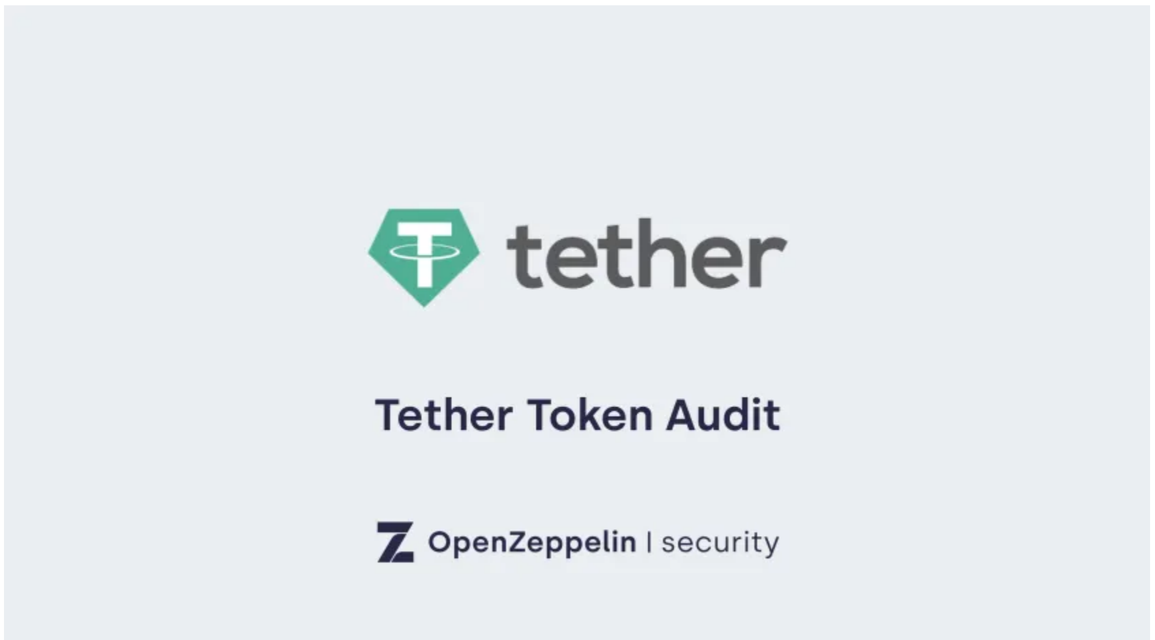
Tether Token Audit

JANUARY 3, 2018 | IN SECURITY AUDITS | BY OPENZEPPELIN SECURITY



The Tether team asked us to review and audit their Tether Token contracts. We looked at the code and now, publish our results.

A previous version of the code was audited by Philip Daian. See his full report here, which includes an overview of the application.
The audited code is located in the tetherto/ether-contracts repository. The version used for this report is commit `9718de4da7b571c1acf822`. We restricted our audit only to the Tether and the Zeppelin contracts, without including the ConsenSys multisig wallet.
Following are our assessment and recommendations, in order of importance.

Update: *The Tether team has followed our recommendations and updated the Tether Token contract. The new version is at commit* `0e2e2ddda17ed4ed20c1d89015906d8276fb38ba` .

Critical severity

No critical severity issues were found.

High severity

No high severity issues were found.

Medium severity

Install OpenZeppelin via NPM and Update

`Contactable` , `Pausable` , `SafeMath` , `Ownable` , `ERC20Basic` , `ERC20` , `BasicToken` , and `StandardToken` were copied from the OpenZeppelin repository.

This violates OpenZeppelin's MIT license, which requires the license and copyright notice to be included if its code is used, and makes it difficult and error-prone to update to a more recent version.

Moreover, the contracts were copied from an old unspecified version (earlier than June 2017, based on the Solidity version pragma). Since then, there have been multiple fixes to the included contracts, which are missing from the Tether repository.
Some of these fixes include:

- Fire an event to signal Ownership transfer (see PR424)
- Remove incorrect short address attack checks (see PR277)
- Check that destination of token transfers is not 0x0 (see PR415)
- Add boolean return flags to ERC20 methods to conform to the standard (see PR308)
- Use `require` checks for token preconditions (see PR466)
- A user can send to themselves more than their current balance (see PR377)

Consider following the recommended way to use OpenZeppelin contracts, which is via the zeppelin-solidity NPM package, and update to the latest version (1.4.0 at the time of this writing). This allows for any bugfixes to be easily integrated into the codebase.

***Update**: OpenZeppelin version 1.4.0 is imported via NPM as of commit `ce34f14d05`. However, due to this issue in the library, a change is needed in the `StandardToken` contract as described in the README. Instead of requiring the developer to manually perform the change, consider forking the library in GitHub and changing it there, or commit a patch diff file to automate the change.*

Low severity

OpenZeppelin standard contracts were modified

Additionally to copying OpenZeppelin's contracts instead of installing them via NPM, some of them were modified.
Fee management was added directly to `BasicToken` and `StandardToken` implementations, instead of implementing them in a new Token contract that extends from `StandardToken`.
We concur with the original observations from Philip Daian in this point:

> One change that would make the code substantially cleaner, more modular, and more upgradeable is the moving of the fee calculation functions to the top-level TetherToken file. Currently, the fee calculation is done independently twice in StandardToken.sol and BasicToken.sol. While the calculation has been verified and tested as working, this repetition of code minorly impacts the upgradeability of the token. Furthermore, modifications directly to the Zeppelin library violate the boundaries drawn in the application diagram in the previous section, potentially making future Zeppelin changes more difficult to integrate and more likely to introduce unintended side effects. While this change has been verified as not security critical, it is our belief that such a change would improve the readability, testability and modularity of the existing codebase.

Furthermore, non ERC20-compliant changes, such as using `MAX_UINT` as an eternal approval magic value, or forcing clients to reduce approval to zero before changing it, were introduced in the StandardToken implementation. Note that, since 1.3.0, OpenZeppelin's StandardToken has the `increaseApproval](https://github.com/OpenZeppelin/zeppelin-solidity/pull/224) and [` decreaseApproval methods to mitigate the latter.

This is not the way OpenZeppelin standard contracts should be used. Making changes to open-source libraries, instead of using them as is, can be dangerous and prevents from integrating bug-fixes into the codebase easily.
Consider extending `StandardToken` in a `StandardTokenWithFees` contract, that adds the fee calculation feature to the token.

***Update**: Fixed in commits `ce34f14d05` and `0e2e2ddda1`.*

Token allowances can be modified while the contract is paused

The method `TetherToken#approve` is missing a `whenNotPaused` modifier. This allows any user to change allowances while the token is paused. A paused token should halt all state-changing operations, except for those to be run in emergency.

Consider adding the `whenNotPaused` modifier to the approve method, or simply using OpenZeppelin's `PausableToken` contract, which integrates the Pausable functionality into a `StandardToken`.

***Update**: Fixed in commit `ce34f14d05`.*

Solidity version

Current code specifies version pragma `^0.4.8` or `^0.4.9`, depending on the file. We recommend changing the solidity version pragma to the latest version (`^0.4.18`) to enforce the use of an up to date compiler. The new compiler includes several bugfixes, including `SkipEmptyStringLiteral`, to which the `0.4.9` compiled code is vulnerable.

Note that this implies updating `constant` modifiers to `pure` / `view`, and updating `throw`s to `revert` / `require` / `assert` as needed.

***Update**: Fixed in commit `ce34f14d05`.*

Unclear responsibilities for BlackList contract

The `BlackList` contract defines methods for adding and removing a user from a blacklist. It also allows the owner to destroy the funds of a blacklisted user, which requires the contract to extend `BasicToken`, in order to access the balances.

However, the contract **does not** enforce that transfer methods cannot be executed by blacklisted users. This is manually implemented in TetherToken L35 and L45. This is not a good design, violating the principle of separation of concerns and modularity.

Consider making the `BlackList` contract independent from token contracts, and implement the `destroyBlackFunds` function in the `TetherToken` directly. Alternatively, consider changing `BlackList` into a `TokenWithBlackList`, extending from `StandardToken`, and adding all the `require(!isBlackListed[msg.sender]) checks there.

***Update**: Fixed in commit `ce34f14d05`.*

Unchecked math operations

There are unchecked arithmetic operations in the functions `issue` and `redeem` in TetherToken.

Even though no overflow should occur due to the additional require guards in both functions, it's always better to be safe and perform operations with correctness assertions.

Furthermore, the potential overflow in the `issue` function should be handled as a `throw` operation instead of a `revert` (see this article for more info). Note that this would be handled automatically by using a checked addition, and by removing the existing require in that function.

Additionally, there is an unchecked arithmetic operation in `BlackList#destroyBlackFunds`. While `totalSupply` should never fall below zero, using a checked subtraction would prevent against potential errors in future upgraded implementations.

Consider using `SafeMath` for all arithmetic operations in the `TetherToken` and `BlackList` contracts.

***Update**: Fixed in commit `ce34f14d05`.*

Incomplete test coverage

Unit test coverage for the Tether deprecation features is quite incomplete. Only the `transfer` method is checked to be properly delegated to the upgraded token.

Consider adding tests to also check the delegation of the `transferFrom`, `balanceOf`, `approve`, `allowance` and `totalSupply` methods.

***Update:** Consider adding tests for the delegation of the `increaseApproval` and `decreaseApproval` methods as well.*

Notes & additional information

- The project has no instructions as to how to run the tests, or the required versions for its dependencies. Consider adding a `package.json` file with the dependencies (such as truffle and zeppelin-solidity), including a test script as well.
  ***Update:** `zeppelin-solidity` dependency added in commit `ce34f14d05`, consider adding Truffle as well.*
- Consider adding a `README` to the project describing its purpose, functionality, structure, architecture, and instructions for development.
  ***Update:** `README` added in commit `ce34f14d05`, consider expanding on project architecture.*
- The short address attack check via the `onlyPayloadSize` modifier is not considered a correct mitigation for the attack, and even potentially harmful when extending contracts. See this issue for more information. Consider removing all uses of the modifier from the codebase.
  ***Update**: Fixed in commit `ce34f14d05`.*
- TetherToken defines the `decimals` public state variable to be uint, which defaults to `uint256`. Consider changing this to `uint8` to comply with the ERC20 specification.
  ***Update**: Fixed in commit `ce34f14d05`.*
- The ERC20 specification suggests emitting a Transfer event from the address 0x0 when minting new tokens. Consider emitting such event in the `TetherToken#issue` function. Additionally, we suggest also emitting a Transfer event **to** the address 0x0 when burning the tokens in the `redeem` and `destroyBlackFunds` functions.
  ***Update**: Fixed in commit `ce34f14d0` for new tokens issued.*
- `TetherToken` defines methods for upgrading the contract, as well as for managing issuance and redeeming of tokens. Consider moving the upgrade mechanism to a separate `UpgradebleToken` contract, and having `TetherToken` extend from it, as a means to separate concerns.

- There are two magic numbers in `TetherToken#setParams`. Consider changing them into constants and define them at the contract level for clarity.
  ***Update**: Fixed in commit `ce34f14d05`.*
- Maximum fees cannot be defined as a fraction of a token, since the `TetherToken#setParams` function accepts a `newMaxFee` which is multiplied by `10**decimals`. Consider accepting the new `maximumFee` value directly, to allow for fraction of tokens to be used.
- Contract `UpgradedStandardToken` extends `StandardToken`. Since it is used as an interface exclusively, consider extending from ERC20, which is the interface implemented by `StandardToken`.
- Consider marking the address parameter in all three blacklist events (`DestroyedBlackFunds`, `AddedBlackList` and `RemovedBlackList`) as `indexed`, to allow a client to listen for changes to their own status.
  ***Update**: Fixed in commit `ce34f14d05`.*
- Function `BlackList#` is unnecessary, since the mapping `isBlackListed` is already flagged as public. Consider removing the getter function, or remove the public modifier in `isBlackListed`.
- Function `BlackList#getOwner` is unnecessary, since the owner is already provided by the parent Ownable contract.
  ***Update**: Fixed in commit `ce34f14d05`.*
- Consider adding a guard to TetherToken#deprecate to check that the `upgradedAddress` is not 0x0, to prevent potential mistakes. Furthermore, consider adding a public flag `isUpgradedToken` to `UpgradedStandardToken`, and check that the code at `upgradedAddress` does contain that flag.
  ***Update:** Added a check that address is not `0x0` in commit `ce34f14d05`.*
- The getters in the `BlackList` contract are not tested. However, since these methods are unnecessary (for they are already automatically generated by Solidity), consider removing them rather than adding tests for them.
- Note that the check that new fees are below a maximum value in order to "ensure transparency" can be easily circumvented, since the token can be upgraded to a version without those limits, and calls are automatically forwarded to the new one.
- Given that decimals is parameterisable in a `TetherToken`, it is possible to update from a token with a number of decimals to another with a different number. This could cause issues in client interfaces listing the tokens. Consider using a fixed amount of decimals, preferably 18 for compatibility with ETH.
  ***Update:** A public property `_totalSupply` was added to the `UpgradedStandardToken` contract in commit `ce34f14d05`. It is not required as part of the interface. Consider removing it.*
- ***Update:** Commit `ce34f14d05` adds the interface `PreviousTokenInterface` with the `oldBalanceOf` method, which is implemented by TetherToken. Consider having TetherToken explicitly inherit from `PreviousTokenInterface if needed`.*

Audited contracts

Following are the MD5 hashes of the audited contracts:

```
dfc0c783ff7a782bbf415f4b4943cbfe contracts/TetherToken.sol
652103fa8d6b9c7d5952770a5abf5b96 contracts/UpgradedStandardToken.sol
3cf622b896dc0990d4f606d8ee9217f1 contracts/BlackList.sol
37e0a81e72f33831e41099c7f5ef4d88 contracts/UpgradedTokenTest.sol
e14829154a7c9bf9f750707692727a51 contracts/zeppelin/ownership/Ownable.sol
a9e3c69db6b3d594691c6f25b4ceec80 contracts/zeppelin/ownership/Contactable.sol
9165ac7dbad97414a00549eb6bb17cba contracts/zeppelin/lifecycle/Pausable.sol
0770d7b5b0bff5cc992e1ccd19b5672d contracts/zeppelin/token/StandardToken.sol
aa0786f69b28548bae8bf69b04f66475 contracts/zeppelin/token/ERC20Basic.sol
ef90a4ebcd66da85d3b0977e3b31f5ed contracts/zeppelin/token/ERC20.sol
5c56ca643931cab56bf1325fb0abdda7 contracts/zeppelin/token/BasicToken.sol
72a05c21bd9108344f9a4207a2223ef2 contracts/zeppelin/SafeMath.sol
```

***Update**: The MD5 hashes of the updated contracts corresponding to commit `0e2e2ddda1` are the following:*

```
5d82de93b5c5fe047d7481d232f35901  BlackList.sol
c4173bcac5359d53c95dc393036c3cba  StandardTokenWithFees.sol
12341f088134abc35ca9e504b03b3453  TetherToken.sol
158de41f417db1785ea4a5ed0136b6c9  UpgradedStandardToken.sol
```

Conclusion

No critical or high severity issues were found. Some changes were proposed to follow best practices and reduce potential attack surface.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Tether Token contracts. We have not reviewed the related Tether project. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.

Security Audits

- If you are interested in smart contract security, you can continue the discussion in our forum, or even better, join the team 🚀

- If you are building a project of your own and would like to request a security audit, please do so here.

RELATED POSTS

SECURITY AUDITS

Metal Token Audit

The Metal team asked us to review and audit their new Metal Token contract code. We looked at their...

READ MORE

by OpenZeppelin Security

SECURITY AUDITS

ERC20 Element Token Audit

The Element Group team asked us to review and audit their ERC20 Element Token contract. We looked...

READ MORE

by OpenZeppelin Security

SECURITY AUDITS

Bax Token Audit

The BABB team asked us to review and audit their Bax Token contract. We looked at the code and now...

READ MORE

by OpenZeppelin Security