SECURITY AUDIT OF TOMOCHAIN SMART CONTRACTS

PUBLIC REPORT

NOV 30, 2018

VeriChains Lab

info@verichains.io

https://www.verichains.io

Driving Technology >> Forward

EXECUTIVE SUMMARY

This private Security Audit Report prepared by VeriChains Lab on Nov 30, 2018. We would like to thank Tomo Chain to trust VeriChains to audit smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the smart contracts. It was conducted on commit *f84175422d79a153a4523872ae316dd502f7c38d* of branch *master* from GitHub repository tomomaster of TomoChain.

Overall, while there are a few security issues, the audited code demonstrates good code quality standards adopted and use of modularity and security best practices.

CONTENTS

ACRONYMS AND ABBREVIATIONS

Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum / Tomochain platform.
Solc	A compiler for Solidity.
EVM	Ethereum Virtual Machine.

AUDIT OVERVIEW

ABOUT TOMO CHAIN

TomoChain is an innovative solution to scalability problem with the Ethereum blockchain, and other blockchain platforms. TomoChain features a 150-Masternodes architecture with Proof of Stake Voting (POSV) consensus for near-zero fee, and instant transaction confirmation.

TomoChain supports all EVM-compatible smart-contracts, protocols, and atomic cross-chain token transfers.

The website of TomoChain is at https://tomochain.com/
White paper (EN) is at https://tomochain.com/docs/technical-whitepaper--1.0.pdf

SCOPE OF THE AUDIT

This audit only focused on identifying security flaws in code and the design of the **TomoChain's smart contracts**. It was conducted on commit *`f84175422d79a153a4523872ae316dd502f7c38d`* of branch *`master`* from GitHub repository `tomomaster` of TomoChain.

Contract source code repository is at:
https://github.com/tomochain/tomomaster/tree/f84175422d79a153a4523872ae316dd502f7c38d/contracts

The scope of the audit is limited to the following 5 source code files:
- contracts/BlockSigner.sol
- contracts/Migrations.sol
- contracts/TomoRandomize.sol
- contracts/TomoValidator.sol
- contracts/libs/SafeMath.sol

This audit **does not** include the security review of (Out of scope)
- Technical whitepaper
- Design and implementation of Tomochain's concensus
- Tomochain's architecture and infrastruture
- Implementation code of TomoChain's client for running masternodes and full-nodes

AUDIT METHODOLOGY

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and Verichains's in-house automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- TimeStamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- Dos with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories, depending on their criticality:

Low	An issue that does not have a significant impact, can be considered as less important
Medium	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
High	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
Critical	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.

AUDITED CONTRACTS OVERVIEW

SAFEMATH

This is library contract for safe integer operations handling from OpenZeppelin.

BLOCKSIGNER.SOL

This is contract for managing blocks' signers, contains only 2 simple functions for a node to sign a block and for everyone to obtain the block's signers.

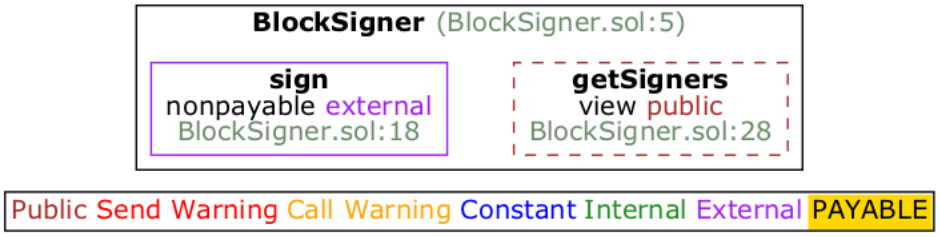

BlockSigner (BlockSigner.sol:5)
sign
nonpayable external
BlockSigner.sol:18
getSigners
view public
BlockSigner.sol:28
Public Send Warning Call Warning Constant Internal External PAYABLE

Figure 1 BlockSinger contract methods

MIGRATIONS.SOL

Contract for managing migrations, created by truffle.

TOMORANDOMIZE.SOL

This is contract for managing randomization seeds for Blocks Verifiers for Double Validation, contains setters for **secret** (the encrypted seed) which only available from block 800 to block 849 of an epoch and **opening** (the key for encrypted seed) which only available after block 850 (an epoch is 900 blocks).

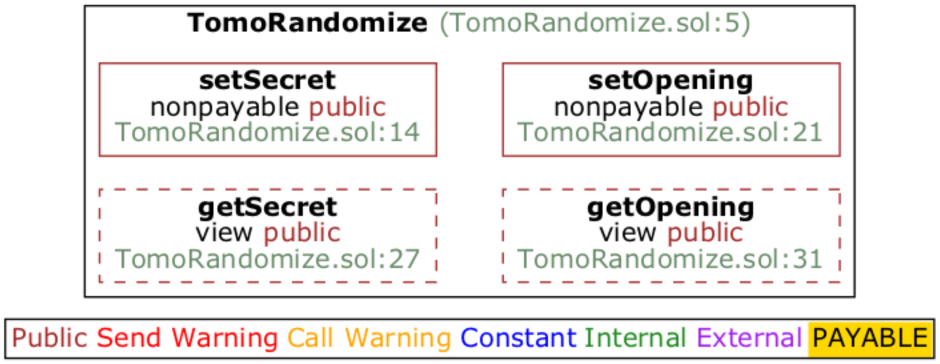


Figure 2 TomoRandomize contract methods

TOMOVALIDATOR.SOL

This is contract for managing master node candidates and voting. Master nodes candidates are either created at contract construction or proposed using **propose** method and can be resigned using **resign** method which will refund all his stake. Everyone can vote for a candidate using **vote** method and can undo anytime by using **unvote** method to get his fund back, the voting weights are stored in *validatorsState[_candidate].voter[_address]*. All refund mechanic is implemented using claim-later method by storing claimable balance at *withdrawsState[_address].caps[_blockNumber]*.

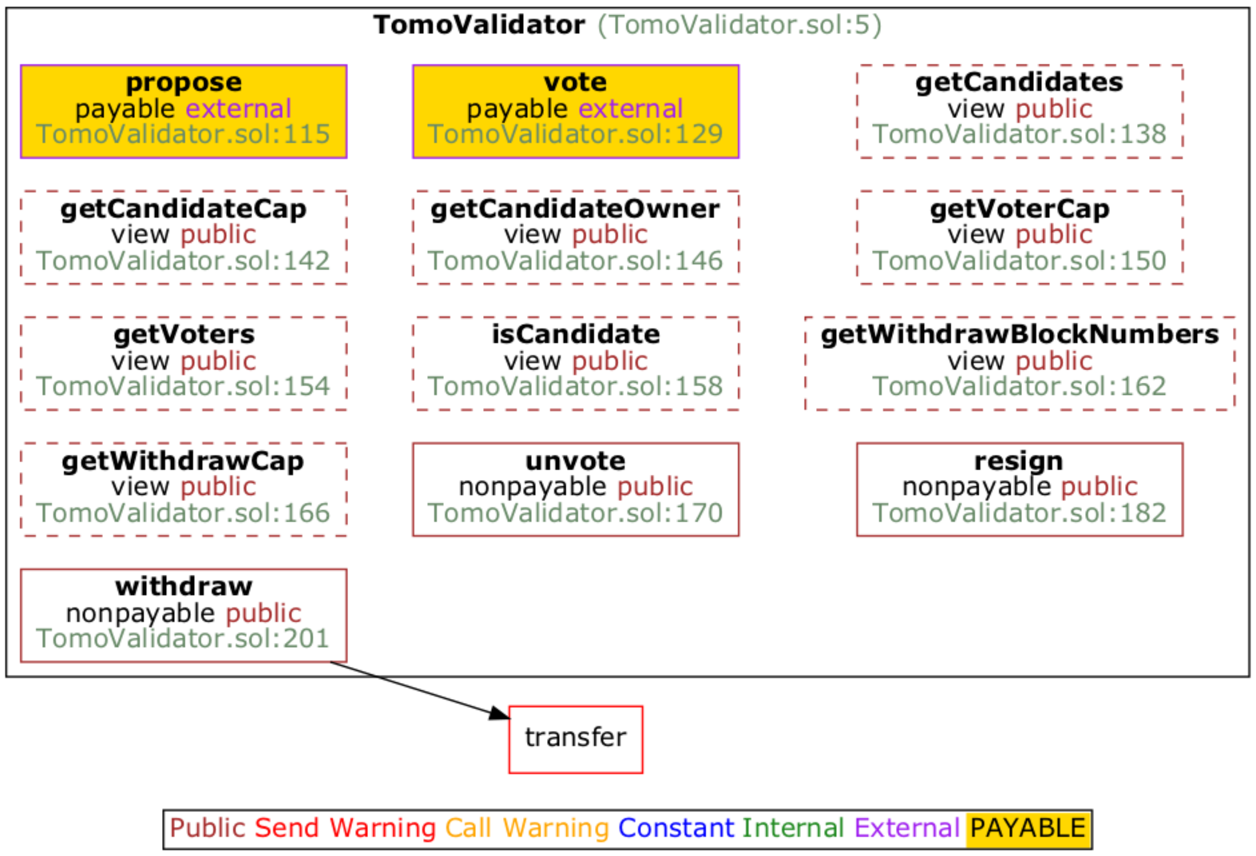

TomoValidator (TomoValidator.sol:5)
propose payable external TomoValidator.sol:115
vote payable external TomoValidator.sol:129
getCandidates view public TomoValidator.sol:138
getCandidateCap view public TomoValidator.sol:142
getCandidateOwner view public TomoValidator.sol:146
getVoterCap view public TomoValidator.sol:150
getVoters view public TomoValidator.sol:154
isCandidate view public TomoValidator.sol:158
getWithdrawBlockNumbers view public TomoValidator.sol:162
getWithdrawCap view public TomoValidator.sol:166
unvote nonpayable public TomoValidator.sol:170
resign nonpayable public TomoValidator.sol:182
withdraw nonpayable public TomoValidator.sol:201
transfer
Public Send Warning Call Warning Constant Internal External PAYABLE

Figure 3 TomoValidator contract methods

AUDIT RESULT

VULNERABILITIES FINDINGS

✓ FIXED **HIGH**
TOMOVALIDATOR VOTE WEIGHT COULD BE RESET AFTER RESIGN

If a candidate is resigned and one of his previous voters propose the candidate again, the voter's weight will be reset to the new value, resulting in the loss of deposit TOMO coins in previous votes.

```
123:      validatorsState[_candidate].voters[msg.sender] = msg.value;
```

RECOMMENDED FIXES
Change the code to retain the previously voted amount

```
123:     validatorsState[_candidate].voters[msg.sender] =
                      validatorsState[_candidate].voters[msg.sender].add(msg.value);
```

NO-FIX **LOW**
TOMOVALIDATOR PARAMETERS VALIDATION

Some caps variables in **TomoValidator**'s constructor are assigned without being checked, which may cause problems if deployed with invalid values of *_minCandidateCap* and *_minVoterCap*. If these variables are initialized with zero, there will be some problems with TomoValidator:

- *voters[address]* array may have duplicated values:

```
129:    function vote(address _candidate) external payable onlyValidVoterCap
onlyValidCandidate(_candidate) {
        validatorsState[_candidate].cap = validatorsState[_candidate].cap.add(msg.value);
        if (validatorsState[_candidate].voters[msg.sender] == 0) {
            voters[_candidate].push(msg.sender);
        }
        validatorsState[_candidate].voters[msg.sender] =
validatorsState[_candidate].voters[msg.sender].add(msg.value);
        emit Vote(msg.sender, _candidate, msg.value);
    }
```

- If attacker keeps calling vote with *value = 0*, *voters[_candidate]* will keep increasing, at some point that the array will become too large that getVoters view cannot return the array.

- The ability to increase the array to arbitrary size also increasing the possibility to overwrite other storage variables' memory **theoretically**. However, practically it is very low probability that this could happen.

RECOMMENDED FIXES

- Add validations to the constructor.

Update: No fix for this issue. Tomochain shall make sure that TomoValidator is deployed with valid initialized values.

LOW
OTHER RECOMENDATIONS

- **✓ FIXED** Initialization of *candidateCount* in constructor of **TomoValidator** contains unnecessary safe-math operation

```
        candidateCount = candidateCount.add(_candidates.length);
```

  which should be changed into

```
        candidateCount = _candidates.length
```

- **✓ FIXED** **TomoValidator**'s contract constructor contains a copy operation from *_candidates* parameter to *candidates* storage variable with push operation:

```
        for (uint256 i = 0; i < _candidates.length; i++) {
            candidates.push(_candidates[i]);
```

  Above copy method consumes more gas than nesseary because push operation increase the length field within each loop and can be optimized into:

```
        candidates.length = _candidates.length;
        for (uint256 i = 0; i < _candidates.length; i++) {
            candidates[i] = _candidates[i];
```

- **✓ FIXED** Within the loop of **TomoValidator**’s contract constructor, the last expression uses *candidates* variable as the array reference parameter while all other operations use *_candidates*. Please make sure the use of variable is consistent.

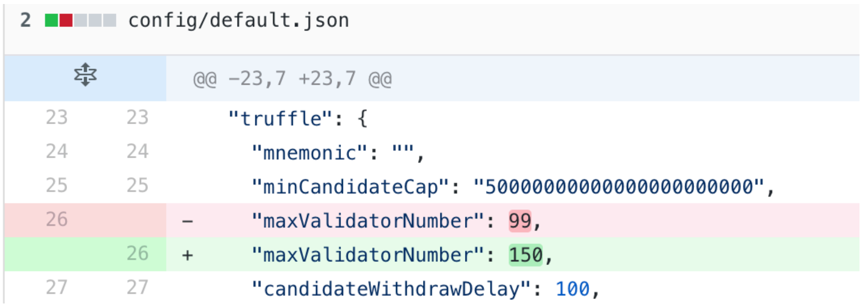
```
111:            validatorsState[candidates[i]].voters[_firstOwner] = minCandidateCap;
```

- **NO-FIX** There are inconsistency in the maxinum number of Masternodes from Tomochain code and technical whitepaper. Please update the paper and code to make it consistent.

  - *_maxValidatorNumber* is initialized with the value of the constant **99** in test script

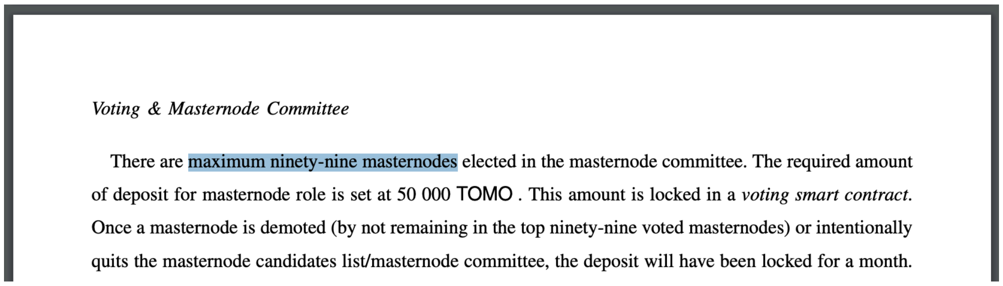
```
tomomaster/test/TomoValidatorTest.js
11:         let validator = await TomoValidator.new([], [], null, minCandidateCap,
minVoterCap, 99, 100, 100)
```

  - *maxValidatorNumber* is configured with the value of the constant **150** in mainnet config (changed from **99** to **150** in commit *d88492b95c0b7c91dffea89e2a9dbecb930f40c1* on Aug 10, 2018)



```
2 config/default.json
@@ -23,7 +23,7 @@
     "truffle": {
       "mnemonic": "",
       "minCandidateCap": "50000000000000000000000",
-      "maxValidatorNumber": 99,
+      "maxValidatorNumber": 150,
       "candidateWithdrawDelay": 100,
```

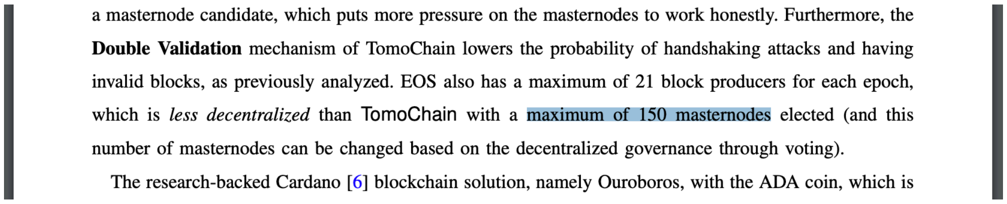
  - TomoChain's technical white paper 1.0, page 5



Voting & Masternode Committee

There are maximum ninety-nine masternodes elected in the masternode committee. The required amount of deposit for masternode role is set at 50 000 TOMO . This amount is locked in a *voting smart contract*. Once a masternode is demoted (by not remaining in the top ninety-nine voted masternodes) or intentionally quits the masternode candidates list/masternode committee, the deposit will have been locked for a month.

- TomoChain's technical white paper 1.0, page 17



a masternode candidate, which puts more pressure on the masternodes to work honestly. Furthermore, the **Double Validation** mechanism of TomoChain lowers the probability of handshaking attacks and having invalid blocks, as previously analyzed. EOS also has a maximum of 21 block producers for each epoch, which is *less decentralized* than TomoChain with a maximum of 150 masternodes elected (and this number of masternodes can be changed based on the decentralized governance through voting).

The research-backed Cardano [6] blockchain solution, namely Ouroboros, with the ADA coin, which is

- **NO-FIX** Solidity contracts can have a special form of comments that form the basis of the Ethereum Natural Specification Format. Please consider changing the comments inside smart contract following https://github.com/ethereum/wiki/wiki/Ethereum-Natural-Specification-Format.

CONCLUSION

Tomo Chain smart contracts have been audited by VeriChains Lab using various public and in-house analysis tools and intensively manual code review. Overall, while there are a few security issues, the audited code demonstrates good code quality standards adopted and use of modularity and security best practices.

LIMITATIONS

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.