BLOCKCHAIN VULNERABILITIES IN PRACTICE

O'rinov Nodirbek Toxirjonovich,Teacher, Department of Information Technology, Andijan **State University**

Saidova Nigora Kamiljonovna-Teacher, Department of Computer Engineering, Andijan State University

Mirabdullayev Izzatillo Isroiljon o'g'li-Student, Computer science teaching methods, Andijan State University

Blockchains are not immune. There are known vulnerabilities in various components of the blockchain ecosystem. This field note describes some of the vulnerabilities seen in smart contracts and node software, their use and how to prevent them, with a focus on the Ethereum ecosystem.

CCS concepts: • ***Security and confidentiality*** → ***Security of*** distributed systems; Anonymization and disinfection of data; Denial of service attacks;

Additional keywords and phrases: blockchain, security, smart contracts, ethereum **ACM**

1. INTRODUCTION

Blockchains are relatively new [13, 36], and there are countless [12, 67] news stories about people losing money due to trade-offs in the components of the blockchain ecosystem. Blockchain technology is not invulnera - BLE and there are actually many known vulnerabilities [10, 49, 50] , just like with any software. This article describes the most common components of the blockchain ecosystem and the vulnerabilities they may contain. The more functions the blockchain offers, the larger the attack surface becomes.

Ethereum [11] , for example, has a virtual machine called the EVM that executes the EVM bytecode. This virtual machine runs smart contracts in an isolated environment. The EVM specification defines over 140 different instructions that smart contract programmers can use. EtherVM [63] provides a link to these instructions. There have been many real-life cases where decentralized applications, also known as DApps [39] , used vulnerable smart contracts that led to the theft of funds [66] . It is also worth noting that vulnerabilities can exist in any component of the blockchain ecosystem, not just smart contracts.

2. BLOCKCHAIN ECOSYSTEM COMPONENTS

The blockchain ecosystem consists of several components. There is usually a **blockchain core** software that will be removed from client software, such as Go Ethereum [4] or Parity [59] in the case of Ethereum.

Each peer participating in a given blockchain network runs this client software, sometimes referred to as a node. Blockchain systems usually contain a wallet that can be implemented in software or hardware. A node and a software wallet can be combined into the same software. Notable hardware wallets include the Ledger Nano X [33] and the Trezor Model T [62] .

Cryptocurrency exchanges are composed of web applications and publicly available REST APIs that can be used to programmatically exchange one cryptocurrency for another. Cryptocurrency is a digital currency protected by cryptography. Exchanges can use hardware security modules to protect private keys. These exchanges often support over a hundred different cryptocurrencies. The exchange requires at least one blockchain node for each supported cryptocurrency. These nodes allow the exchange to make transactions through various cryptocurrencies and control the transactions made on the blockchain to know when it has received cryptocurrencies.

Some blockchains support smart contracts: arbitrary code executed by multiple nodes of the blockchain network. These contracts are the main building block for building decentralized applications. DApps usually consist of an external application, which can be a web application, a desktop application, or a mobile application, as well as one or more smart contracts. These smart contracts contain methods that an external application can call - by sending a transaction to the contract address - to perform actions such as reading or writing data to the blockchain.

Finally, **e-commerce websites** that want to accept payments in cryptocurrencies such as bitcoins [36] also need a solution that provides this functionality. They can use existing solutions [7, 19] or create their own. In any case, such a solution requires the use of a blockchain node for each cryptocurrency in order to properly accept payments. All of these components present in the blockchain ecosystem can stack and create a large attack surface.

3. KNOWN VULNERABILITIES

There are many known vulnerabilities in blockchain ecosystems. This field note focuses on smart contracts and the main blockchain vulnerabilities.

3.1 Major blockchain vulnerabilities

3.1.1 Manipulation with the consensus mechanism. Blockchain consensus mechanisms such as proof of work or proof of stake have been attacked [38] . 51% attack is a well-known proof-of-work attack on blockchains in which an attacker who controls most of the network's computing resources can discard blocks mined by someone else, giving priority to their own blocks. Please note that this attack does not allow an attacker to create fake transactions or create money out of thin air. That, however, let them do double costs - ING. In practice, such attacks are extremely expensive on blockchain networks with high hashing rates [23] , such as Bitcoin, and are therefore quite rare. One possible prevention method is to perform a formal check [61] of the consensus mechanism.

3.1.2 Main vulnerabilities of the cryptosystem. Blockchain wallets usually work with a pair of public and private keys for signing and are as secure as the underlying cryptosystem they use. The public key algorithm (ECDSA [32] , EdDSA [6] , Schnorr [48] , ElGamal [24]) used for these keys has known attacks [8, 27, 29, 35, 45, 47] that can be applied in some cases. There are real use cases [10, 22] . To reduce the attack surface, it is necessary to use a library that implements the required cryptosystem with protection against side-channel attacks. One example is the mbedTLS library [34] , which includes side-channel attack protection for deterministic ECDSA signatures.

3.1.3 Incorrect blockchain magic check. Some blockchains have multiple offshoots, such as the mainnet and one or more testnets. The magical value of the blockchain is used to uniquely identify the chain, thus linking transactions to a specific chain. The node software must check if the transactions received have the expected magical value that is relevant to the current chain. If there is no such check, an attacker can replay a transaction originally executed on a different chain, thereby creating transactions intended for a different chain.

3.1.4 Incorrect verification of the transaction nonce. Each transaction must be unique within a given blockchain. The transaction nonce is used by host implementations to ensure uniqueness. Poor node implementation may allow transactions to be repeated on the same chain. Such a vulnerable implementation could be exploited by an attacker who receives a transaction worth N, which can be replayed over and over again until there is enough funds left in the original wallet to complete the transaction. Risk mitigation - checking the uniqueness of all received transactions.

3.1.5 Denial of Service. Proof-of-Work blockchains with an auto-configurable block target [60] can be vulnerable to a denial of service attack. Indeed, if the minimum target is not defined, uncaught floating point undercompletion can occur, and the target block can be rounded to zero, making it impossible to mine new blocks and render the blockchain useless. Some DApps can become so popular [65] that congestion can occur in the underlying blockchain. One way to avoid network congestion is to use a high-bandwidth smart contract platform [68] .

3.1.6 Public key and address mismatch. The public key of a blockchain wallet can usually be obtained from the wallet address. However, some implementations [3] truncate the public key to obtain an address. If addresses are not tied to a specific key pair, this can be a problem as there are multiple key pairs with the same wallet address. In this case, you can pick up another key pair that controls the target wallet in a reasonable amount of time, depending on the length of the address. To prevent such attacks, you need to make sure that each unique wallet address is tied to a single key pair.

3.2 Vulnerabilities in smart contracts

As of July 2020, Ethereum is still considered the most popular DApp platform [56] . Most DApps use Ethereum and the platform has the largest number of daily active users and smart contracts. After Ethereum, the most widely used are EOSIO [25] and Steem [58] . Ethereum's popularity is most likely due to the fact that it was the first blockchain to offer Turing-complete (excluding gas cap) smart contracts with reasonably low latency (stable average block time of 13 s) acceptable for most decentralized applications. Ethereum smart contracts are usually written in high-level languages such as Solidity [55] or Vyper [64] . Both languages are compiled to EVM bytecode. There are several other blockchains [1, 14, 25, 31, 37, 57] that support Nowa - days smart contracts . The rest of this section lists known smart contract vulnerabilities that can appear on any of the aforementioned smart contract platforms, but uses Ethereum to provide examples [20, 28, 54] .

Re-entry. The re-entry vulnerability [12] can be exploited when the withdrawal function of contract F synchronously calls the default function of another untrusted contract D. Indeed, D could call F again before F updates its state. If F does not update its state before making the outer call to D, then it may be vulnerable (Listing 1). Indeed, a malicious external contract could call the remove () function, which would call the default malicious contract function, which could call remove () again before the balance was updated, thus withdrawing more funds than it should have.

```
1   output function (uint x) {
2         is required (leftovers [msg.sender]> = x);
3         msg.sender.call.value (x) ();
4         balances [msg.sender] - = x;
5   }
```

Listing 1. Vulnerable removew () function for reentry. Lines 3 and 4 should be reversed to eliminate the vulnerability.

In 2016, the smart contract part of the DAO [12] was vulnerable to re-entry and was attacked, which led to the loss of 3.6 million ETH, or about 50 million US dollars at that time. It was this event that caused

Ethereum hard fork. The original Ethereum chain was renamed Ethereum Classic [18] , and the new fork was given the original name Ethereum.

3.2.1 Questions of arithmetic. The Solidity smart contract language does not catch integer overflows by default. If not caught, overflow can lead to unexpected behavior (Listing 2). Unsigned integers are represented in Solidity by 256 bits. Therefore, an unsigned integer arithmetic operation can be used that causes the result to be greater than $2^{256} - 1$ or less than 0 [5, 42, 43] .

```
output function (uint x) {
      is required (leftovers [msg.sender] - x> 0);
      msg.sender.transfer (x);
      balances [msg.sender] - = x;
}
```

Listing 2. Removew () function vulnerable to integer overflow. What happens if x is large?

One solution to this problem is to use safe functions provided by an external smart contract library, such as the SafeMath functions of OpenZeppelin [40], or to use a language that has built-in overflow protection, such as Vyper [64] .

3.2.2 Unprotected SELFOSTRACT. The SELFDESTRUCT EVM instruction renders the smart contract unusable and sends the contract balance to the address specified in the parameter. If the contract has a self-destruct function - EK, then it must be protected with care [67] , and it must be ensured that only authorized users can invoke the code. As a general rule, the use of *self-destruct* should be avoided whenever *possible* . If this cannot be avoided, then it should be used with extreme caution.

3.2.3 Problems with visibility. It is recommended that you explicitly mark the visibility of the function for all functions and variables. The default visibility is public to functions in Solidity. If the visibility of a function is not explicitly marked, then a developer can easily infer that the function is private when in fact it is public, causing unexpected behavior, such as allowing an attacker to invoke allegedly private code.

Likewise, any data that is written to the EVM storage area is visible to everyone because it is stored on the blockchain. Any call to another contract, including function arguments, is also public because it creates a transaction. Secrets should not be stored openly in the EVM storage.

3.2.4 Weak randomness. Generating random numbers in smart contracts is a difficult task [30, 44] . Smart contract developers looking for random numbers may be tempted to use predictable chain data as a source of randomness. Such chaining data includes block number, block hash, and block timestamp. All of these values can be manipulated by block miners and should not be used to generate random numbers. The SmartBillions contract [53] is a famous example of a bad block hash that allowed attackers to exploit the contract and withdraw all lottery money. One possible solution is to use a secure random number generator for smart contracts such as RANDAO [44] or RBGC [16] .

3.2.5 Dependence on transaction order (pre-launch). Deal order dependence can lead to dishonest re - remuneration for labor [9, 52] in the case of smart contracts. Imagine that a smart contract implements a quiz where players are asked to solve a problem. The first player to submit the correct solution to the problem may claim a prize (contract balance). Now suppose Alice worked day and night for a month and finally found a solution to the problem. She commits a transaction to submit her decision. The attacker sees Alice's solution in the transaction pool and immediately sends the same solution, but with a higher transaction fee. The attacker transaction is first picked by the miners because of the higher fees paid by the attacker, and this transaction is inserted into the block before Alice's transaction. Thus, the attacker is the first to solve the problem and can claim a prize instead of

Alice. To prevent this attack, a fix-disclose scheme [26] can be used , which allows players to safely disclose a solution to a problem.

3.3 Education and further reading

There are many educational tools out there on blockchain security. FumbleChain [51] contains tutorials and tasks based on a specially vulnerable blockchain written in Python to enable developers to learn about basic blockchain security. Ethernaut [41] is a wargame focused on the Ethereum smart contract vulnerabilities in Solidity. There are several good articles on blockchain security [2, 17, 46] .

4 CONCLUSION

Selected vulnerabilities related to the main blockchain and smart contracts are described in detail. How these Wool - nerabilities could be used and methods of weakening to prevent such attacks were presented. Blockchains are not immune, and a combination of dedicated smart contract tools [15, 21] and general software analysis tools such as static code analyzers for the underlying blockchain should be used to automatically detect simple problems. Good practices like software testing also apply to blockchain development. To maximize the detection of more complex vulnerabilities, you should conduct a thorough code review or third-party security audits.

REFERENCES

[1] Algorand. 2018. Algorand. Retrieved from https://www.algorand.com/.

[2] Ayman Alkhalifah, Alex Ng, A. S. M. Kayes, Mohammad Chowdhury, MamounAlazab, and Paul Watters. 2019. A Taxonomy of Blockchain Threats and Vulnerabilities. DOI: https://doi.org/10.20944/preprints201909.0117.v1

[3] J. P. Aumasson. 2018. Blockchains: How to Steal Millions in 2**64 Operations. Retrieved from https://research.kudelskisecurity.com/2018/01/16/blockchains-how-to-steal-millions-in-264-operations/.

[4] Various authors. 2013. Go Ethereum—Official Go implementation of the Ethereum protocol. Retrieved from Retrieved from https://github.com/ethereum/go-ethereum.

[5] Eric Banisadr. 2018. How 800k Evaporated from the PoWH Coin Ponzi Scheme Overnight. Retrieved from https://medium.com/@ebanisadr/how-800k-evaporated-from-the-powh-coin-ponzi-scheme-overnight-1b025c33b530?.

[6] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2011. High-speed high-security signatures. *Cryptology ePrintArchive, Report 2011/368.*

Retrieved from https://eprint.iacr.org/2011/368.

[7] Bitpay. 2016. Bitpay. Retrieved from https://bitpay.com/.

[8] Daniel Bleichenbacher. 2000. Generating ElGamal signatures without knowing the secret key.https://crypto.ethz.ch/publications/files/Bleich96.pdf.

[9] Ivan Bogatyy. 2017. Implementing Ethereum trading front-runs on the Bancor exchange in Python. Retrieved from https://hackernoon.com/front-running-bancor-in-150-lines-of-python-with-ethereum-api-d5e2bfd0d798.

[10] Joachim Breitner and Nadia Heninger. 2019. Biased Nonce Sense: Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies. *Cryptology ePrintArchive, Report 2019/023.* Retrieved from https://eprint.iacr.org/2019/023.

[11] VitalikButerin. 2015. Ethereum. Retrieved from https://ethereum.org/.

[12] VitalikButerin. 2016. Critical Update Re: DAO Vulnerability. Retrieved from https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/.

[13] VitalikButerin. 2019. Hard Problems in Cryptocurrency: Five Years Later. Retrieved from https://vitalik.ca/general/2019/11/22/progress.html.

[14] Cardano. 2014. Cardano. Retrieved from https://www.cardano.org/en/home/.

[15] ChainSecurity. 2018. Chaincode Scanner. Retrieved from https://chaincode.chainsecurity.com/.

[16] Krishnendu Chatterjee, Amir KafshdarGoharshady, and ArashPourdamghani. 2019. Probabilistic smart contracts: Secure randomness on the blockchain. In *Proceedings of the IEEE International Conference on Blockchain and Cryptocurrency (ICBC'19).* DOI: https://doi.org/10.1109/bloc.2019.8751326

[17] *Vincent Chia, Pieter Hartel, Qingze Hum, Sebastian Ma, Georgios Piliouras, DaniëlReijsbergen, Mark Van Staalduinen, and PawelSzalachowski. 2018. Rethinking blockchain security: Position paper. In Proceedings of the IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) andIEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData'18). IEEE, 1273-1280.*

[18] Ethereum classic. 2016. Ethereum Classic. Retrieved from https://ethereumclassic.org/roadmap.

[19] Coinbase. 2018. Coinbase Commerce. Retrieved from

https://commerce.coinbase.com/.

[20] ConsenSys. 2016. Known Attacks. Retrieved from https://consensys.github.io/smart-contract-best-practices/known_attacks/.

[21] ConsenSys. 2016. Smart contract security tools. Retrieved from https://consensys.github.io/smart-contract-best-practices/security_tools/.

[22] Nicolas T. Courtois, Filippo Valsorda, and Pinar Emirdag. 2014. Private Key recovery combination attacks: On extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor RNG events.https://eprint.iacr.org/2014/848.

[23] Crypo. 2018. PoW 51% Attack Cost. Retrieved from https://www.crypto51.app/.

[24] T. ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Info. Theory* 31, 4 (July 1985), 469-472. DOI: https://doi.org/10.1109/TIT.1985.1057074

[25] EOS. 2018. EOSIO. Retrieved from https://eos.io/.

[26] ShayanEskandari, SeyedehmahsaMoosavi, and Jeremy Clark. 2019. SoK: Transparent Dishonesty: front-running attacks on Blockchain.https://arxiv.org/abs/1902.05164.

[27] P. Fouque, D. Masgana, and F. Valette. 2009. Fault attack on Schnorr-based identification and signature schemes. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'09*). 32-38. DOI: https://doi.org/10.1109/FDTC.2009.36

[28] NCC Group. 2018. Decentralized Application Security Project. Retrieved from https://dasp.co/.

[29] R. A. Haraty, H. Otrok, and A. Nasser Kassar. 2005. Attacking ElGamal-based cryptographic algorithms using Pollard's rho algorithm. In *Proceedings of the 3rd ACS/IEEE International Conference onComputer Systems and Applications, 2005.91.* DOI: https://doi.org/10.1109/AICCSA.2005.1387082

[30] Tjaden Hess and NiklasFeurstein. 2016. How can I securely generate a random number in my smart contract? Retrieved from https://ethereum.stackexchange.com/questions/191/how-can-i-securely-generate-a-random-number-in-my-smart-contract/207#207.

[31] Hyperledger. 2016. Hyperledger Fabric. Retrieved from https://www.hyperledger.org/projects/fabric.

[32] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Info. Secur.* 1, 1 (Aug. 2001), 36-63. DOI:https://doi.org/10.1007/s102070100002

[33] Ledger. 2019. Ledger Nano X. Retrieved from Retrieved from https://shop.ledger.com/pages/ledger-nano-x.

[34] ARM Limited. 2009. mbed TLS. Retrieved from https://tls.mbed.org/.

[35] Hiraku Morita, Jacob C. N. Schuldt, Takahiro Matsuda, Goichiro Hanaoka, and Tetsu Iwata. 2015. On the Security of the Schnorr Signature Scheme and DSA against Related-Key Attacks. *Cryptology ePrint Archive, Report 2015/1135.* Retrieved from https://eprint.iacr.org/2015/1135.

[36] Satoshi Nakamoto. 2009. Bitcoin: A peer-to-peer electronic cash system. Cryptography Mailing list. Retrieved from https://metzdowd.com.

[37] Neo. 2014. Neo. Retrieved from https://neo.org/.

[38] ocminer. 2018. Network Attack on XVG / VERGE. Retrieved from https://bitcointalk.org/index.php?topic=3256693.0.

[39] State of the DApps. 2014. State of the DApps. Retrieved from https://www.stateofthedapps.com/.

[40] OpenZeppelin. 2016. OpenZeppelin. Retrieved from https://github.com/OpenZeppelin/openzeppelin-contracts.

[41] OpenZeppelin. 2017. Ethernaut. Retrieved from https://ethernaut.openzeppelin.com/.

[42] PeckShield. 2018. New batchOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018a10299). Retrieved from https://medium.com/@peckshield/alert-new-batchoverflow-bug-in-multiple-erc20-smart-contracts-cve-2018-10299-511067db6536.

[43] pirapira. 2016. Exception on overflow. Retrieved from https://github.com/ethereum/solidity/issues/796#issuecomment-253578925.

[44] RANDAO. 2015. RANDAO: A DAO working as RNG ofEthereum. Retrieved from https://github.com/randao/randao.

[45] Y. Romailler and S. Pelissier. 2017. Practical fault attack against the Ed25519 and EdDSA signature schemes. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'17).* 17-24. DOI: https://doi.org/10.1109/FDTC.2017.12

[46] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHunNyang, and Aziz Mohaisen. 2019. Exploring the attack surface of blockchain: A systematic overview. Retrieved from https://arXiv:1904.03487.

[47] Jorn-Marc Schmidt and Marcel Medwed. 2009. A fault attack on ECDSA. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'09).* 93-99. DOI: https://doi.org/10.1109/FDTC.2009.38

[48] Claus Schnorr. 1991. Efficient signature generation by smart cards. *J. Cryptol.* 4 (Jan. 1991), 161-174. DOI:https://doi.org/10.1007/BF00196725

[49] Eric Schorn. 2018. Smart Contract (in)Security—Bad Arithmetic. Retrieved from https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2018/november/smart-contract-insecurity-bad-arithmetic/.

[50] Eric Schorn. 2018. Smart Contract (in)Security—Fat Fingers. Retrieved from https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2018/december/smart-contract-insecurity-fat-fingers/.

[51] Kudelski Security. 2019. FumbleChain: A Purposefully Vulnerable Blockchain. Retrieved from https://github.com/kudelskisecurity/fumblechain.

[52] Emin Gun Sirer and Phil Daian. 2017. Bancor Is Flawed. Retrieved from http://hackingdistributed.com/2017/06/19/bancor-is-flawed/.

[53] SmartBillions. 2017. SmartBillions—World's first multi-billion-dollar decentralized global blockchain lottery. Retrieved from https://twitter.com/smartbns.

[54] SmartContractSecurity. 2018. Smart Contract Weakness Classification and Test Cases. Retrieved from https://swcregistry.io/.

[55] Solidity. 2014. Solidity. Retrieved from https://solidity.readthedocs.io/.

[56] DApp Statistics. 2014. DApps Statistics. Retrieved from https://www.stateofthedapps.com/stats/.

[57] Steem. 2016. Steam—Write a smart contract. Retrieved from https://github.com/harpagon210/steemsmartcontracts-wiki/blob/master/Write-a-

Smart-Contract.md.

[58] Steem. 2016. Steem. Retrieved from https://steem.com/.

[59] Parity Technologies. 2015. Parity—The fast, light, and robust EVM and WASM client. Retrieved from https://github.com/paritytech/parity-ethereum.

[60] Theymos. 2010. Block target and difficulty. Retrieved from https://en.bitcoin.it/wiki/Target.

[61] Pierre Tholoniat and Vincent Gramoli. 2019. Formal Verification of Blockchain Byzantine Fault Tolerance. Retrieved from https://arxiv:1909.07453.

[62] Trezor. 2018. TrezorModelT. Retrieved from https://shop.trezor.io/product/trezor-model-t.

[63] Unknown. 2018. EtherVM - Ethereum Virtual Machine Opcodes. Retrieved from https://ethervm.io/.

[64] Vyper. 2016. Vyper. Retrieved from https://vyper.readthedocs.io/en/latest/index.html.

[65] Joon Ian Wong. 2017. The ethereum network is getting jammed up because people are rushing to buy cartoon cats on its blockchain. Retrieved from https://qz.com/1145833/cryptokitties-is-causing-ethereum-network-congestion/.

[66] Joseph Young. 2018. Another ICO Hacked: KICKICO Loses 8 Million After Smart Contract Breach. Retrieved from https://finance.yahoo.com/news/another-ico-hacked-kickico-loses-120331205.html.

[67] Michael Yuan. 2017. I accidentally killed it (and evaporated 300 million). Retrieved from https://medium.com/cybermiles/i-accidentally-killed-it-and-evaporated-300-million-6b975dc1f76b.