Passwort Decrypting Cluster

Beitrag von „Goron“ vom 25. Dezember 2012, 16:46

Die Anleitung habe ich vor längerer Zeit mal ins Netz gestellt. Auf die Schnelle nur geringfügig editiert, also nicht gleich schlagen! Wenns zu knapp geschrieben ist, erläutere ichs etwas ausführlicher, wenn ich Ende der Woche wieder im Büro bin ...

Passwort von mysql vergessen? Rar Archiv verschlüsselt? Zur Zeit geht es leider nur auf einer Maschine, aber ich porte gerade openmpi: damit lässt sich dann die Power aller im Netz zur Verfügung stehenden Maschinen nutzen, kurz: Passwort-decrypting-Cluster from scratch

Prequel:
Ihr solltet sha und md5 verstanden haben und wissen, wie diese Algorithmen funktionieren!!! Ich werde hier keinen Grundkurs abhalten, ich stelle hier lediglich ein poc (proof of concept), also eine Machbarkeitsstudie und meine Erfahrungen damit zur Verfügung ...

Da ich nicht gänzlich wahnsinnig bin, hier die Schritt für Schritt Anleitung für Menschen, die wissen, was sie tun. Die Anleitung ist absolut korrekt und nach bestem Wissen und Gewissen geschrieben.

However, als Basis dient Mac OS X Lion 10.7.5, weshalb noch ein paar Vorarbeiten von Nöten sind:

- CLItools laden und installieren
- wget sourcen laden und compilieren
- MacPorts laden und installieren

Damit kanns dann schon losgehen:

Mittels wget besorgen wir uns die johntheripper Sourcen:

wget http://download.openwall.net/p...john-1.7.7-jumbo-6.tar.gz

Den Jumbo-Patch braucht ihr, da erst seid Jumbo5 die grundsätzliche Mehrkernunterstützung mit an Bord ist!

Diese lassen sich mittels "make macosx-x86-64" compilieren und verfügen über einen Standardsatz an Entschlüsselungsalgorithmen ...

Der Haken an der Sache ist, dass das Ganze nur auf einem Kern läuft! Da wir wahrscheinlich nicht so alt werden wie Methusalem kompilieren wir zunächst MPI in den guten alten John, damit wir alle Prozessoren/Kerne der eigenen Hardware nutzen können:

Mittels "port install openmpi" beschäftigen wir zunächst unsere Maschine für eine Weile -> Zeit für Kaffee!

Tjaaaaa, dauert länger als erwartet ... später mehr ....

So ein driet!!! Port tut zwar, was ich von ihm verlangt habe, aber aus irgendeinem Grunde fehlt das "mpirun". *grrrrr*

Dann halt anders:

Porten funktioniert nur halb, deshalb müssen wir uns von hier: http://www.open-mpi.org/softwa...oads/openmpi-1.4.3.tar.gz die Sourcen zu OpenMPI besorgen und von Hand kompilieren:

- make all (!!!!) und
- make install

Damit haben wir dann auch das später benötigte "mpirun" zur Verfügung.

Nun editieren wir das "Makefile" und schalten die beiden folgenden Zeilen scharf:

CC = mpicc -DHAVE_MPI -DJOHN_MPI_BARRIER -DJOHN_MPI_ABORT
MPIOBJ = john-mpi.o

Nun wieder neu kompilieren und siehe da, er verteilt die Last ....

Aufgerufen wird das Ganze dann mittels:

mpirun -n 8 ./john hash.txt

Die Zahl "8" sagt ihm, wieviele Kerne des System er nutzen soll, die Datei "hashes.txt" enthält zeilenweise die zu dekodierenden hashes.

Nun der eigentlich interessante Teil: das Cluster!
Seelig, wer auf Linux, oder OS X zurückgreifen kann, die Mausschubser unter uns sind leider aussen vor und haben den Text umsonst gelesen ;P

Voraussetzungen für das Cluster:

- 1- X Clients mit Linux, oder OS X
- jtr auf die selbe Art und Weise im selben Pfad (!!!) kompiliert

Um die Kommunikation zwischen Linux und OS X zu ermöglichen müsst ihr dafür sorgen, dass der integrierte "Hydra-proxy" im selben Pfad verfügbar ist:

ln -s /usr/local/bin/hydra_pmi_proxy /opt/local/bin/hydra_pmi_proxy

Jetzt legen wir uns eine Datei "notes.txt" an, in die wir zeilenweise die Namen, oder IP-Adressen der zu verwendenden Clients schreiben. Wichtig hierbei: localhost unbedingt auch hinzufügen, sonst rechnen nur alle anderen Maschinen und euer Rechner macht Pause .... 

Gestartet wird das Dekodieren dann mittels:

mpirun -f nodes.txt -n 18 john-1.7.9-jumbo-5/run/john john-1.7.9-jumbo-5/run/hashes.txt

---

Nun eine Alternative für langsame Hashes, AMD User haben hier GANZ KLAR die Nase vorn:

Kompiliert ihr die Sourcen statt wie oben angegeben mit folgendem Befehl:

make macosx-x86-64-opencl

Bringt ihr ihm gleichzeitig bei, eure GPU anstelle von eurer CPU zu nutzen. Theoretisch sollte auch ein Mischbetrieb funktionieren, ich habe bloss noch nicht rausgefunden wie ☹

Für NVidias CUDA Support könnt ihr mit

make macosx-x86-64-gpu

kompilieren, die Resultate waren bei mir allerdings mehr als enttäuschend ....

Um die Berechnung von der GPU ausführen zu lassen, starten wir mit:

./john –format=sha512crypt-opencl

Da wir wissen, dass der Algorithmus auf der GPU schneller zu knacken ist, wissen wir welcher es ist und geben diesen mit an, spart Zeit, Strom und Nerven ;P

Haben wir mehrere leistungsfähige Maschinen im Netz, lassen sich analog zu oben, auch mehrere GPU´s parralel nutzen:

mpirun -f nodes.txt ./john –format=sha512crypt-opencl

Auf die Schnelle wars das jetzt erstmal ... bei Fragen und Anregungen ...

Gruß

Goron

ps.: Das Beste kommt natürlich zum Schluß!!! Da es die FREIEN Sourcen sind, hat die Sache natürlich einen Haken: das Passwort darf mx. 8 Stellen haben. Ist das Passwort länger .... passiert halt einfach nix; es rechnet sich in einer Endlosschleife tot.

Beitrag von „Dr. Ukeman“ vom 25. Dezember 2012, 23:47

eine Sehr nette Anleitung werde ich mal bei Gelegenheit testen.
Mal schaun wie lange der i7 mit den c2Quad und dem C2Duo vll sogar incl raspi und netbook brauchen für mein Wlan Passwort. Allemal ein Interessantes Projekt.

Beitrag von „Goron“ vom 26. Dezember 2012, 00:16

Ein Core2Duo E6600 2,4 Ghz, brauch 1 sek. für ein 6 stelliges Passwort (nur Kleinbuchstaben), eine 8400GS 34 sek. ....

Groß- und Kleinschreibung + Sonderzeichen vergrößern die Zeit ungemein .... 15+ Zeichen sind hart würded ich mal meinen, aber ich hab noch nicht investiert. Wenn also jemand Lust hat und die Vollversion kauft UND auch noch die sourcen opfert, teste ich gerne weiter ....