Machine Translated by Google

Анализ контрактов Broxus TON-DEX

Подготовлено Прувендо

2021/10/26

Управляющее резюме

В ходе этого проекта, согласно замыслу авторов проекта, мы выполнили
аудит смарт-контрактов Solidity, разработанных в рамках децентрализованной биржи токенов
(ДЭКС). Документ предназначен для использования авторами договоров по их собственному желанию для публичного или
частные цели.

В ходе аудита серьезных уязвимостей в экосистеме смарт-контрактов Broxus не выявлено.
были найдены. Несколько незначительных проблем, которые не ставят под угрозу экосистему смарт-контрактов Broxus
но по-прежнему требуют улучшения, перечислены в этом документе в следующих разделах.

Machine Translated by Google

Machine Translated by Google

Источник данных

Исходный код смарт-контрактов доступен на Github по адресу:
https://github.com/broxus/ton-dex/tree/master/contracts Хэш фиксации 02943f5f05d280be795427f1adf01ab5f466616e сделан 11.08.2021.

Однако на момент подачи этого документа было сделано два новых коммита.

Мотивация

Создание экосистемы DEX — одна из важных целей сети Free TON. Здесь мы предоставляем список публичных мероприятий, организованных в виде конкурсов субуправления DeFi (https://defi.gov.freeton.org) . : 1. #6 Архитектура и дизайн FreeTon DEX a. Продолжительность: 28 октября - 15 ноября 2020 г., 23:59 UTC b. 10 положительно оцененных представлений c. Победитель (7,66 балла): «Книга заказов Dex Design and Architecture» на

https://firebasestorage.googleapis.com/v0/b/ton-labs.appspot.com/o/documents%2Fapplication%2Fpdf%2Fin0jwkdweijkhjr6xdb-Orderbook%20Dex%20Design%20and%20Architecture.pdf?alt=media&token=84824133-7817-49ca-abd7-a4122ee8a900 2. Конкурс №16 по внедрению FreeTon DEX, этап 1

a. Продолжительность: 27 января - 20 марта 2021 г., 23:59 UTC b. 7 сабмитов с положительной оценкой c. Победитель (8,45 балла): в

https://firebasestorage.googleapis.com/v0/b/ton-labs.appspot.com/o/documents%2Fapplication%2Fpdf%2Fr5arcsbzehkm9nixl7-FreeDEX%20contest%20_%20SVOI%20submission%20_%20TonSwap.pdf?alt = медиа и токен = 4f1984d3-1c02-4c06-8c42-bd5d983af090

3. №23 FreeTon DEX, этап 2 внедрения a. 9 мая - 18 июня 2021 г., 23:59 UTC b. 4 сдачи с положительной оценкой c. Победитель (8,12 балла) на

https://firebasestorage.googleapis.com/v0/b/ton-labs.appspot.com/o/documents%2Fapplication%2Fpdf%2Fjqyuwqjmvqgkq2ky5ej-II%20FreeDEX%20contest%20_%20SVOI%20submission%20_%20TonSwap.pdf?alt=media&token=d64e7847-24eb-4022-a722-99108c3751e5 4. #38 Внедрение FreeTON DEX: Этап 3

Machine Translated by Google

а. 4 августа - 27 сентября 2021 г., 23:59 UTC b. 5
представлений с. Идет голосование

Контракты Broxus являются частью этой деятельности и имеют самую зрелую систему, уже работающую в производстве, см. информацию в Интернете:

- https://broxus.medium.com/ - https://broxus.com/ - https://hub.forklog.com/broxus-zapustila-ton-swap-2-0-chto-novogo-v-dex-na-blokchejne-svobodnaya-tonna/ (на русском)
- https://www.facebook.com/4BitcoinAddicts/posts/broxus-a-leading-developer-of-the-freet на-проекте-представил-вторую-версию/3064450753801402/
- https://twitter.com/broxus

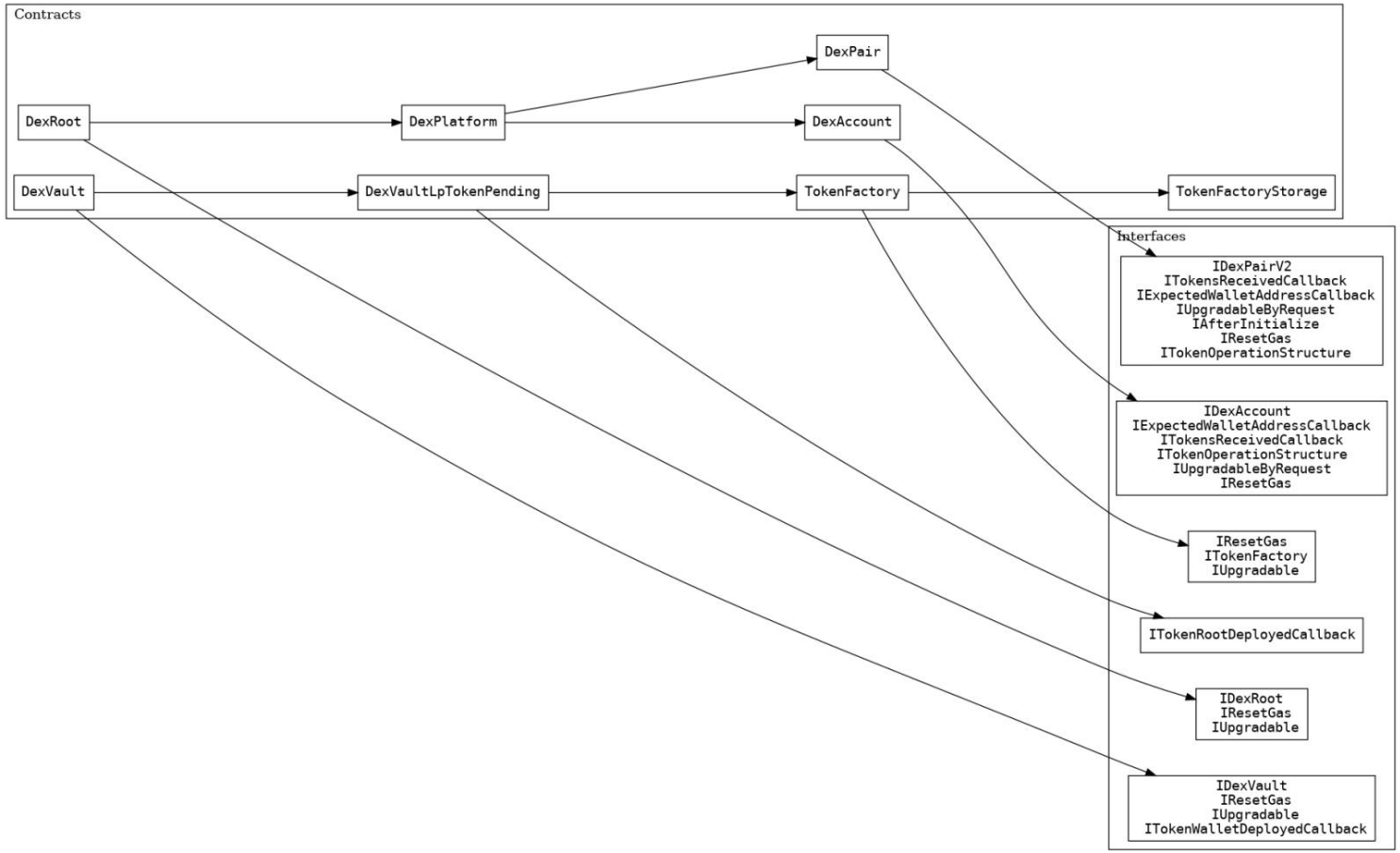
Поскольку основной целью бирж является манипулирование пользовательскими токенами, анализ таких контрактов требует расследования, чтобы снизить риск сбоя и предотвратить возможные атаки.

Структура программы

Этот раздел содержит описание иерархии контрактов, график вызовов (см. отдельный файл) одной из важных траекторий вызовов (развертывание токена поставщика ликвидности).

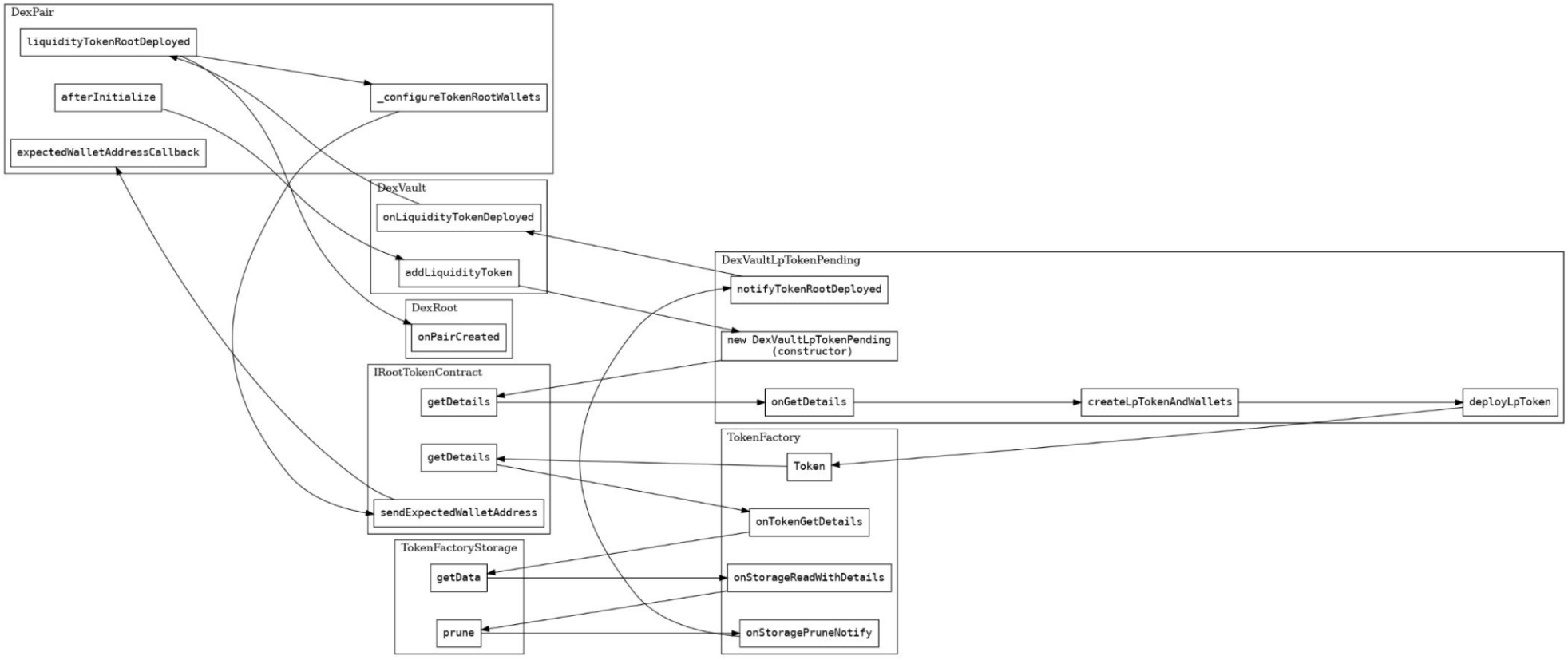
Machine Translated by Google

Структура контрактов с интерфейсами и последовательностью развертывания



Machine Translated by Google

Траектория развертывания токена LP DexPair -> ...-> DexPair (с обратными вызовами)



Machine Translated by Google

Список проблем

Объяснение

Контракты были проанализированы с точки зрения логических ошибок, тогда как каждая функция была проанализирована с точки зрения удобочитаемости, адекватных имен переменных и функциональной инкапсуляции. Также был проведен поиск дублирующегося кода[1] для предотвращения нарушения принципа DRY. алгоритма для осознания соответствия конечного алгоритма и потенциальной цели использовался здравый смысл. Однако, когда здравый смысл противоречит коду контракта, мы не можем гарантировать, что это ошибка, мы отмечаем это место, чтобы привлечь больше внимания разработчиков.

Большинство функций хорошо читаются, имена переменных говорят сами за себя, логику контракта можно извлечь из его кода при наличии общих знаний языка программирования Solidity и предмета области использования контракта.

Мы также следуем следующему принципу: лучше обратить внимание на несуществующее, чем пропустить что-то важное. Только автор договора может судить о реальной серьезности обнаруженных проблем.

Общий:

1. Поскольку ответственность является относительно новой функцией, ее использование для геттеров обычно не рекомендуется, и некоторые члены сообщества предлагают использовать обратные вызовы.
2. Использование флага IGNORE_ERRORS при отправке сообщения снижает строгость условий контракта. Если предполагается, что могут возникать ошибки, мы предлагаем предотвратить их, используя соответствующие запросы.
3. Предложите использовать побитовые операторы типа || для настроек флаги вместо арифметического сложения 4. Предложить не использовать флаг 2 в tvm.rawReserve, это снижает детерминированность поведения контракта, так как резервирует минимум от баланса и заданного аргумента, и например в случае, когда мы резервируем весь баланс weCopy of Аудит контракта Broxus DEXКопия аудита контракта Broxus DEXё недостаточно средств для выполнения функции
5. buildPairParams, buildInitParams, buildAccountParams в каждом контракте содержат много дублированного кода (на самом деле они буквально идентичны). Мы предлагаем использовать отдельную библиотеку или общего предка
    a. (tvm.hash(_buildInitData(

[1] https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

Machine Translated by Google

PlatformTypes.Account,

_buildAccountParams(account_owner)))) - предлагаю производить расчет адресов в отдельной функции, т.к. она имеет дубликаты DexRoot:

6. deployAccount: при развертывании учетной записи не учитывается адрес отправителя, а только данные, передаваемые в качестве аргументов. Предложите сделать больше проверок здесь.
7. deployAccount: улучшить проверку достаточного баланса (только V1) 8. install*:
предложить проверить, достаточно ли баланса 9. Функция forceUpgradeAccount
слишком жесткая и позволяет владельцу root выполнять
обновление, не позволяя пользователю выбирать и приводит к потере данных
10. setVaultOnce: не проверять, что new_vault не равно нулю
11. requestUpgradeAccount: не генерирует события, однако аналогичные функции (forceUpgradeAccount, upgradePair) делают DexAccount:

12. getWalletData: предложить использовать необязательный тип для
возврата 13. addPair: генерируемое событие соответствует начальному порядку валют (левый и правый корни), а не порядку, в котором они хранятся (который сортируется по адресу) 14. ожидаемыйWalletAddressCallback
принимает сообщение value и не возвращает его обратно в случае ошибки (или отправляет на send_gas_to)

15. tokensReceivedCallback: if(_balances.exists(token_root)) { _balances[token_root] += tokens_amount; } else
{ _balances[token_root] = tokens_amount;

} немного избыточный код , «+=» будет работать в обоих случаях

16. DexAccount.depositLiquidity:
выдать DepositLiquidity(left_root, left_amount, right_root, right_amount, auto_change); DexPair.depositLiquidity: выдать DepositLiquidity(left_amount, right_amount, lp_tokens_amount); События с одинаковыми именами и разными аргументами

ДексВалт:

17. конструктор: слишком публичная функция, потенциально может привести ко множеству подобных хранилищ DexPair:

18. exchange: слишком дублированный код в ветках, предложите рефакторинг 19.
setFeeParams (onlyRoot): root на самом деле не вызывает эту функцию

Machine Translated by Google

a. предложить также проверки: знаменатель > 0, числитель <= знаменатель (?)

20. afterInitialize: если DexPair не полностью развернут в случае неудачного развертывания токена Lp, единственный способ заставить его работать — это вызывать deployPair снова и снова, что нелогично, поскольку сама пара фактически развернута (но Lp жетон).
    a. Если что-то случится с Pair (повреждение по неизвестной причине), можно повторно развернуть ту же рабочую пару с помощью вызова upgrade, а после этого вызвать deployPair для инициализации корня Lp, если он равен нулю (это тоже немного сбивает с толку, однако это не ошибка)
21. tokensReceivedCallback: использует небольшие значения TON 10, 30, 44 (нанотон) для определения поведения. Предложите использовать более понятный способ. 22. DepositLiquidity: require(lp_supply != 0 || (left_amount > 0 && right_amount > 0), DexErrors.WRONG_LIQUIDITY); require((left_amount > 0 && right_amount > 0) || (auto_change && (left_amount + right_amount > 0)), DexErrors.AMOUNT_TOO_LOW);

может быть реорганизован в

```
require((left_amount > 0 && right_amount > 0) || ((lp_supply != 0)&&(auto_change && (left_amount + right_amount > 0)), Error);
```

Однако это уменьшает количество ошибок 23. DepositLiquidity: Требование lp_supply != 0 кажется технически разумным, но предлагает использовать более экономически обоснованные значения в качестве нижней границы (на самом деле мы не видим здесь хорошего решения, поэтому мы можем рассмотреть это в будущее эволюции контрактов)

Ожидание DexVaultLpToken:

24. конструктор: предлагается использовать строгое назначение pending_messages = 2, а не += для дать больше гарантий присваиванию ценности a.
    deploy_value не используется
25. Переменные для создания символа LP могут быть константами, а не переменными:

```
строка LP_TOKEN_SYMBOL_PREFIX = "TONSWAP-LP-"; строка LP_TOKEN_SYMBOL_SEPARATOR = "-"; uint8 LP_TOKEN_DECIMALS = 9;
```

TokenFactory: 26. Применение из onStorageReadWithDetails: TokenFactoryStorage(msg.sender).prune обратные вызовы (onStoragePruneNotify,

Machine Translated by Google

onStoragePruneReturn) кажутся избыточными, обратный вызов getData (onStorageReadWithDetails) содержит все необходимые параметры для определения правильной ветки в функции, и требования совпадают. Предложить рефакторинг

Квадратный корень и решение уравнений

Начнем с квадратного корня.

Используемый метод - это вавилонский метод нахождения квадратного корня, адаптированный для целочисленных аргументов. Алгоритм следующий: функция _sqrt(uint256 x) private pure возвращает

```
(uint256) {
        если (x == 0) вернуть 0;
        иначе если (x <= 3) вернуть 1;
        uint256 z = (x + 1) / 2;
        uint256 y = x;
        в то время как (r < y)
        {
                y = r;
                r = (x / r + r) / 2;
        }
        вернуть y;
    }
```

Это обычно называют методом Ньютона (а также методом Герона). Известно, что сходимость ньютоновских методов сильно зависит от начального приближения. Чем меньше разница между реальным квадратным корнем и начальным значением, тем быстрее сходится метод. В контракте используется начальное значение (x + 1)/2. Однако можно найти и более близкий к настоящему корневому вариант.

Machine Translated by Google

Чтобы мотивировать результаты, давайте использовать 2 ^ [log2 X/2] в качестве начального значения.

x = 12345678 23,5575027 = log2 -> (23+1)/2->12->2^12 = 4096
1. 6172839 4096
2. 3086420 3555
3. 1543211 3513
4. 771609 3513
5. 385812 3513
6. 192921 3513
7. 96492 3513
8. 48309 3513
9. 24282 3513
10. 12395 3513
11. 6695 3513
12. 4269 3513
13. 3580 3513
14. 3514 3513
15. 3513 3513
16. 3513 3513

Мы видим, что если использовать половину аргумента (первый столбец), сходимость достигается на 16-й шаг, а если начать с 2^(([log2 (X)]+1)/2), то же самое достигается на 4-м. [x] здесь означает, что функция пола возвращает наибольшее целое число, меньшее аргумента. Значение [log2 (X)]+1 — это просто количество двоичных цифр в числе. В TBM есть инструкция

- **B602** — **BITSIZE** $(x - c)$, computes smallest $c \geq 0$ such that x fits into a c-bit signed integer $(-2^{c-1} \leq c < 2^{c-1})$.

который возвращает это значение. Однако мы знаем, что в какой-то момент стандартной библиотеки не существует. функция, которая делает то же самое. Также возможно найти это значение с помощью процедуры дихотомии используя правый битовый сдвиг со сложностью O (log (log (X))).

Machine Translated by Google

Чтобы мотивировать больше, мы берем большее число, чтобы продемонстрировать конвергенцию.

x = 12 345 678 912 345 600	log2= 53,45485569
1. 6 172 839 456 172 800	134217728
2. 3 086 419 728 086 400	113100101
3. 1 543 209 864 043 200	111128599
4. 771 604 932 021 604	111111112
5. 385 802 466 010 810	111111110
6. 192 901 233 005 421	111111110
7. 96 450 616 502 742	
8. 48 225 308 251 434	
9. 24 112 654 125 844	
10. 12 056 327 063 177	
11. 6 028 163 532 100	
12. 3 014 081 767 073	
13. 1 507 040 885 584	
14. 753 520 446 887	
15. 376 760 231 635	
16. 188 380 132 201	
17. 94190098868	
18. 47095114969	
19. 23547688556	
20. 11774106420	
21. 5887577482	
22. 2944837192	
23. 1474514752	
24. 741443729	
25. 379047296	
26. 205808791	
27. 132897475	
28. 112896869	
29. 111125233	
30. 111111111	
31. 111111110	
32. 111111110	

В первом случае алгоритм сходится на 32-м шаге, а вариант на базе log2 — на 6-м. Поэтому мы предлагаем улучшить начальное приближение, чтобы значительно ускорить сходимость.

Machine Translated by Google

Решение квадратного уравнения

```
        функция _solveQuadraticEquationPQ (uint256 p, uint256 q) частная чистая
возвращает (uint128) {
                uint256 D = math.muldiv(p, p, 4) + q;
                uint256 Dsqrt = _sqrt(D);
                если (Dsqrt > (p/2)) {
                        вернуть uint128 (Dsqrt - (p/2));
                } иначе {
                        вернуть uint128((p/2) - Dsqrt);
                }
        }
```

1. uint256 D = math.muldiv(p, p, 4) + q; as (p:uint256) p^4+q может переполнять uint256 размера 2. Поскольку (D:uint256) sqrt(D) должен соответствовать uint128 размера 3. если sqrt работает правильно, Dsqrt > (p/2) всегда вторая ветвь не должно существовать

    a. Если он есть, то это намекает на некорректно работающий sqrt, и нет никаких причин для присваивать возвращаемому значению что угодно, кроме 0.

4. В любом случае значение uint128((p/2) - Dsqrt) не является осмысленным значением (второй корень равно -(p/2) + Dsqrt, что отрицательно) 5.

Функция _sqrt может возвращать значение uint128 вместо uint256.

Machine Translated by Google

Заключение

На данный момент мы не обнаружили серьезных уязвимостей, которые могли бы поставить под угрозу средства клиента, задействованные при работе с Broxus. Система кажется надежной, пригодной для использования в экосистеме Free TON и основана на широко известном Uniswap AMM.

Плюсы: 1. Контракты хорошо структурированы 2.
Названия переменных в большинстве случаев говорят сами за себя
3. События отправляются, и они удовлетворительны 4. Все функции
тщательно проверяют баланс перед переводами и резервируют минимальный баланс контракта для
предотвращение недополнения 5. Все функции корректно проверяют права доступа 6. Почти все константы именованы и говорят сами за себя (есть специальная библиотека с константами)

7. Геттеры готовы выполнить взаимодействие по контракту 8.
Мы считаем, что нет необходимости перепроектировать архитектуру
9. Мы не обнаружили ни одной проблемы, которую мы могли бы решить как критическую или основную

Минусы: 1. Некоторые функции достаточно длинные (например, tokensReceivedCallback 437 LOC и некоторые
200+) и сложно реализовать во время код-ревью.
2. Взаимодействие контрактов с обратными вызовами иногда занимает достаточно много времени (см. траекторию развертывания токена Lp с 19 функциями подряд)
3. Возникает повторяющийся
код 4. Некоторые проблемы обнаружены и описаны на предыдущих страницах

Все найденные проблемы обсуждались с авторами во время звонков в Zoom и в Telegram-канале.

Наконец, мы предполагаем, что соответствующий обходной путь желателен, и мы можем рекомендовать контракты для использования в производстве со следующей оговоркой:

Некоторое поведение контракта (возможно, не полное) настоятельно рекомендуется пройти формальную проверку, особенно: - функции с математическими вычислениями; - длинные контракты (более 6-ти подряд)
цепочка вызовов (в т.ч. обратные вызовы); - процедура развертывания и эффекты обновления кода.