Smart Contract Security Audit Report

DOS Network Token

SECBIT

Mar 9, 2019

1. *Introduction*

DOS Network Token is a token contract deployed on Ethereum. SECBIT Labs conducted an audit from Mar 6th, 2019 to Mar 9th, 2019, including an analysis of the contract in 3 areas: **code bugs**, **logic flaws** and **risk assessment**. The assessment shows that DOS Network Token contract has no critical security risks, and SECBIT team has some tips on logical implementation, potential risks and code revising(see part 4 for details).

Type	Description	Level	Final Status
Implementation Vulnerability	The `approve()` function has a certain security risk and may be exploited maliciously in certain scenarios	Medium	Fixed
Implementation Vulnerability	In `transferFrom()` function, the locked tokens can be transferred	High	Fixed
Implementation Vulnerability	When `perNodeLockedAmount` is changed, `lockedSupply` may not match the actual amount of total locked tokens	Medium	Fixed
Logic Vulnerability	The implementation of `transferFrom()` and `approve()` functions does not conform to the logic of the transfer and approve operations	Medium	Fixed

2. Contract Information

This part describes basic contract information and code structure.

2.1 Basic Information

The following list shows basic information of DOS Network Token:

Name	DOS Network Token
Symbol	DOS
Address	0x70861e862e1ac0c96f853c8231826e469ead37b1
Line	825
Initial Version	GitHub (commit e196a17)
Final Version	GitHub (commit e6a5938b)
Stage	Deployed

2.2 Contract List

The following content shows the contracts included in DOS Network Token project:

Name	Lines	Description
DSAuthEvents	4	Events in DSAuth contract
DSAuthority	5	Interface for external authority management
DSAuth	42	Authoity management for DSAuth
DSNote	26	Note of events
DSMath	67	Library for safe math calculation
DSStop	14	Stop function for contract
ControllerManager	40	Management of controllers
ERC20	12	ERC20 interface
Managed	26	Authority Management
TokenController	17	Interface for external control of token operations
LockdropController	49	Contract with lock and drop functions
DOSToken	109	Main contract

3. Contract Analysis

This part describes details of contract code assessment, including 3 items: sum of tokens, authorities of contract accounts, functions of the contract.

3.1 TotalSupply

The sum of tokens in the contract is initialized to 1,000,000,000 and mutable. Here is a conclusion of ways to change the sum of tokens in the contract:

- Increasing

  Owner and authorized accounts could mint tokens, the balance of the account and the totalSupply would increase accordingly.

- Decreasing

Owner and authorized accounts could burn tokens from itself or approved accounts within the balance of the account and the approved amount. The balance of the corresponding account and the totalSupply would decrease accordingly.

3.2 Contract Account

There are 5 types of accounts in DOS Network Token: common account, approved account, authorized account, owner and manager.

- Common Account
  - Description

    All accounts holding DOS Network Token
  - Authority
    - Transfer tokens in its own balance
    - Authorize other accounts to transfer its own token balance
  - Method of Authorization

    Every account can be the common account
- Approved Account
  - Description

    Accounts authorized to transfer tokens from other accounts
  - Authority
    - Transfer tokens from other accounts within the approved allowance
  - Method of Authorization

    Approved by other accounts
- Authorized account
  - Description

    Accounts authorized to perform certain actions in the contract
  - Authority
    - All common accounts' authorities
    - Mint tokens to any account
    - Burn tokens from itself or other accounts within the approved allowance
    - Set owner
    - Set authority contract
    - Lock and release tokens
  - Method of Authorization

    Owner or accounts authorized by the authority contract

- Owner
  - Description

    The owner of the contract, creator of the contract
  - Authority
    - All authorized accounts' authorities
  - Method of Authorization

    The creator of the contract, or set by the authorized account
- Manager
  - Description

    The owner of the contract, creator of the contract
  - Authority
    - All common accounts' authorities
    - Adjust token tranfer and approve amount when set as a contract
  - Method of Authorization

    The creator of the contract, or set by manager

3.3 Feature Analysis

As a token contract, DOS Network Token meets with ERC20 contract standards and implements additional functions. We can divide the key contract features into several parts:

- Transfer

  Any account can perform a transfer of the balance in its account.
- Approve

  Approve other accounts to manage part of the balances.
- TransferFrom

  The approved account can transfer tokens within the allowance in the corresponding account.
- Mint

  Authorized accounts can mint tokens into any account.
- Burn

  Authorized accounts can burn tokens within its own balance or within the approved allowance of other accounts.
- Lock and drop

  Authorized accounts can lock and drop tokens in any account.

4. Audit Detail

This part describes the process and detailed results of the audit, also demonstrates the problems and potential risks.

4.1 Audit Process

The audit strictly followed the audit specification of SECBIT Lab. We analyzed the project from code bug, logical implementation and potential risks. The process consists of four steps:

- Fully analysis of contract code line by line.
- Evaluation of vulnerabilities and potential risks revealed in the contract code.
- Communication on assessment and confirmation.
- Audit report writing.

4.2 Audit Result

After scanning with SECBIT Solidity Static Analysis Extension & sf-checker (internal version) developed by SECBIT Labs and Mythril, the auditing team performed a manual assessment. The team inspected the contract line by line and the result could be categorized into twenty-one types:

Number	Classification	Result
1	Normal functioning of features defined by the contract	✓
2	No obvious bug (e.g. overflow, underflow)	✓
3	Pass Solidity compiler check with no potential error	✓
4	Pass common tools check with no obvious vulnerability	✓
5	No obvious gas-consuming operation	✓
6	Meet with ERC20	✓
7	No risk in low level call (call, delegatecall, callcode) and in-line assembly	✓
8	No deprecated or outdated usage	✓

9	Explicit implementation, visibility, variable type and Solidity version number	✓
10	No redundant code	✓
11	No potential risk manipulated by timestamp and network environment	✓
12	Explicit business logic	✓
13	Implementation consistent with annotation and other info	✓
14	No hidden code about any logic that is not mentioned in design	✓
15	No ambiguous logic	✓
16	No risk threatening the developing team	✓
17	No risk threatening exchanges, wallets and DApps	✓
18	No risk threatening token holders	✓
19	No privilege on managing others' balances	✓
20	No minting method	×
21	Correct managing hierarchy	✓

4.3 Issues

- **The `approve()` function has a certain security risk and may be exploited maliciously in certain scenarios.**
    - Level: **Medium**
    - Type: Implementation Vulnerability
    - Description:

      The `approve()` function directly adjusts the approved allowance by updating the value in `_approvals`. The approved account may launch `re-approval` attack, that is, the approved account can spend repeatedly the approved allowance by controlling the order of transactions in the miners' block.

      The code related to the problem is as follows:

```
function approve(address guy, uint wad) public stoppable
returns (bool) {
    // Adjust token approve amount if necessary.
    if (isContract(manager)) {
        wad =
ControllerManager(manager).onApprove(msg.sender, guy, wad);
        require(wad > 0, "approve-disabled-by-
ControllerManager");
    }

    _approvals[msg.sender][guy] = wad;

    emit Approval(msg.sender, guy, wad);

    return true;
}
```

- Consequence:

  The approved account may launch `re-approval` attack and influence the interest of common accounts.

- Suggestion:

  It is recommended to modify `approve()` function to require the value to be 0 when modifying `allowance`. Or add `increaseApproval()` and `decreaseApproval()` functions to avoid `re-approval` attacks.

- Status:

  Fixed.

- **In `transferFrom()` function, the locked tokens can be transferred.**
  - Level: **High**
  - Type: Implementation Vulnerability
  - Description:

    In `transferFrom()` function, The transfer amount is only checked after `onTransfer()` function is executed. The user can transfer the amount beyond the balance of his account and then break the limit of locked tokens.

    The code related to the problem is as follows:

```
function transferFrom(address src, address dst, uint wad)
public stoppable returns (bool) {
    // Adjust token transfer amount if necessary.
    if (isContract(manager)) {
        wad = ControllerManager(manager).onTransfer(src,
dst, wad);
```

```
        require(wad > 0, "transfer-disabled-by-
ControllerManager");
    }

    if (src != msg.sender && _approvals[src][msg.sender] !=
uint(-1)) {
        require(_approvals[src][msg.sender] >= wad, "token-
insufficient-approval");
        _approvals[src][msg.sender] = sub(_approvals[src]
[msg.sender], wad);
    }

    require(_balances[src] >= wad, "token-insufficient-
balance");
    _balances[src] = sub(_balances[src], wad);
    _balances[dst] = add(_balances[dst], wad);

    emit Transfer(src, dst, wad);

    return true;require(_balances[src] >= wad, "token-
insufficient-balance");`
}
```

  - Consequence:

    The locked tokens can be transferred, which affects the interests of the project party.

  - Suggestion:

    It is recommended to check the amount of tokens to transfer before the execution of `onTransfer()`. The suggested code is as follows:

    `require(_balances[src] >= wad, "token-insufficient-balance");`.

  - Status:

    Fixed according to the suggestion.

- **When `perNodeLockedAmount` is changed, `lockedSupply` may not match the actual amount of total locked tokens.**

  - Level: **Medium**
  - Type: Implementation Vulnerability
  - Description:

    The calculation of `lockedSupply` is accumulated according to the amount of each user's lock. When `perNodeLockedAmount` is changed, the amount of each user's actual locked tokens changes, and `lockedSupply` does not change accordingly.

- Consequence:

  `lockedSupply` does not match the actual amount of total locked tokens.

- Suggestion:

  It is recommended to increase the variable to record each user's locked amount or the total number of locked users. Then the calculation of `lockedSupply` can be consistent with the amount of total locked tokens.

- Status:

  Fixed according to the suggestion. `lockedNode` is added to count locked accounts.

- **The implementation of `transferFrom()` and `approve()` functions does not conform to the logic of the transfer and approve operations.**

  - Level: **Medium**
  - Type: Logic Vulnerability
  - Description:

    In `transferFrom()` and `approve()` functions, The amount of tokens to transfer or approve is reduced by `onTransfer()` and `onApprove()` functions. Then the reduced amount is the actual amount to transfer and approve. The amount of actual transferred and approved tokens is inconsistent with the initial amount.

  - Consequence:

    The logic implementation of function is unclear and easy to be misunderstood by users

  - Suggestion:

    It is recommended to increase the function to calculate the total amount of locked tokens of users. Transfer can be checked with balance and the amount of locked tokens of accounts.

    The recommended code is as follows:

```
function calLocked(address _from) public view returns(uint)
{
    if (lockdropList[_from]) {
        return perNodeLockedAmount;
    } else {
        return 0;
    }
}

function calAllLocked(address _from) public returns(uint) {
    uint locked = 0;
```

```
    for (uint i = 0; i < controllers.length; i++) {
        locked =
add(locked,TokenController(controllers[i]).calLocked(_from))
;
    }
    return locked;
}

function transferFrom(address src, address dst, uint wad)
public stoppable returns (bool) {
    // Adjust token transfer amount if necessary.
    locked = 0;
    if (isContract(manager)) {
        locked =
ControllerManager(manager).calAllLocked(src);
    }

    require(_balances[src] >= add(locked, wad), "token-
insufficient-balance");

    if (src != msg.sender && _approvals[src][msg.sender] !=
uint(-1)) {
        require(_approvals[src][msg.sender] >= wad, "token-
insufficient-approval");
        _approvals[src][msg.sender] = sub(_approvals[src]
[msg.sender], wad);
    }

    _balances[src] = sub(_balances[src], wad);
    _balances[dst] = add(_balances[dst], wad);

    emit Transfer(src, dst, wad);

    return true;
}

function approve(address guy, uint wad) public stoppable
returns (bool) {
    // Adjust token approve amount if necessary.
    locked = 0;
    if (isContract(manager)) {
        locked =
ControllerManager(manager).calAllLocked(msg.sender);
    }

    require(_balances[src] >= add(locked, wad), "token-
insufficient-balance");

    _approvals[msg.sender][guy] = wad;
```

```
        emit Approval(msg.sender, guy, wad);

        return true;
    }
```

- Status:

  Fixed.

4.4 Risks

SECBIT team found the following risk after assessing DOS Network Token contract:

- Authorized accounts has the authority to mint tokens, which may influence the exchange's decision on listing the token.
  - Level: **Medium**
  - Description:

    Authorized accounts has the authority to mint and burn tokens, which may not meet the requirements of some exchanges. If necessary, consider removing the function of minting and burning tokens.

5. Conclusion

SECBIT team had found no critical code bug or flaw after analyzing DOS Network Token contract. DOS Network Token implements common token functions(transfer, approve) with additional functions(mint, burn, Lock and drop) by specific project targets. The contract reveals 4 code issues and 1 potential risk as demonstrated above. Meanwhile, SECBIT Labs holds the view that the code of DOS Network Token is of high quality.

Disclaimer

SECBIT smart contract audit service assesses the contract's correctness, security and performability in code quality, logic design and potential risks. The report is provided "as is", without any warranties about the code practicability, business model, management system's applicability and anything related to the contract adaptation. This audit report is not to be taken as an endorsement of the platform, team, company or investment.

APPENDIX

Vulnerability/Risk Level Classification

Level	Description
High	Severely damage the contract's integrity and allow attackers to steal ethers and tokens, or lock ethers inside the contract.
Medium	Damage contract's security under given conditions and cause impairment of benefit for stakeholders.
Low	Cause no actual impairment to contract.
Info	Relevant to practice or rationality of the contract, could possibly bring risks.

SECBIT Lab is devoted to construct a common-consensus, reliable and ordered blockchain economic entity.

 http://www.secbit.io

 audit@secbit.io

 @secbit_io