Ambisafe
Software

Blockchain School

2018

Урок 4. Пишем игру OXO

Обрабатываем поступление и отправку денег для контрактов, разбираемся, как устроена память Solidity, и как работать с библиотеками в Ethereum.

План

1. Вызовы в другой контракт
2. Удаление данных из контракта
3. Storage and Memory
4. Отправка денег из контракта
5. Задания

1. Вызовы в другой контракт

DelegateCall – это вызов функции другого контракта, но при этом мы остаемся в своем контракте. Таким образом, если эта функция меняет какие-то записи в storage, то storage меняется у того контракта, который вызвал функцию, но не у того, в котором она прописана.

Библиотеки в Solidity – это обычные контракты. Их функции описывают какую-то логику, которую потом можно использовать из других контрактов с помощью DelegateCall.

Функция DelegateCall очень полезна, но она также открывает возможности для взлома. Какая бы логика не была у вашего контракта, к нему может обратиться другой контракт, вызывая публичные функции. Учитывайте это, когда пишете логику контракта.

Например, вы наверняка слышали про уязвимость в контракте DAO. Функция **splitDAO** в их контракте реализовывала логику выдачи вознаграждения и работала нормально, но не учитывала, что будет, если к ней обратится контракт. В итоге, один из пользователей написал код, который рекурсивно выкачивал деньги из контракта, потому что в функции не стояла защита от этого. В коде сначала отправлялись деньги на счет пользователю, а потом его виртуальный счет в контракте обнулялся. Чтобы обойти эту уязвимость, нужно сначала обнулять счет пользователя и только потом отправлять ему деньги.

Об этой и о других уязвимостях вы можете прочесть тут:

https://consensys.github.io/smart-contract-best-practices/

2. Удаление данных из контракта

Selfdestruct – функция, которая удаляет контракт, его функции и данные в storage. Вызывая эту функцию, вы получаете небольшой refund газа за то, что освобождаете хранилище в Ethereum.

Рекомендуется реализовывать эту функцию так:

```
function kill(address _to) public onlyowner {
    selfdestruct(_to);
}
```

В **_to** вы передаете адрес, на который отправите весь эфир, который был в этом контракте.

P.S. Если кто-то попытается отправить на ваш контракт деньги таким способом, то вы никак не сможете это предотвратить. Нет никакого способа запретить это сделать. Казалось бы *“А зачем?”*. Затем, что ваш код может начать работать неправильно. Например, если вы ожидаете, что у вас 0 на счету контракта и если вам кто-то отправил 0.000000000000001 эфиринки, то это вроде вас богаче не сделало, а код не работает. Так что, никогда не пишите код исходя из того, будто вы знаете сколько на его счету может быть денег. И никогда не предполагайте, что если у него нет **payable** функций, то значит денег на его счету всегда будет ноль.

Внимание!

Перед тем как отправить транзакцию, убедитесь, что вы понимаете, к чему это приведет. Лучше всего сначала проверить контракт или транзакцию в тестовой сети. Вы можете навредить не только чужому контракту, но и своему.

3. Storage and Memory

Storage – хранилище контракта.

Memory – то, что хранится в памяти текущего вызова к контракту. Как только вызов закончился, память стерлась.

Запись одной пустой ячейки в storage стоит 20 000 газа. Изменение одной ячейки в storage стоит 5000 газа. Запись одной ячейки в memory стоит 3 газа. Если вы собираетесь производить большое количество манипуляций с изменением значения переменной в транзакции, то это лучше всего делать в memory. Но после всех изменений с переменной, нужно не забывать все-таки записать ее в storage.

```
contract Test {

    struct Entity {
        string name;
        uint value;
    }

    Entity[] public entities;

    function addEntity(string _name, uint _value) public returns(bool) {
        entities.push(Entity(_name, _value));
        return true;
    }

    function increaseValue(uint _id) public returns(bool) {
        Entity storage ent = entities[_id];
        ent.value += 1; //5000 gas
        ent.value += 1; //5000 gas
        ent.value += 1; //5000 gas
        ent.value += 1; //5000 gas
        ent.value += 1; //5000 gas
        return true;
    }

    function decreaseValue(uint  id) public returns(bool) {
        Entity memory ent = entities[_id];
        ent.value -= 1; //20 gas
        ent.value -= 1; //20 gas
        ent.value -= 1; //20 gas
        ent.value -= 1; //20 gas
        ent.value -= 1; //20 gas
        ent.value -= 1; //20 gas
        entities[ id].value = ent.value; //5000 gas
        return true;
    }
}
```

Важно. Так как memory является ссылкой на сущность, то запись:

```
uint memory a;
```

не имеет смысла, потому что ей не на что ссылаться. В memory можно работать только с объектами.

4. Отправка денег из контракта

Чтобы отправить деньги из контракта на какой-то адрес (кошелька или контракта), можно использовать две функции: **transfer()** и **send()**.

```
msg.sender.transfer(value);
msg.sender.send(value);
```

Note: При **transfer()** и **send()** контракту получателя будет доступно всего 2300 газа. Поэтому fallback функция должна использовать до 2300 газа, если хочет принять эфир. То есть, лучше ей вообще остаться пустой.

Эти две функции отличаются тем, что **transfer()** делает **revert()** в случае, если деньги нельзя отправить по какой-то причине (например, если их не достаточно на счету), а **send()** возвращает **false** в случае неудачи и **true** в случае успеха.

Таким образом, используя **send()**, вы можете добавлять сообщение об ошибке с помощью ивентов, но не забывайте о том, что бывают случаи, когда **revert()** необходим. В таких случаях все-таки используйте **transfer()**.

Команда **msg.value** указывает на сумму эфира в определенной транзакции. Благодаря ей мы можем узнать, сколько эфира прислали вам в результате транзакции.

Команда **this.balance** позволяет прочесть свой баланс, то есть увидеть количество средств на счету.

5. Задания

Задание 1

Сделать игру крестики-нолики на умном контракте.

Есть два адреса (два человека или два контракта), которые играют друг с другом в крестики-нолики. Они играют не на интерес, а на эфир. Нужно сделать так, чтобы выигравший получил свой выигрыш и чтобы в случае ничьей оба игрока забрали свои ставки.

Задание 2

Написать тесты к этому контракту.

Задавайте вопросы в нашем Телеграм-чате!

Также, можете заливать готовый код в github и кидать ссылку в телеграм, чтобы все желающие могли проверить, все ли правильно, и указать на возможные проблемы.