Wisecracker™

A high performance distributed cryptanalysis framework

A Technical White Paper

October 30 2012

Written by
Vikas N Kumar

Introduction

Cryptanalysis can be performed in various ways such as by using number theory to discover flaws in existing cryptographic algorithms or by using brute force to break the cryptography by using numerous computers. Discovering number theory related flaws might be near impossible to find or might need a Ph.D. in mathematics, but using brute force might make it easy to break such algorithms. Wisecracker aims to tackle cryptanalysis by using the brute force method.

However, brute force cryptanalysis, depending on the algorithm, might still need large scale supercomputers or massively parallel clusters of computers to be able to work in a reasonable amount of time that is countably finite. Some algorithms might take months or years to crack while some might take minutes or seconds. Cracking algorithms that might take weeks or months is possible by those who have access to supercomputers and massively parallel clusters such as the researchers at government agencies like the NSA and big corporations like Google®, Facebook® and Twitter®. An average or hobbyist security researcher does not have access to such computer systems and is not able to perform such research at a low cost. Wisecracker aims to solve this problem for researchers with less resources by optimizing with intelligent software.

Wisecracker consists of a framework for a researcher to use the computational power of commercial off-the-shelf (COTS) graphical processing units (GPUs) across multiple multi-processor and multi-core systems by using OpenCL and MPI in tandem. Currently, a sample application to crack the MD5 one-way cryptographic hashes for strings of 6-8 printable ASCII characters is provided as a demonstration of how to use the framework. These kinds of strings are used for storing passwords by web developers. More applications such as for cracking SHA-1, SHA-2 and other cryptographic algorithms will be added as time progresses.

With a software framework like Wisecracker a security researcher can leverage the technology of distribution across multiple heterogeneous GPUs and CPUs already provided as the core of the framework, and focus on the implementation of the cryptographic algorithms. The framework has a C and C++ API, and the user has to implement the portion of the code that will be distributed on the GPUs and/or CPUs using OpenCL.

Wisecracker is supported on Linux, Windows and Mac OSX operating systems and depends on OpenCL and optionally MPI for distribution across multiple systems.

This document gives a high level description of the capabilities and internal designs of Wisecracker and does not describe the API itself. For the API, please refer to the API documentation provided separately. An example application for cracking MD5 checksums for 6-8 character strings will be explained in this document in the later sections.

Distribution Across Systems

Heterogeneous and homogeneous clusters of computers with or without GPUs can be handled by Wisecracker, as long as the appropriate OpenCL libraries have been installed. OpenCL implementations provided by Intel®, AMD®, NVIDIA® and Apple® are supported. Within a single system the framework uses OpenCL to distribute the workload and across systems it uses MPI to distribute the workload. Testing for Wisecracker has been done with the OpenMPI implementation of MPI, although any standards compliant MPI implementation, such as MPICH, should work just as well.

The notion of a *system* for Wisecracker is a single computer which might have one or more GPUs and has one or more CPUs. If the system has a CPU version of OpenCL installed such as that of Intel, AMD or Apple, then the system will use the CPUs for computing. If the system has the OpenCL implementation of GPUs installed such as that of AMD or NVIDIA, then the system will use the GPUs for computing. If the system has both the CPU and GPU implementations installed, it will use any or all depending on how the user makes the device selection in the API.

If the user has only one system, he does not need MPI as a dependency to use Wisecracker. A security researcher might have various systems each with different computational capabilities, some with multiple CPUs and some with a combination of CPUs and GPUs. With this framework, they can leverage all the possible systems they have for performing their analysis. The beauty of OpenCL is that it is designed to be cross platform across a variety of architectures and hence Wisecracker can run on any system that supports OpenCL.

Nowadays, cloud computing companies like Amazon® provide high end GPU computing resources which can also be used by a security researcher for just a few dollars an hour. Wisecracker is supported on the Amazon EC2® GPU cluster virtual machines (VM) out of the box. It comes with a simple setup script that can setup the Amazon GPU VM with all the pre-requisites needed for Wisecracker and applications built with Wisecracker to run.

Let us take the example of the sample application provided with the Wisecracker source code called `wisecrackmd5`. This application takes an MD5 cryptographic checksum that has been generated by a character string that is of 6-8 characters in length and finds the string that has generated the MD5 checksum. Many websites store their passwords as MD5 checksums and this application can be used to reverse an MD5 checksum into a password.

Consider the case of a 6-character password that might have any of the 94 printable ASCII characters in it, i.e. alphabets from A-Z in both upper and lower case, digits from 0-9 and all the remaining special characters like punctuation marks. For a 6-character password using 94 characters, the number of possible tries that will generate the given MD5 checksum are 94^6 which is 689,869,781,056 or approximately ***690 billion*** combinations!

On a single system with 2 Intel Xeon 5030 processors with 8 cores this takes on an average about 7 hours to reverse the MD5 sum into the source string.
However, using the powerful NVIDIA GPUs on a single Amazon's EC2 cluster VM takes about 20 minutes to reverse. When Wisecracker is built with MPI support, the user can run this application on 2 Amazon EC2 cluster VMs and it takes about ***3 minutes*** to reverse on average! That is a near exponential decline with just twice the power.

This happens because the task distribution uses a ***divide-and-conquer*** model between systems to distribute the tasks. When each system receives its task range, it then uses a ***round-robin*** task block distribution for each OpenCL device on that system based on the device's compute capabilities provided by the OpenCL API (a product of the number of compute units and work group size).

Amazon's VMs have 2 GPUs each. When a single system is being used the tasks are distributed in the range `[1, 690billion]` between 2 GPUs by dividing the range into smaller blocks of size based on the product of the compute units and workgroup size. So if a GPU's compute units are 32 in number and have a work group size of 2048, it gets `32 × 2048` task blocks to work on. So one by one each GPU will keep computing on it successive task blocks that it is given until it finds the solution.

When you use 2 systems, the tasks are distributed between each system as the ranges `[1, 345billion]` and `(345billion, 690billion]`. Once each system gets its task range, it distributes work between its GPUs in a similar fashion as task blocks based on compute units and work group size.

Let's say you want to recover the string `z@bD1g` and it might be in the index range `[345billion, 690billion]`. If you were to run this on 1 Amazon GPU VM the program will have to compute for `[1, 345billion]` range first and then get to the `(345billion, 690billion]` range.

However, if you distribute this on 2 VMs you will hit upon the solution faster because the second system is starting from 345 billion and you might not need to compute all the 345 billion possible values on each VM. You are saved from the needless computation of the `[1, 345billion]` range in full as done in the single system operation.

Hence the 2 VMs give a bigger decline in time rather than 1 VM because of the way the work is distributed for the MD5 example, `wisecrackmd5`.

On an average with different sets of strings the runtime drastically goes down because of the fact that the search buckets are smaller and start at different points.

NOTE: *We define average as the average time taken to extract the source string from the MD5, and that the code running on the CPUs is the* ***same*** *OpenCL that will run on the GPUs. Comparisons with highly optimized CPU code using SSE2 intrinsics has not been done at this time.*

As we can see, with the power of GPUs across multiple systems and the intelligent distribution of tasks by Wisecracker we can exponentially reduce the time taken to try out various combinations for MD5

reversing. This is true for any other algorithm as well although the time taken might vary based on the number of combinations to be tried out and the computational requirements of the algorithm itself per combination.

Domain Decomposition

The distribution of billions or trillions of combinations or tasks gets done by Wisecracker which uses a ***master-slave model*** to distribute the workload across multiple GPUs and CPUs. Figures 1 and 2 denote the distribution across GPUs and CPUs of a single system and across multiple systems respectively.

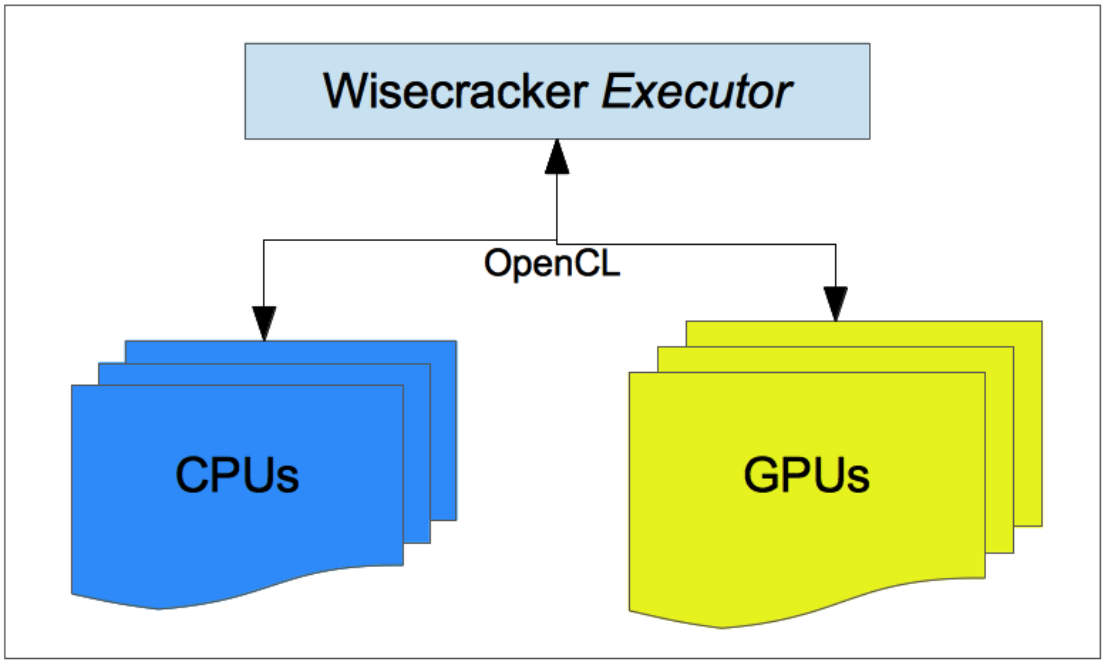

Figure 1. Distribution across a single system with OpenCL

When an application is written using Wisecracker, it provides a set of callbacks for Wisecracker's internal runtime, called *executor*, to invoke during execution. Some of these callbacks inform Wisecracker about the number of tasks that Wisecracker has to distribute. When running across multiple systems using MPI, the application runs copies of itself (called *slaves*) on each of those systems. The initiating system is called the *master*. Hence, there is an *executor* on each slave system communicating with the master *executor* system. Note that on a single system there is no slave *executor* and the master *executor* does all the required work in the same way a slave would perform.

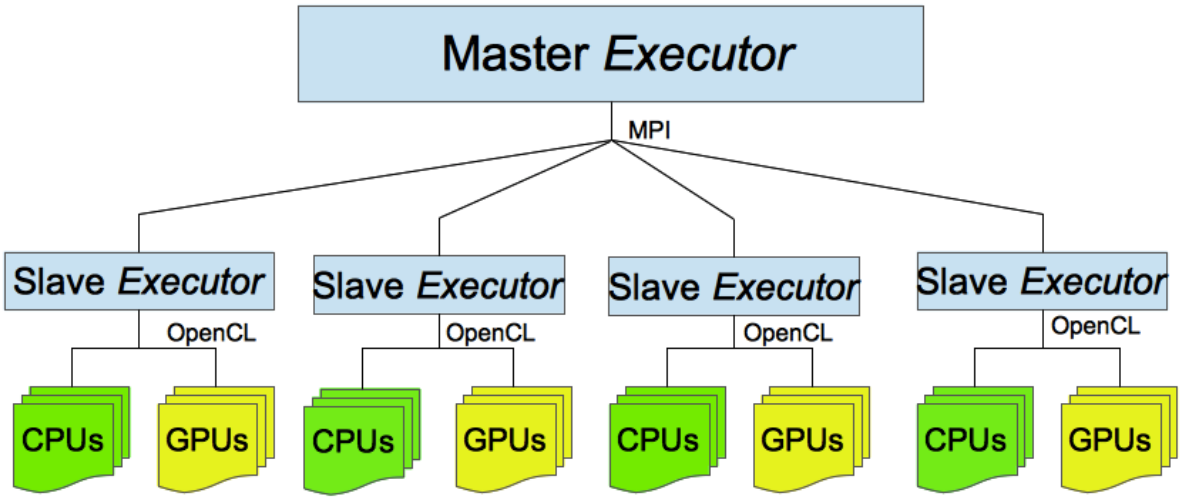

Figure 2. Distribution across multiple systems with MPI and OpenCL

The master *executor* queries each slave system's details and gets the device information about every system. Once the device information is retrieved from all the systems, the master *executor* will then break up the tasks as per the capabilities of each slave system and then inform each slave *executor* the task range they will be executing.

When a slave *executor* receives a task range, it divides it up between its multiple devices and starts executing the workload by invoking the correct callback per device per range. The execution is non-blocking and event based by using OpenCL events. When a task range has completed execution, the results are communicated back to the master *executor* which will then invoke the callbacks in the master application appropriately informing the application that the results have arrived. This goes on until all the tasks have been completed or until the master or any of the slaves call for a stop to the execution based on certain conditions.

The domain decomposition is **not** very complex and depends on the compute capabilities of the OpenCL devices on each system and the total number of tasks to be executed. The user can always modify or customize the domain decomposition by modifying the *executor* source code of Wisecracker themselves, so as to attack their problem as desired.

The master *executor* process does **not** execute any of the tasks because it spends all its power collecting results from every slave which might send the results in multiple blocks for efficiency purposes.

Example Application: `wisecrackmd5`

As part of the Wisecracker framework, a sample application to reverse an MD5 checksum of a 6-8 character string (such as a password) is provided. The user can learn how to use the Wisecracker API by looking at the source code of this application and referring to the API document. Here we explain how to run this application after it has been compiled into an executable.

Single System Run

Let's say the user wants to reverse the MD5 sum `7e7cf8ad5181911e63d8bfc12d0a0747` which is the MD5 sum for the string `a6cd@F` (without the `NULL` terminating character).
If the user wants to run `wisecrackmd5` on a single system, all they need to do is run the following command at the shell prompt:

```
$./wisecrackmd5 -N 6 -C alnumspl -M 7e7cf8ad5181911e63d8bfc12d0a0747
```

Depending on the system properties, this might take a few hours on a system with 8 CPUs to maybe 20 minutes on the Amazon EC2 GPU VM to return the string `a6cd@F`.

To expedite the run for testing, the user can use the `-p` option to provide a hint to the program, like below:

```
$./wisecrackmd5 -N 6 -C alnumspl -M 7e7cf8ad5181911e63d8bfc12d0a0747
-p a6c
```

This might take a few seconds to complete as the number of combinations to try exponentially reduces to 830,584 from 689,869,781,056.

To view all the options provided by the `wisecrackmd5` application you can use the `-h` option:

```
$./wisecrackmd5 -h
```

Multiple System Run

For a multiple system run, the user has to setup MPI to work correctly across the systems. Each system will need to have OpenCL and MPI installed and also the `wisecrackmd5` application present in the ***same path*** on both systems.

For Linux and Mac OSX systems, we recommend using MPI over `ssh` on systems. It is necessary for `ssh` to work using public key authentication rather than passwords. For Windows, we recommend using Microsoft's recommended MPI or using OpenMPI's instructions for setting up MPI on Windows. Details of setting up MPI are out of the scope of this document.

Once MPI has been setup, the user should create a host file with the names or IP addresses of the systems that will be running `wisecrackmd5`. An example host file that runs with 1 master and 2 slaves will look like this:

```
192.168.1.2 slots=2
192.168.1.3
```

The reason the first system IP address has 2 slots is because the master *executor* uses 1 extra slot and does not perform computations, and hence we need to run 1 slave on that machine. The user might choose not to run a slave on the master machine.

To run the `wisecrackmd5` application using MPI, the user has to run the following command for 1 master and 2 slaves (3 processes) with the host file being as above will look like this:

```
$ mpiexec -np 3 -hostfile ./path/to/hostfile ./wisecrackmd5 -N 6 -C
alnumspl -M  7e7cf8ad5181911e63d8bfc12d0a0747
```

If we run the above command with 2 Amazon EC2 GPU VMs and number of processes as 3, it takes about 160-180 seconds to retrieve the string `a6cd@F`.

If the user runs the MPI compiled application with 1 process, the master *executor* automatically switches to single system mode and will perform the computation itself. As a debugging method, the application can be run with 2 processes and it will run as 1 master and 1 slave which can help debug the communications between processes.

Conclusion

Wisecracker is an easy to use framework for security researchers to perform their own cryptanalysis by leveraging multiple systems with multi-core processors and GPUs. They can either use the provided sample applications or can write their own or enhance the samples themselves. The open source nature of Wisecracker allows the user to have the freedom to perform research as desired. The presence of cloud computing VMs such as those provided by Amazon can be leveraged to perform cryptanalysis on demand and Wisecracker helps the users by making distribution easier across systems.

The source code of wisecracker can be downloaded from https://github.com/vikasnkumar/wisecracker. Professional level technical support can be obtained by contacting the developers at Selective Intellect LLC by emailing them at wisecracker@selectiveintellect.com.