DOI: 10.51932/9785907271739_106
УДК 004.056.53

В.А. Кардаков
(г. Казань, Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ)

БЕЗОПАСНОСТЬ СМАРТ – КОНТРАКТОВ ДЛЯ САПР

SMART CONTRACT SAFETY FOR CAD

Рассмотрена важность безопасности смарт-контрактов и их уязвимости. В данный момент все чаще и чаще ведутся разговоры о применении блокчейн – технологии в различных сферах: коммерческая деятельность, строительство, IT и т.д. Также блокчейн можно применить и в САПР: благодаря смарт-контрактам можно формировать, контролировать, предоставлять владение информацией.

At the moment, more and more often there are talks about the use of blockchain technology in various fields: commercial activity, construction, IT, etc. Blockchain can also be used in CAD: thanks to smart contracts, it is possible to form, control, using information ownership. Therefore, the security of smart contracts and their vulnerabilities are important.

Ключевые слова: блокчейн, смарт-контракты, информационная безопасность, Ethereum.

Keywords: blockchain, smart contracts, information security, Ethereum.

Каких-то нескольких лет назад о блокчейн – технологии знало малое количество людей. Однако блокчейн уже применяется в финансовом секторе, в банковской сфере, реестре и для осуществления патентов, авторских прав. Так же мы можем использовать блокчейн в САПР. Например, в блоке может находится уникальная информация о чертежах, также какие-то новые алгоритмы размещения, трассировки, компоновки и т.д. В общем и целом, в блоке может находится любая информация, которая важна в САПР.

Что такое блокчейн? Блокчейн – это распределенный реестр, который обеспечивает безопасность и неизменность данных и представляет собой постоянно растущую последовательность блоков. В каждом блоке имеется хэш – сумма, которая высчитывается от всех транзакций, входящих в блок, и именно благодаря ей происходит неизменность данных, т.к. хэш – сумма одного блока зависит от хэш – суммы предыдущего блока. Благодаря данной взаимосвязи невозможно изменить данные внутри самого блокчейна [1].

Смарт – контракт – это, по сути, фрагмент кода, который хранится в блокчейне. Благодаря смарт – контракту имеется возможность читать данные, писать данные в блокчейн.

Одним из самых популярных блокчейн – технологии, которая направлена на работу со смарт – контрактами, является Ethereum. Только за первую половину 2018 года количество смарт – контрактов Ethereum возросло в два

раза, в сравнении с 2017 годом. Соответственно, с увеличением количества использования какой – либо технологии возрастает количество новых уязвимостей. Например, благодаря уязвимости контрактов была совершена кража 30 миллионов долларов из Parity и 53 миллиона долларов из DAO. Именно во время проектирования или построения проектов, которые будут использовать смарт-контракты в САПР, нужно обеспечить безопасность информации в смарт – контрактах от шпионажа, взлома и т.д.

Сам код контракта в Ethereum пишется на языке Solidity, после этого происходит его компиляция до байт – кода для вертуальной машины Ethereum [2]. За вычисления, которые производит контракт, разработчик платит узлу некое количество валюты (газа). И имеется прямая связь: чем выше сложность вычисления, тем больше газа оплачивает разработчик.

Для защиты информации у блокчейна имеются базовые методы защиты: Proof of Work и Proof of Stake. Однако имеются уже известные уязвимости, которые нужно учитывать при моделировании и разработки блокчейн – системы для САПР.

1. «Состояние гонки». Данная уязвимость заключается в ошибках написания кода, когда возможен повторный вызов внешнего кода, во время выполнения кода контракта. Следовательно, вызовы функции могут взаимодействовать способами, которые ломают логику контракта (рис. 1).

```
1  mapping (address => uint) private userBalances;
2
3  function withdrawBalance() public {
4          uint amountToWithdraw = userBalances[msg.sender];
5          require(msg.sender.call.value(amountToWithdraw)()); // 1
6  /* В строчке 1 вызывается внешний код, который может быть вызван повторно
7     до завершения первого вызова */
8          userBalances[msg.sender] = 0;
9  }
```

Рис. 1. «Состояние гонки»

2. Ошибка при переполнении типов данных. Переполнение может изменить значение, которую разработчик упустил при моделировании логики контракта (рис. 2).

```
mapping (address => uint256) public balanceOf;
// Небезопасный вариант
function transfer(address _to, uint256 _value) {
    /* Проверка доступности суммы для отправки */
    require(balanceOf[msg.sender] >= _value);
    /* Add and subtract new balances */
    balanceOf[msg.sender] -= _value;
    balanceOf[_to] += _value;
}
// Безопасный вариант
function transfer(address _to, uint256 _value) {
    /* Проверка доступности суммы для отправки и проверка на переполнение */
    require(
    balanceOf[msg.sender] >= _value &&
    balanceOf[_to] + _value >= balanceOf[_to]
    );
    /* Изменить балансы */
    balanceOf[msg.sender] -= _value;
    balanceOf[_to] += _value;
}
```

Рис. 2. Ошибка при переполнении типов данных.

3. Атака с возможностью возврата информации. Данная уязвимость происходит во время попытки передачи информации обратно, т.е. предыдущему человеку. Однако происходит ошибка, когда контракт принимает начальное положение, но доступ к контракту остается за тем, кто переводил информацию обратно, а не наоборот (рис. 3). Так же может происходить ошибка с количеством газа контракта [3].

```
contract auction {
    address highestBidder;
    uint highestBid;
    function bid() {
        if (msg.value < highestBid) throw;
        if (highestBidder != 0) {
        /*если этот вызов постоянно возвращает ошибку,
        никто не может сделать ставку*/
            if (!highestBidder.send(highestBid)) throw;
        }
        highestBidder = msg.sender;
        highestBid = msg.value;
    }
}
```

Рис. 3. Атака с возможностью возврата информации.

4. Уязвимости мультиподписей. Наиболее распространены во многих смарт – контрактов мультиподписи. В Ethereum мультиподпись имеет совместную подпись и находится в смарт – контракте. Если отсутствует проверка на повторную инициализацию блока, то любой человек, который вызывает функцию инициализации, может стать полноправным владельцем информации, который находится в блоке (рис. 4).

```
function initWallet(address[] _owners, uint _required, uint _daylimit) {
    initDaylimit(_daylimit);
    initMultiowned(_owners, _required);
}
```

Рис. 4. Уязвимость мультиподписи

Таким образом, нужно учитывать возможные уязвимости и слабые места при моделировании и создании системы, использующая блокчейн – технологию, а именно смарт – контракты. Это касается не только для САПР, но и для остальных сфер, где блокчейн может пригодиться для решения внутренних задач. Так же стоит учитывать, что при увеличении количества пользователей и компаний, которые используют смарт – контракты, тем выше шанс выявить злоумышленникам новые дыры и ошибки в информационной безопасности. Зная существующие проблемы, мы можем прогнозировать новые уязвимости, которые имеют места быть, для создания новых методов защиты и обновлении существующих.

Список литературы

1. *Уильям Могайар.* Блокчейн для бизнеса / Уильям Могайар, В. Бутерин. – Москва: ООО «Издательство «Эксмо», 2018. – 26 с.

2. Как работает Эфириум (Ethereum)? [Электронный ресурс]. – 21 октября 2017. – URL: https://habr.com/ru/post/407583 (дата обращения: 01.10.2020).

3. Безопасность и тестирование смарт-контрактов. [Электронный ресурс]. – 01 ноября 2017. – URL: https://inaword.ru/blokchejn/bezopasnost-i-testirovanie-smart-kontraktov/ (дата обращения: 02.10.2020).

Материал поступил в редколлегию 12.10.20.