Status Network Token

June, 2017

By Pablo Yabo and Ismael Bejarano

Introduction 3

Summary of discoveries 3

Audit details 3

Reviewed files: 3

Non reviewed files 4

Contracts Relationship Diagram 4

Individual files 5

ContributionWallet.sol 5

DevTokenHolder.sol 5

DynamicCeiling.sol 5

Owned.sol 5

SafeMath.sol 5

SGT.sol (Status Genesis Token) 6

SGTExchanger.sol 6

SNT.sol (Status Network Token) 6

SNTPlaceHolder.sol 6

StatusContribution.sol 6

Conclusions 7

References 8

Introduction

CoinFabrik was hired to audit contracts written by Status.im. The result of this security review is reflected in this document.

The base of the audited contracts are from https://github.com/status-im/status-network-token at commit 2152b17aa2ef584a2aea95533c707a345c6ccf69.

Summary of discoveries

The tokens are based in the MiniMeToken.sol (https://github.com/Giveth/minime).

A non issue we found is the use of block.timestamp to record the end of the contribution period. Usually using the block.timestamp is considered a security issue because its value can be manipulated by the miner. In this case it is only used when the developers try to vest their tokens so it is not possible to exploit it by a miner. In other places it is used as a flag to indicate that the contribution stage has concluded.

Also, in transferable modifier of SNTPlaceHolder it is uses block.timestamp to enable transfer only one week after the contribution stage has finalized. It is not an issue because it is not used for a secure calculation.

Secure mathematical functions can be deployed as a library. In the audited files they are included in every contract and will cause duplicated code and higher fees at deployment. But SafeMath functions can be shared between them.

We were not in charge of auditing the MiniMeToken.sol contract, but we found the methods transfer, transferFrom, and approve are vulnerable to the recently discovered "Short Address Attack".

MiniMeToken has a payable fallback function that will proxy the funding to the token controller. We believe this add unnecessary complexity to the StatusContribution contract, and it is safer if the token contract doesn't have to handle Ether.

Audit details

Reviewed files:

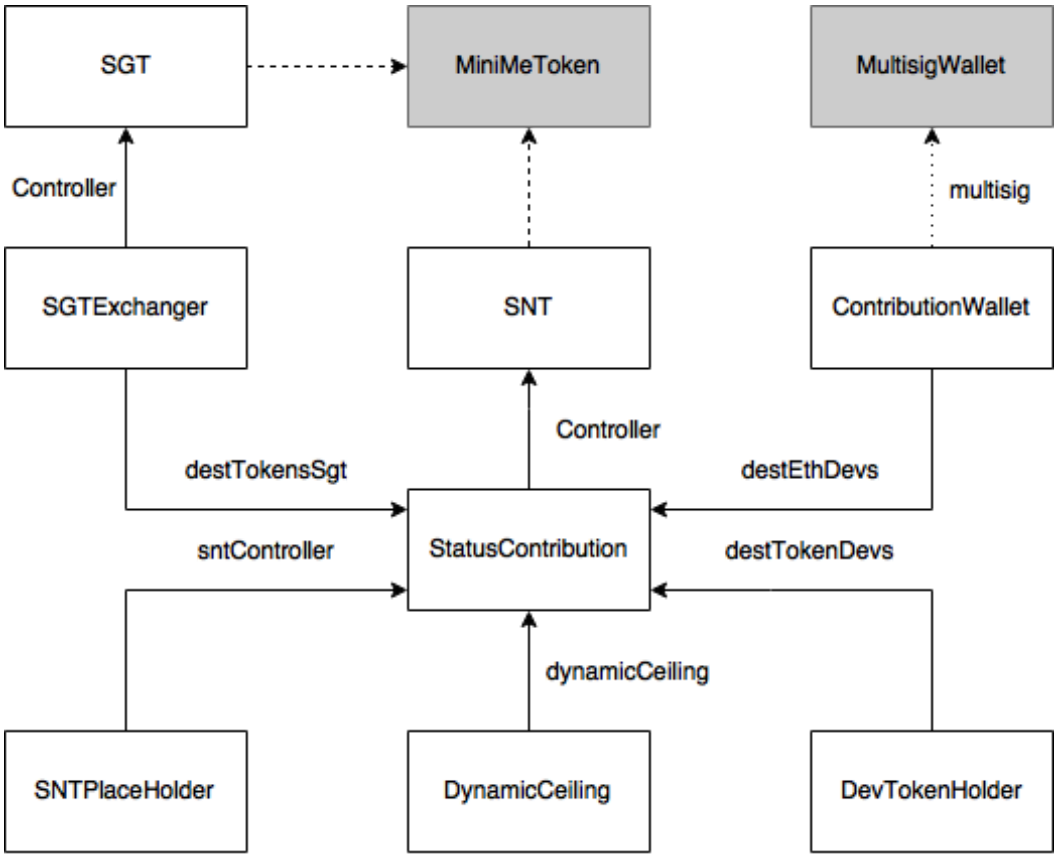
- ContributionWallet.sol
- DevTokensHolder.sol
- DynamicCeiling.sol
- Owned.sol
- SafeMath.sol

- SGT.sol
- SGTExchanger.sol
- SNT.sol
- SNTPlaceHolder.sol
- StatusContribution.sol

Non reviewed files

- MiniMeToken.sol
- MultisigWallet.sol

Contracts Relationship Diagram



The main piece is the token contract SNT.sol “Status Network Token”, it inherits from MiniMeToken. Contracts that inherits from MiniMeToken have a controller entity which will receive notifications of tokens transfers and Ether payments.

The StatusContribution as the controller of SNT during the funding stage, and as such has control on the behavior of the token sale. After the funding stage is over it will switch the controller responsibility to SNTPlaceHolder.

The SGT.sol “Status Genesis Token” is a token for early adopter that have contributed to the project. It is already deployed at 0xd248B0D48E44aaF9c49aea0312be7E13a6dc1468 (https://etherscan.io/address/0xd248B0D48E44aaF9c49aea0312be7E13a6dc1468#readContract), it also implemented on top of MiniMeToken. They will be able to exchange their SGT tokens for SNT token through the SGTExchanger contract after the funding stage is complete.

The DevTokenHolder contract will hold the token allocated to the developers after the contribution stage is complete. It has the particularity the tokens will be frozen for a period of six months after the contribution stage, and then they will be released in a controlled way until after 24 months of the finalization of the contribution stage.

The ContributionWallet will hold the Ether funds until the contribution stage is complete, after which it will allow the multisig wallet to withdraw its funds.

Individual files

ContributionWallet.sol

It will only allow the multisig wallet to retrieve its balance only after the contribution stage is complete, either the funding is complete or the finalization block has already been mined.

DevTokenHolder.sol

Contract that allows the developers to access SNT tokens only six months after the contribution has finalized.

It uses block.timestamp as a source to calculate a time interval to determine the amount of tokens the developers can retrieve. It is not safe to use a value that can be manipulated by a miner to make a secure calculation. The risk is nonexistent because the method collectTokens can only called by the owner of the contract.

DynamicCeiling.sol

Used by StatusContribution to define the cap of the contribution stage. It is defined in such a way that the values of the cap will only be revealed close to the date of the ICO.

Owned.sol

Base contract to allow define an owner address, and provides the modifier onlyOwner so only the owner can execute certain functionality.

SafeMath.sol

A contract that provides basic mathematical operations with validation of the results.

An enhancement we propose it to deploy this contacts as a library to allow the sharing of the code between contracts.

SGT.sol (Status Genesis Token)

It adds functionality on top of MiniMeToken to generate tokens in batches through the multiMint method.

SGTExchanger.sol

It is a token controller for SGT it allows the conversion of SGT tokens to SNT tokens.

SNT.sol (Status Network Token)

Token on top of MiniMeToken, it doesn't provide extra functionality.

SNTPlaceHolder.sol

It will replace StatusContribution after the funding phase. It allow token transfers and will decline sending Ether to the SNT contract.

It uses block.timestamp as activation time for the modifier transferable at line 79. As said previously It is not recommendable to use block.timestamp for secure calculations. It is more secure to use the block number.

```
function transferable(address _from) internal returns (bool) {
  // Allow the exchanger to work from the begining
  if (activationTime == 0) {
    uint f = contribution.finalized();
    if (f>0) {
      activationTime = safeAdd(f, 1 weeks);
    } else {
      return false;
    }
  }
  return (getTime() > activationTime) || (_from == sgtExchanger);
}
```

StatusContribution.sol

Controller of SNT token contract during the funding phase. It will allow payments to the token contract and will forbid transfer of tokens.

The method `finalize` at line 306 uses `now` to mark the finalization of the contribution stage. It is not considered secure to depend on a value that can be manipulated by the miner to his advantage.

```
    function finalize() initialized {
        // ...
        finalized = now;
```

```
        // ...
    }
```

It is a non issue because the only place it is used as a timestamp is from DevTokensHolder.sol by the function collectTokens at line 66.

```
    function collectTokens() onlyOwner {
        // ...
        uint finalized = contribution.finalized();

        if (finalized == 0) throw;
        if (safeSub(getTime(), finalized) <= months(6)) throw;

        // ...
    }
```

Otherwise is used only as a flag to indicate the contribution has completed.

Since SNT forwards payments made to it to the StatusContribution, there is some unnecessary complexity added to the doBuy method in the case of a refund.

We think this will add a burden to the maintenance of the contract, since this behavior is originated in the MiniMeToken, it will also add time to the testing and validation of the function.

```
    function doBuy(address _th, uint _toFund, bool _guaranteed) internal {
        // ...
        if (toReturn>0) {
            // If the call comes from the Token controller,
            // then we return it to the token Holder that.
            // Otherwise we return to the sender.
            if (msg.sender == address(SNT)) {
                _th.transfer(toReturn);
            } else {
                msg.sender.transfer(toReturn);
            }
        }

        // ...
    }
```

MiniMeToken.sol

This file was out of the scope of the security audit but we noticed following the rest of the code that the contract TokenController in this file, which implements `transfer`, `transferFrom` and `approve` is not protected against the vulnerability “ERC20 Short Address Attack” (http://vessenes.com/the-erc20-short-address-attack-explained/). It should verify the payload size that matches the size of the function parameters:

```
modifier onlyPayloadSize(uint size) {
  if(msg.data.length < size + 4) {
```

```
    throw;
  }
  _;
}

function transfer(address _to, uint _value) onlyPayloadSize(2*32) returns (bool success) {
  ...
}
function transferFrom(address _from, address _to, uint _value) onlyPayloadSize(3*32)
returns (bool success) {
  ...
}
function approve(address _spender, uint _value) onlyPayloadSize(2*32) returns (bool
success) {
  ...
}
```

Solidity Optimizer Bug Check

Contracts use Solidity v0.4.11 which corrects the optimizer bug (https://blog.ethereum.org/2017/05/03/solidity-optimizer-bug/).

Conclusions

The source code of the contracts is very well documented and the methods are extensively commented. We can state they are some of the most well documented contracts in the public use.

The contracts seems to be well maintained. For example, they were updated to require the newest version of the solidity compiler 0.4.11 released on May 3.

We have to remark that all the contracts provide ways to reclaim arbitrary tokens/Ethers mistaken sent to them. This is a serious issue when a contract does not have a refund function, the tokens are effectively lost.

References

1. $ 77000 are lost in ERC20 GNO tokens: https://www.reddit.com/r/ethereum/comments/6c68mw/new_record_holder_appears_lets_congratulate/
2. A total of 67316.2838 ETH ended up trapped in contract: https://www.reddit.com/r/ethereum/comments/6ettq5/statement_on_quadrigacx_ether_contract_error/
3. The ERC20 Short Address Attack Explained: http://vessenes.com/the-erc20-short-address-attack-explained/