SMART CONTRACT AUDIT (HTTPS://BLOG.COINFABRIK.COM/CATEGORY/SMART-CONTRACTS/SMART-CONTRACT-AUDIT-SMART-CONTRACTS/)

DreamTeam Token Audit

Matias Bardelli (https://blog.coinfabrik.com/author/matias-bardelli/)

June 12, 2018 (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/dreamteam-token-audit/)

CoinFabrik was asked to audit the contract for the DreamTeam token and token sale (https://blog.coinfabrik.com/dreamteam-smart-contract-for-players-compensation/) (previously audited). Firstly, we will provide a summary of our discoveries and secondly, we will show the details of our findings.

Contents

Summary

The contract audited is in the DreamTeam repository at https://github.com/dreamteam-gg/smart-contracts (https://github.com/dreamteam-gg/smart-contracts). The audit is based on the commit *616ed2538526001d25b75680e4d3bd8f3c4deac2*, and updated to reflect the changes made at commit *da4f4dcce51f3e1086ee9ef09ceacfdc9f761272*.

The audited contract is contracts/token/DreamTeamToken.sol, which is the token contract. It implements the standard ERC20 functionality, an *approveAndCall* method, a *rescueLostTokens* method for recovering mistakenly sent tokens and complementary methods for distinguishing between a transaction's signer and an executor.

We found some problems facing future possible developments and maintainability issues, which would have proven harmful in the future, but they were resolved in the latest commit audited by us.

The following analyses were performed:

- Misuse of the different call methods: call.value(), send() and transfer().
- Integer rounding errors, overflow, underflow and related usage of SafeMath functions.
- Old compiler version pragmas.
- Race conditions such as reentrancy attacks or front running.
- Misuse of block timestamps, assuming anything other than them being strictly increasing.
- Contract softlocking attacks (DoS).
- Potential gas cost of functions being over the gas limit.
- Missing function qualifiers and their misuse.
- Fallback functions with a higher gas cost than the one that a transfer or send call allows.
- Fraudulent or erroneous code.
- Code and contract interaction complexity.
- Wrong or missing error handling.
- Overuse of transfers in a single transaction instead of using withdrawal patterns.
- Insufficient analysis of the function input requirements.

Detailed findings

Minor severity

Overflow possibility in minting functions

In the *multiMint* function there are three situations that can cause an overflow: in the *balanceOf* mapping values, in the *total* variable, and in the *totalSupply* variable. Using safeMath would work to avoid this and make code more readable:

```
for (uint i = 0; i < recipients.length; ++i)
        {
            balanceOf[recipients[i]] = balanceOf[recipients[i]].add(amounts[i]);
            total = total.add(amounts[i]);
            emit Transfer(0x0, recipients[i], amounts[i]);
        }
        totalSupply = totalSupply.add(total);
```

Similarly, the same happens in the *lastMint* function when assigning a value to *remaining*, *balanceOf[tokenDistributor]* and *totalSupply*:

if (fractionalPart <= remaining)   remaining = remaining.sub(fractionalPart); // Remove the fractional part to round the totalSupply   balanceOf[tokenDistributor] = balanceOf[tokenDistributor].add(remaining);   emit Transfer(0x0, tokenDistributor, remaining);   totalSupply = totalSupply.add(remaining);   tokenDistributor = 0x0; // Disable multiMint and lastMint functions forever

This issue was fixed in the last revision sent to us.

Removal of semantic description in hash prefixes

The *sigDestinationTransfer*, *sigDestinationTransferFrom*, *sigDestinationApprove* and *sigDestinationApproveAndCall* variables are defined according to this pattern:

```
bytes32 public variable = keccak256(

    "type description",

    "type description",

//...

);
```

These variables are used in the following snippet, implementing EIP 712 (https://github.com/0xProject/EIPs/blob/master/EIPS/eip-signTypedData.md) data signing:

```
  require(

            signer == ecrecover(

                keccak256(

                    signDest == sigDestination.transfer

                        ? sigDestinationTransfer

                        : signDest == sigDestination.approve

                            ? sigDestinationApprove

                            : signDest == sigDestination.approveAndCall

                                ? sigDestinationApproveAndCall

                                : sigDestinationTransferFrom,

                    data

                ),

                v, r, s

            )

        );
```

We recommend changing this to follow the pattern used in the discussion of the Ethereum Improvement Proposal:

```
bytes32 public variable = keccak256(

    "type field",

    "type field",

//...

);
```

Furthermore, this change would improve maintainability, since possible future documentation changes don't get reflected in the contract code.

It was decided that this behavior will not be modified in the future by the development team.

Enhancements

Input parameter type in rescueLostTokens

The *rescueLostTokens* function receives as input a DTT token and a value for it. Adjacent comments state that it should be an ERC20 token, but this is not the case. While this doesn't really affect the contract, as the function being called is available in both interfaces, it'd be more legible and less prone to errors if instead of accepting a DTT token it accepted as input a token of type ERC20.

This was corrected in the last revision sent to us.

Use of SafeMath

The contract does not use SafeMath (https://github.com/OpenZeppelin/openzeppelin-solidity). While it is guarded against overflow in some functions, it was found that it could still overflow in others, namely multiMint and lastMint. It'd be best if the team considered using a well-tested library for doing math protecting against overflow. The protection against overflow should be consistent through the contract.

This observation was corrected in the last revision sent to us.

Use of modifiers for common require clauses

The contract uses require clauses mixed in the function code. In general, the contract's legibility could be improved by putting frequently used clauses in modifiers, by typing:

```
modifier onlyDistributorAndCurrent()
 {
    require(tokenDistributor != 0x0 && tokenDistributor == msg.sender);
    _;
 }

modifier onlyRescueAccount()
 {
    require(msg.sender == rescueAccount);
    _;
 }
```

This observation was corrected in the last revision sent to us.

Conclusion

We found the contracts to be simple and straightforward and to have an adequate amount of documentation. Signature functions were found to be adequate. The token implementation supports three signature standards, which covers all existing signing methods known for today.

Contact with the development team was made, and the issues were resolved promptly in the latest commit sent to us.

References

- EIP 712: https://github.com/0xProject/EIPs/blob/master/EIPS/eip-signTypedData.md (https://github.com/0xProject/EIPs/blob/master/EIPS/eip-signTypedData.md)
- EIP 712 discussion (ongoing): https://github.com/ethereum/EIPs/pull/712 (https://github.com/ethereum/EIPs/pull/712)
- SafeMath: https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol (https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol)
- Example usage of ecrecover with EIP 712: https://github.com/ukstv/sign-typed-data-test/blob/master/contracts/SignTypedData.sol (https://github.com/ukstv/sign-typed-data-test/blob/master/contracts/SignTypedData.sol)

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/stasis-token-smart-contract-audit/)

Stasis Token Smart Contract Audit (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/stasis-token-smart-contract-audit/)

Coinfabrik has been hired to audit the smart contracts for Stasis sale, the Stable Euro…

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/dmtoken-token-sale-smart-contract-audit/)

DMToken Token Sale Smart Contract Audit (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/dmtoken-token-sale-smart-contract-audit/)

Coinfabrik was hired to audit the contract in terms of its security. First of all,…

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-

Rightmesh Token Sale Smart Contract Audit (Master) (https://blog.coinfabrik.com/smart-contracts/smart-contract-smart-contracts/rightmesh-token-sale-smart-contract-audit-master/)

audit-smart-contracts/rightmesh-token-sale-smart-contract-audit-master/)

Coinfabrik was asked to audit the contracts for the RightMesh Token sale. In the first...

(https://blog.coinfabrik.com/smart-contracts/security-audit-send-sdt-token-sale-ico-smart-contract/)

Security Audit - Send (SDT) Token Sale ICO Smart Contract (https://blog.coinfabrik.com/smart-contracts/security-audit-send-sdt-token-sale-ico-smart-contract/)

Coinfabrik was asked to audit the smart contracts for the Send Token sale. In the...

Tags:

audit (https://blog.coinfabrik.com/tag/audit/)

dreamteam (https://blog.coinfabrik.com/tag/dreamteam/)

Token (https://blog.coinfabrik.com/tag/token/)

SHARE ON

(https://www.facebook.com/sharer/sharer.php?u=https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/dreamteam-token-audit/)
(https://twitter.com/intent/tweet?text=DreamTeam%20Token%20Audit&url=https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/dreamteam-token-audit/)
(https://pinterest.com/pin/create/button/?url=&media=https://blog.coinfabrik.com/wp-content/uploads/2018/04/dreamteam.png&description=DreamTeam+Token+Audit)
(https://www.linkedin.com/shareArticle?mini=true&url=https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/dreamteam-token-audit/&title=DreamTeam%20Token%20Audit&source=CoinFabrik%20Blog)

← PREVIOUS ARTICLE

CryptoCup: Smart Contract Audit (https://blog.coinfabrik.com/smart-contracts/cryptocup-audit/)

NEXT ARTICLE →

Casper CST Token Sale Security Audit (https://blog.coinfabrik.com/smart-contracts/casper-cst-token-sale-security-audit/)

You may also like

(https://blog.coinfabrik.com/smart-contracts/magic-bridge-audit/)

Magic Bridge Audit (https://blog.coinfabrik.com/smart-contracts/magic-bridge-audit/)

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/mintingfactoryv2-baseupgradablemarketplace-kodav3upgradablegatedmarketplace/)

2 months ago

Smart Contract Audit (https://blog.coinfabrik.com/category/smart-contracts/smart-contract-audit-smart-contracts/)

MintingFactoryV2, BaseUpgradableMarketplace & KODAV3UpgradableGatedMarketplace (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/mintingfactoryv2-baseupgradablemarketplace-kodav3upgradablegatedmarketplace/)

(https://blog.coinfabrik.com/feed/)

(https://ar.linkedin.com/company/coinfabrik)

(https://twitter.com/coinfabrik)

(https://www.youtube.com/channel/UC2GmjCr7aEz-il31kqOy9aw)

(https://www.facebook.com/CoinFabrik/)

(https://www.reddit.com/r/CoinFabrik/)

(https://github.com/coinfabrik)