УДК 004.65

СПОСОБ МИНИМИЗАЦИИ РИСКА РАСКРЫТИЯ ПАРОЛЯ ПОЛЬЗОВАТЕЛЯ В БАЗАХ ДАННЫХ

Нортенко Дмитрий Вячеславович, Тимофеев Дмитрий Сергеевич
Государственный ВУЗ «Национальный горный університет» www.nmu.org.ua
dimanortenko@gmail.com

Рассмотрим нетипичную реализацию хранения хешей в реляционной базе данных и определим наиболее оптимальный для неё алгоритм хеширования. Особенность данной реализации заключается в значительном увеличении времени, необходимого злоумышленнику, для подбора пароля к хешу.

Ключевые слова – база даннях; пароль; хеш-функция; "соль"; коллизия.

ВСТУПЛЕНИЕ

Допустим, для повышения безопасности проводится, так называемое, "замедленное хеширования паролей" с использованием соли. Для реализации подобного метода хеширования чаще всего применяют такие адаптивные криптографические хеш-функци, как bcrypt, scrypt и PBKDF2. После получения сводок сообщений (хешей) возникает ряд вопросов. Например, что дальше делать с хешем, и где его нужно хранить.

Соль и хеш должны быть помещены в специальное надежное, энергонезависимое, зарезервированное хранилище. Обычно в роли такого хранилища выступает реляционная база данных. Какие таблицы необходимо использовать, и как их организовать – главный вопрос данной статьи.

НЕДОСТАТКИ СТАНДАРТНЫХ РЕШЕНИЙ

Чаще всего для решения поставленной задачи используют таблицу, со столбцами: USER_ID, HASH, SALT. Или же поступают еще проще, поместив HASH и SALT в основную таблицу пользователей (Users). В обоих вариантах соль находится в отношении "один к одному" с пользователями. Получив доступ к соли и хешу конкретного пользователя, злоумышленник может довольно легко найти пароль, используя перебор по словарю. Так-как преобладающая часть пользовательских паролей слабы, даже "замедленное хеширование" не может гарантировать должного уровня защиты.

Значит, необходим сбособ проверки пароля с минимальным риском раскрытия любого пароля пользователя. Не следует забывать о том, что злоумышленник, скорее всего, обладает достаточной вычислительной мощностью и статическим дампом базы данных, то есть имеет полный доступ к системе.

АЛЬТЕРНАТИВНЫЙ ВАРИАНТ

Помещаем все хеши в таблицу следующего вида:

```
CREATE TABLE [Hashes]
[HASH] [BINARY] (20) NOT NULL,
CONSTRAINT [PK_Hashes] PRIMARY KEY
NONCLUSTERED ([Hash] ASC)
```

Оператор CONSTRAINT языка SQL объявляет ограничения ссылочной целостности, накладываемые на таблицу Hashes. Следует заметить, что внешненего ключа нет. Сзвязь между хеш-кодами и пользователями перестает существовать. Теперь неизвестно, какому пользователю, какой хеш принадлежит. Соль так и остается в отношении "один к одному" с пользователями. Такая реализация хранения и проверки паролей подразумевает следующие действия.

При создании учетной записи нового пользователя или же при изменении пароля существующего, генерируем случайную, уникальную соль и сохраняем её в таблице Users. Затем хешируем пароль с этой солью, а результат заносим в таблицу Hashes. Когда пользователь попытается войти в систему, находим соль, соответствующую его имени, и проделываем знакомую операцию с введенным паролем. Если пароль был введен правильно, то полученный хеш совпадет с одним из множества архивных хешей таблицы Hashes и пользователь успешно войдет в систему. Получается вместо проверки нового хеша с хешем конкретного пользователя, будет проверяться, есть ли вообще такой хеш-код в таблице.

Для начала в таблицу с хешем можно записать несколько миллиардов случайных фиктивных значений, которые невозможно отличить от настоящих пользовательских хеш-кодов. По мере необходимости, количество фиктивных значений можно смело увеличивать до сотен миллиардов, пока не иссякнут вычислительные ресурсы системы.

Выгода от такой реализации хранения хеша очевидна. Помимо упомянутого выше отсутствия связи между хешем и пользователями, есть главное преимущество – значительное усложнение поиска пароля. Злоумышленнику придется проверять каждое значение из таблицы хеша, а для этого понадобится не только огромная процессорная мощь, но и немалый объем оперативной памяти. Если таблица полностью не поместиться в ОЗУ злоумышленника, то скорость перебора паролей значительно упадет. Таблицу с хешами можно нарастить до таких размеров, что копирование значительной ее части окажется, практически невыполнимой задачей.

Стоит отметить, что хеширование осуществляется серверами приложения, в то время, как обработка запросов исполняется базой данных. Так происходит разделение нагрузки, необходимой злоумышленнику для перебора.

КОЛЛИЗИЯ

В отличии от систем, где пароль хранится в открытом виде, во всех системах, использующих хеш-функции, есть вероятность пройти аутентификацию с неверным паролем. Для “хороших” хеш-функций частота возникновения коллизий близка к теоретическому минимуму.

Рассматриваемая система с фиктивными значениями может вызвать недоверие в связи с увеличенной вероятностью получить коллизию. Чем больше фиктивных хеш-кодов заносится в таблицу, тем больше шанс их совпадения с прохешированным неправильным паролем. И все-таки не смотря на рост вероятности коллизии, она остается на столько малой, что ею можно смело пренебречь.

Для примера возьмем устаревший, криптографически взломанный, 128-битный алгоритм хеширования MD5. Расчитаем для него вероятность коллизии в об щем случае:

$$1 - (2^{128}! / (2^{(128*n)} * (2^{128} - n)!)) = 2^{-128}$$

То есть, используя пароль с однозначным MD5 хешем, вероятность входа в систему с неправильным паролем составляет 2^{-128}

Тепер сравним с рассматриваемой схемой, где пароль хешируется алгоритмом SHA-1, а в таблицу занесено миллиард (10^{30}) фиктивных хешей. Рассчитаем вероятность коллизии:

$$2^{-160} * 2^{30} = 2^{-130}$$

Получаем, что в рассматриваемой схеме с миллиардом фиктивных хеш-кодов, вероятность возникновения коллизии в 4 раза менше, чем в первом случае.

Теперь рассмотрим увеличение вероятности коллизии с ростом количества фиктивных хешей. Допустим, таблицу с 2^{30} хешей (1Gb) расширяем в 1024 раза. Получается таблица с 2^{40} хешей (1Tb). Вероятность коллизии возросла на десять двоичных порядков с 2^{-98} до 2^{-88} Оба значения слишком малы, поэтому подобным ухудшением безопасности можно пренебречь.

ПЕРЕЧЕНЬ ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Блог (Электрон. ресурс) / Способ доступа: URL: http://www.opine.me/a-better-way-to-store-password-hashes. – A better way to store password hashes.

2. Коллективный блог (Электрон. ресурс) / Способ доступа: URL: http://web-bricks.ru/comments/about_hashes_and_passwords. – О хешах и безопасном хранении паролей

3. Новостной сайт, коллективный блог (Электрон. ресурс) / Способ доступа: URL: http://habrahabr.ru/post/100138. – Замедление хеширования паролей. Зачем?

4. Блог web-программиста (Электрон. ресурс) / Способ доступа: URL: http://ekimoff.ru/74. – md5() + соль. Хранение паролей в базе данных

5. Новостной сайт, коллективный блог (Электрон. ресурс) / Способ доступа: URL: http://habrahabr.ru/post/139974. – Немного о хэшах и безопасном хранении паролей