Создание подделок виртуальных валют на базе протокола CryptoNote посредством использования уязвимостей дерева Меркла

Ян Мачета (Jan Macheta), Саранг Ноезер (Sarang Noether) *, Шурэ Ноезер (Surae Noether)† и Хавьер Смус (Javier Smooth)
Исследовательская лаборатория Monero (Monero Research Lab)
12 сентября 2014

Аннотация

4 сентября 2014 г. на сеть криптовалюты Monero была проведена атака нового типа. В результате атаки сеть была разбита на два отдельных подмножества, каждое из которых отрицало легитимность другого. Это имело несметное количество последствий, не все из которых известны по сей день. У злоумышленника было достаточное количество времени, чтобы, например, создать подделки. В этом бюллетене описаны недостатки исходного кода CryptoNote, которые сделали проведение этой атаки возможным, а также решение, изначально предложенное Рафалем Фриманом (Rafal Freeman) из Tigusoft.pl, а впоследствии и командой CryptoNote, описаны исправления, внесённые к данному моменту в кодовую базу Monero. Помимо этого, нами подробно рассматривается результат использования защитного блока в сети. Данный исследовательский бюллетень не проходил независимого анализа и отражает исключительно результаты внутреннего исследования.

1 Вступление

4 сентября 2014 г. на сеть криптовалюты Monero была проведена атака нового типа, ставшая причиной невиданного до сих пор феномена. Человек, стоявший за атакой, должен был хорошо знать код Monero, а также деревья Меркла, должен был обладать базовыми знаниями в области криптографических хеш-функций.

Код Monero появился как форк исходного кода Bytecoin в апреле 2014 г. до того, как был выпущен исходный код CryptoNote. Код Bytecoin оказался несколько размытым и плохо прокомментированным, поэтому каждый кусок кода рассматривался с определённой долей скептицизма. Поэтому любой злоумышленник, знающий код Monero, предположительно, также точно мог знать исходные коды Bytecoin и CryptoNote.

Дерево Меркла является структурой данных, в которой каждый узел, не являющийся «листом» дерева, получает метку, и эта метка является хешем дочерних узлов[3]. Чтобы построить дерево Меркла, нами были взяты некоторые блоки данных (транзакции), которые мы хешировали. Затем мы хешировали полученные блоки по два за раз. И мы повторяли этот процесс, пока не достигли желаемого. Дерево Меркла можно рассматривать в качестве футбольной турнирной таблицы, в которую вписываются криптографические хеши. Возникает естественный вопрос: а что ели у нас не будет точного, чётного, кратного двум, делимого количества транзакций? Как мы увидим, в этом и кроется уязвимость.

*sarang.noether@protonmail.com
†surae.noether@protonmail.com

Криптовалюты на базе протокола CryptoNote, как в принципе и большинство криптовалют, используют дерево Меркла для построения хеша блока транзакций (который впоследствии заносится в заголовок блока). Злоумышленником было отмечено, что неверная реализация распространённого алгоритма округления до количества кратного двум может быть использована при вычислении хеша блока с целью генерирования двух отдельных блоков с одним и тем же хешем. Всё это, безусловно, должно быть невозможным, так как конфликт криптографических хеш-функций является очень редким явлением. Если говорить о том, насколько редко такое происходит, то в данном случае будет уместно сравнение с тем, «сколько элементарных частиц существует во вселенной», и обычно такие числа недостаточны для описания того, насколько редко происходят подобные конфликты. В целом конфликты хешей подразумевают ошибку в реализации, а не сбой хеш-функций, если предположить, что мы используем правильную хеш-функцию [2].

Насколько известно авторам, такая атака произошла лишь единожды. Несмотря на то, что целью стала Monero, любая монета, использовавшая исходный код CryptoNote до 4 сентября 2014, может пострадать в результате неправильной реализации алгоритма, в результате которого будет возможно провести атаку. Есть лишь два исключения. У Fantomcoin и Moneta Verde данная уязвимость была тайно устранена несколько месяцев назад. Опять же, насколько известно авторам, все популярные монеты на базе протокола CryptoNote реализовали изменения, описанные Р. Фриманом и командой CryptoNote, хотя мы и не проводили тщательного анализа их кода. Цель настоящего документа состоит в описании той части кода, которая сделала проведение атаки возможным, а также в рассмотрении эффектов атаки. Немного иной подход к рассмотрению атаки содержится в работе [1], например.

2 Уязвимость исходного кода CryptoNote

В этом разделе мы немного вольно используем терминологию, когда говорим об *исходном коде CryptoNote*, так как форк Monero от Bytecoin произошёл ещё до того, как появился исходный код CryptoNote. Но мы не думаем, что это как-то может запутать читателя. Уязвимость исходного кода CryptoNote можно понять, используя аналогию с футбольной турнирной таблицей, которая была приведена нами выше: что если данные, которые нам необходимо хешировать, не будут чётными, кратными двум? Мы просто берём всё, что *выходит за пределы* кратного двум, и играем в рамках меньшей турнирной таблицы первого круга, где все остальные получают *пустой*[‡] номер в следующем круге. Безусловно, это просто помогает перейти к следующему меньшему кратному двум, но в конечном счёте этот процесс закончится отбором двух команд, и мы продолжим работу с нашей турнирной таблицей в нормальном режиме. Для того чтобы реализовать такую стратегию, для начала нам необходимо определить, сколько вообще команд примут участие.

Конечно же, чтобы сделать это, нам необходимо округлить *размер, длину или меру* текущего набора данных до ближайшего значения кратного двум, но обязательно меньшего, чем текущий размер данных. Все данные *выше* этого значения (в значениях индексов) должны быть включены в данные ниже этого значения при помощи криптографических хеш-функций. В этом и кроется уязвимость. Разделяющий индекс, служащий для отделения *выше* от *ниже*, был вычислен неверно. Злоумышленник заметил, что это позволит игнорировать некоторые данные при вычислении хеша блока, а также что два отдельных блока могут получить один и тот же хеш. Это произошло не из-за того, что хеш-функции использовались как-то не так, а, как было описано выше, из-за неправильного измерения размера «реальных» данных, которые использовались для построения дерева Меркла.

Злоумышленником была использована следующая часть кода из `src/crypto/tree-hash.c`:

[‡]Не байт.

src/crypto/tree-hash.c:

```
46  size_t cnt = count - 1;
48  for (i = 1; i < sizeof(size_t); i <<= 1) {
49          cnt |= cnt >> i;
50  }
51  cnt &= ~(cnt >> 1);
```

Задача этого кода, версии широко используемого алгоритма, состоит в округлении значения `count` до самого большого значения кратного двум, которое *строго* должно быть меньше значения `count`, и установки этого значения как `cnt`[4].

Input	Output
100	64
255	128
512	256
513	512
1205	1024

Чтобы понять, как работает этот сегмент кода, мы сначала рассмотрим лёгкий пример, а затем сравним результаты с тем, как на самом деле работает алгоритм. Предположим, мы хотим определить самое большое значение кратное двум, которое *строго* меньше целого числа 1205. Мы определяем x как двоичное число, полученное уменьшением 1205 и присваиваем каждому биту справа от самого старшего бита значение 1:

$$1204_{10} = 10010110100_2 \Rightarrow x = 11111111111_2$$

Затем определяем y как двоичное число, полученное путём одного сдвига вправо по x получением его отрицательного значения:

$$y = \overline{x \gg 1} = \overline{01111111111}_2 = 10000000000_2$$

В результате, используя операцию логического умножения *И*, получаем xy:

$$xy = 11111111111_2 \cdot 10000000000_2 = 10000000000_2 = 1024_{10}$$

Теперь сравним этот простой пример с алгоритмом применительно к вышеуказанному блоку кода из `src/crypto/tree-hash.c`. Предположим, что действительно так для большинства современных архитектур, что `sizeof(size_t) = 8`. В этом случае мы имеем $i \in \{1, 2, 4\}$, поэтому цикл повторяется три раза. Рассмотрим работу кода из `src/crypto/tree-hash.c` с использованием данных выборки $\texttt{count} = 513_{10}$:

$$\begin{aligned}
\texttt{cnt} &= 513 - 1 = 512_{10} = 1000000000_2 \\
i = 1 : \texttt{cnt} &= 1000000000_2 + 0100000000_2 = 1100000000_2 \\
i = 2 : \texttt{cnt} &= 1100000000_2 + 0011000000_2 = 1111000000_2 \\
i = 4 : \texttt{cnt} &= 1111000000_2 + 0000111100_2 = 1111111100_2
\end{aligned}$$

Заметьте, ввиду того, что цикл повторялся всего три раза, мы не стали изменять состояние двух самых младших битов. Алгоритм работает дальше:

$$\texttt{cnt} = 1111111100_2 \cdot 100000001_2 = 1000000000_2 = 512_{10}$$

Мы получаем правильный результат. Предположим, однако, что вместо этого мы будем использовать данные выборки $\texttt{count} = 514_{10}$:

$$\begin{aligned} \texttt{cnt} &= 514 - 1 = 513_{10} = 1000000001_2 \\ i = 1 : \texttt{cnt} &= 1000000001_2 + 0100000000_2 = 1100000001_2 \\ i = 2 : \texttt{cnt} &= 1100000001_2 + 0011000000_2 = 1111000001_2 \\ i = 4 : \texttt{cnt} &= 1111000001_2 + 0000111100_2 = 1111111101_2 \end{aligned}$$

Как и до этого последние два бита остаются без изменений. Алгоритм работает дальше:

$$\texttt{cnt} = 1111111101_2 \cdot 1000000001_2 = 1000000001_2 = 513_{10}$$

Это очевидно неправильно. Как мы и ожидали, алгоритм выдал целое число 512. Эта неправильная реализация ранее описанного алгоритма округления стала причиной появления уязвимости в исходном коде CryptoNote. Давайте рассмотрим последствия неправильной реализации такого округления на примере кода, приведённого ниже:

src/crypto/tree-hash.c:

```
47  char (*ints)[HASH_SIZE];
52  ints = alloca(cnt * HASH_SIZE);
53  memcpy(ints, hashes, (2 * cnt - count) * HASH_SIZE);
54  for (i = 2 * cnt - count, j = 2 * cnt - count; j < cnt; i += 2, ++j) {
55    cn_fast_hash(hashes[i], 64, ints[j]);
56  }
57  assert(i == count);
58  while (cnt > 2) {
59    cnt >>= 1;
60    for (i = 0, j = 0; j < cnt; i += 2, ++j) {
61      cn_fast_hash(ints[i], 64, ints[j]);
62    }
63  }
64  cn_fast_hash(ints[0], 64, root_hash);
```

В этом блоке кода строки с 47 по 56 отвечают за выполнение следующих задач. Мы распределяем последовательность хешей по `ints`. Мы берём то, что, *как нам кажется* , является данными оригинальных **хешей**, которые пока не вышли за пределы необходимой нам структуры кратности двум, и мы просто используем `memcpy`, чтобы скопировать данные в `ints`. Тем не менее величина $2 \cdot \texttt{cnt} - \texttt{count}$, как было описано ранее, является завышенной оценкой количества данных. Например, известная уязвимость, описанная выше, округлила бы 514 до 513, а не до 512. Следовательно, устанавливая $\texttt{cnt} = 513$ и $\texttt{count} = 514$, мы видим, что мы копируем элементы $2 \cdot \texttt{cnt} - \texttt{count} = 512$, заполняя всю последовательность, которая используется в последующих кругах хеширования. Тем не менее нам следовало вычислить $\texttt{cnt} = 512$, скопировав набор данных $2 \cdot \texttt{cnt} - \texttt{count} = 510$. Таким образом, баг, о котором говорилось ранее, приводит к тому, что появляются два набора данных, которые полностью игнорируют друг друга. К слову, первые хеши $2 \cdot \texttt{cnt} - \texttt{count}$ должны получать *«пустой номер»*, если следовать спортивной аналогии, и переходить в следующий круг, не принимая участия в игре (или, скорее, должны быть

хешированы другими данными). Так как значение `cnt` было переоценено, слишком много данных прошло дальше без надлежащей обработки, оставив остальные данные неиспользованными. Безусловно, мы по-прежнему хешируем оставшиеся данные вместе, но эти хеши не используются.

Следует отметить невероятный уровень технической подготовки, необходимой для обнаружения и использования этого бага. Кто-то, совершивший эту атаку, был очень и очень умным, настолько умным, что не только нашёл, но и использовал этот глубоко скрытый баг. Ему был хорошо известен алгоритм, и это явно было сделано со злым умыслом.

3 Внесённые исправления

4 сентября 2014 г., в день атаки, Рафаль Фриман из `Tigusoft.pl`[§] сообщил о первом решении для нейтрализации уязвимости. Решение не было уникальным, так как в исходном коде CryptoNote было реализовано другое решение[¶]. Решение CryptoNote предполагало «быстрое исправление» в `src/crypto/tree-hash.c` при помощи завершающего условия `8·sizeof(size_t)`, в результате чего цикл стал проходить через все биты в `cnt`. Тем не менее без адекватной проверки размера `cnt` можно выбрать достаточно большое значение, чтобы провести ту же самую атаку. Кроме того, использование системного кода попросту является критикуемой практикой в области вычислительной техники и криптографических протоколов. И, опять же, как и в случае с обфускацией кода, считается хорошей манерой комментирование отдельных участков вашего кода, и по обоим пунктам команду CryptoNote можно признать виновной.

Кодовая база Monero, напротив, устанавливает проверку размера `cnt` и использует следующий алгоритм округления:

`src/crypto/tree-hash.c:`

```
47  assert( count >= 3); // cases for 0,1,2 are handled elsewhere
49  size_t tmp = count - 1;
50  size_t jj = 1;
51  for (jj=1 ; tmp != 0 ; ++jj) {
52    tmp /= 2;
53  }
54  size_t cnt = 1 << (jj-2);
56  assert( cnt > 0 );   assert( cnt >= count/2 ); assert( cnt <= count );
57  assert( ispowerof2_size_t( cnt ));
58  return cnt;
```

Алгоритм определяет количество значений кратных двум и меньше `cnt` и производит соответствующий сдвиг битов, чтобы восстановить правильное значение кратное двум. Этот алгоритм не страдает тем же ограничением размера, что решение, предложенное командой CryptoNote.

4 Последствия атаки

Когда практически одновременно было опубликовано два блока с одинаковым хешем, один блок остался в одной части сети, а другой блок — в остальной её части (при этом узлы, ещё не получившие блок,

[§]См. коммит в репозитории Monero `2ef0aee81d20c002ed50d6dec4baceee1ac40b44`

[¶]См. коммит в репозитории CryptoNote `6be8153a8bddf7be43aca1efb829ba719409787a`

игнорировались). Простая проверка хешей транзакций в другой половине сети могла вызвать ошибку. Это разбило сеть на две отдельных части, которые не признавали легитимность друг друга. Какое-то время сеть была разделена надвое. Читатель может попытаться назвать это форком блокчейна. Но будьте бдительны. Называть это форком было бы неверно. На самом деле, форк происходит, когда две конкурирующих цепочки транзакций в дереве блоков (обе при этом являются легитимными) конкурируют за хешрейт сети. В конечном счёте одна из них побеждает, так как сеть использует метод выбора «самого длинного блокчейна», предложенный Сатоши Накомото (Satoshi Nakomoto). Кроме того, несмотря на всю теоретическую сложность, вполне возможно, что со временем сеть переключится с одного блокчейна на другой, если он станет достаточно длинным.

Две части сети отказывались признавать легитимность блокчейна, с которым работала другая часть. Это было, как если бы внезапно сеть Monero переключилась на абсолютно новый блокчейн монеты на базе протокола CryptoNote. Оказалось, что, например, криптовалютные биржи или коммерсанты, все работали в одной сети с Monero.

Это имело несметное количество последствий, не все из которых известны по сей день. Например, баланс многих пользователей удвоился. Если у вас была 1.0 XMR до проведения атаки, то теперь у вас было по 1.0 XMR в каждой версии сети. Кто-то может усмотреть в этом сценарий поделки монеты: проводящий атаку опубликовал два блока с одним и тем же хешем, и теперь вместо одной сети стало две, а злоумышленник удвоил свой баланс. У него было некоторое время, чтобы продать «подделку» Monero из новой сети на бирже вроде Poloniex за Bitcoin и немедленно вывести средства. Безусловно, при этом у злоумышленника сохранялся его настоящий баланс в «старой» сети. За исключением MintPal, большинство ведущих бирж, торгующих Monero, смогли заморозить транзакции во время проведения атаки.

По неизвестным до сих пор причинам большинство ведущих майнинговых пулов Monero закончили с одной стороны сети; это могло стать просто результатом демографической стохастичности после разделения сети надвое, или же это могло быть детерминированным результатом скорости, с которой проходили блоки. На самом деле, если два блока с одинаковым хешем одновременно попадают в сеть, и если один блок находится большим пулом до того, как будет найден другой из-за стохастичности, то, вероятнее всего, именно этот блок и будет признан большинством сети, а не другой. С другой стороны, если два блока с одинаковым хешем попадают в есть *практически* одновременно, но случайность исключается, первый блок станет доминантным.

Как бы то ни было, меньшая часть сети, конечно же, имела значительно более низкую мощность хеширования и начала выдавать ужасные ошибки. Всё усложнялось также и тем, что было проблемой ещё до разделения. Так как протокол CryptoNote игнорирует сильно отличающиеся значения в блоке при вычислении сложности, то у сети B уйдет от примерно трёх дней до недели, чтобы скорректировать сложность для компенсации. Прибыльность любого блока, застрявшего со стороны B, сильно упадает. Это в совокупности с потоком ошибок привело к рассинхронизации большинства узлов с их блокчейном, к тому, что пришлось перезагружать компьютеры, и подобным вещам. В этот момент стороне B ничего не оставалось, кроме как исчезнуть, но последствия атаки на блоке 202612 до сих пор находят отклик в сети Monero. 6 сентября 2014 г. командой разработчиков Monero был выпущен патч, гарантирующий, что подобная атака никогда более не будет проведена, и майнеры смогут идентифицировать сторону сети A (то есть версию блока 202612 со стороны A), а также идентифицировать сторонние узлы с версией блока 202612 со стороны B.

Вполне возможно, что есть и другие, пока ещё неизвестные последствия сохранения блока 202612 в блокчейне. Также, возможно, есть другие, пока ещё неизвестные последствия использования транзакций, действительность которых была подтверждена в этом блоке в качестве ложных элементов при

создании новых кольцевых подписей, если владельцы этих транзакций (предположительно, являющиеся злоумышленниками) продолжат тратить эти транзакций с нулевыми миксинами.

Список литературы

[1] Werner Albert. *Monero Network Exploit Post-Mortem (Анализ уязвимости сети Monero)*, 2014 (accessed Sept 15, 2014). `https://forum.cryptonote.org/viewtopic.php?f=7&t=270`.

[2] Ross Anderson. The classification of hash functions (Классификация хеш-функций). 1993.

[3] Ralph C Merkle. A digital signature based on a conventional encryption function (Цифровая подпись на базе традиционной функции шифрования). In *Advances in Cryptology—CRYPTO'87*, pages 369–378. Springer, 1988.

[4] John Viega and Matt Messier. *Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Input Validation & More (Поваренная книга для безопасного программирования на C и C++: рецепты для криптографии, аутентификации, валидации входных данных и многое другое).* O'Reilly Media, Inc., 2003.