Взлом паролей с применением GPU.

Оглавление

[Введение 3](#_Toc479674609)

[GPU 4](#_Toc479674610)

[Ахитектура 4](#_Toc479674611)

[Классические задачи для GPU 5](#_Toc479674612)

[Кластер Джереми Госни 6](#_Toc479674613)

[Взлом паролей 7](#_Toc479674614)

[Атака по словарю 8](#_Toc479674615)

[Атака Bruteforce 8](#_Toc479674616)

[Атака с помощью радужных таблиц 9](#_Toc479674617)

[Взлом алгоритма MD5 9](#_Toc479674618)

[Hashcat 11](#_Toc479674619)

[Рассмотрим пример. 12](#_Toc479674620)

[Заключение 14](#_Toc479674621)

[Проблемы развития GPU-вычислений 14](#_Toc479674622)

[Итог 15](#_Toc479674623)

[Источники 16](#_Toc479674624)

Введение

Современная криптография включает в себя три направления: шифрование с закрытым ключом, шифрование с открытым ключом и хеширование. Сегодня мы поговорим о том, что такое хеширование.

В целом под хешированием понимают преобразование входных данных произвольной длины в выходную битовую строку фиксированной длины. Чаще всего хеш-функции применяют в процессе аутентификации пользователя (в базе данных обычно хранится хеш пароля вместо самого пароля) и для вычисления контрольных сумм файлов, пакетов данных и т. п.

Хеши паролей были придуманы для решения одной задачи. Если пользователи должны вводить пароли для того, чтобы войти в систему, то надо где-то в системе эти пароли хранить. Каким же образом сделать это так, чтобы они не хранились в виде открытого текста, но при этом можно было бы проверить, правильно ли пользователи их набирают при входе в систему? Решением является однонаправленное шифрование паролей с помощью хеша. Идея однонаправленного хеширования состоит в том, что пароль достаточно легко зашифровать, но почти невозможно восстановить его по хешу.

В последнее время даже эта система защиты паролей стала подвергаться атакам и взломам, что являет собой серьезную проблему. Большинство применяющихся хеш-алгоритмов настолько устарели, что вообще не должны применяться. Взломщики паролей всё больше и больше используют мощные GPU и облачные платформы. Поэтому вопрос изучения потенциальных возможностей GPU и его слабых мест является на сегодняшний день весьма актуальным. Ведь зная возможности противника, проще защититься от него.

GPU

GPU, или Графический процессор, является частью видео системы компьютера. Типичные функции ГПУ - помочь с рендерингом 3D графики и визуальных эффектов.

Микроархитектура GPU построена совсем иначе, чем у обычных CPU, и в ней изначально заложены определенные преимущества. Задачи графики предполагают независимую параллельную обработку данных, и GPU изначально мультипоточен. Микроархитектура спроектирована так, чтобы эксплуатировать имеющееся в наличии большое количество нитей, требующих исполнения.

Если говорить о применимости GPU, то ситуация такова: существует не так много задач, для которых требуется большая вычислительная мощность и сильно не хватает обычного двухъядерного процессора.

Несколько лучше перспективы GPU с точки зрения использования в профессиональных приложениях и секторе рабочих станций, так как там больше потребности в производительности.

Больше всего GPU востребованы в сфере научных вычислений. Уже построено несколько суперкомпьютеров на базе GPU, которые показывают очень высокий результат в тесте матричных операций. Научные задачи так многообразны и многочисленны, что всегда находится множество, которое прекрасно ложится на архитектуру GPU, для которого использование GPU позволяет легко получить высокую производительность.

Ахитектура

GPU состоит из нескольких десятков (30 для Nvidia GT200, 20 — для Evergreen, 16 — для Fermi) процессорных ядер, которые в терминологии Nvidia называются Streaming Multiprocessor, а в терминологии ATI — SIMD Engine. Также можно называть их минипроцессорами.

Лучше всего представлять GPU как некий многоядерный процессор с большим количеством ядер, исполняющих одновременно множество нитей.

Каждый минипроцессор имеет локальную память, размером 16 KБ для GT200, 32 KБ — для Evergreen и 64 KБ — для Fermi (по сути, это программируемый L1 кэш). Она имеет сходное с кэшем первого уровня обычного CPU время доступа и выполняет аналогичные функции наибыстрейшей доставки данных к функциональным модулям. В GPU локальная память служит для быстрого обмена данными между исполняющимися нитями.

Одна из обычных схем GPU-программы такова: в начале в локальную память загружаются данные из глобальной памяти GPU. Это просто обычная видеопамять, расположенная (как и системная память) отдельно от «своего» процессора — в случае видео она распаяна несколькими микросхемами на текстолите видеокарты. Далее несколько сотен нитей работают с этими данными в локальной памяти и записывают результат в глобальную память, после чего тот передается в CPU. В обязанность программиста входит написание инструкций загрузки и выгрузки данных из локальной памяти. По сути, это разбиение данных для параллельной обработки.

Но кроме локальной памяти в минипроцессоре есть ещё одна область памяти, во всех архитектурах примерно в четыре раза бо́льшая по объему. Она разделена поровну между всеми исполняющимися нитями, это регистры для хранения переменных и промежуточных результатов вычислений. На каждую нить приходится несколько десятков регистров. Точное количество зависит от того, сколько нитей исполняет минипроцессор.

И ещё одна важная черта GPU: «мягкая» векторность. Каждый минипроцессор обладает большим количеством вычислительных модулей, но все они могут выполнять только одну и ту же инструкцию, с одним программным адресом. Операнды же при этом могут быть разные, у разных нитей свои. Например, инструкция сложить содержимое двух регистров: она одновременно выполняется всеми вычислительными устройствами, но регистры берутся разные. Предполагается, что все нити GPU-программы, осуществляя параллельную обработку данных, в целом движутся параллельным курсом по коду программы. Таким образом, все вычислительные модули загружаются равномерно.

Классические задачи для GPU

Одна из самых классических задач для GPU — это задача вычисления взаимодействия N тел, создающих гравитационное поле. Также если требуется также рассчитать траектории комет и объектов пояса астероидов, это задача для GPU, так как объектов достаточно, чтобы создать необходимое количество параллельных потоков расчета.

GPU также хорошо себя проявит, если необходимо рассчитать столкновение шаровых скоплений из сотен тысяч звезд.

Ещё одна возможность использовать мощность GPU в задаче N тел появляется, когда необходимо рассчитать множество отдельных задач, пусть и с небольшим количеством тел. Например, если требуется рассчитать варианты эволюции одной системы при различных вариантах начальных скоростей. Тогда эффективно использовать GPU удастся без проблем.

Но мы будем говорить о применении GPU не для физических задач, и не для видеообработки. Поговорим о взломе паролей с помощью графических процессоров.

Кластер Джереми Госни

Многие знают, что безопасность - лишь вопрос усилий, которые требуется приложить к взлому. Все это верно для двух сторон, для защищающей и для взламывающей. На конференции Passwords^12 Джереми Госни (Jeremi Gosney) продемонстрировал кластер GPU, позволяющий взламывать пароли за считанные секунды. В нынешней версии кластера используется 14 видеокарт, но теоретически возможно распределение нагрузки до 128 GPU.

Кластер "перемалывает" 180 млрд. хэшей MD5 в секунду. Также можно просчитывать до 63 млрд. хэшей SHA1. Намного более сложные хэши Bcrypt и Sha512cryt просчитываются медленнее - до 71 000 или 364 000 комбинаций в секунду. Довольно интересна производительность кластера по взлому паролей LM и NTLM, которые используются для аутентификации под Windows. Кластер может просчитывать 384 млрд. хэшей NTLM в секунду, что позволяет взломать пароль Windows из восьми символов менее чем за шесть часов. Программное обеспечение не оптимизировано под хэши LM, поэтому они взламываются со скоростью до 20 млрд. хэшей в секунду, но даже при этом 14-символьный пароль можно взломать за шесть минут из-за слабого алгоритма.

Системы, подобные этой, не годятся для атак серверов в реальном времени, однако могут быть использованы для взлома ранее похищенных зашифрованных паролей. Госни в своё время стал известен. когда через дыру в безопасности были похищены 6,5 миллионов паролей LinkedIn и он стал первым экспертом по безопасности, который успешно обработал эти данные. Вместе с коллегой Госни смог восстановить 90% паролей.

Взлом паролей

В простейшем виде взлом паролей очень похож на обычный вход в систему. Производятся попытки угадать пароль, хешируя его и сравнивая полученный хеш с существующим. Если хеши совпали - пароль взломан. Основное отличие между взломом и обычным входом в систему состоит в том, что при взломе производятся сотни тысяч, если не миллионы таких сравнений в секунду.

В мирных целях можно использовать взлом паролей для восстановления забытых паролей от онлайн аккаунтов, также это используется системными администраторами для профилактики на регулярной основе.

Для взлома паролей в большинстве случаев используется перебор. Программное обеспечение генерирует различные варианты паролей и сообщает если был найден правильный. В некоторых случаях персональный компьютер способен выдавать миллионы вариантов в секунду. Программа для взлома пароля на пк проверяет все варианты и находит реальный пароль.

Время, необходимое для взлома пароля пропорционально длине и сложности этого пароля. Поэтому рекомендуется использовать сложные пароли, которые трудно угадать или подобрать. Также скорость перебора зависит от криптографической функции, которая применяется для генерации хэшей пароля. Поэтому для шифрования пароля лучше использовать Bcrypt, а не MD5 или SHA.

Вот основные способы перебора пароля, которые используются злоумышленниками:

* Атака по словарю - для атаки используется файл, который содержит список слов. Программа проверяет каждое из слов, чтобы найти результат;
* Атака Bruteforce - можно не использовать словарь, а перебирать все комбинации заданных символов;
* Атака с помощью радужных таблиц - в атаке используются предварительно вычисленные хэши, поэтому она быстрее.

Есть и другие методы взлома паролей, основанные на социальной инженерии, но сегодня мы остановимся только на атаках без участия пользователя. Чтобы защититься от таких атак нужно использовать только сложные пароли.

Атака по словарю

В качестве пароля очень часто выбирается какое-то слово. Программа автоматического перебора паролей проверяет слова, содержащиеся в заданном файле со словарем (существует огромное количество доступных словарей такого рода для разных языков). Словарь из двухсот тысяч слов проверяется такой программой за несколько секунд.

Многие пользователи считают, что если применить к задуманному слову некоторое простое преобразование, например, написать его задом наперед или русскими буквами в английской раскладке или намеренно сделать ошибку, то это обеспечит безопасность. На самом деле, по сравнению с подбором случайного пароля подбор пароля по словарю с применением различных преобразований (сделать первую букву заглавной, сделать все буквы заглавными, объединить два слова и т.д.) делает невыполнимую задачу вполне возможной.

Надежный пароль не должен строиться на основе слов естественного языка.

Атака Bruteforce

Самая простая (с технической точки зрения) атака на пароль – перебор всех комбинаций допустимых символов (начиная от односимвольных паролей). Современные вычислительные мощности позволяют перебрать все пароли длиной до пяти-шести символов за несколько секунд.

Некоторые системы не позволяют реализовать атаки, основанные на переборе, поскольку реагируют на несколько попыток неправильно набранного пароля подряд.

Однако существует множество систем, позволяющих бесконечный перебор. Например, к защищенному паролем файлу (архив rar или zip, документ Microsoft Office и т.д.) можно пробовать разные пароли бесконечно. Существует множество программ, которые позволяют автоматизировать эту процедуру: Advanced RAR Password Recovery, Advanced PDF Password Recovery, Advanced Office XP Password Recovery. Кроме того, многие программы хранят хэш пароля в доступном файле. Например, таким образом клиент для работы с электронной почтой (работающий на общедоступном компьютере) может хранить пароли пользователей. Существуют способы похитить файл, содержащий хэши паролей доступа к операционной системе. После этого можно заниматься подбором паролей уже в обход системы, с помощью специальных программ.

Важной характеристикой пароля, затрудняющей полный перебор, является его длина. Современный пароль должен иметь длину не менее 12 символов.

Два лишних символа в пароле увеличивают время перебора в 40000 раз, а четыре символа — уже в 1.600.000.000 раз. Однако вычислительные мощности компьютеров постоянно растут (еще несколько лет назад безопасным считался пароль длиной 8 символов).

Атака с помощью радужных таблиц

Радужная таблица - это список предварительно вычисленных хэшей (числовых значений зашифрованных паролей), используемых большинством современных систем. Таблица включает в себя хэши всех возможных комбинаций паролей для любого вида алгоритма хэширования. Время, необходимое для взлома пароля с помощью радужной таблицы, сводится к тому времени, которое требуется, чтобы найти захэшированный пароль в списке. Тем не менее, сама таблица огромна и для просмотра требует серьезных вычислительных мощностей. Также она будет бесполезна, если хэш, который она пытается найти был усложнен добавлением случайных символов к паролю до применения алгоритма хеширования.

Стоит сказать о возможности существования усложненных радужных таблиц, но они были бы настолько велики, что их было бы трудно использовать на практике. Они, скорее всего, работали бы только с набором заранее заданных "случайных величин", при этом пароль должен состоять менее чем из 12 символов, иначе размер таблицы будет непомерно велик, даже для хакеров государственного уровня.

Взлом алгоритма MD5

Одним из наиболее известных и широко используемых алгоритмов хеширования является MD5.

Алгоритм MD5 представляет собой 128-битный алгоритм хеширования. Это значит, что он вычисляет 128-битный хеш для произвольного набора данных, поступающих на его вход. Этот алгоритм разработал профессор Рональд Ривест из Массачусетского технологического института в 1991 году для замены менее надежного предшественника — MD4. Алгоритм был впервые опубликован в апреле 1992 года в RFC 1321. После этого MD5 стал использоваться для решения самых разных задач, от хеширования паролей в CMS до создания электронно-цифровых подписей и SSL-сертификатов.

О том, что алгоритм MD5 можно взломать, впервые заговорили в 1993 году. Исследователи Берт ден Боер и Антон Боссиларис показали, что в алгоритме возможны псевдоколлизии. Через три года, в 1996-м, Ганс Доббертин опубликовал статью, в которой доказал наличие коллизий и описал теоретическую возможность взлома MD5. Это был еще не взлом, но в мире начались разговоры о необходимости перехода на более надежные алгоритмы хеширования, например SHA1 (на момент написания этой статьи уже было доказано, что коллизии имеются и в этом алгоритме, поэтому рекомендую использовать SHA2) или RIPEMD-160.

Непосредственный взлом MD5 начался 1 марта 2004 года. Компания CertainKey Cryptosystems запустила проект MD5CRK — распределенную систему поиска коллизий. (Коллизией в криптографии называют два разных входных блока данных, которые для одной и той же хеш-функции дают один и тот же хеш. Каждая функция на выходе дает последовательность битов определенной длины, которая не зависит от размера первоначальных данных.) Целью проекта был поиск двух сообщений с идентичными хеш-кодами. Проект завершился 24 августа 2004 года, когда четыре независимых исследователя — Ван Сяоюнь, Фэн Дэнгуо, Лай Сюэцзя и Юй Хунбо — обнаружили уязвимость алгоритма, позволяющую найти коллизии аналитическим методом за более-менее приемлемое время. Первого марта 2005 года было продемонстрировано первое использование указанной уязвимости на практике. В том же году Властимил Клима опубликовал алгоритм, позволяющий обнаруживать коллизии на обычном ноутбуке за несколько часов. В 2006 он пошел дальше. Восемнадцатого марта 2006 года исследователь обнародовал алгоритм, находящий коллизии за одну минуту! Этот метод получил название «туннелирование». В 2008 году на конференции Chaos Communication Congress была представлена статья о методе генерации поддельных сертификатов X.509. Фактически это был первый случай реального использования коллизий в алгоритме MD5.

Большая работа была также проделана и для ускорения взлома хешей. В 2007 году Кевин Бриз представил программу, использующую Sony PlayStation3 для взлома MD5. Он сумел добиться очень неплохих результатов: 1,4 миллиарда MD5-хешей генерировались всего лишь за одну секунду! Уже через два года, в 2009-м, на BlackHat USA вышла статья об использовании GPU для поиска коллизий, что позволяло повысить его скорость в несколько раз, особенно если он выполнялся с помощью нескольких видеокарт одновременно.

Hashcat

Hashcat —применение GPU для перебора паролей. Позволяет многократно увеличить скорость перебора паролей при использовании различных алгоритмов шифрования. В программе используется распараллеленый алгоритм перебора, базированный на технологии CUDA (NVidia) и OpenCL (AMD/ATI). Программа кросс-платформенная работает на всех версиях Windows, Linux и Max OS X. Недавно ее автор перевел программу в разряд Open Source, и теперь она доступна на GitHub, так что каждый желающий может присоединиться к работе над ее новыми версиями. Hashcat предлагает множество моделей атак для получения эффективного и комплексного покрытия пространства хешей. Есть следующие режимы:

* Атака брут-форсом (перебором)
* Комбинаторная атака
* Атака по словарю
* Атака по отпечаткам
* Гибридная атака
* Атака по маске
* Перестановочная атака
* Атака основанная на правиле
* Табличная атака
* Атака с переключением раскладки

Hashcat имеет около 85 опций, 230 хеш-режимов. Особенности Hashcat

* Бесплатная
* Открытый исходный код (лицензия MIT)
* Мультиплатформенная (Linux, Windows и OSX)
* Мультиплатформенная (CPU, GPU, DSP, FPGA и т.д.)
* Одновременный взлом множества хешей
* Использует множество устройств на одной системе
* Использует устройства разных типов на одной системе
* Поддержка распределённых систем взлома (с помощью дополнительного сегмента)
* Поддержка сессий
* Поддержка восстановления
* Поддержка шестнадцатеричных солей и шестнадцатеричных наборов символов
* Поддержка автоматического порядка пространства ключей цепей Маркова
* И многое другое

В настоящее время, Hashcat объединила в себе две ранее существовавшие отдельные ветки программы. Одна так и называлась Hashcat, а вторая называлась oclHashcat (а ещё раньше oclHashcat была разделена на собственно oclHashcat и cudaHashcat). В настоящее время абсолютно все версии слиты в одну, которая при восстановлении паролей использует центральный процессор и видеокарту.

22 августа вышло обновление, включающее в себя парсер для Skype.

Рассмотрим пример.

Для примера пусть у нас будет хеш пароля, сохраненный в базе данных WordPress. Считаем, что нам известно, что админ использует цифровой пароль из 8-ми знаков. Это около 100,000,000 комбинаций. Обычные переборщики дают хешрейт около 4 тысяч в секунду. Получаем 7 часов перебора. Терпимо, но можно и лучше.

Используем hashcat gui.

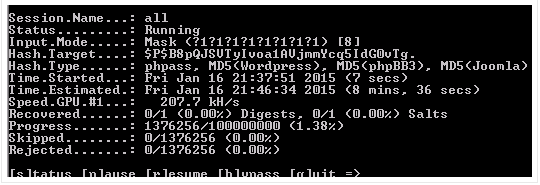


Рис.1

Поясним некоторые опции, с которыми загружается oclHashcat:

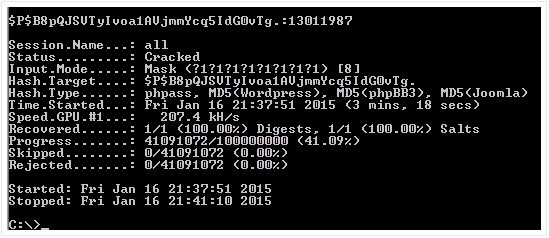
* **-a 3** – режим атаки брут-форс (перебор);
* **-m 400** – тип хеша – md5 wordpress;
* **-i --increment-min=8 –increment-max=8** – длина пароля;
* **-1 01234567890** – используемые в пароле символы (-1 – первый набор);
* **tmp8355.tmp** – временный файл, содержащий наш хеш;
* **-1 ?d ?1?1?1?1?1?1?1?1** – маска пароля. Десятичные цифры из первого набора, 8 штук;
* **-w 3 --gpu-temp-abort=80 --gpu-temp-retain=70** – режим работы с высокой доступностью пользовательского окружения, температура остановки перебора и температура догона.
* **--session=all** – определяет имя сессии (для возможного продолжения перебора);
* **--force** – игнорируем предупреждения;

Итак, выполняем команду:

  
Рис.2

Видим статусное окно, в котором показана вся интересующая нас информация. Важное, на что тут можно обратить внимание – Speed.GPU.#1 – 207.7 kH/s – скорость подбора, 207700 вариантов в секунду. Сто миллионов вариантов делим на это число, получится чуть больше восьми минут. Намного более приятный результат.

Через некоторое время:

  
Рис.3

Через 3 минуты 18 секунд пароль сдался. Статус Cracked. oclHashcat решила задачку и выдала результат: 13011987.

Однако при достаточной сложности паролей, взлом будет уже гораздо более времяемким.

Заключение

Проблемы развития GPU-вычислений

В качестве заключения можно сказать о недостатках GPU, чтобы несколько сгладить сложившееся, возможно, впечатление об их всемогуществе.

Выше уже говорилось о том, что все нити GPU-программы, осуществляя параллельную обработку данных, в целом движутся параллельным курсом по коду программы. А что будет, если нити из-за ветвлений в программе разошлись в своем пути исполнения кода? Происходит так называемая сериализация. Тогда используются не все вычислительные модули, так как нити подают на исполнение различные инструкции, а блок вычислительных модулей может исполнять, как мы уже сказали, только инструкцию с одним адресом. И, разумеется, производительность при этом падает по отношению к максимальной.

Ещё слабые стороны GPU – ограниченный объем видеопамяти: примерно в 1 ГБ для обычных GPU. Одним из главных факторов, снижающих производительность GPU-программ, является необходимость обмена данными между CPU и GPU по медленной шине, а из-за ограниченного объема памяти приходится передавать больше данных. И тут перспективной смотрится концепция AMD по совмещению GPU и CPU в одном модуле: можно пожертвовать высокой пропускной способностью графической памяти ради легкого и простого доступа к общей памяти. Вообще, общая память GPU и CPU существенно расширит область применения GPU, сделает возможным использование его вычислительных возможностей в небольших подзадачах программ.

Главным недостатком GPU является то, что в нем нет стека, где могли бы хранится параметры функций и локальные переменные. Из-за большого количества нитей для стека просто нет места на кристалле. Действительно, так как GPU одновременно выполняет порядка 10000 нитей, при размере стека одной нити в 100 КБ совокупный объем составит 1 ГБ, что равно стандартному объему всей видеопамяти. Тем более нет никакой возможности поместить стек сколько-нибудь существенного размера в самом ядре GPU. Например, если положить 1000 байт стека на нить, то только на один минипроцессор потребуется 1 МБ памяти, что почти в пять раз больше совокупного объема локальной памяти минипроцессора и памяти, отведенной на хранение регистров.

Поэтому в GPU-программе нет рекурсии, и с вызовами функций особенно не развернешься. Все функции непосредственно подставляются в код при компиляции программы. Это ограничивает область применения GPU задачами вычислительного типа. Иногда можно использовать ограниченную эмуляцию стека с использованием глобальной памяти для рекурсивных алгоритмов с известной небольшой глубиной итераций, но это нетипичное применение GPU. Для этого необходимо специально разрабатывать алгоритм, исследовать возможность его реализации без гарантии успешного ускорения по сравнению с CPU.

Итог

Вывод из всего вышесказанного можно сделать следующий: использование графических процессоров может приводить к существенному повышению производительности различных вычислений. При этом выигрыш в скорости по сравнению с использованием архитектуры с CPU может достигать нескольких десятков раз. А добиться еще большего повышения производительности можно с помощью создания кластеров GPU — в таком случае она растет практически линейно.

Однако и у GPU пока есть свои недостатки, что создает некоторые ограничения на их применение.

Источники

* <http://ashep.org/2012/giganstkij-gpu-klaster-dlya-podbora-parolej/#.WOXn3e-hqko>
* <http://rus-linux.net/MyLDP/sec/Password-Cracking-with-GPUs-1.html>
* <http://www.ixbt.com/video3/rad.shtml>
* <https://xakep.ru/2013/10/13/md5-hack/>
* <https://en.bitcoin.it/wiki/Why_a_GPU_mines_faster_than_a_CPU>
* <http://www.hardwareluxx.ru/index.php/news/hardware/grafikkarten/24187-gpu-cluster-pass-hack.pdf>
* <https://hashcat.net/hashcat/>
* <https://kali.tools/?p=578>
* <https://litl-admin.ru/utility/rabota-s-programmoj-hashcat-hashcat-gui-oclhashcat.html>