BackTrack 4 CUDA Guide

Written by

Pureh@te

Table of Contents

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

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:

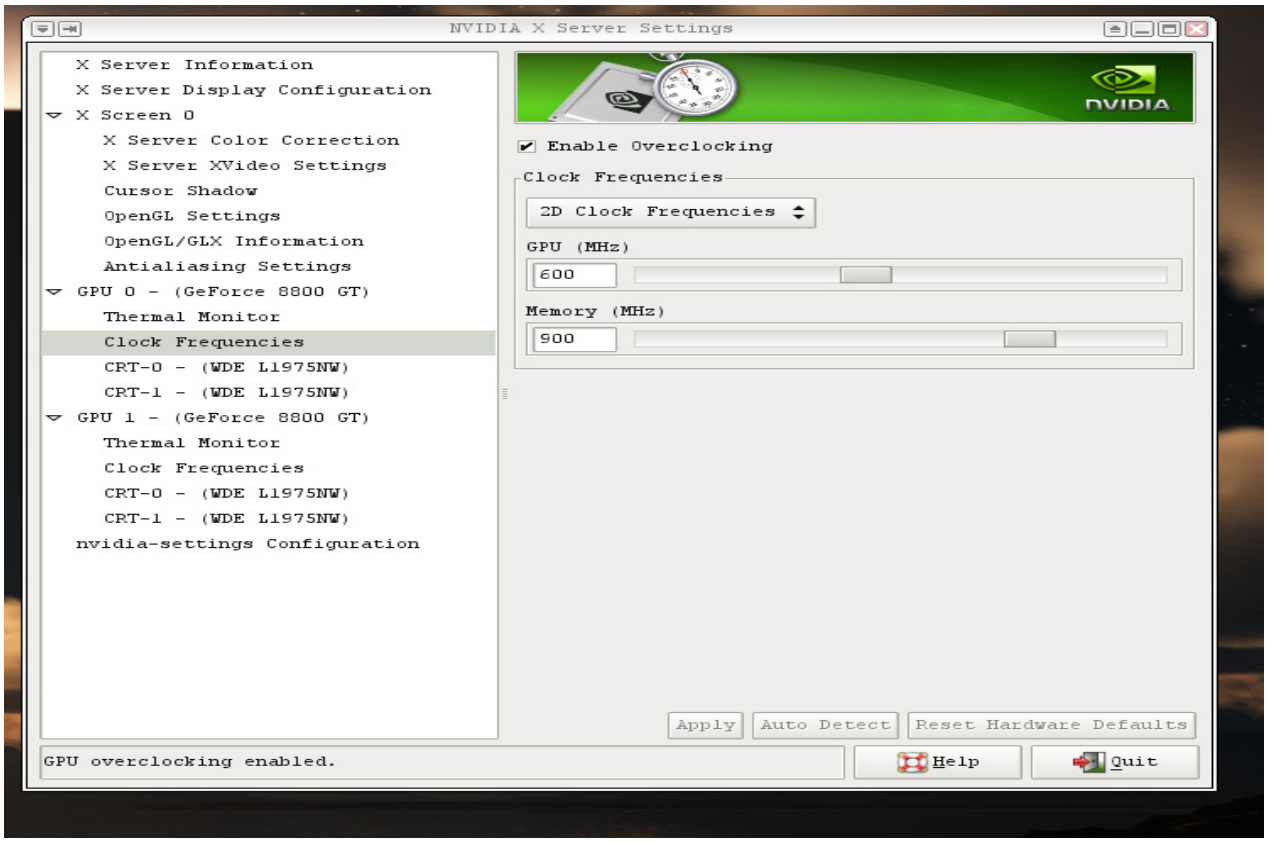
There are two ways to overclock your video card in Linux. The first way is to use the nvidia-settings tool which comes with the nvidia-driver. In order to do this you need to edit your xorg.conf in order to unlock the option.

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

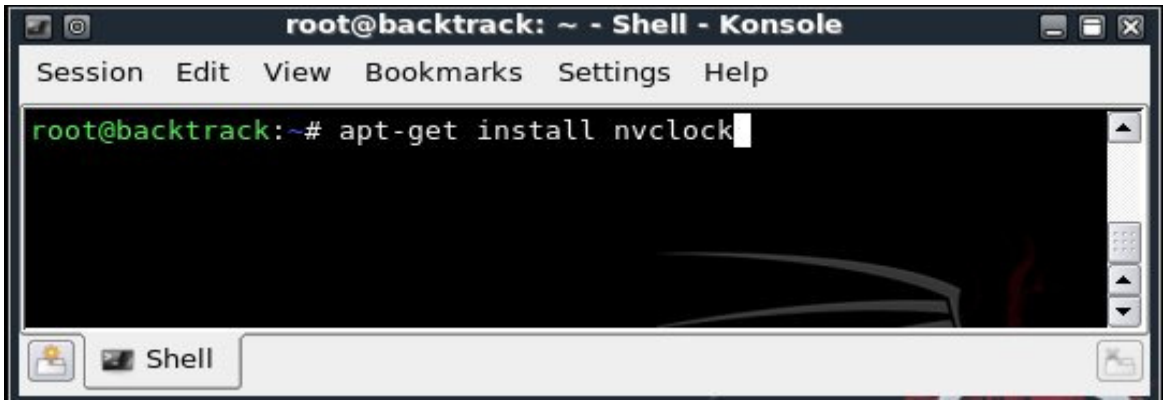
and find the section that looks like this:

```
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 you card in linux is to use the nvclock command line utility.



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

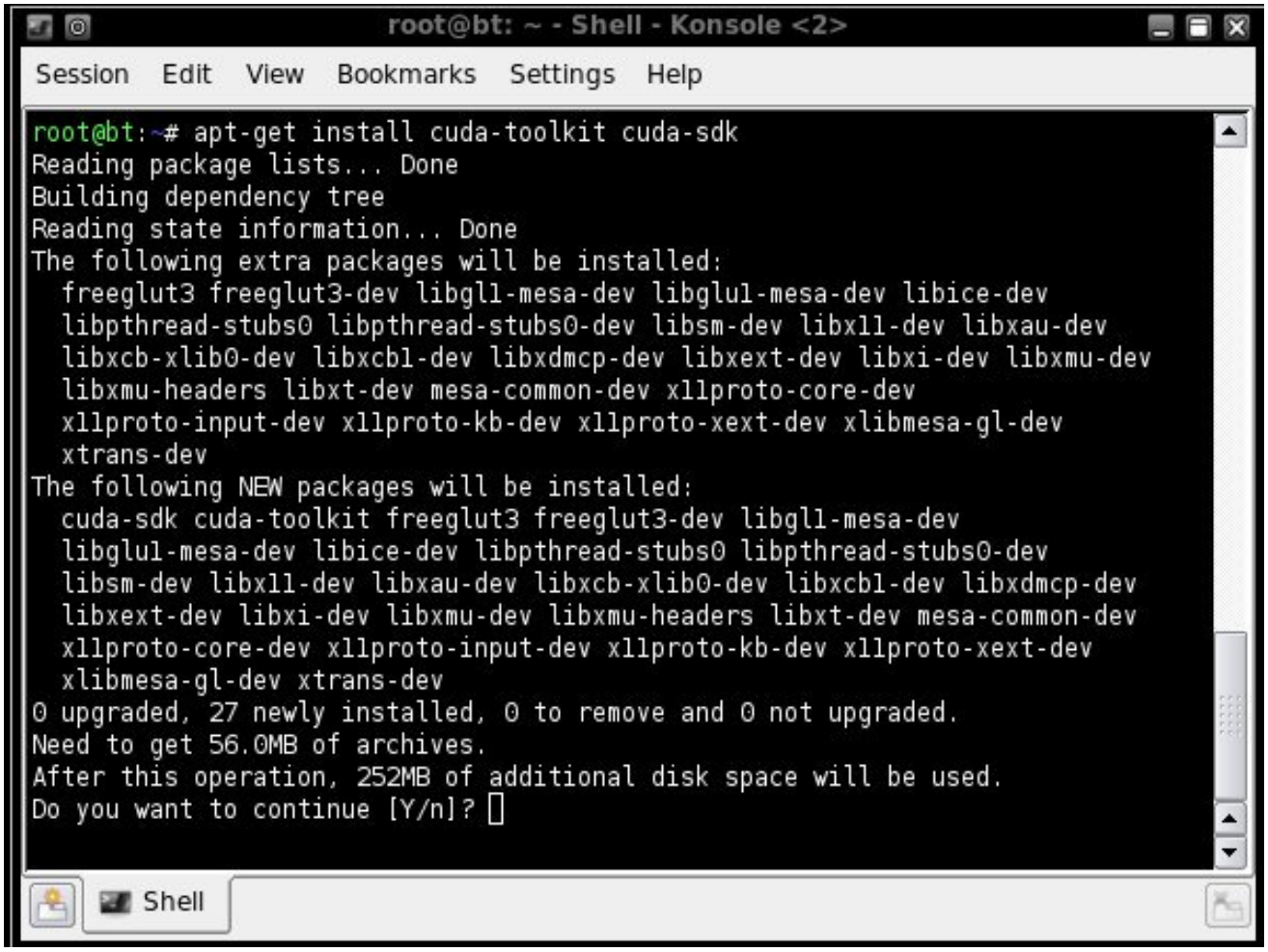
```
root@bt ~ $ nvclock
NVClock v0.7

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                      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 :

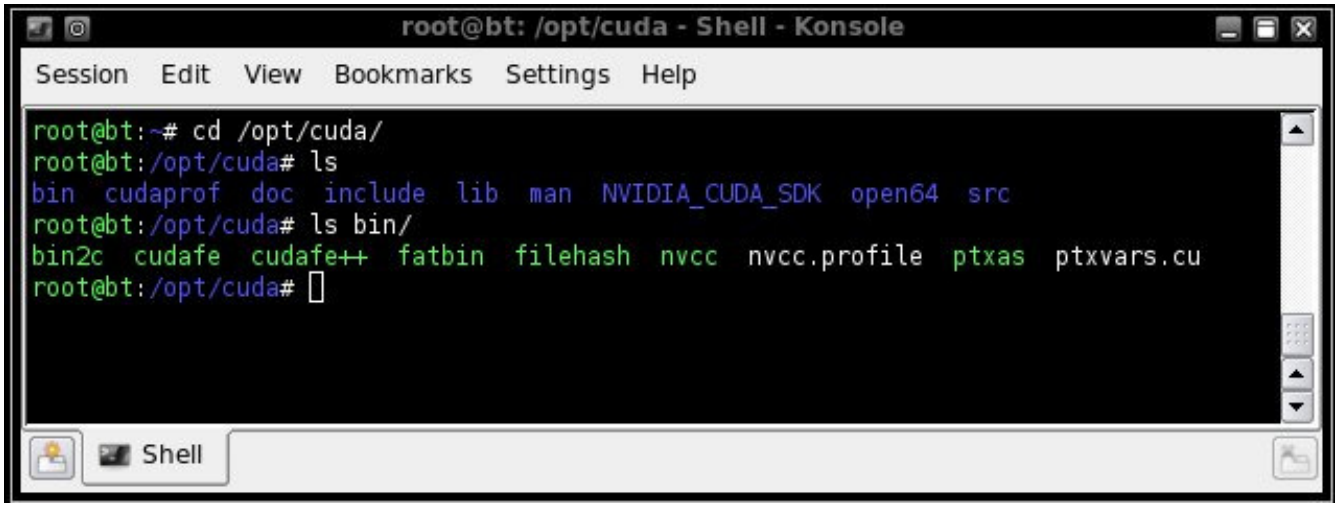
Now that we have our driver installed and the clock settings to our liking, its time to get our CUDA development environment set up. This is not necessary if you are only interested in running a tool such as Pyrit however if you want to build any CUDA applications you will need this environment.

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.

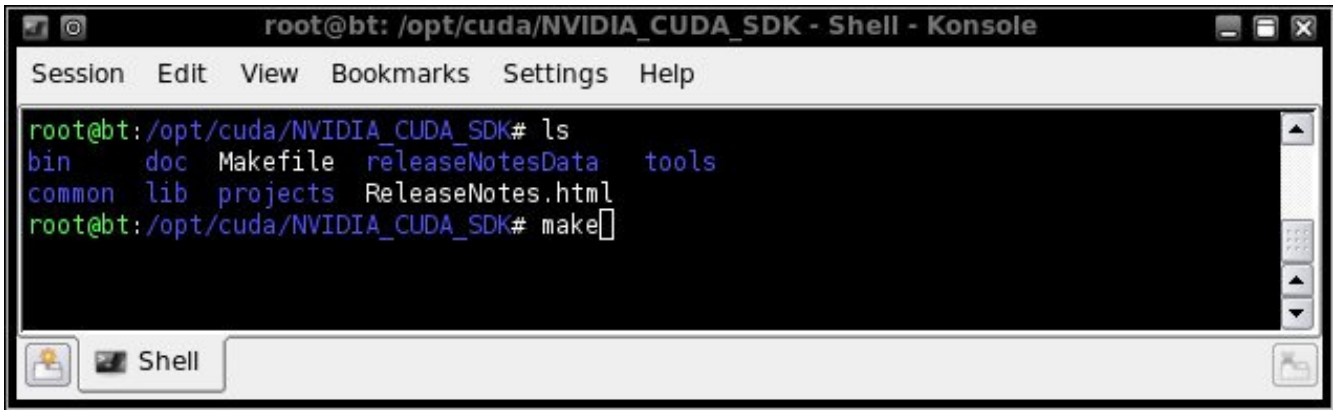


Once this is finished installing you will have every thing you need to build or program your own CUDA applications. I will provide some helpful programing links at the end of this document because how to program in CUDA is beyond the scope of this document. I will show some basic navigation.

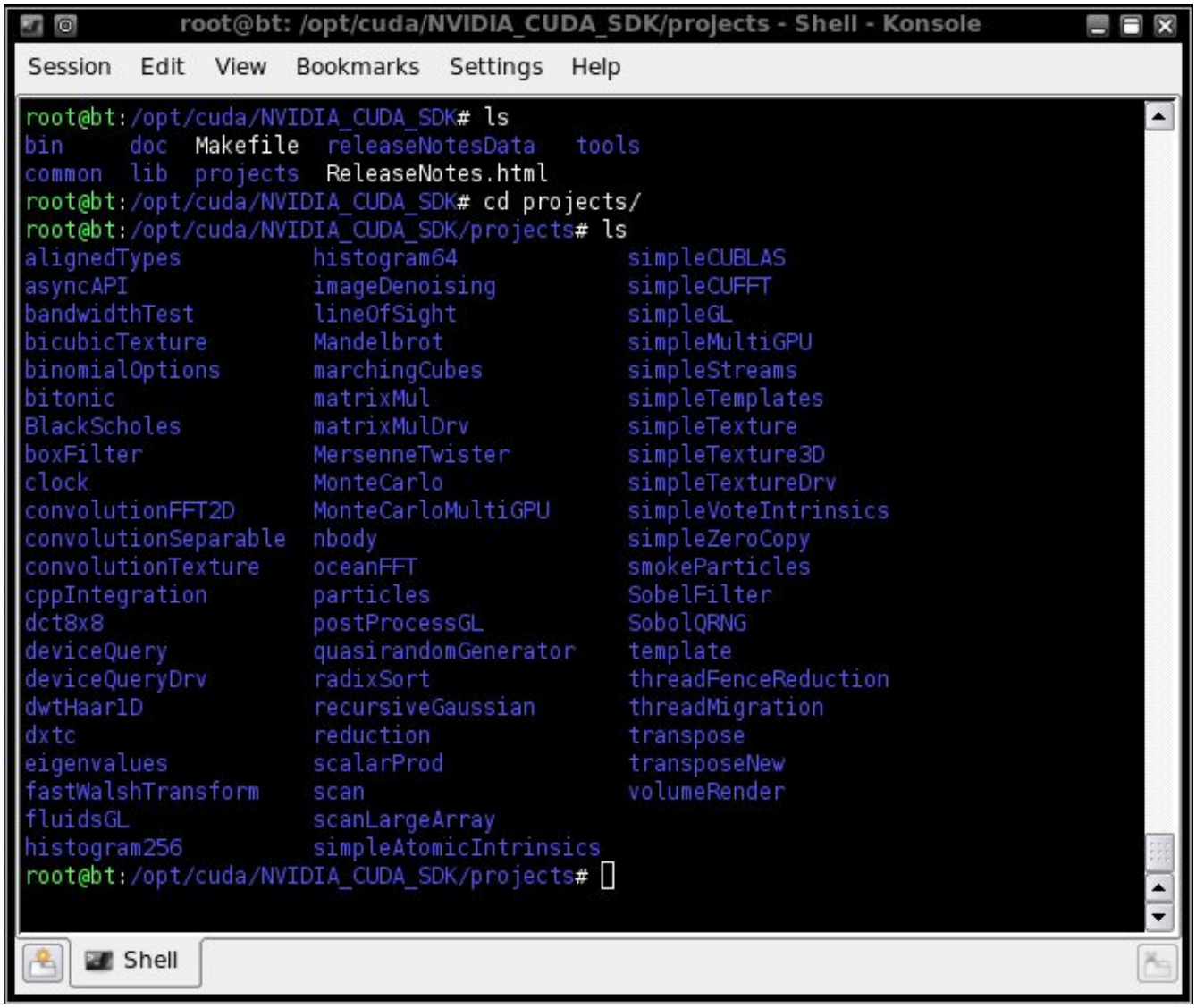
The initial environment is in /opt/cuda and in /opt/cuda/bin are the build tools for CUDA. The binary nvcc is the Nvidia compiler which is used to build applications.



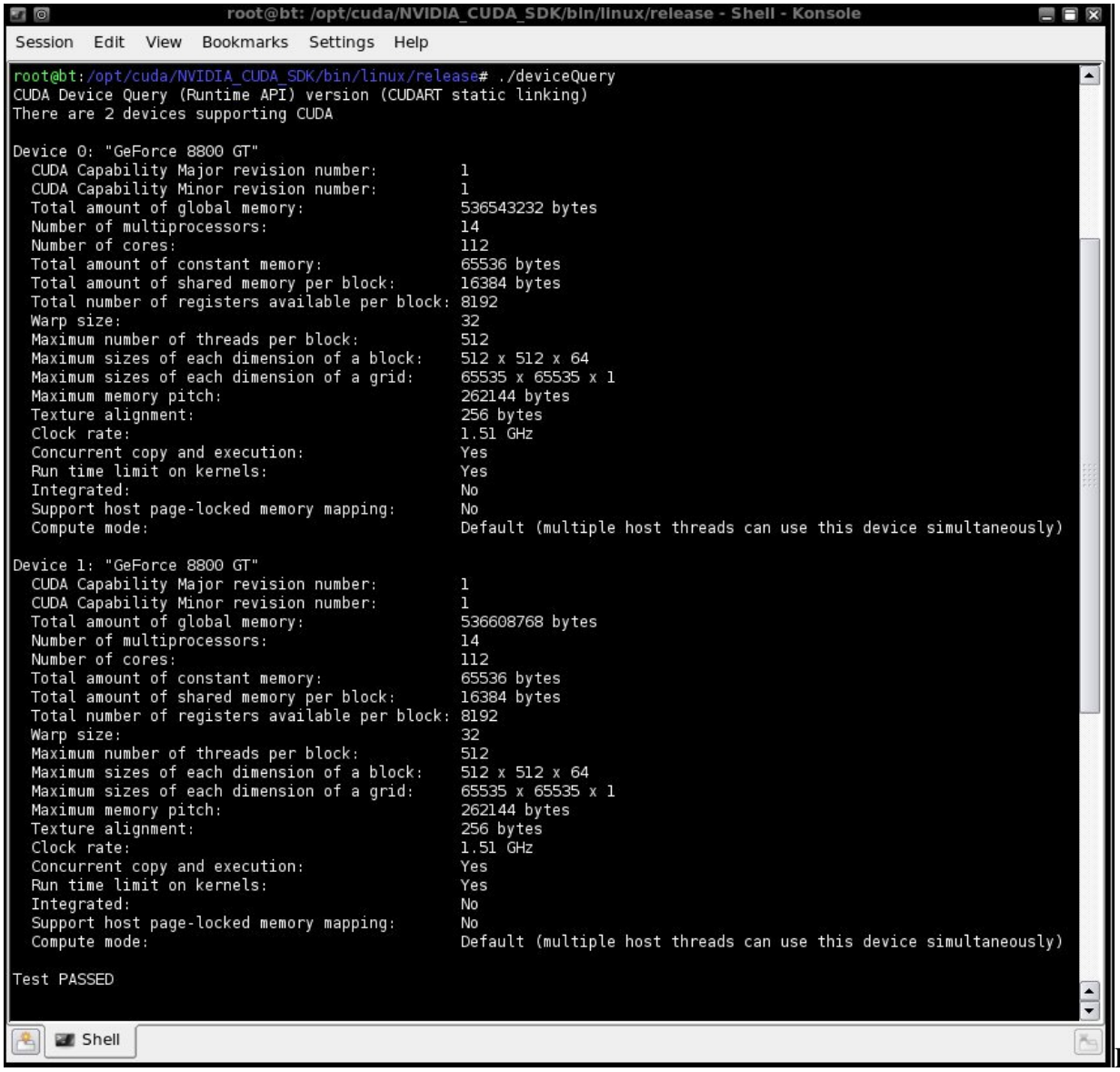
The cuda-sdk package contains code samples to help you get started programing in CUDA. This environment is located in /opt/cuda/NVIDIA_CUDA_SDK. I have already built all the tools in this folder for you however if you would like to build them yourself simply navigate to the SDK folder and issue the command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



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.



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.



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

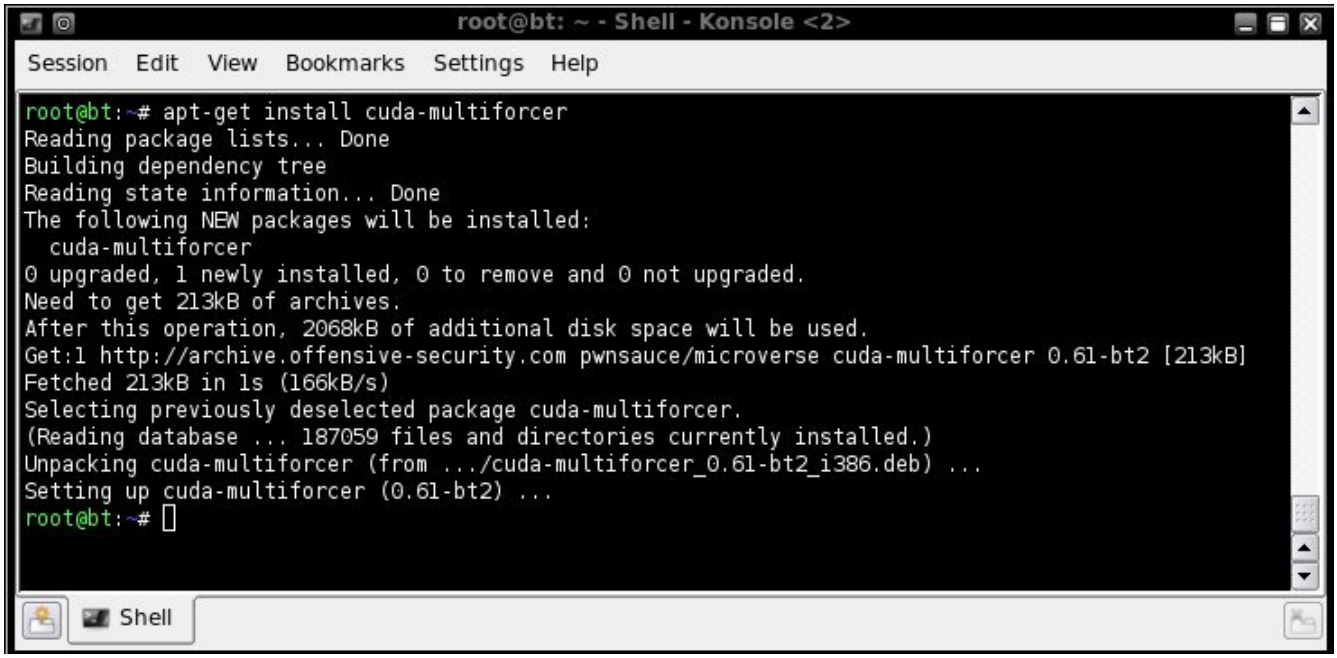
As I said before, issuing the “make” command in the root sdk directory will build all the sample tools. Anything built will appear in the release directory.

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

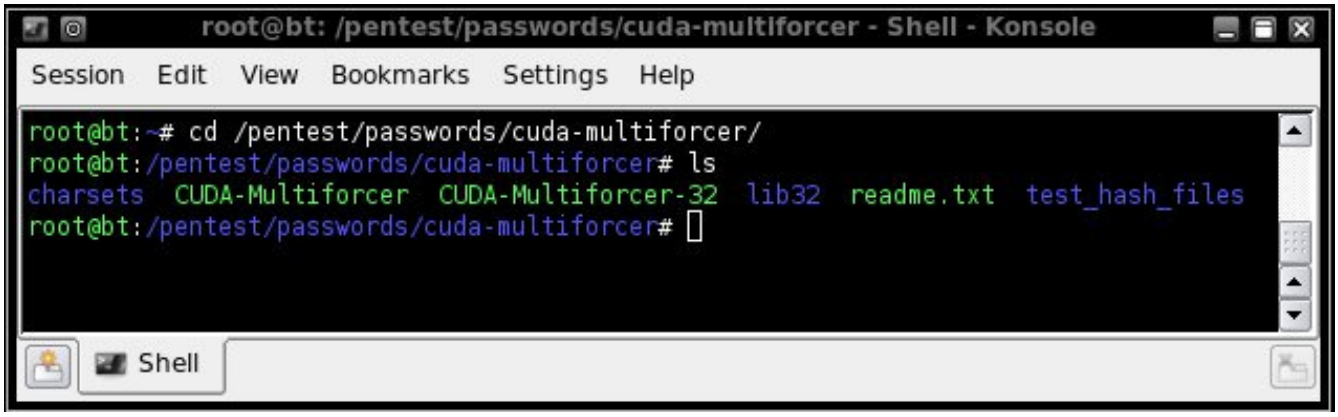
CUDA Tools

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.

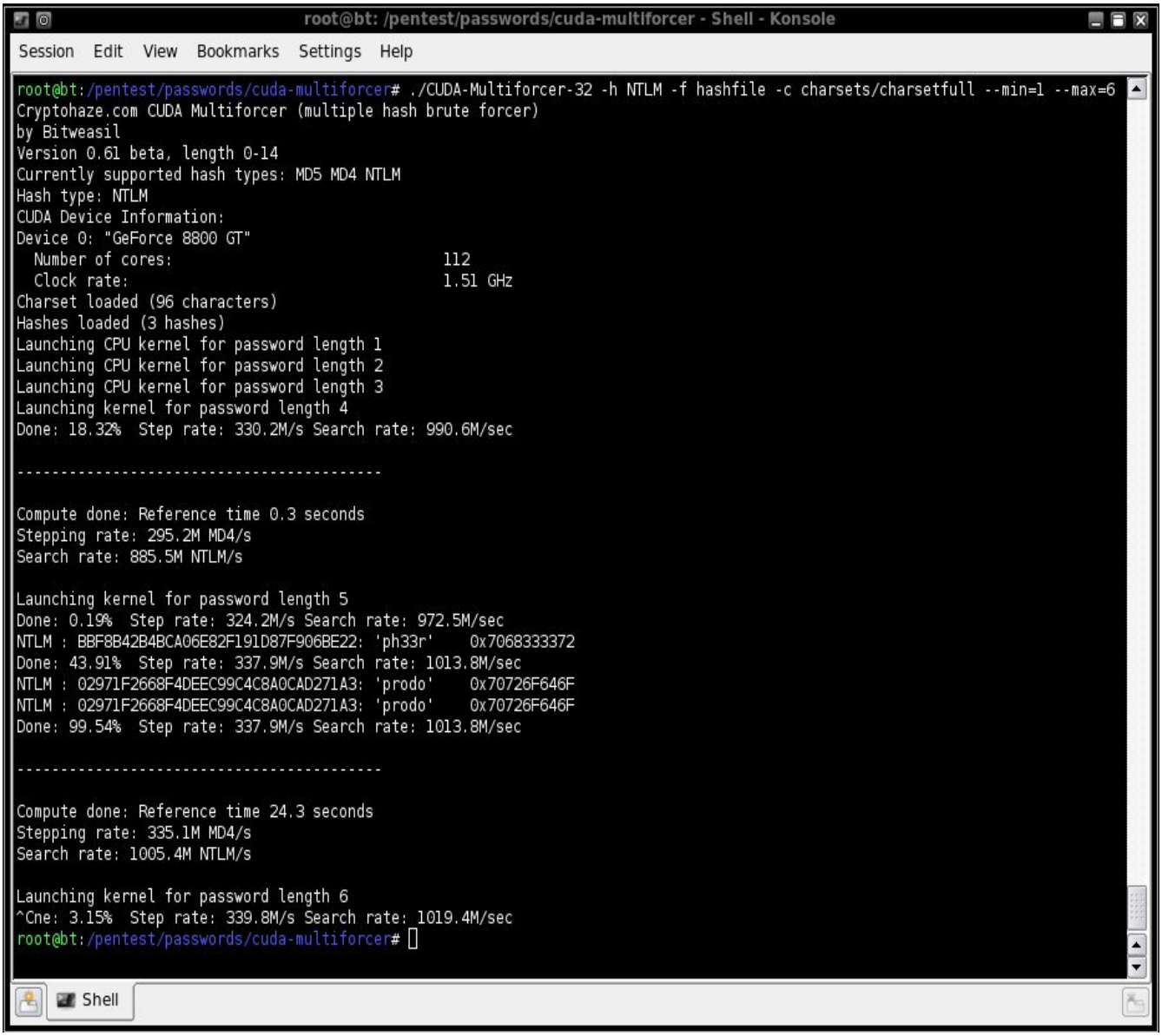


You can either launch the tool from the KDE menu or you can directly navigate to the folder in the /pentest directory.



In order to start cracking you need to create a file ion 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.



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

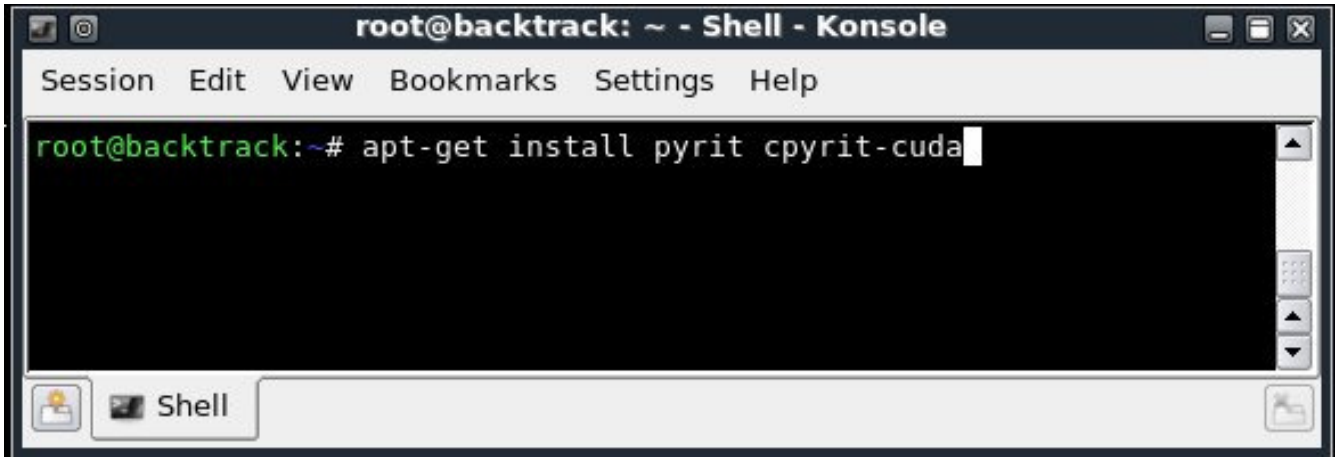
What is pyrit?

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.

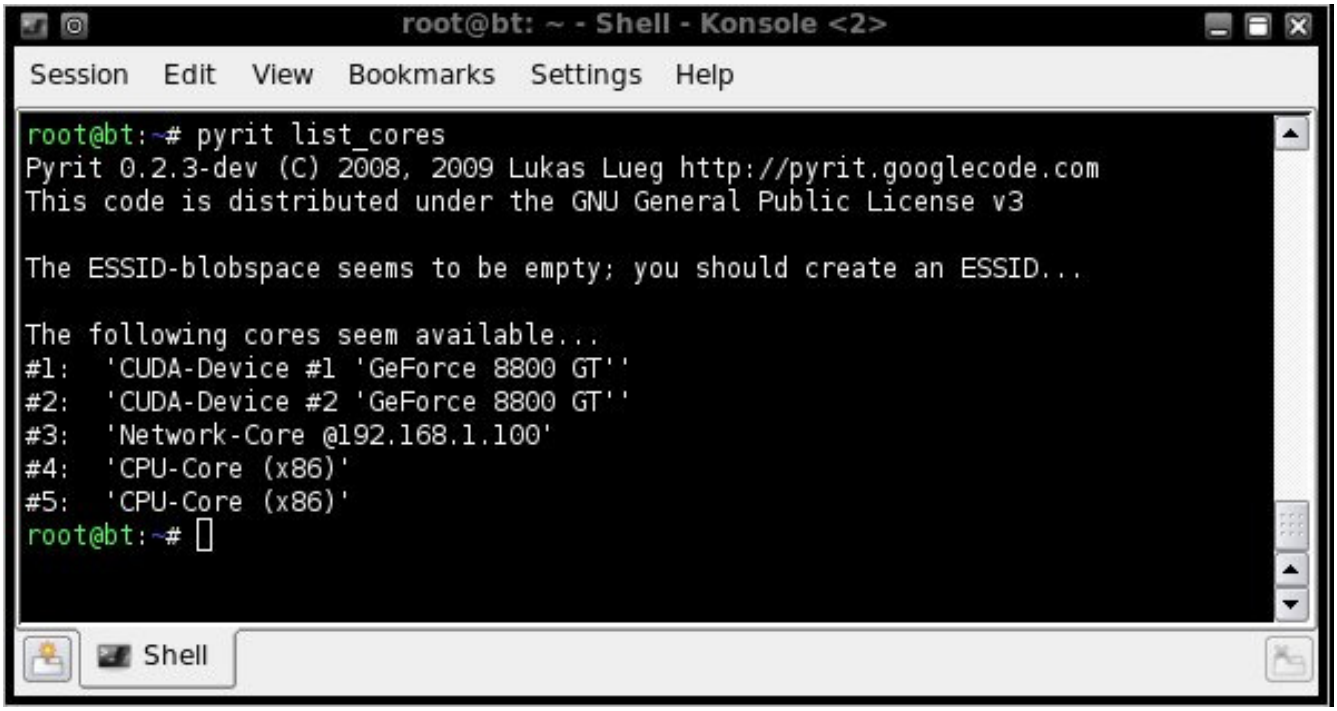
Up and running with pyrit

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.



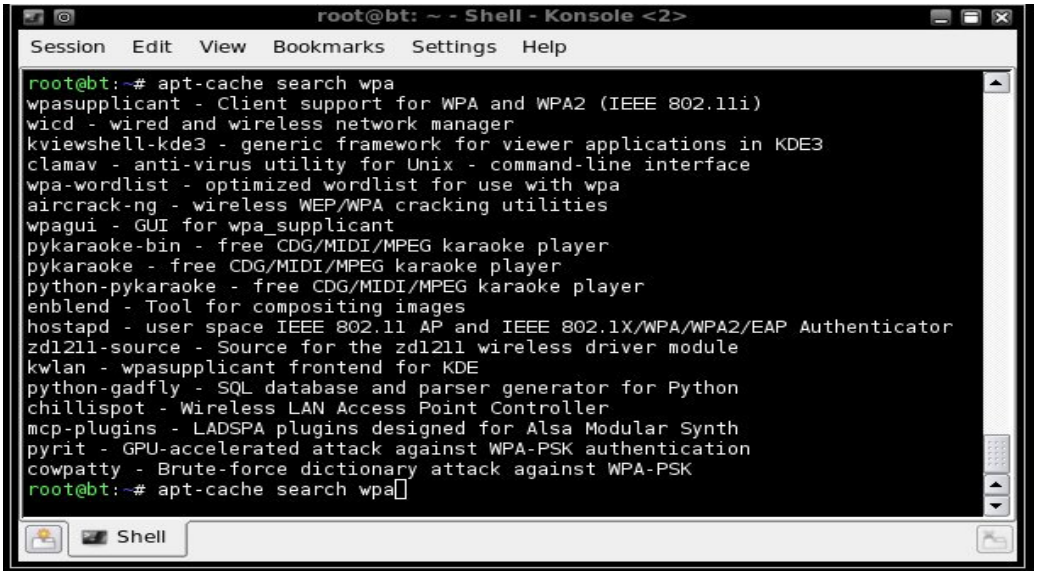
Making sure Pyrit is working:

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



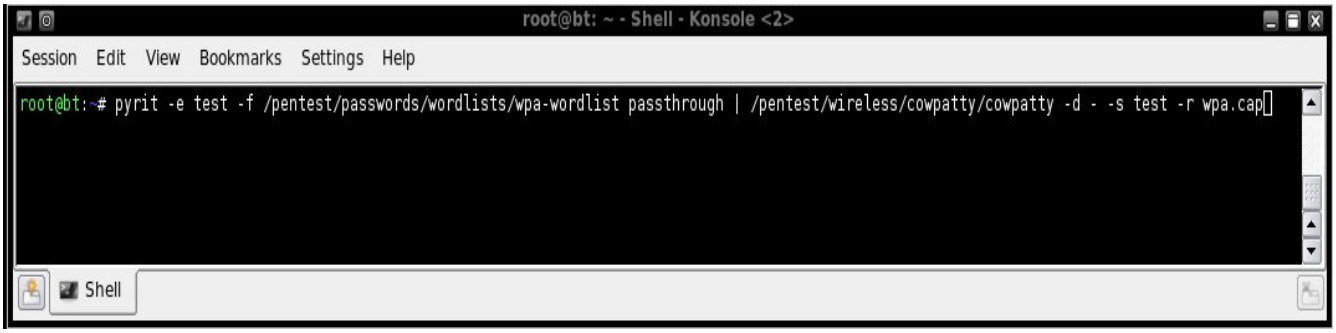
Dont worry about #3 networkcore in this picture yet. There will be more on that feature later on in the document. As you can see we have 2 8800 GT cards and we are using two of the four CPU cores as well. So now lets try a bench mark to make sure the Nvidia CUDA core gtes loaded and is working properly. In order to do that simply run root@bt~# pyrit benchmark.

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



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:



In case the text on the picture is to 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 a 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:

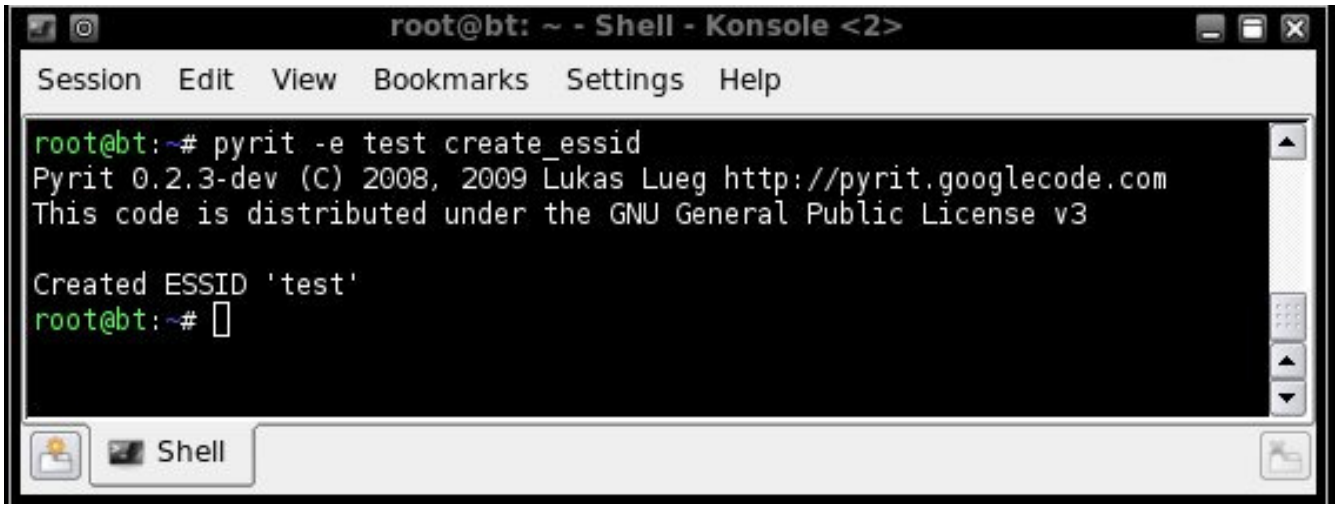
```
root@bt ~ $ /pentest/passwords/crunch/crunch 8 8 123456 | pyrit -e NETGEAR -f -
passthrough | cowpatty -d - -r wpa-01.cap -s NETGEAR
cowpatty 4.3 - WPA-PSK dictionary attack. <jwright@hasborg.com>

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.

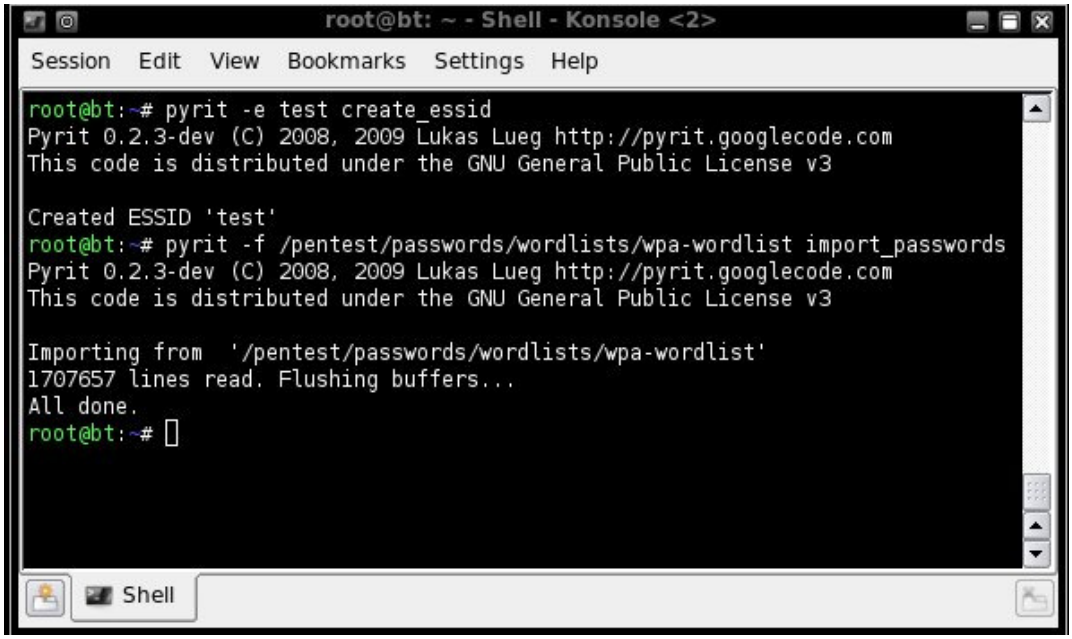
Batch Mode:

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.

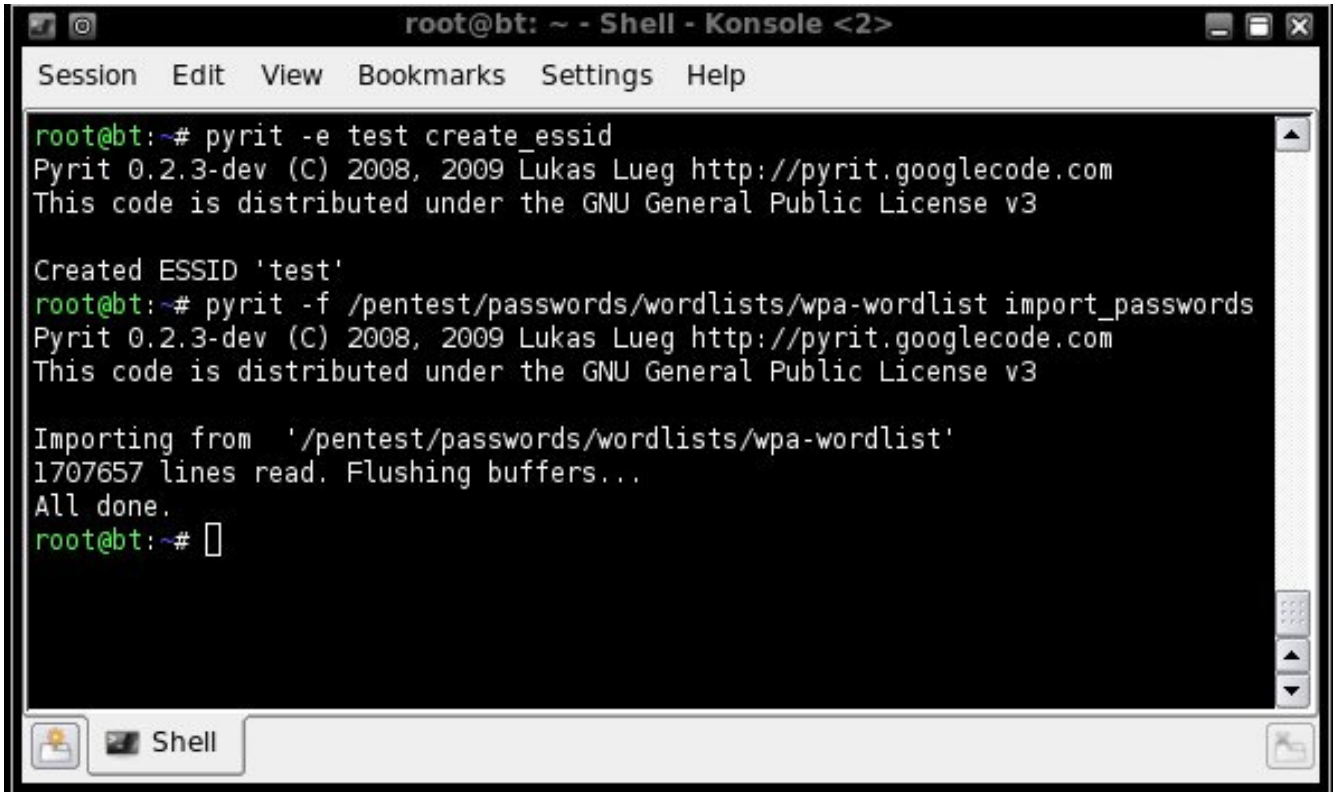
First we add our essid:



Next we import some passwords:

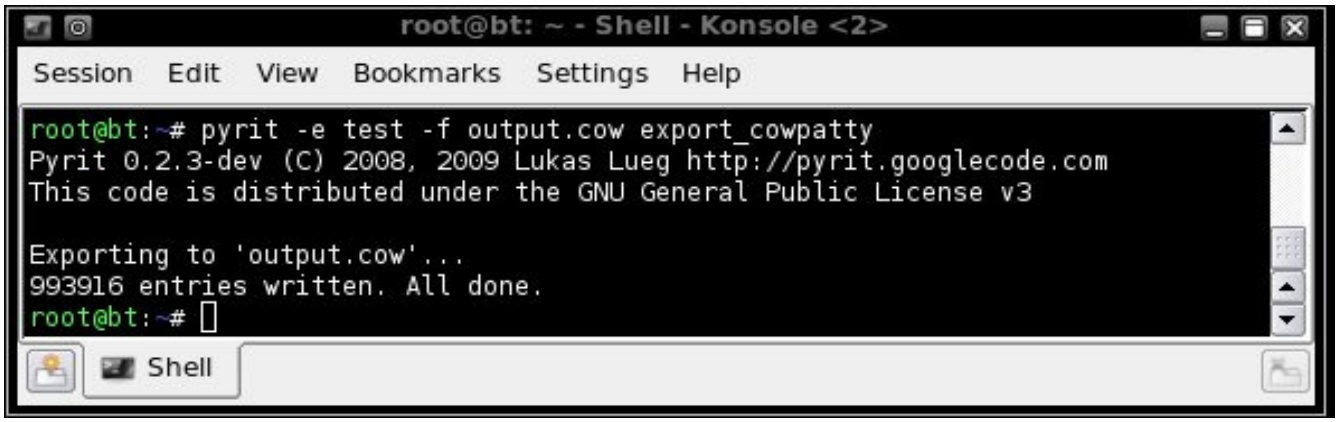


Next we start the batch processing:

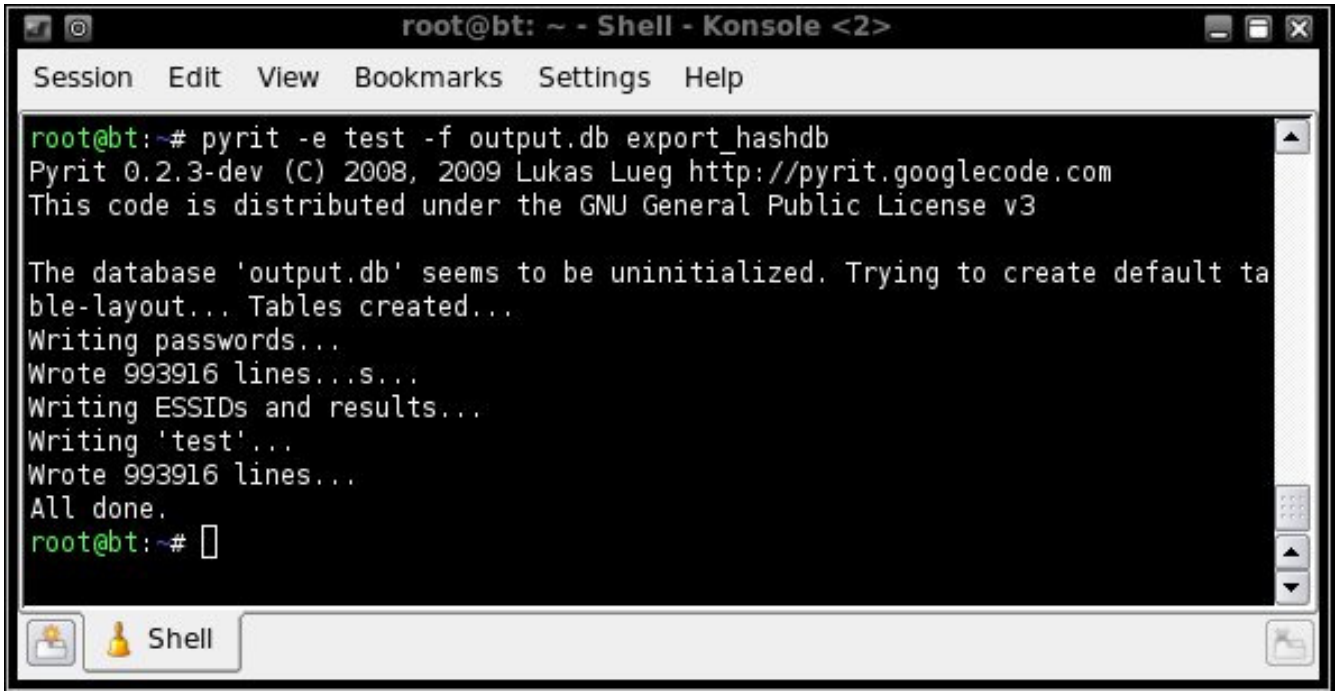


At this point we have a choice. We can either export to a cowpatty format or a aircrack-ng format. The cowpatty way is quite a bit faster due to some sqlite limitations however I will showcase both methods.

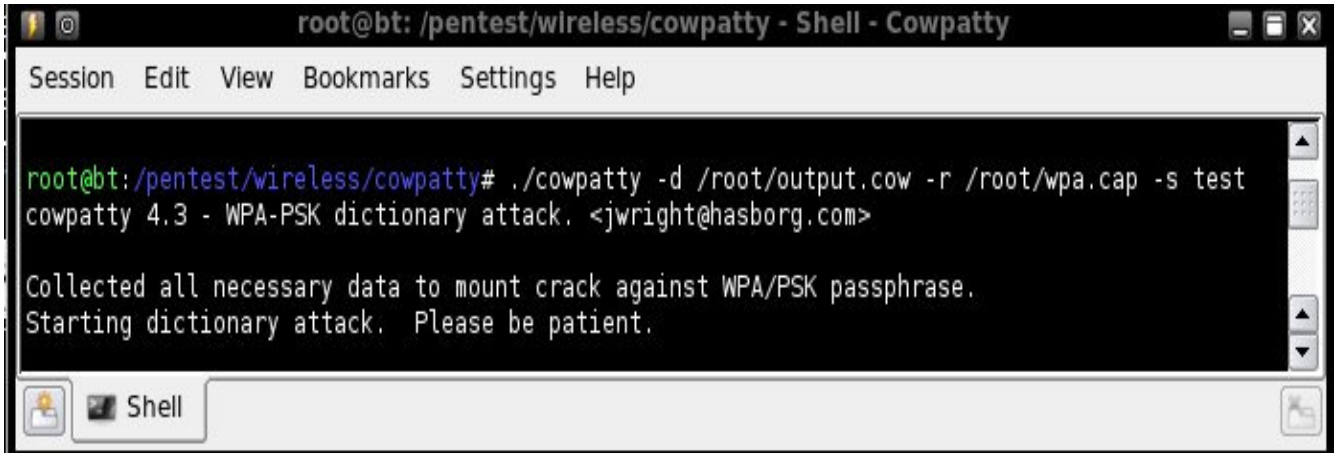
Here is the syntax for cowpatty:



And here is the syntax for aircrck-ng



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



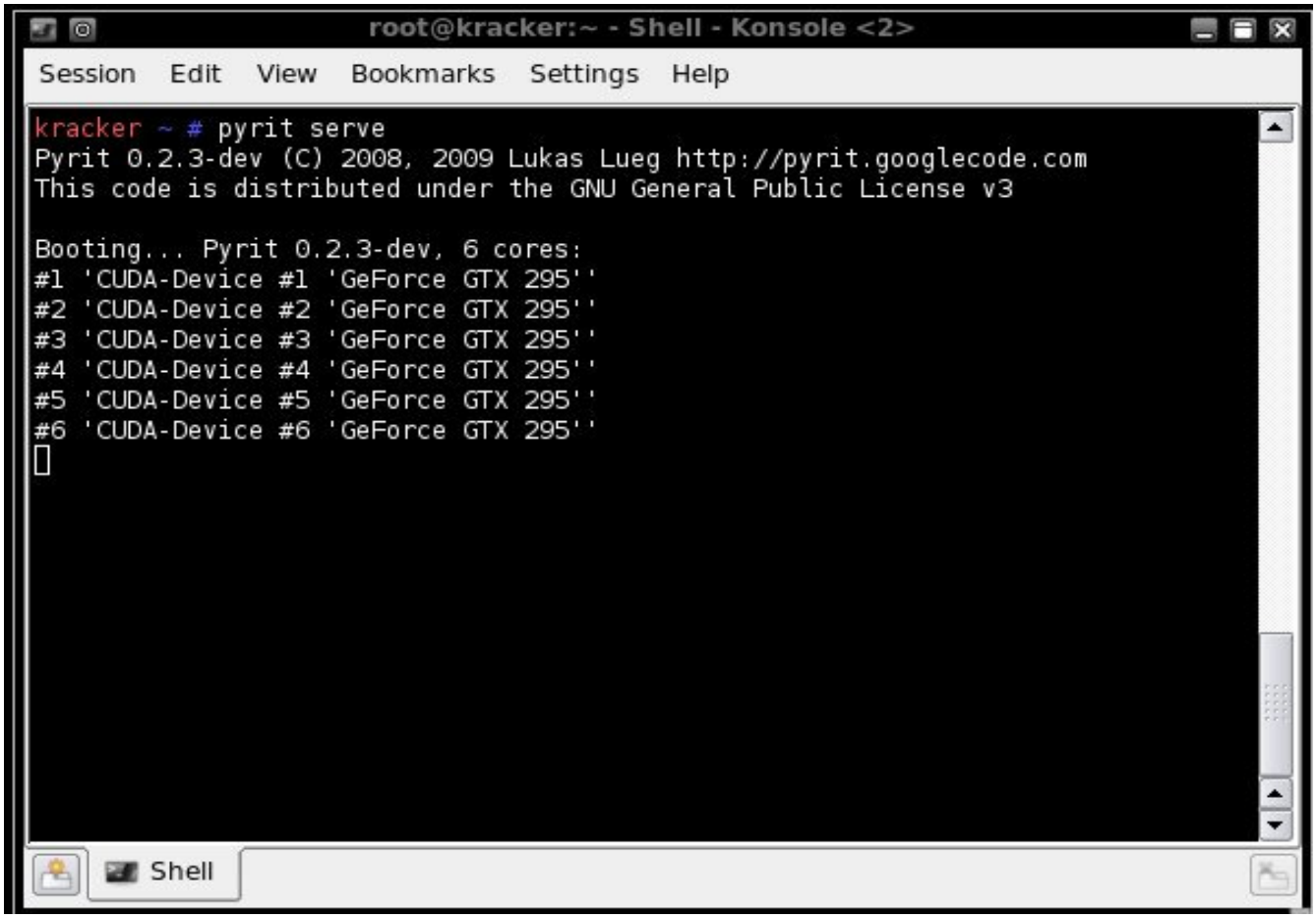
Server / Client Mode:

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…

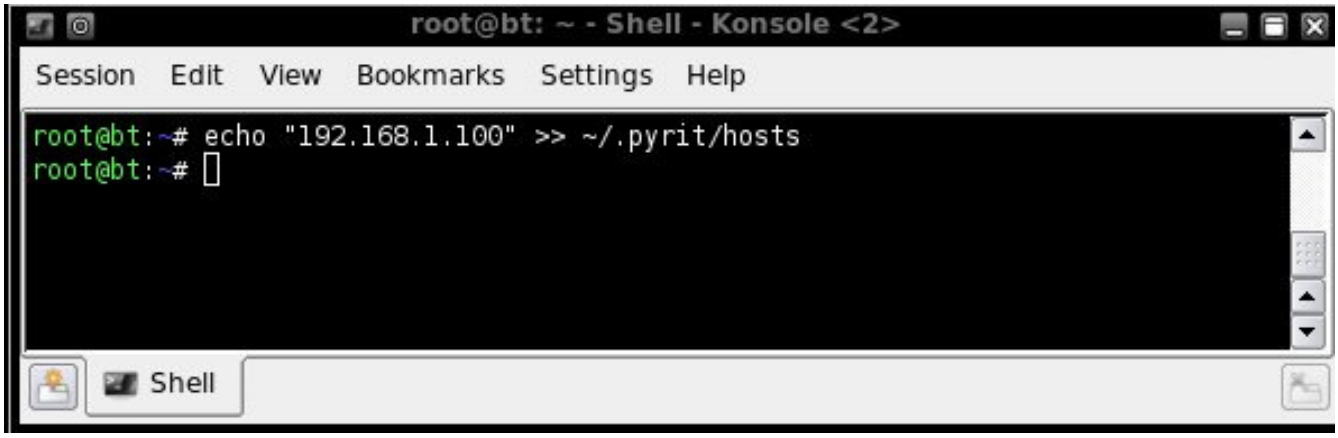
On the servers, the machines with the fast hardware:

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…

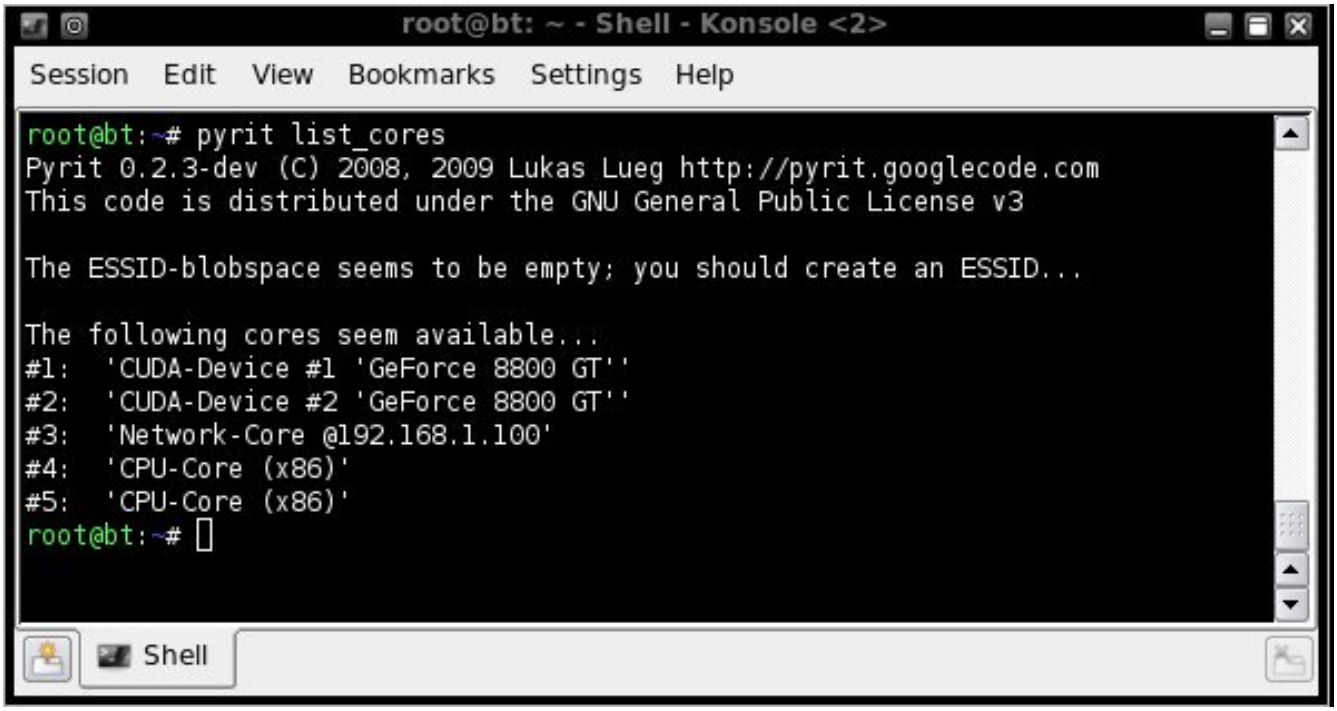


On the client, the machine that hosts the database:

- 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.



Now run "list_cores" and see if your network is available:



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
```

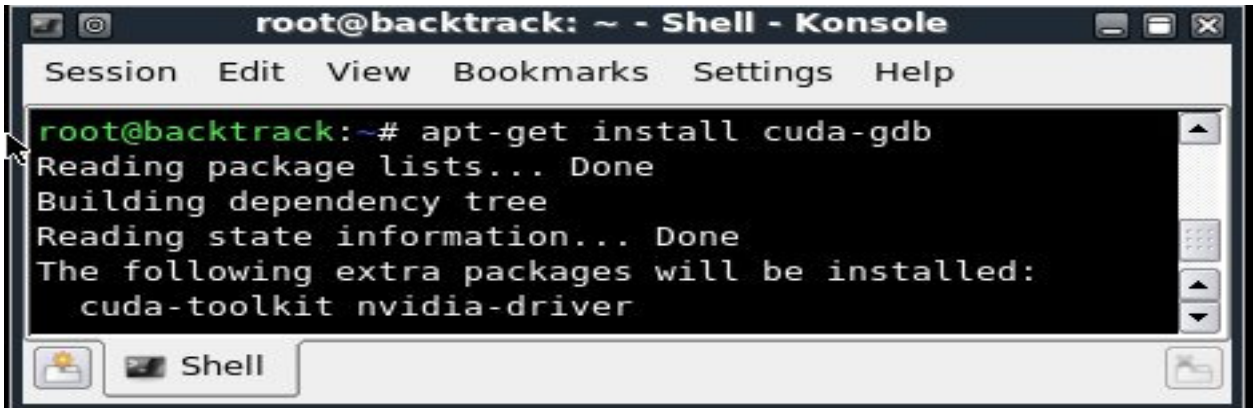
The -p switch is what adds the CUDA function to aircrack-ng. I have tested the tool and it does work but like I said its underdevelopment and could use some optimization. In my testing pyrit was still quite a bit faster however your milage may vary.

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:



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

Useful Links:

- 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://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, 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 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