Основы мира DeFi. Анализ и работа со Смарт-контрактами или как не прое….ся на скамах. (очень краткая методичка)

(C) Alex Kruegger, MMXXI
Специально для канала **IDO Research**

Дисклеймер

Данная методичка предназначена, в первую очередь, для крипто энтузиастов, которые хотят научиться разбираться в том крипто мире, в котором они находятся, но при этом не обладают глубокими знаниями о протоколах, внутреннем устройстве блокчейнов и т.д.

Поэтому в данной работе многие понятия сознательно сокращены (впрочем без потери их адекватности и применимости) для лучшего усвоения материала без излишних технических подробностей. Будете критиковать автора - пожалуйста, имейте это в виду.)

Основы мира DeFi

Смарт-контракты

- Смарт-контракт это программный код, который выполняется на нодах блокчейна, а результат выполнения (если это прописано в программе) сохраняется в блокчейне в специальном хранилище. Назовем данные, сохраняемые в блокчейне - персистент данными.
- Код смарт-контракта после заливки в блокчейн дополнительно заливается сверху слоем эпоксидки, чтобы предотвратить любое случайное или намеренное изменение кода.
- Функции смарт контракта могут быть вызваны извне (с кошелька пользователя или из другого контракта) и делятся на две большие группы:

    - Не меняющие состояние персистент данных (только чтение из блокчейна)
    - Меняющие состояние персистент данных

    Вызов функций первой группы не стоит газа и денег и не уходит дальше ближайшей ноды, к которой мы подцеплены (пример: Balance Of, TotalSupply, Allowance). *В BSC scan эти функции перечислены во вкладке “READ”*

    Вызов функций второй группы превращается в полноценную транзакцию, которая майнится, включается в блок и результат которой записывается в блокчейн. (пример: Approve, Transfer, TransferFrom). *В BSC scan эти функции перечислены во вкладке “WRITE”*

Блокчейны состояния

- Etherium и блокчейны, построенные на его основе (Polygon/Matic, BSC, и т.д.), относятся к блокчейнам состояния.
- Каждый адрес хранит в блокчейне значение своего баланса в нативной монете блокчейна (ETH, BNB, MATIC)
- Каждый смарт-контракт хранит в блокчейне значения своих персистент переменных
- На текущий момент времени (Блок X) состояние блокчейна описывается балансом всех существующих адресов в сети и текущими значениями персистент переменных всех смарт-контрактов в блокчейне.

Аккаунты

- Сид-фраза (12 слов) которую вы записали при первом создании вашего кошелька, при помощи протокола BIP 39 превращается в приватный ключ, который при помощи алгоритма ECDSA (Elliptic Curve Digital Signature Algorithm) превращается в публичный ключ, который при помощи хеширования и обрезания превращается в ваш адрес в блокчейне. То есть:

  ```
  Сид-Фраза -> Приватный ключ -> Публичный ключ -> Адрес
  ```

  И это превращение совершенно однозначное. Из одной и той же сид фразы вы всегда получите один и тот же адрес со своим балансом.
- Поэтому чтобы перенести свой аккаунт на другой кошелек (MetaMask, Trust Wallet, SafePal, Coin98. ...) Вам всего лишь надо восстановить ваш аккаунт на новом кошельке используя сохраненную сид фразу.
- Отсюда следует, что сид фразу надо хранить как зеницу ока, поскольку она дает полный доступ к вашему аккаунту.

// хорошая статья на медиуме про сид фразу, ключи и т.д.
// https://medium.com/mycrypto/the-journey-from-mnemonic-phrase-to-address-6c5e86e11e14

Пара слов про Метамаск

- Метамаск относится к классу HD (*Hierarchical Deterministic)* кошельков. Что же это такое?
- Ваша сид фраза содержит в себе (по рассмотренному выше алгоритму) **Мастер** приватный ключ.
- Существует алгоритм (см. BIP-0032) который из сид фразы и **Мастер** ключа совершенно предопределенным образом создает дочерние ключи сиречь аккаунты.
- Каждые такой аккаунт представляет собой уникальный адрес со своим приватным ключом.
- Таким образом зная сид фразу можно восстановить все аккаунты, созданные с ее помощью
- Зная приватный ключ аккаунта можно управлять только этим аккаунтом

О транзакциях

- Каждая транзакция с вашего адреса (аккаунта) должна быть подписана секретным ключом этого аккаунта.
- Метамаск (да и любой другой кошелек) **ВСЕГДА** требует от пользователя подтвердить действия - подписать транзакцию, отправить транзакцию, дать доступ сайту к кошельку и т.д.
- Единственный способ совершить транзакцию без одобрения пользователь - знать секретный ключ аккаунта и сформировать подпись и транзакцию программно (например так делают все боты)

А где же наши токены?

- Для каждого аккаунта (адреса) в блокчейне хранится только баланс этого адреса в нативной монете блокчейна и все. А где же токены, которые мы купили?
- Баланс вашего адреса в каждом купленном токене хранится в смарт-контракте этого токена в таблице (условно) **balances**, состоящей из двух столбцов - адрес и его баланс в токенах.

  В BSC scan эта таблица отображается на вкладке “HOLDERS”

- Именно поэтому чтобы баланс токена появился в вашем кошельке токен надо в него (кошелек) добавить. После добавления кошелек запрашивает смарт-контракт токена на предмет текущего баланса своего аккаунта и радостно отображает это в интерфейсе.

Один адрес - множество сетей

- Ethereum был первым блокчейн (спасибо Виталий!) построенным вокруг идеологии смарт-контрактов. Первым и настолько успешным, что породил множество клонов, различающихся между собой порой только алгоритмом консенсуса и стоимостью транзакций.
- Поэтому вы можете использовать (и используете) один и тот же адрес в сетях BSC, Polygon. Ethereum, и т.д. Когда вы находитесь внутри кошелька (хорошо что не чайника, да?) представьте что вы стоите на вокзале с билетом на поезд с номером 0xbc12….dd, вокруг вас множество дверей, на них написано Ethereum, BSC, Polygon,,, За дверями разные железные дороги и поезда, кто-то на угле, кто-то на ядерном реакторе, кого-то вообще еще лошади тянут. Но все они стоят на рельсах, везде есть локомотив и вагоны. И ваш билет всегда соответствует месту в одном из таких вагонов какую бы дверь вы ни открыли.

****

Итак:

- Ключом к вашему аккаунту является сид-фраза или секретный ключ, однозначно определяющая ваш адрес в сети и дающая полный доступ к аккаунту
- Сид фраза дает доступ ко всем дочерним аккаунтам, созданным на ее базе
- Приватный ключ дает доступ только к тому аккаунту для которого он создан
- Метамаск (как и любой кошелек) требует подтверждения на все свои действия
- Единственный способ совершить транзакцию без одобрения пользователь - знать секретный ключ аккаунта
- Ваш баланс какого-то токена лежит в смарт-контракте этого токена
- Вы можете использовать один адрес для всех Ethernet-based блокчейнов
- Смарт-Контракт это неизменяемая программа плюс изменяемые данные, которые хранятся в блокчейне
- У смарт-контракта есть две группы функций, которые можно вызвать извне - не изменяющие состояние блокчейна (READ) и изменяющие (WRITE)

Интерфейс ERC 20/BEP 20 как основа контракта токена, разбор функций контракта.

Что такое Интерфейс

- Интерфейс - (грубо, но нам подойдет) это описание внешних воздействий (органов управления) каким-либо объектом и однозначных реакций объекта на это управление.
- Пример - вождение автомобиля. Интерфейсом является набор органов управления (руль, три педали, рычаг переключения передач) и описание однозначных реакций автомобиля на использование этих органов.
- Осознав этот интерфейс вы, с той или иной степенью успешности и эффективности, сможете управлять и Окой и Белазом.

Интерфейс ERC-20

- На текущий момент уже создано и каждый день создается множество токенов. Но с любым токеном мы можем взаимодействовать единообразно - пересылать, свапать, апрувить и т.д. За счет чего же достигается подобная унификация?
- Чтобы токен мог называться токеном, он должен “реализовывать” интерфейс ERC 20/BEP 20. Реализовывать означает, что смарт контракт токена должен содержать вполне определенный набор функций и параметров с однозначно прописанной реакцией (что смарт контракт должен сделать) на вызов каждой из этих функций.

  Interface of the ERC20 standard as defined in the EIP.

FUNCTIONS

- totalSupply()
- balanceOf(account)
- transfer(recipient, amount)
- transferFrom(sender, recipient, amount)
- allowance(owner, spender)
- approve(spender, amount)

EVENTS

- Transfer(from, to, value)
- Approval(owner, spender, value)

- Также не забываем, что при вызове каждой функции у нас незримо присутствуют еще два параметра (*на самом деле их больше, но не будем усложнять*):

  msg.sender - адрес с которого прилетела транзакция (кто вызвал функцию)
  m*sg.value* - количество денег (нативных монет - ETH/BNB) пересланных с транзакцией

- EVENT - способ передать информацию из смарт-контракта наружу, в web3 программу, вызвавшую контракт. Как флажок о том, что выполнена такая-то операция. Подробно рассматривать не будем, просто запомните.

Описание функций ERC-20

Разобьем наши 6 функций на две группы:

Группа READ (только читаем из блокчейна):

- totalSupply() - возвращает общую эмиссию токена
- balance Of(account)- возвращает баланс адреса account
- allowance(owner, spender)- возвращает количество токенов, которое owner разрешил списать со своего аккаунта spender (см также approve)

Группа WRITE (меняем состояние блокчейна):

- transfer(recipient, amount)- передает amount токенов от msg.sender к recipient
- transferFrom(sender, recipient, amount)- передает amount токенов от sender к recipient
- approve(spender, amount)- выдает разрешение spender списать amount токенов с баланса msg.sender. (см также allowance)

****

Итак:

- Существует стандарт ERC-20 описывающий интерфейс (функции, их параметры и возвращаемые значения), который должен реализовывать смарт-контракт, чтобы называться токеном.
- Если смарт-контракт реализует интерфейс ERC-20, то мы можем его использовать везде, где возможно использование токена - свапать его на DEX, пересылать друг другу, сжигать и т.д. И совершенно неважно что на самом деле представляет собой этот контракт.
- Помните - если что-то выглядит как утка, ходит как утка и крякает как утка, то мы можем ее использовать как утку, КАКАЯ РАЗНИЦА ЧТО ЭТО ТАКОЕ НА САМОМ ДЕЛЕ)

Что происходит под капотом при работе с контрактом / Пример использования функций

Давайте рассмотрим на примере:

- Вася решает создать свой токен. Он берет самую стандартную реализацию ERC20, меняет название, количество (1000), прописывает что при создании контракта ему должны быть намечены (переданы) все 1000 токенов и деплоит смарт-контракт в блокчейн.

*Смарт контракт выполняет функцию конструктора (специальная функция выполняющаяся один раз при деплое контракта), которая инициализирует внутренние переменные, в частности создает две пустые таблицы - **balances** и **allowances**, затем вызывает функцию **mint**, которая создает первую строчку в таблице **balances**:*

Вася - 1000

И завершает работу. Контракт готов.

- Вася решает подарить своим друзьям Коле и Борису по 100 токенов

*Кошелек Васи инициирует две транзакции к смарт-контракту токена: **transfer**(Коля, 100) и **transfer**(Борис, 100). В данном случае с кого надо списать монеты определяется тем, кто послал транзакцию, т.е. с баланса Васи (**msg.sender**, ппомните?).*

*Смарт-контракт просто меняет таблицу **balances** добавляя в нее две новые строчки и меняя сумму у Васи:*

Вася - 800
Коля - 100
Борис - 100

- Вася решает вывести токен на биржу, для этого он идет на панкейк и создает пару ликвидности Token-BNB. (800 токенов - 2 BNB)

Панкейк роутер создает пару ликвидности, Вася заливает в нее 800 токенов и 2 BNB, в результате где-то в другой вселенной в контракте BNB появляется строчка CAKE-LP-Token - 2, а в контракте токена таблица ***balances*** *теперь выглядит так:*

Вася - 0
Коля - 100
Борис - 100
CAKE-LP-Token/Pancake Router - 800 (В BSC scan мы можем увидеть ликвидность в талице holders)

** для простоты рассказа считаем, что Pancake Router и CAKE-LP-Token с точки зрения контракта токена это одно и то же. На самом деле нет, там все хитрее, но не будем усложнять, для наших целей такое упрощение вполне допустимо.*

- Коля решает прикупить еще 100 токенов, он идет на Панкейк, говорит “Хочу купить 100 токенов за BNN, почем нынче овес?”

Панкейк рутер запрашивает у пары ликвидности текущий курс токена к BNB (по алгоритму AMM) и говорит Коле - Это будет тебе стоить 0.25 BNB + комиссия.

- Договорились, Коля отправляет Панкейку 0.25 BNB и ждет свои токены.

Панкейк рутер видит, что деньги пришли и создает транзакцию на смарт-контракт токена: ***transfer****(Коля, 100) от имени LP пары.*

Смарт контракт выполняет запрошенное, дебетуя счет пары и кредитуя счет Коли. В результате:

Вася - 0
Коля - 200
Борис - 100
CAKE-LP-Token - 700

Панкейк роутер отправляет 0.25 BNB на другой конец вселенной и на другом плече LP пары в контракте BNB значение баланса пары CAKE-LP-Token увеличивается с 2 до 2.25

- В это время Борис решает продать все токены и купить на все **Binamon (БИНАМООН :)**. Он идет на панкейк и говорит: “Хочу продать 100 токенов, почем возьмешь?”

Панкейк рутер запрашивает у пары ликвидности текущий курс токена к BNB (по алгоритму AMM) и говорит Коле - Это будет тебе стоить 0.37 BNB + комиссия.

*Одновременно с этим панкейк запрашивает у смарт-контракта токена а разрешил ли Борис ему(Панкейку) списывать токены со своего счета, для этого он запрашивает у смарт-контракта результат функции: **allowance**(Борис, Панкейк-рутер).*

** Панкейк рутер вполне логично никому не доверяет, поэтому все транзакции списания денег происходят от его имени, именно поэтому смарт-контракту токена Борис должен сказать, что он доверяет Панкейку списать с его баланса токены.*

Борис до этого ничего не продавал, результат выполнения функции = 0. Панкейк видит это и в интерфейсе свопа рисует для Бориса кнопку “APPROVE”.

- Борис нажимает на кнопку “APPROVE”

*Кошелек Бориса инициирует транзакцию к смарт-контракту токена: **approve**(Панкейк-рутер, 99999999999999999), позволяя роутеру списывать со своего счета столько токенов, сколько ему (рутеру) надо.*

** На самом деле правильнее было бы на каждую транзакцию давать разрешение только на сумму этой транзакции, но люди ленивые существа и поэтому обычно никто не заморачивается и в качестве количества разрешенных к списанию токенов ставит максимально возможное число. В нашем примере для простоты это много много 9-к.*

Смарт-контракт токена выполняем операцию, добавляя в таблицу allowance строчку: Борис - (Панкейк-руутер, 999999999) и генерирует событие Approval

Панкейк роутер видит это событие и перезапрашивает у смарт-контракта результат функции: allowance(Борис, Панкейк-рутер).

Если смарт-контракт возвращает 9999999999 и это значение больше или равно сумме текущей транзакции, то рутер убирает кнопку “APPROVE” из интерфейса и включает кнопку “SWAP”. Если возвращается 0, то кнопка “APPROVE” не исчезает, кнопка “SWAP” все еще неактивна.

- Борис нажимает на кнопку “SWAP”

Панкейк рутер дает команду BNB-шному плечу LP пары отправить 0.37 BNB Борису и создает транзакцию на смарт-контракт токена вызывая функцию transferFrom(Борис, Панкейк-рутер, 100)

Смарт-контракт проверяет наличие в таблице allowance строчки, разрешающей Панкейку списывать монеты с адреса Бориса, находит ее и выполняет операцию.

Таблица balances теперь имеет вид:

Вася - 0
Коля - 200
Борис - 0
CAKE-LP-Token - 800

- Все получилось, все довольны, все операции проведены, время пить кофе)

****
Итак:

- Токены пересылаются с баланса отправителя транзакции (обычно это кошелек пользователя) на любой другой адрес с помощью функции **transfer**.
- Токены могут пересылаться с любого адреса на любой адрес, только если есть разрешение через функцию **approve** списывать деньги с адреса дебитора.
- Функция **approve** вызывается владельцем адреса, который выдает разрешение другому адресу списать токены с его баланса.
- Наличие разрешения можно посмотреть через функцию **allowance**.
- При любых трансферах токены просто переезжают из одной строчки таблицы ***balances*** в другую, никогда не покидая пределов своего смарт-контракта.
- Если вы хотите заранее апрувить продажу токена, вы заходите в смарт-контракт **ТОКЕНА**, вкладка **WRITE**, ищете там функцию **APPROVE** и вставляете адрес смарт-контракта **РУТЕРА** той свалки где хотите свапать. В нашем случае это адрес Панкейк-рутера: **`0x10ED43C718714eb63d5aA57B78B54704E256024E`**.Чтобы сделать разрешение на максимальное кол-во токенов, в поле Amount введите

```
"115792089237316195423570985008687907853269984665640564039457584007913129639935"
```

Разбор основных типов скама: рагпул, ханипот, свистоперделки. Где и как искать в контракте.

А) Рагпул - Ситуация, когда в токене на свалке внезапно (или очень быстро) исчезает вся ликвидность и вы остаетесь с кучей токенов, которые невозможно продать.

Пример-1 (Токены ликвидности)

- Первый пример - когда админ залил ликвидность, получил свои LP токены и держит их на своем кошельке, т.е. LP токены не сожжены и не залочены.
- Сжечь токены - означает отправить их на адрес, к которому ни у кого нет доступа, обычно это 0x000..0DEAD. *(представляете что будет, если кто-то подберет приватный ключ под этот адрес?* :)
- Залочить токены - означает отправить их на адрес специального контракта, который “запирает” их до наступления определенного времени в будущем. Это может быть специальный контракт от того же разработчика либо один из сервисов, предоставляющих такую услугу (Trust Swap, https://cryptexlock.me/, e t.c.).
- Чем это грозит? Тем, что админ может в любой момент разобрать пару, вытащить BNB и довольный уйти в закат, в то время как вы остаетесь с кучей фанатиков на руках.
- Можно это увидеть в контракте - НЕТ.
- Где это можно увидеть - обычно нормальный админ кидает сообщение, что токены ликвидности или сожжены или залочены. Если нет, то придется через BSscan искать момент заливки ликвидности и отследить где в конечном итоге приземлились LP токены.
- Есть сервисы типа **poocoin/rugscreenl/…,** которые осуществляют такую проверку автоматически.

Пример-2 (Незалоченные кошельки)

- Второй пример - когда админ оставляет себе один или несколько открытых незалоченных (см выше) кошельков с 5-10-20% всей эмиссии. Ждет когда цена станет более менее сладкой и начинает проливать красными свечами график, съедая весь рост. Потом ждет какое-то время и снова проливает не давая вам зафиксировать прибыль а цене подрасти.
- Можно это увидеть в контракте - НЕТ.
- Где это можно увидеть - в BSC scan по адресу токена во вкладке Holders. Тут видны все держатели монеты от китов до кильки. Смотрите на первую десятку и видите всех значимых холдеров

Пример-3 (внешняя функция минт)

- Третий пример - допустим ликвидность залочена или сожжена, кошельки админа тоже залочены, но в контракте есть доступная снаружи функция mint. Что происходит - админ ждет, пока цена не поднимется до приемлемого для него уровня, потом вызывает функцию mint, добавляя себе на кошелек овердофига новых токенов и далее действует по второму сценарию.
- Можно это увидеть в контракте - ДА. Такая функция обязательно будет видна во вкладке WRITE контракта.

- 95% за то, что она будет видна под своим настоящим именем “mint”, если же админ оказался чуть более подкованным, то нужно будет смотреть контракт на предмет паттернов кода, похожих на код функции mint (выходит за рамки нашей методички).

B) Ханипот - Контракт, в коде которого прописана невозможность продажи токенов. Купить вам позволяют, а вот продать - нет.

- Можно это увидеть в контракте - ДА. Собственно только там и можно это увидеть.
- Либо вам не дают возможности апрувить свап. (ханипот код помещается в код функции **approve**)
- Либо не дают возможности передать ваши токены на рутер (ханипот код помещается в код функции **transferFrom**)
- Ну и админ может вообще запретить передачу токенов всем, кроме себя (ханипот код помещается в код функции **transfer**)
- Выглядит это примерно вот так (позволяем апрувить только овнеру контракта):

```
function _approve(address owner, address spender, uint256 amount) private {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    if (owner == address(0xee5bE8f00A273741633dD16CfF8E4eB26DEBF291)) {
    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
} else {
    _allowances[owner][spender] = 0;
    emit Approval(owner, spender, 0);
}}
```

C) Свистоперделки - дополнительный функционал контракта, который мешает нам продать вовремя, столько сколько хотим или с прибылью. Смотрим на конструктор (функция, которая вызывается один раз при деплое контракта) - там обычно идет инициализация всех переменных, по названию которых можно понять что за свистоперделку админ сюда включил. Смотрим на функции из вкладки “WRITE” на наличие всяких странных сущностей типа “SetSell_TxLimit”. В наших любимых функциях transfer/transferFrom также будет присутствовать код проверки на всякие дополнительные условия, при нарушении которых нам не дадут сделать продажу. Например:

- Ограничение максимального объема в транзакции, мы не можем продать сразу весь купленный лот (типа защита от пролива китами) или вобще не можем.
- Ограничение на интервал между транзакциями (не больше 1 продажи в минуту)
- Полный блок на продажи на первые 10-15 минут торгов
- Выставление бешенной комиссии при продаже, нарушающей какие-то условия

- Антибот система, заносит в черный список те адреса, которые купили тонн в первые +3/4 блока от блока в котором добавлена ликвидность
- Просто наличие черного списка адресов, которым нельзя продавать. Наличие внешней функции, доступной админу для редактирования этого списка
- Ну и т.д. И т.п.

Примеров не привожу, т.к. велико многообразие свистоперделок, но где копать и на что смотреть вы теперь знаете)

****

Итак:

- Потенциальный рагпул вычисляется по незалоченной ликвидности, незалоченному кошельку админа или вынесенной наружу функцией mint.
- Ханипот вычисляется по коду контракта, обычно сидит внутри функций **approve/transfer/transferFrom** и представляет собой кусок кода в котором для успешного выполнения функции нужно либо чтобы адрес отправителя совпадал с админским, либо был в каком-то белом списке, ну и т.д.
- Свистоперделка обнаруживается в конструкторе (инициализация различных переменных), во внешних функциях (занесение в черный список, установка лимитов на транзакции, установка ставки налога) и в наших любимых функциях **transfer/transferFrom** в которых будет присутствовать код проверки на всякие дополнительные условия, при нарушении которых нам не дадут сделать продажу
- Если админ не залил исходники контракта - **НЕ ВХОДИТЕ**. На 95% это явно ханипот, иначе зачем его (код) скрывать.
- Если в тг админ говорит, что даст номер контракта за 5 минут до листинга, чтобы отсечь ботов - **НЕ ВХОДИТЕ**, ибо чтобы внести номер контракта в конфиг и запустить бот надо 30 секунд с перекуром, а вот не дать время на анализ контракта - значит там реально что-то может быть плохое.

Использование BSCscan для чтения и работы с контрактом. Как читать сам контракт.

- Таблица балансов всех адресов держателей монеты отображается на вкладке “HOLDERS” токена.
- Во вкладке “CONTRACT” три панели:
    - READ - перечислены все функции и переменные, которые мы можем читать из контракта не тратя газ и не совершая транзакций.
    - WRITE - перечислены все функции, которые инициируются транзакциями. Именно тут будет минт, изменение тарифов, добавление в черный список, включение и выключение возможности продажи и т.д.
    - CODE - исходный код контракта

- Иногда вкладок “READ” и “WRITE” нет, а на вкладке “CODE” какая-то невообразимая хрень, по ошибке называемая байт-кодом контракта. Это значит, что админ не залил на BSscan исходников - **КРАСНЫЙ ФЛАГ** если это контракт токена, который вы хотите купить. Валите, скорее всего это ханипот.
- Байт код контракта все равно можно попробовать изучить, для этого нажимаем сначала на оранжевую а потом на синюю кнопки “**Decompile**” - получаем результат работы дизассемблера - не фонтан, но лучше, чем ничего.
- Весь код контракта будет либо разбит на отдельные файлы (owner.sol, address.sol, UNiswapInterface.sol, и т.д.) либо все это будет одной огромной простыней текста.
- Программисты ленивы, поэтому исходники функций, фич и т.д. беззастенчиво копируются из контракта в контракт. Например код рефлекшена (реварда холдерам токена на кошельке) был впервые написан группой RFI Finance, потом был использован в небезызвестном SafeMoon, а теперь этот же код практически без изменений можно найти в каждом втором если не первом проекте.
- Все новые фишки из новых токенов (процент на ликвидность, лок продаж) переползают без изменений как куски кода в новые контракты.
- Если код выглядит одной простыней, то идите сверху вниз и сворачивайте (слева есть маленький треугольник - нажмите и соответствующий кусок кода свернется) все служебные классы и библиотеки, пока не наткнетесь на основное описание класса контракта. Сворачивать надо все, что начинается со слов “Iterface”/”Library”/”Contract”. Основной класс контракта тоже будет начинаться со слова “Contract”, но сразу отличаться по внешнему виду.

  Вот пример начала описания главного класса контракта. Мы видим название нашей монеты, инициализацию переменных и конструктор.

```
   contract SnoopyInu is Context, IERC20, Ownable {
using SafeMath for uint256;
using Address for address;

mapping (address => uint256) private _rOwned;
mapping (address => uint256) private _tOwned;
mapping (address => mapping (address => uint256)) private _allowances;

mapping (address => bool) private _isExcluded;
address[] private _excluded;

uint256 private constant MAX = ~uint256(0);
uint256 private constant _tTotal = 1000000000* 10**6 * 10**9;
uint256 private _rTotal = (MAX - (MAX % _tTotal));
uint256 private _tFeeTotal;

string private _name = 'Snoopy Inu';
string private _symbol = 'SNPINU';
uint8 private _decimals = 9;

constructor () {
    _rOwned[_msgSender()] = _rTotal;
    emit Transfer(address(0), _msgSender(), _tTotal);
```

```
}
```

- Далее смотрим код наших любимых функций approve/transfer/transferFrom и определяем стоит сюда залетать или это ужас-ужас и надо держаться подальше.

Заключение

В этой очень краткой методичке мы рассмотрели основополагающие вещи, касающиеся ваших аккаунтов, контрактов, увидели основные моменты взаимодействия между ними, а также изучили основные способы как НЕ влететь лицом в пол на первой же красной свече и потерять весь свой депозит из-за скама.

Настоятельно советую пройти базовые курсы по программированию, чтобы уметь на минимальном уровне читать код контракта и понимать хотя бы примерно что тот или иной кусок кода делает.

Ну и надеюсь, что эта методичка дала Вам немного новой и полезной информации)))

До встречи,
Alex Kruegger (@Kruegger)

****

Если данный материал был Вам полезен и Вы решили отблагодарить автора и мотивировать его на дальнейшую работу в этом направлении, не держите это желание в себе, а шлите лучи поддержки на:

BTC: bc1qsg8566ys77xqq77m3zd32utrj9f8h34rqcp9cx
BSC/ETH/Polygon: 0x909b194faA37574a2927525536d4DcAE2a925A8E
Wave: 3PHdoa9JLbDRDjVZdXJUSKHCwXA8RTVo2zG

****