DevCon School

Технологии будущего

Погружение в Ethereum

Сергей Лоншаков
Blockchain разработчик, Airalab

Цели

Создание умного контракта в Ethereum Blockchain

Пример инфраструктуры из нескольких видов умных контрактов

Проверка умного контракта в Ethereum Blockchain

Рекомендации от ведущих разработчиков Ethereum.

Выдержки из презентаций с devcon2 в Шанхае

Известные атаки на умные контракты на Ethereum

Атака по глубине стека, DoS при вызове исключения, условия гонки, DoS при переполнении лимита газа.

Создание умного контракта в Ethereum Blockchain

#msdevcon

DevCon School
Технологии будущего

Контракт – это аккаунт

Есть два типа аккаунтов в Ethereum:

Аккаунт пользователя (Externally owned account сокр. EOAs): учетная запись контролируемая закрытым ключом, и если у вас есть закрытый ключ, связанный с EOA у вас есть возможность отправлять эфиры и сообщения от EOA.

Контракт: учетная запись, которая имеет свой собственный код, и управляемая с помощью кода.

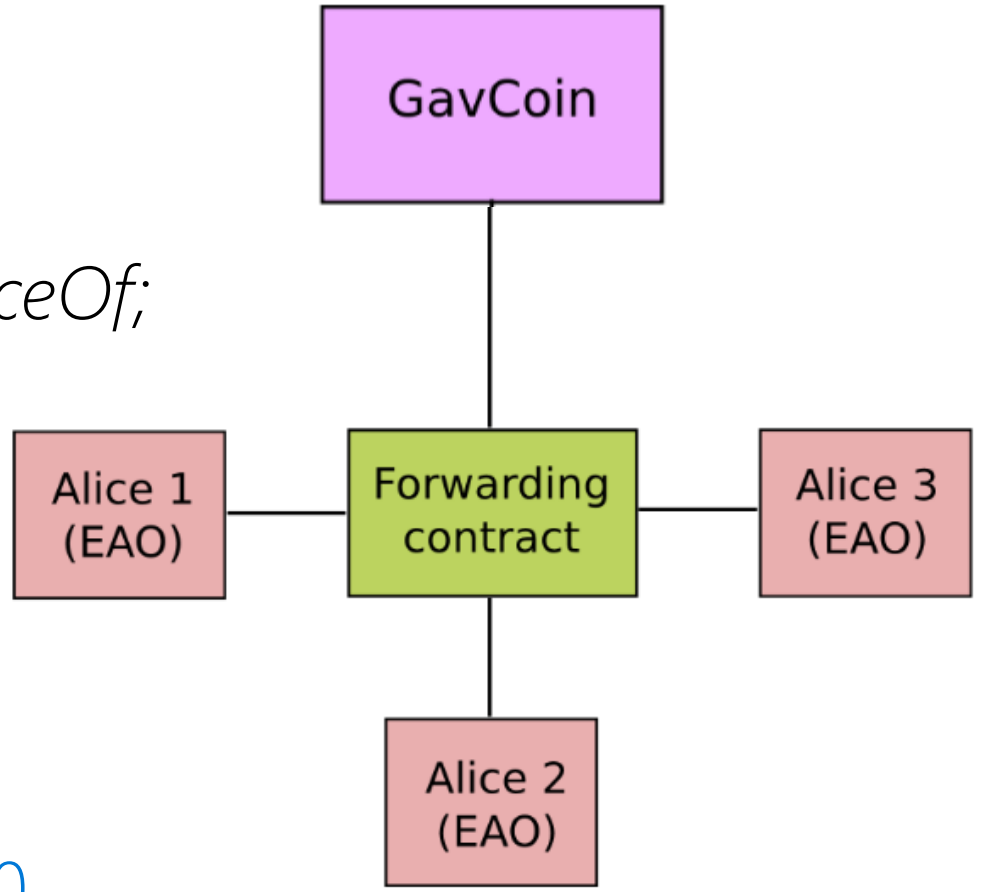
Токены

contract MyToken {

/ This creates an array with all balances */*

mapping (address => uint256) public balanceOf;

}

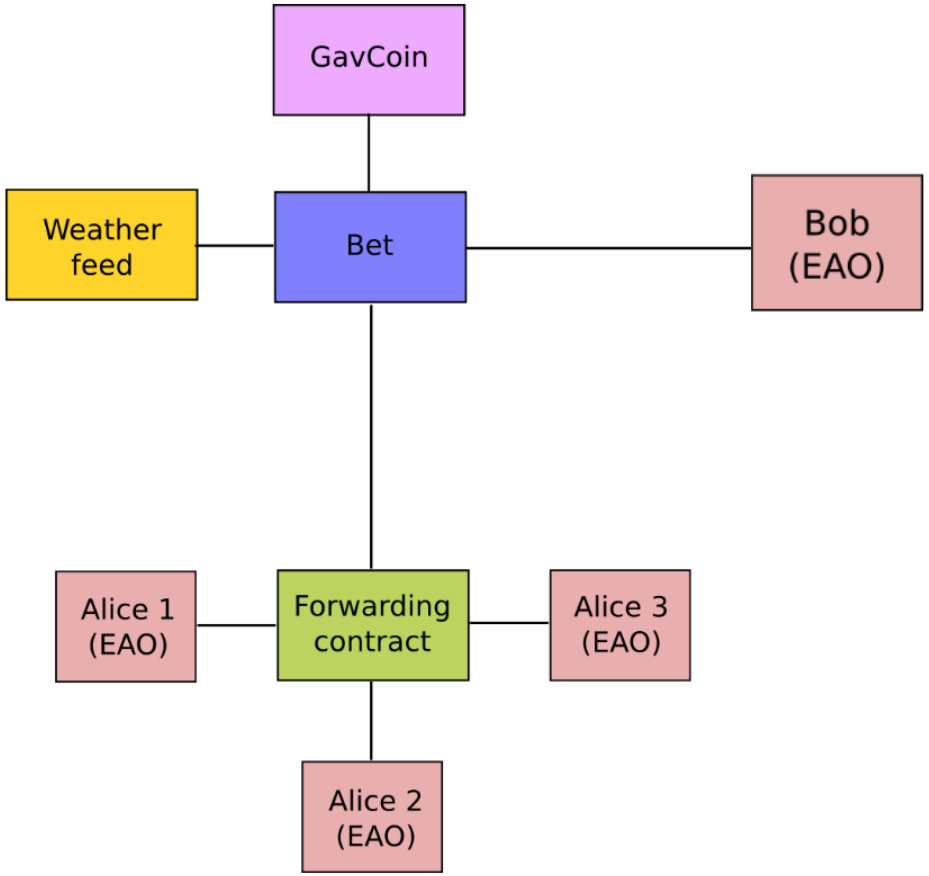


Описание стандарта ERC20:
https://github.com/ethereum/EIPs/issues/20

Управление взаимоотношениями

1. Конгресс / Совет директоров /Демократия
2. Escrow / Аккредитив
3. p2p + Страхование / Кредитование / Финансирование

Управление взаимоотношениями



Alice и Bob заключают пари на 100 GavCoin с триггером на основе данных о температуре в Москве.

Если температура в Москве не будет выше 35 градусов в течении года, то Bob получит 100 GavCoin, иначе их получит Alice.

Демонстрация

Разбираем ERC20 токен

Проверка умного контракта в Ethereum Blockchain

Рекомендации от ведущих разработчиков Ethereum

#msdevcon

DevCon School
Технологии будущего

Обобщенные рекомендации к проверке кода контрактов на Ethereum платформе

1. Избегайте внешних вызовов когда это возможно, очень часто они являются причиной уязвимости
2. Отдавайте приоритет изменению состояния над внешним вызовом вызовом (включая .send())
3. Изолируйте внешние вызовы в отдельной транзакции, например, метод withdraw для вывода средств
4. Будьте внимательны при делении целых чисел (округление происходит к ближайшему целому)
5. При делении на ноль возвращается **ноль**, проверяйте аргументы самостоятельно

Обобщенные рекомендации к проверке кода контрактов на Ethereum платформе

6. Будьте внимательны к переполнению целых чисел (особенно в сравнениях) и приведению знаковых к беззнаковым в JS
7. Будьте осторожны при переборе динамических массивов, это может потребовать большое количество газа
8. Будьте осторожны с привязкой логики контракта ко времени блока, оно устанавливается майнером
9. Подумайте о способе обновления контракта в будущем
10. Используйте метки остановки работы контракта в случае чрезвычайной ситуации, например обнаружении уязвимости

Обобщенные рекомендации к проверке кода контрактов на Ethereum платформе

11. Разделяйте критически важные вызовы во времени, например, вывод большого количества средств не чаще раза в неделю

12. Используйте формальную верификацию контрактов

Известные атаки на контракты Ethereum платформы

#msdevcon

DevCon School
Технологии будущего

DoS при переполнении лимита газа

Идея: объем газа в блоке ограничен, если для транзакции требуется газа больше, чем помещается в блок - она никогда не будет исполнена.

DoS при переполнении лимита газа

```
struct Payee {
        address addr;
        uint256 value;
}

Payee payees[];

uint256 nextPayeeIndex;
function payOut() {
        uint256 i = 0; // при достаточно большом размере payees возможно
        превысить лимит газа while
        (i < payees.length) { payees[i].addr.send(payees[i].value); i++;
        }
}
```

DoS при переполнении лимита газа

Противодействие: избегать итерации по большим массивам данных, либо переносить перебор на программную логику вне контракта; если невозможно избавиться от перебора, необходимо разбить его на несколько шагов, либо выполнять действия по запросу, например, добавить метод `withdraw` для вывода средств.

Атака по глубине стека

Идея: разрешенная глубина стека составляет 1024, вызовы глубже не будут выполнены, однако транзакция не прервется; атакующий может вызвать код с глубиной стека 1023, в таком случае вызовы из уязвимого кода, например `send()` не будут исполнены.

Атака по глубине стека

```
// INSECURE
contract auction {

        mapping(address => uint) refunds;

        // [...]

        function withdrawRefund(address recipient) {
                uint refund = refunds[recipient];
                refunds[recipient] = 0;
                recipient.send(refund); // эта строка исполнится не так, как ожидается
        }
}
```

```
// ATTACKER contract xxx {
        function foo(address target, address recipient, uint iter) {
                if (iter < 1023) foo(target, recipient, iter+1);
                else auction(target).withdrawRefund(recipient);
        }
}
```

Атака по глубине стека

Противодействие: минимизация вызовов внутри методов, приоритет записи и учета над вызовом другого метода; а также `.send()` возвращает `false` если не может быть исполнена, необходимо при каждой отправке средств проверять возвращаемое значение.

Условия гонки

Идея: внешний вызов может произвести неконтролируемые изменения в данных контракта.

Условия гонки

```
// INSECURE
mapping (address => uint) private userBalances;

function withdrawBalance() public {
                uint amountToWithdraw = userBalances[msg.sender];  // в этом месте внешний
                контракт может вызвать метод withdrawBalance снова
                if (!(msg.sender.call.value(amountToWithdraw)()))               { throw; }
                userBalances[msg.sender] = 0; }
```

```
/// ATTACKER
contract xxx {
                uint iter;
                address target;

                function foo(address _target) {
                                iter = 0;
                                target = _target;
                                token(_target).withdrawBalance(); }
                                function () payable { if (iter < 10) // Withrawal 10 times
                                token(_target).withdrawBalance();
                }
}
```

Условия гонки

Противодействие: для недоверенного кода(`msg.sender`) приоритет в использовании `.send()` над `.call.value()`, так как количество газа для `.send()` очень ограничено и не может быть использовано для эксплуатации уязвимости.

Q&A

Погружение в Ethereum

Сергей Лоншаков, sergeylonshakov@gmail.com

#msdevcon

DevCon School
Технологии будущего