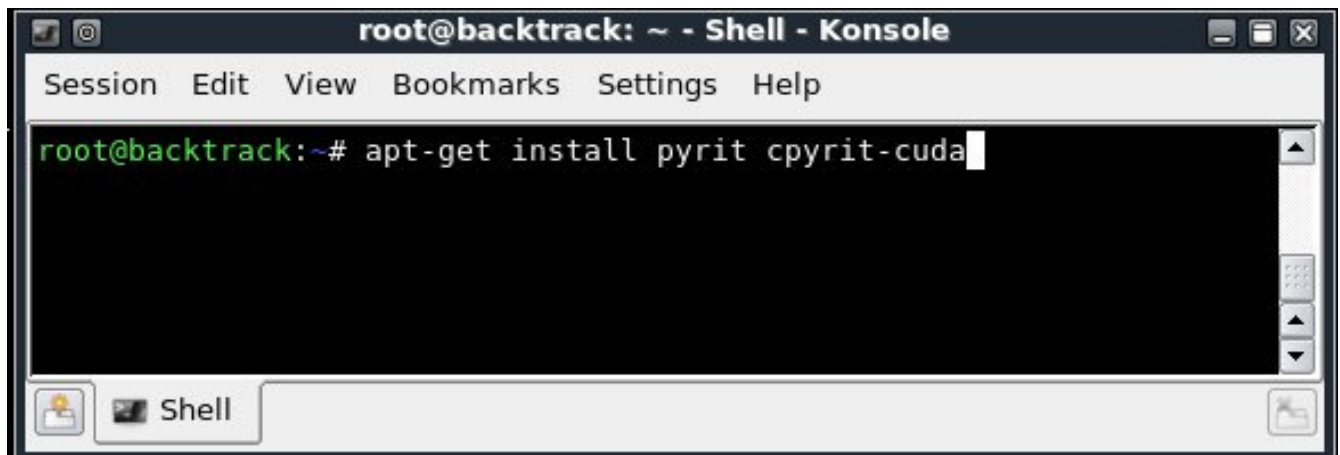
## What is pyrit?

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

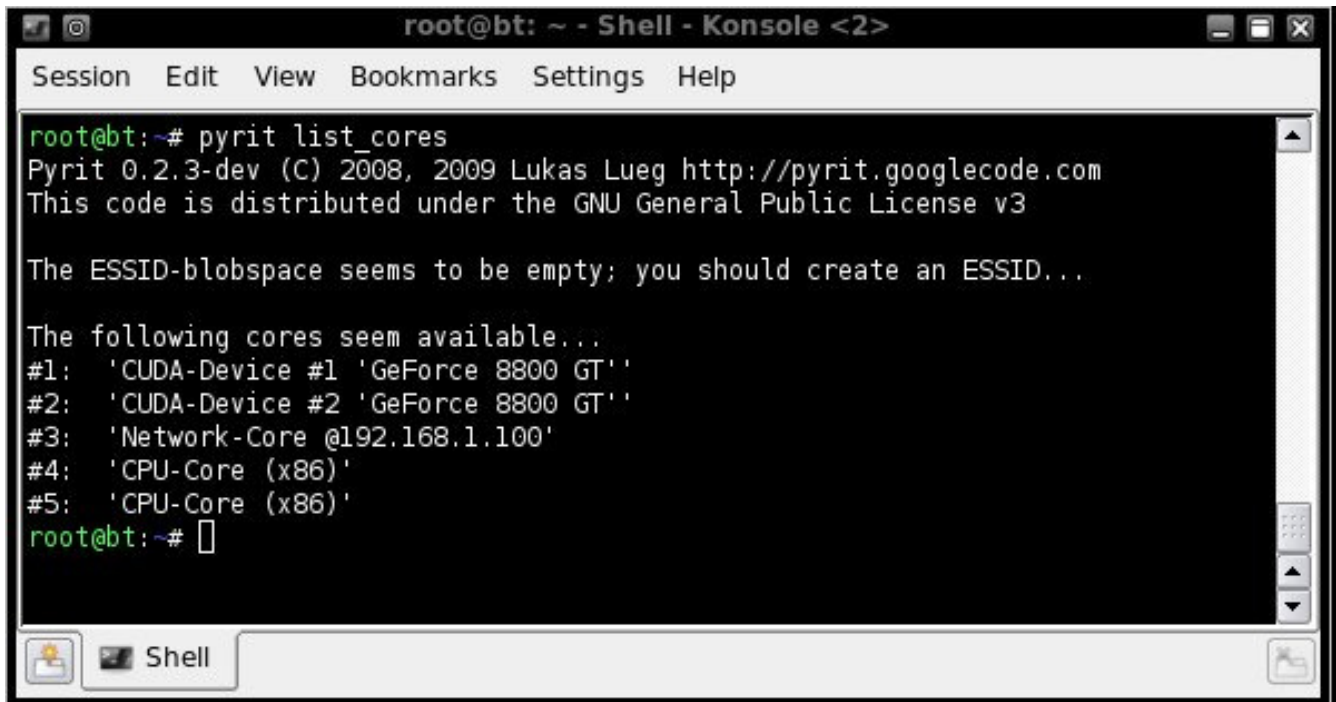
Pyrit is already included in the backtrack .iso however the cuda core is not. In order to make sure we have the most recent version of both we will need to apt-get them.



```
root@backtrack: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
root@backtrack:~# apt-get install pyrit cpyrit-cuda
```

## Making sure Pyrit is working:

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

A terminal window titled 'root@bt: ~ - Shell - Konsole <2>' showing the output of the 'pyrit list\_cores' command. The output lists five available cores: two GeForce 8800 GT GPUs, a network core at 192.168.1.100, and two x86 CPU cores. The terminal also shows the Pyrit version (0.2.3-dev) and license information.

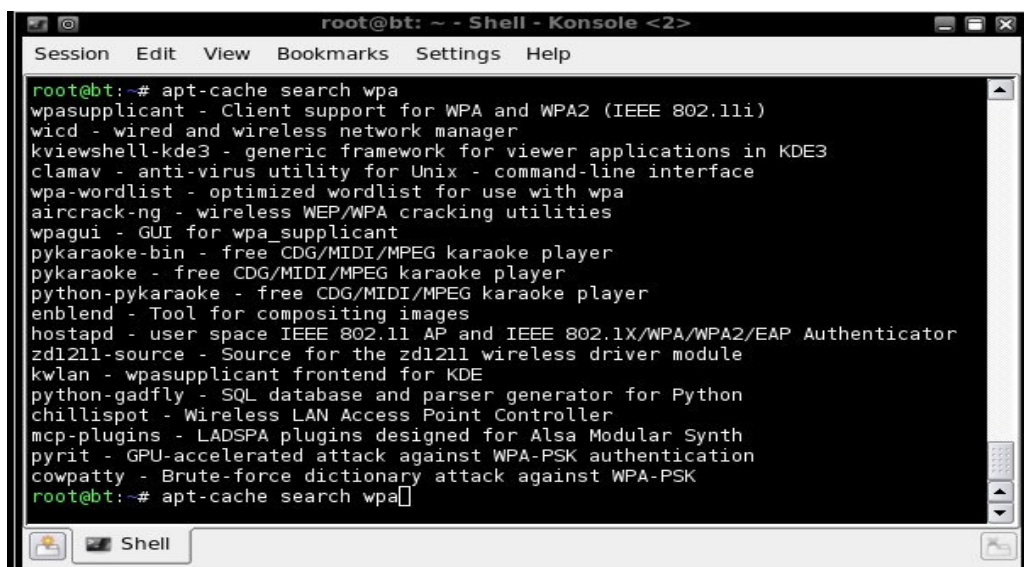
```
root@bt:~# pyrit list_cores
Pyrit 0.2.3-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

The ESSID-blobspace seems to be empty; you should create an ESSID...

The following cores seem available...
#1: 'CUDA-Device #1 'GeForce 8800 GT''
#2: 'CUDA-Device #2 'GeForce 8800 GT''
#3: 'Network-Core @192.168.1.100'
#4: 'CPU-Core (x86)'
#5: 'CPU-Core (x86)'
root@bt:~#
```

Don't worry about #3 networkcore in this picture yet. There will be more on that feature later on in the document. As you can see we have 2 8800 GT cards and we are using two of the four CPU cores as well. So now let's try a benchmark to make sure the Nvidia CUDA core gets loaded and is working properly. In order to do that simply run `root@bt~# pyrit benchmark`.

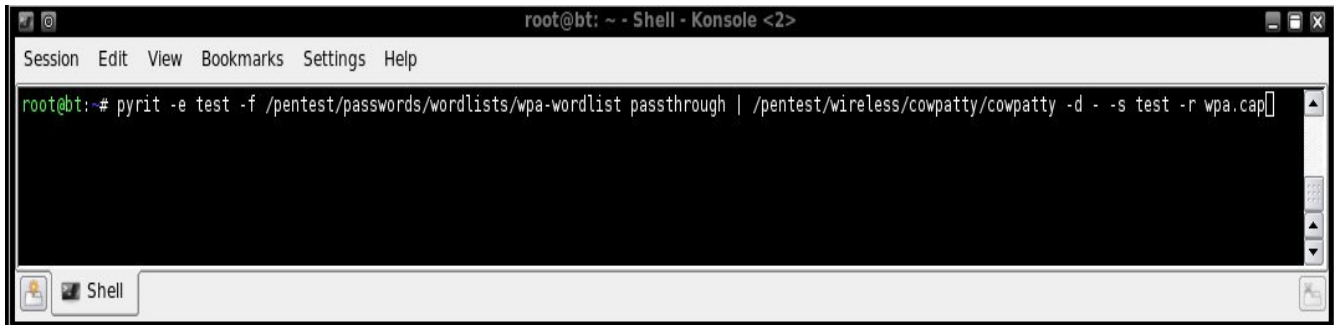
We have created an optimized wpa password list for users to get started with. This was too big for the .iso however we can easily grab it from the repo with apt-get

A terminal window titled 'root@bt: ~ - Shell - Konsole <2>' showing the output of the 'apt-cache search wpa' command. The output lists various packages related to WPA, including wpa\_supplicant, wicd, kviewshell-kde3, clamav, wpa-wordlist, aircrack-ng, wpa\_gui, pykaraoke-bin, pykaraoke, python-pykaraoke, enblend, hostapd, zd1211-source, kwlan, python-gadfly, chillispot, mcp-plugins, pyrit, and cowpatty.

```
root@bt:~# apt-cache search wpa
wpa_supplicant - Client support for WPA and WPA2 (IEEE 802.11i)
wicd - wired and wireless network manager
kviewshell-kde3 - generic framework for viewer applications in KDE3
clamav - anti-virus utility for Unix - command-line interface
wpa-wordlist - optimized wordlist for use with wpa
aircrack-ng - wireless WEP/WPA cracking utilities
wpa_gui - GUI for wpa_supplicant
pykaraoke-bin - free CDG/MIDI/MPEG karaoke player
pykaraoke - free CDG/MIDI/MPEG karaoke player
python-pykaraoke - free CDG/MIDI/MPEG karaoke player
enblend - Tool for compositing images
hostapd - user space IEEE 802.11 AP and IEEE 802.1X/WPA/WPA2/EAP Authenticator
zd1211-source - Source for the zd1211 wireless driver module
kwlan - wpa_supplicant frontend for KDE
python-gadfly - SQL database and parser generator for Python
chillispot - Wireless LAN Access Point Controller
mcp-plugins - LADSPA plugins designed for Alsa Modular Synth
pyrit - GPU-accelerated attack against WPA-PSK authentication
cowpatty - Brute-force dictionary attack against WPA-PSK
root@bt:~# apt-cache search wpa
```

## 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:

A screenshot of a terminal window titled "root@bt: ~ - Shell - Konsole <2>". The terminal shows a command being entered: `root@bt:~# pyrit -e test -f /pentest/passwords/wordlists/wpa-wordlist passthrough | /pentest/wireless/cowpatty/cowpatty -d - -s test -r wpa.cap`. The terminal interface includes a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The command prompt is `root@bt:~#`. The terminal output is currently blank.

In case the text on the picture is too small the command looks like this:

```
root@bt~# pyrit -e (essid) -f (path/to/wordlist) passthrough |  
/pentest/wireless/cowpatty -d - -s (essid) -r (path/to/capfile)
```

Here is an example from one of my older tests:

```
r00t@infected ~ $ pyrit -e NETGEAR -f final-wordlist.txt passthrough | cowpatty  
-d - -r wpa-01.cap -s NETGEAR
```

```
cowpatty 4.3 - WPA-PSK dictionary attack. <jwright@hasborg.com>
```

```
Collected all necessary data to mount crack against WPA/PSK passphrase.
```

```
Starting dictionary attack. Please be patient.
```

```
Using STDIN for hashfile contents.
```

```
key no. 10000: 123456pnb
```

```
key no. 20000: 1Tokenof
```

```
...
```

```
key no. 970000: waegbarer
```

```
key no. 980000: withstood
```

```
key no. 990000: yc26njw4xd
```



fread: Success

Unable to identify the PSK from the dictionary file. Try expanding your passphrase list, and double-check the SSID. Sorry it didn't work out.

990100 passphrases tested in 104.51 seconds: 9473.97 passphrases/second

Although the key was not recovered you can see how it works.

### **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:

```
root@bt ~ $ /pentest/passwords/crunch/crunch 8 8 123456 | pyrit -e NETGEAR -f -  
passthrough | cowpatty -d - -r wpa-01.cap -s NETGEAR
```

```
cowpatty 4.3 - WPA-PSK dictionary attack. <jwright@hasborg.com>
```

Collected all necessary data to mount crack against WPA/PSK passphrase.

Starting dictionary attack. Please be patient.

Using STDIN for hashfile contents.

key no. 10000: 11131143

key no. 20000: 11335211

key no. 30000: 11453262

...

key no. 1660000: 66342333

key no. 1670000: 66512215

Unable to identify the PSK from the dictionary file. Try expanding your passphrase list, and double-check the SSID. Sorry it didn't work out.

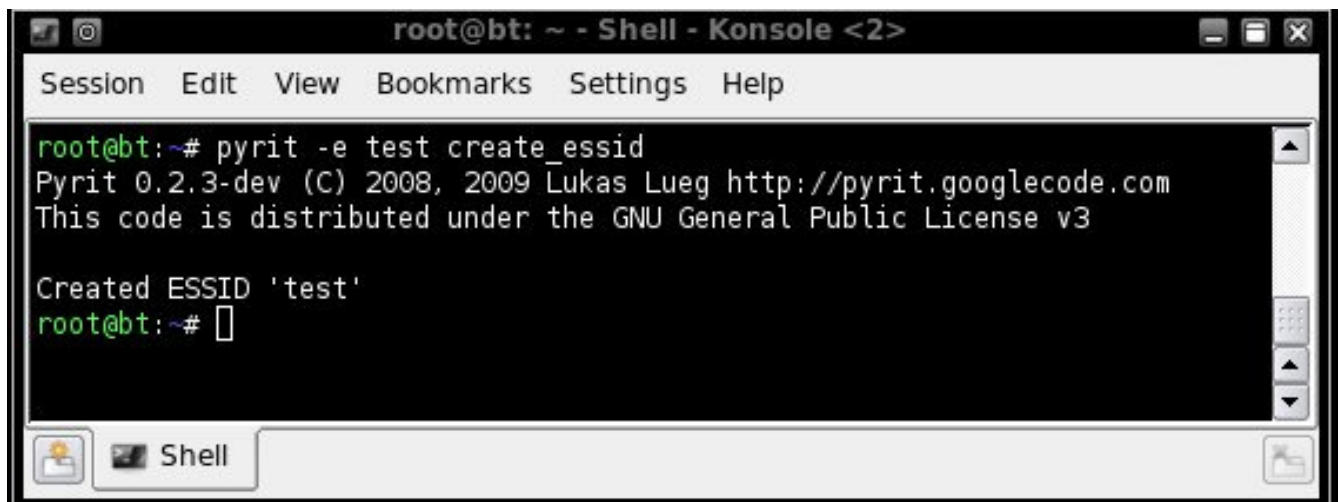
1670168 passphrases tested in 171.54 seconds: 9736.04 passphrases/second

As you can see the attack did not work however it is possible to create bruteforce lists on the fly and pipe them straight into Pyrit. This type of attack may become more useful in the future when WPA is further exploited or as WPA attacks become better.

## 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.

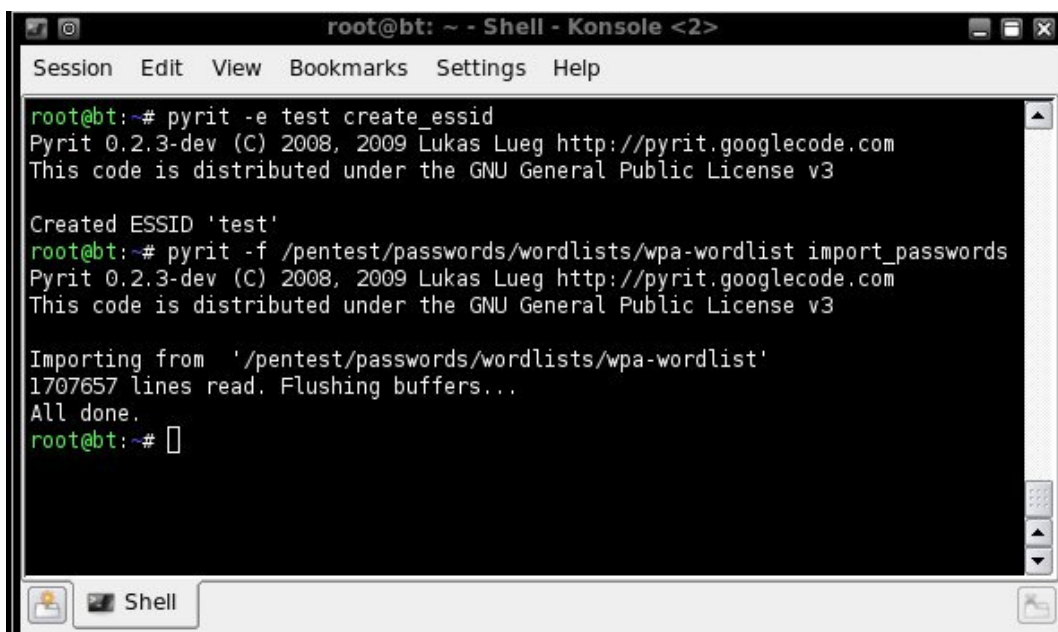
First we add our essid:



```
root@bt: ~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
root@bt:~# pyrit -e test create_essid
Pyrit 0.2.3-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Created ESSID 'test'
root@bt:~#
```

Next we import some passwords:

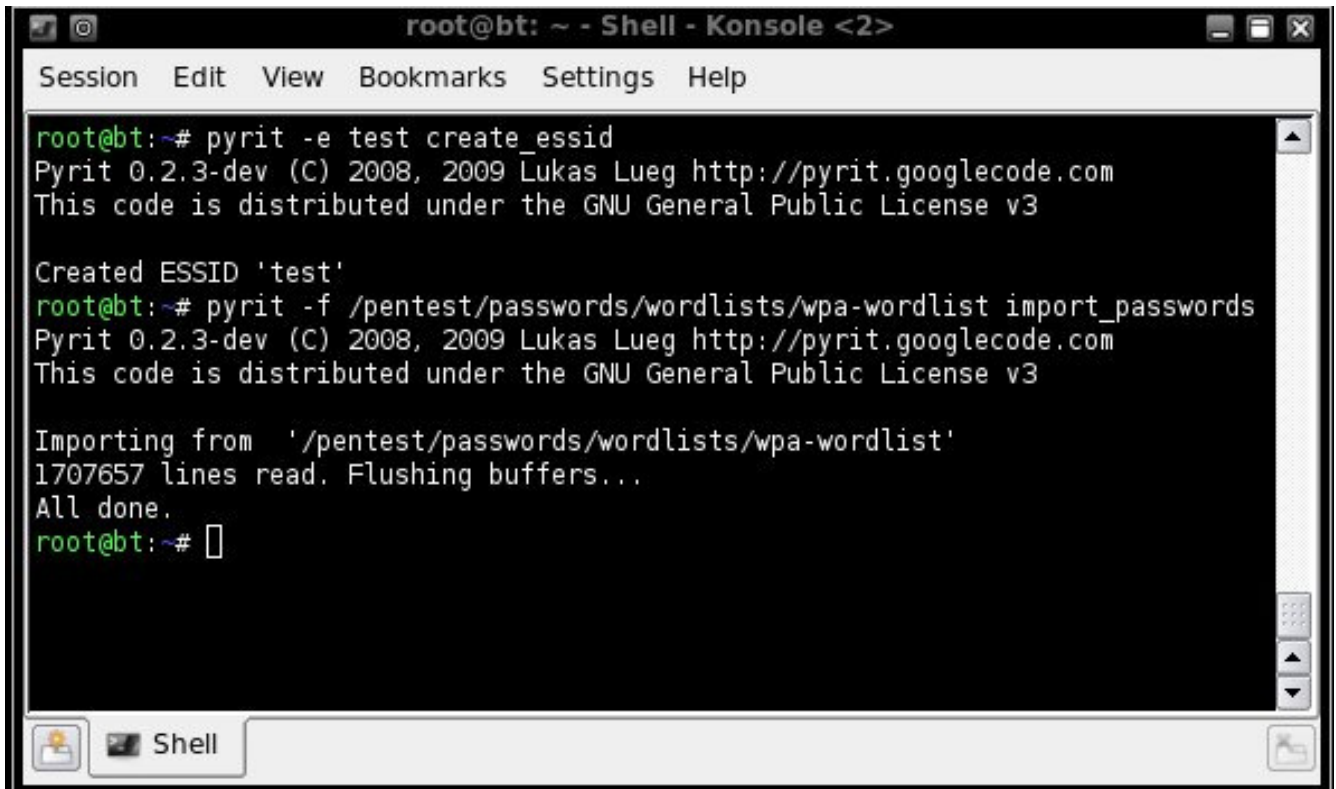


```
root@bt: ~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
root@bt:~# pyrit -e test create_essid
Pyrit 0.2.3-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Created ESSID 'test'
root@bt:~# pyrit -f /pentest/passwords/wordlists/wpa-wordlist import_passwords
Pyrit 0.2.3-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Importing from '/pentest/passwords/wordlists/wpa-wordlist'
1707657 lines read. Flushing buffers...
All done.
root@bt:~#
```

Next we start the batch processing:



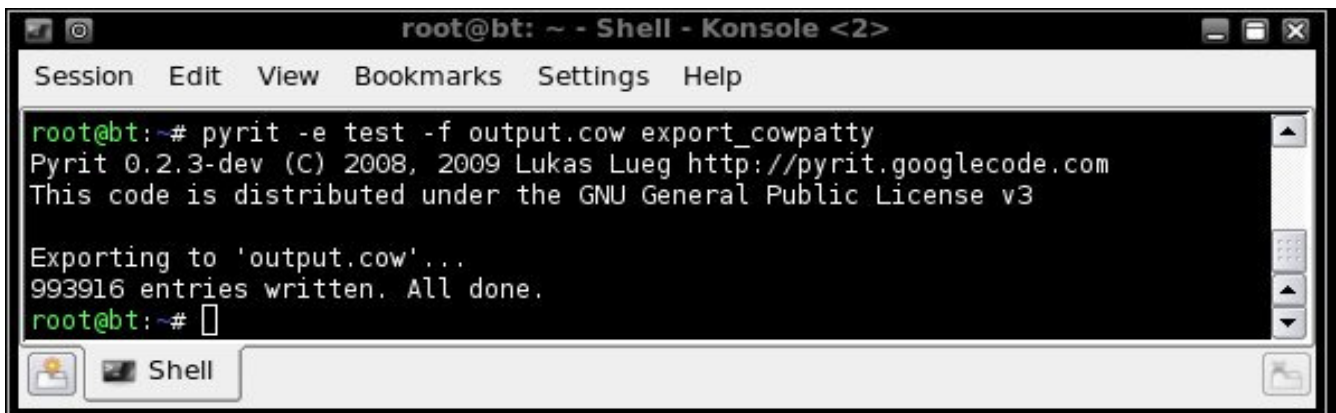
```
root@bt: ~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
root@bt:~# pyrit -e test create_essid
Pyrit 0.2.3-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Created ESSID 'test'
root@bt:~# pyrit -f /pentest/passwords/wordlists/wpa-wordlist import_passwords
Pyrit 0.2.3-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Importing from '/pentest/passwords/wordlists/wpa-wordlist'
1707657 lines read. Flushing buffers...
All done.
root@bt:~#
```

At this point we have a choice. We can either export to a cowpatty format or a aircrack-ng format. The cowpatty way is quite a bit faster due to some sqlite limitations however I will showcase both methods.

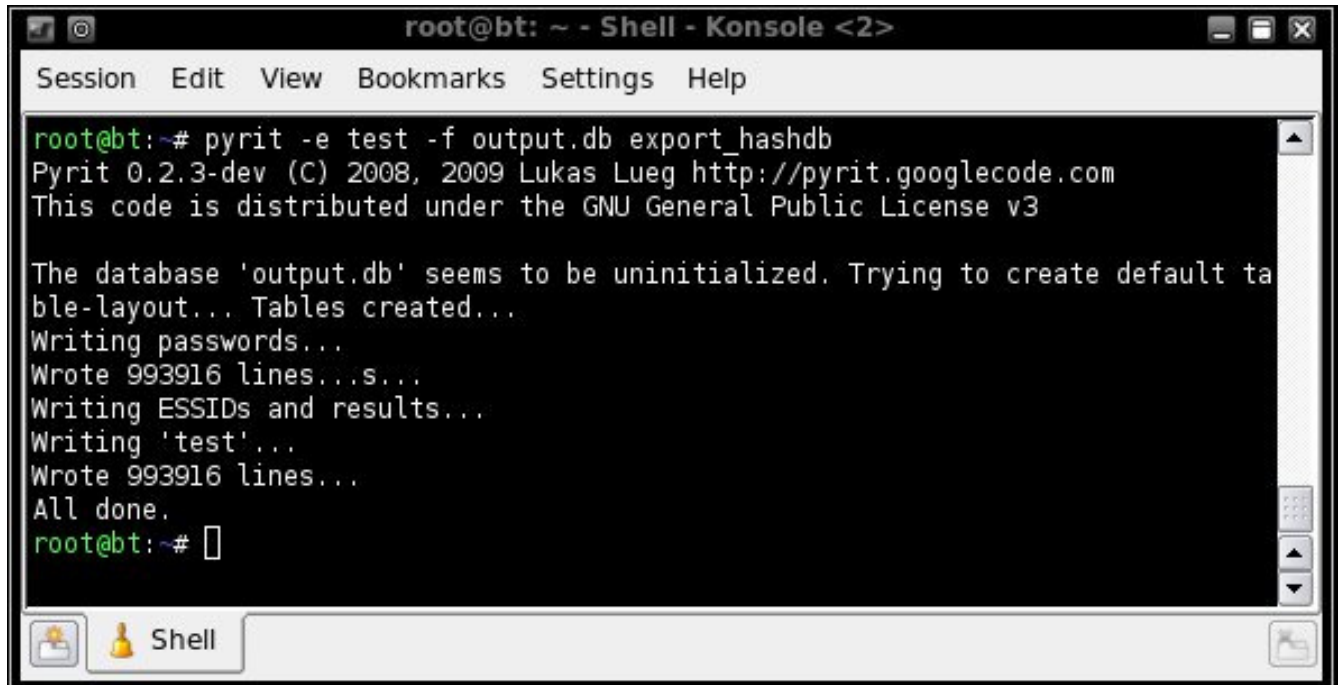
Here is the syntax for cowpatty:



```
root@bt: ~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
root@bt:~# pyrit -e test -f output.cow export_cowpatty
Pyrit 0.2.3-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Exporting to 'output.cow'...
993916 entries written. All done.
root@bt:~#
```

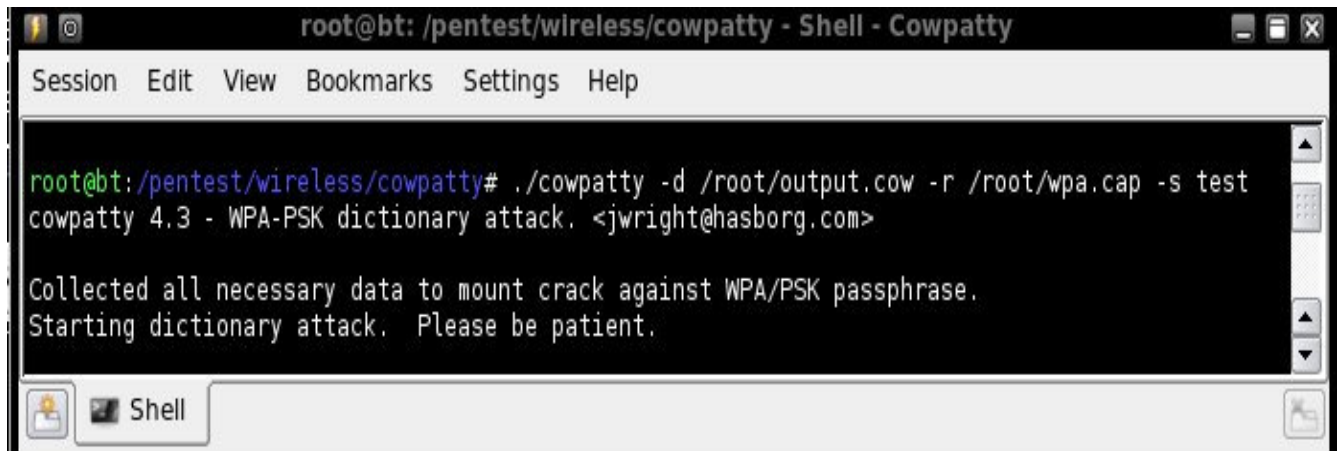
And here is the syntax for aircrack-ng



```
root@bt: ~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
root@bt:~# pyrit -e test -f output.db export_hashdb
Pyrit 0.2.3-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

The database 'output.db' seems to be uninitialized. Trying to create default ta
ble-layout... Tables created...
Writing passwords...
Wrote 993916 lines...s...
Writing ESSIDs and results...
Writing 'test'...
Wrote 993916 lines...
All done.
root@bt:~#
```

Once we have our table saved we can send it to the cracker:



```
root@bt: /pentest/wireless/cowpatty - Shell - Cowpatty
Session Edit View Bookmarks Settings Help
root@bt:/pentest/wireless/cowpatty# ./cowpatty -d /root/output.cow -r /root/wpa.cap -s test
cowpatty 4.3 - WPA-PSK dictionary attack. <jwright@hasborg.com>

Collected all necessary data to mount crack against WPA/PSK passphrase.
Starting dictionary attack. Please be patient.
```

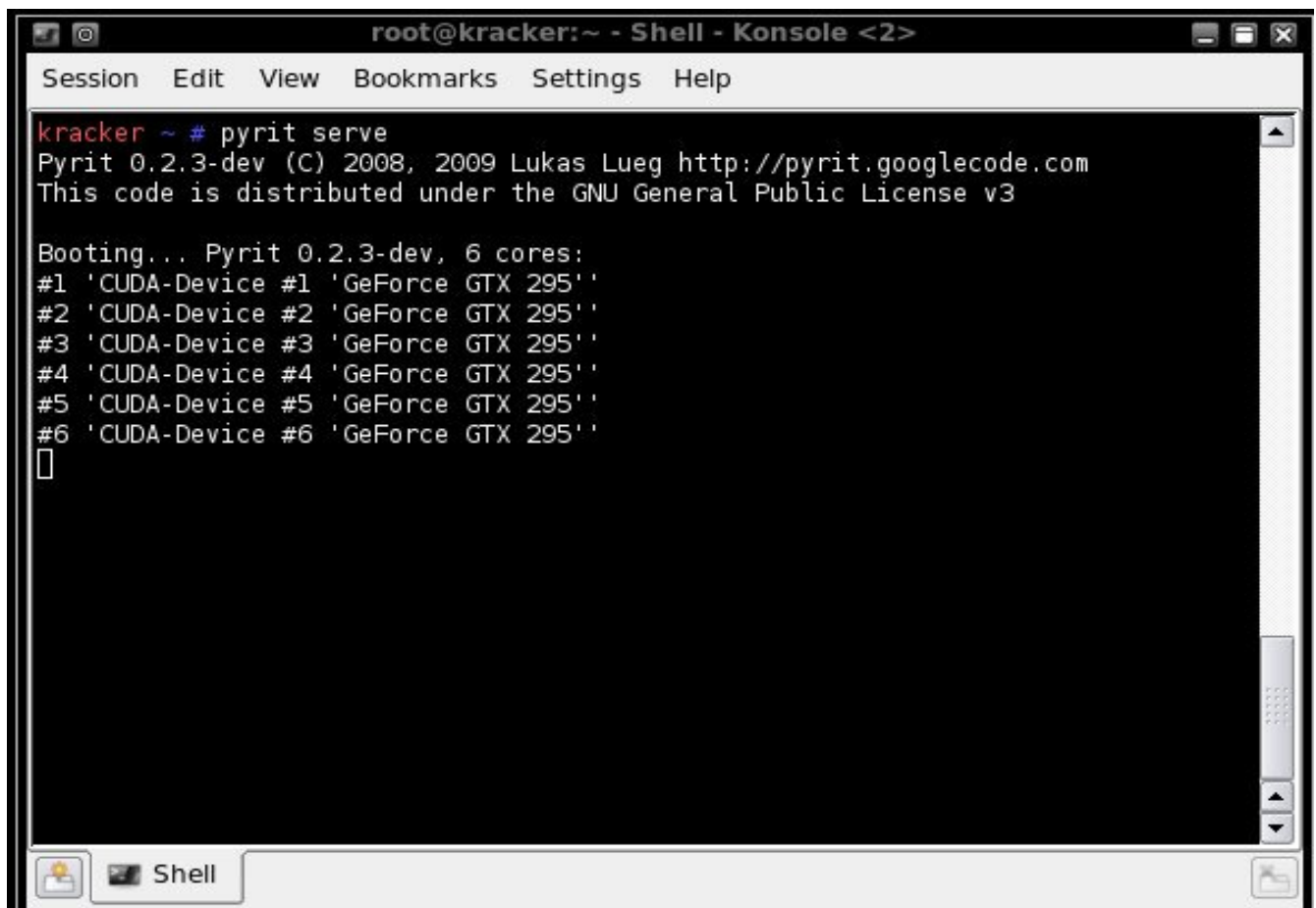
## Server / Client Mode:

Pyrit now includes support for clustering multiple machines over the local network. This feature was often requested as it allows to use hardware much more effectively.

Pyrit has a new command `'serve'` that starts a server on the current host. A server listens for connections on port 19935 (setup those firewalls...) and can use the local hardware to compute for other clients. Clients can use multiple servers and each server can support multiple clients simultaneously. This is not a distributed database! The clients transfer their workunits to the servers and the servers compute the results and send them back. Bandwidth is a problem: 10.000 PMKs/s require about 30kb/s from the client to the server and about 300kb/s from the server to the client. This makes internet-connections too slow for most of us...

On the servers, the machines with the fast hardware:

Start Pyrit with `'pyrit serve'`. The server uses all available (local!) hardware just like a pyrit-session would do... Kill it with `ctr+c` when you are done. Beware that clients which are still waiting for results from that server will die...

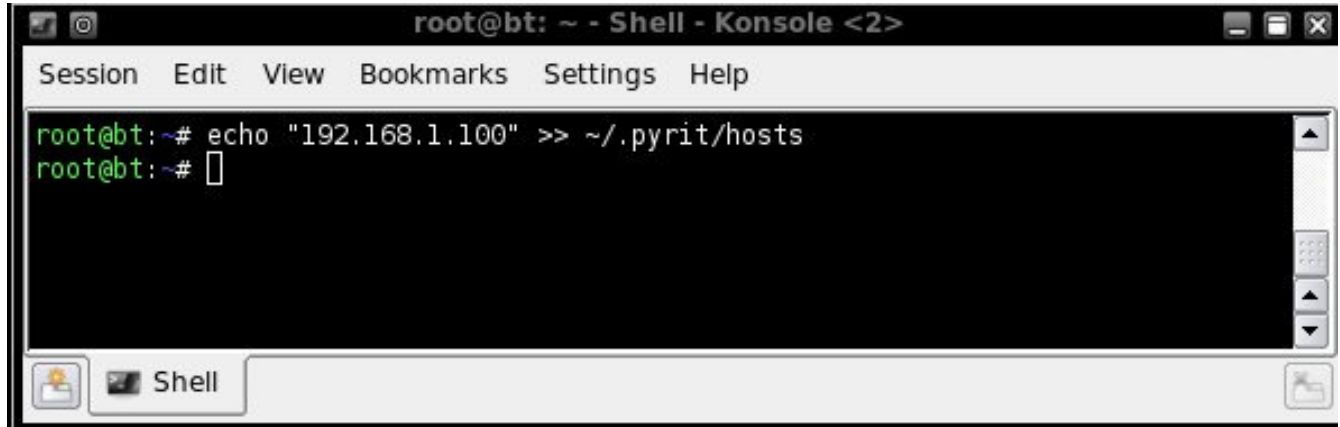


```
root@kracker:~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
kracker ~ # pyrit serve
Pyrit 0.2.3-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

Booting... Pyrit 0.2.3-dev, 6 cores:
#1 'CUDA-Device #1 'GeForce GTX 295' '
#2 'CUDA-Device #2 'GeForce GTX 295' '
#3 'CUDA-Device #3 'GeForce GTX 295' '
#4 'CUDA-Device #4 'GeForce GTX 295' '
#5 'CUDA-Device #5 'GeForce GTX 295' '
#6 'CUDA-Device #6 'GeForce GTX 295' '
█
```

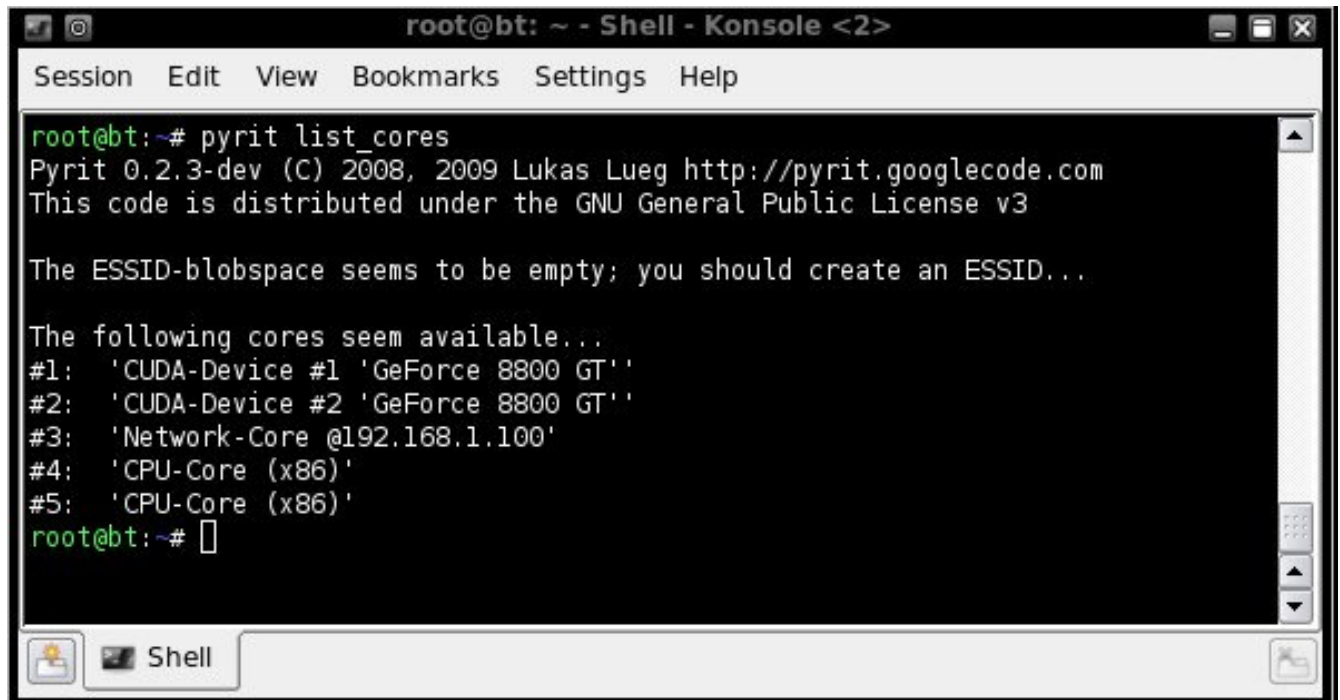
On the client, the machine that hosts the database:

- Edit `~/pyrit/hosts`. Add one IP/hostname per line for every server you have.
- Check if the server is reachable by opening `http://[Server-IP]:19935/` in your web-browser.
- Run `pyrit list_cores`. It should list the new Network-Cores.
- The servers do not have to be online when you start Pyrit. Inactive servers get ignored...
- Use Pyrit like you would normally do. All functions (benchmark/batchprocess/passthrough) use the servers transparently and without further interaction.



```
root@bt: ~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
root@bt:~# echo "192.168.1.100" >> ~/.pyrit/hosts
root@bt:~#
```

Now run “list\_cores” and see if your network is available:



```
root@bt: ~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
root@bt:~# pyrit list_cores
Pyrit 0.2.3-dev (C) 2008, 2009 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3

The ESSID-blobspace seems to be empty; you should create an ESSID...

The following cores seem available...
#1: 'CUDA-Device #1 'GeForce 8800 GT''
#2: 'CUDA-Device #2 'GeForce 8800 GT''
#3: 'Network-Core @192.168.1.100'
#4: 'CPU-Core (x86)'
#5: 'CPU-Core (x86)'
root@bt:~#
```

I think the implementation is already quite reasonable; however you should expect some rough edges like unhandled exceptions/crashes caused by network timeouts and such...

\* Text was taken from <http://pyrit.wordpress.com/> which is the official pyrit blog

## 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.

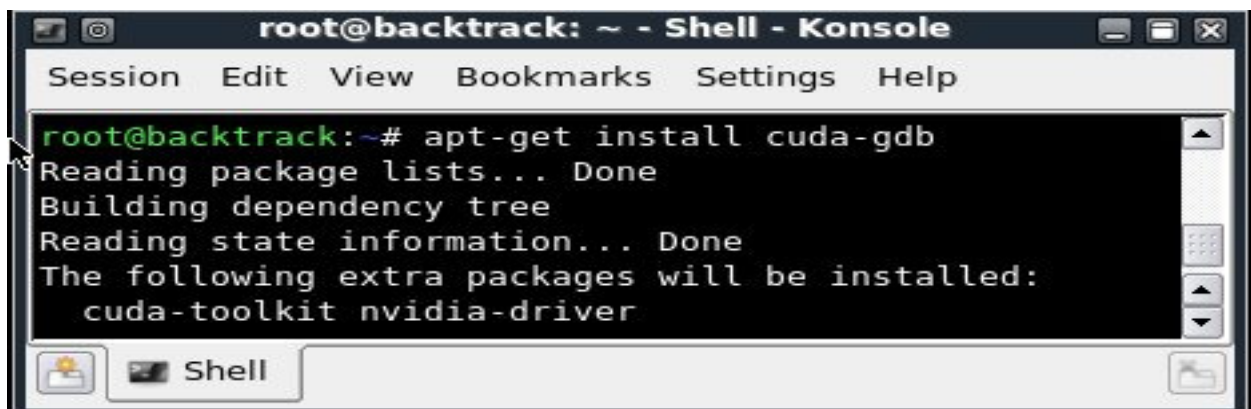
Special thanks to Zermelo and fnord0 for testing and posting the results of this tool.



## Cuda Debugger:

CUDA-GDB is a ported version of GDB: The GNU Debugger, version 6.6. The goal of its design is to present the user with an all-in-one debugging environment that is capable of debugging native host code as well as CUDA code. Therefore, it is an extension to the standard i386 port that is provided in the GDB release. As a result, standard debugging features are inherently supported for host code, and additional features have been provided to support debugging CUDA code. CUDAGDB is supported on 32-bit Linux

Installing the debugger is easy:

A screenshot of a terminal window titled "root@backtrack: ~ - Shell - Konsole". The window has a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The terminal output shows the command "apt-get install cuda-gdb" being executed. The output includes "Reading package lists... Done", "Building dependency tree", "Reading state information... Done", and "The following extra packages will be installed: cuda-toolkit nvidia-driver". The terminal prompt is "root@backtrack:~#".

```
root@backtrack:~# apt-get install cuda-gdb
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  cuda-toolkit nvidia-driver
```

NVCC, the NVIDIA CUDA compiler driver, provides a mechanism for generating debugging information necessary for CUDA-GDB to work properly. The “-g -G” option pair must be passed to the CUDA compiler when compiling an application in order to debug with the CUDA debugger (cuda-gdb). For example:

```
nvcc -g -G foo.cu -o foo
```

Start the CUDA debugger by entering the following command at a shell

prompt:

```
bt~# cuda-gdb (program name)
```

\* The complete .pdf on using the CUDA debugger can be found here

[http://developer.download.nvidia.com/compute/cuda/2\\_1/cudagdb/CUDA\\_GDB\\_User\\_Manual.pdf](http://developer.download.nvidia.com/compute/cuda/2_1/cudagdb/CUDA_GDB_User_Manual.pdf)



## 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://forums.remote-exploit.org/>
- <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 Remote-exploit-dev team, thebaron , ebfe, Synok, Gomet 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 thank anyone else who I have forgotten because I have ADD.

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

Thanks

<3

Pureh@te

25