CONFIDENTIAL

Security Audit

of GLOBCOIN's Smart Contracts

November 21, 2018

Produced for

GLOBCOIN
BRINGING STABLECOINS TO THE NEXT LEVEL

by

CHAINSECURITY

Table Of Content

Foreword 1

Executive Summary 1

Audit Overview 2

1. Methodology and Scope of the Audit 2
2. Depth of Audit 2
3. Terminology 3

Limitations 4

System Overview 5

1. Token Overview 5
2. Token Sale Overview 5
3. Extra Token Features 5

Best Practices in GLOBCOIN's project 6

1. Hard Requirements 6
2. Soft Requirements 6

Security Issues 7

1. Unrestricted writes to `KeyValueStorage` H ✓ Fixed 7

Trust Issues 8

1. Owner can wipe funds of any address H ✓ Acknowledged 8
2. Unrestricted change of GLX implementation H 8

Design Issues 9

1. Outdated compiler version M ✓ Fixed 9
2. No technical specifications M 9
3. Unused contracts: `PublicProxy`, `TransferableStorage` L ✓ Fixed 9

4. Unsafe contracts M ✓ Fixed 9

5. Limited test coverage M 10

6. Libraries with `public` functions M ✓ Fixed 10

7. Unused function in `WhitelistableTokenDelegate` L 10

8. `OwnableDelegate` is not used M 10

9. Duplicate `onlyOwner` check in `mint` L ✓ Fixed 11

Recommendations / Suggestions 12

Foreword

We first and foremost thank GLOBCOIN for giving us the opportunity to audit their smart contracts. This documents outlines our methodology, limitations, and results.

– ChainSecurity

Executive Summary

The GLOBCOIN smart contracts have been analyzed under different aspects, with a variety of tools for automated security analysis of Ethereum smart contracts and expert manual review.

Initially provided contracts contained a considerable amount of unused code, were lacking specification and documentation and had few test cases. During the course of the audit unused files were removed, tests and deployment scripts were also supplied. This lead to an improvement in the overall project quality.

Several severe security and design issues were uncovered which GLOBCOIN successfully addressed. CHAINSECURITY however is concerned that GLOBCOIN may not be fully aware of the consequences and implications of their upgradeability requirements. We recommend to investigate and reflect more on the proxy-storage setup to avoid any issues when upgrading the implementation.

Audit Overview

Methodology and Scope of the Audit

ChainSecurity's methodology in performing the security audit consists of four chronologically executed phases:

1. Understanding the existing documentation, purpose, and specifications of the smart contracts.

2. Executing automated tools to scan for generic security vulnerabilities.

3. Manual analysis covering both functional (best effort based on the provided documentation) and security aspects of the smart contracts by one of our ChainSecurity experts.

4. Writing the report with the individual vulnerability findings and potential exploits.

The scope of the audit is limited to the following source code files. All of these source code files were received on November 14, 2018 and an updated version on November 20, 2018:

SHA-256 checksum	File
5afda0a15df14f225504d49501f2944208a10ea28bb42f31dc9b3dae1da7bd21	./Libraries/WhitelistingTokenLib.sol
4fe11f1f01af6eaa798538cb5a8f67ddc534e0840efbd9658020bf31fb4690d6	./Libraries/StandardTokenLib.sol
f417c2dedce09478b8441f3ffc4665b5e32531ad2084c9cb514fff71b34290df	./Libraries/BasicTokenLib.sol
c70531fceeebd23a01969a940029c9ac6a8731cbcd33b8820e0a6fe6de687347	./Implementations/GlobCoinToken.sol
86e61e47f87762ca664e42eec215fd8a79a1a09533fe7da1d1456d53299cfefe	./Implementations/GenericErc20Token.sol
54ce2c12e41cfd53299d032026c2dc65c9f9399e7b8a48c663d19b63c19e34a9	./Implementations/DetailedToken.sol
1cd09c380e6966411c7fb9bc1ec4926eeec5090b7d9d9a678a13b05f0efa9fdc	./Core/Storage/StorageState.sol
5bb6bea35445d6bfcab6099280f10d586cf3fca4a030e94a9c32131c1cfce204	./Core/Storage/KeyValueStorage.sol
7ca669de23e4756e917e93b9d4f8be9aaa52178ad2acc7fd2e7e0991666de707	./Core/Storage/StorageLib.sol
2c709a98c7a60c3ba63b899380e595ef8fe72be61e4a23b3a80d82f0b4772171	./Core/Storage/StorageConsumer.sol
a66feeb6ff1b47c907d6f9592344c9896b7a3df311944a7dd38781d8e523f2f3	./Core/Proxy/BaseProxy.sol
7b6140865e8dec55e9e41ce7f29c6a982fbbafeacfc5d85c8efecdcc768eb535	./Core/Proxy/OwnableProxy.sol
67cfc7cc5a8275b660edf7a92a109f72d02d0ca785785c4fe7461d8a4009ab69	./Delegates/PausableTokenDelegate.sol
1838a60d47a76451e70f74a1fc78d9308cc61ee727780548a6804ceaf7bb0f13	./Delegates/BurnableTokenDelegate.sol
506e31f6656c48e1a59ece9497b686e79ee898bf00223407dca3c629674345c5	./Delegates/WhitelistableTokenDelegate.sol
2f45148939e8e9e3ef2ae147a223b0bd79f4fd0981e1fbe6d9e268d35742cbc9	./Delegates/BasicTokenDelegate.sol
c11c0694ccd0823c0903e733fcf0bc1fbb05dee67b5befbbdaa3702114cc6576	./Delegates/StandardTokenDelegate.sol
cb9746b17e1d783f225a9e3d1311a6472d8706549da361baa673b6d98100a3c0	./Delegates/WipeableTokenDelegate.sol
3df40c45fdd8ee15760fe2f01765a1e625445855d32572adea0fda539b2edf40	./Delegates/MintableTokenDelegate.sol

Depth of Audit

The scope of the security audit conducted by ChainSecurity was restricted to:

- Scan the contracts listed above for generic security issues using automated systems and manually inspect the results.

- Manual audit of the contracts listed above for security issues.

Terminology

For the purpose of this audit, we adopt the following terminology. For security vulnerabilities, we specify the *likelihood*, *impact* and *severity* (inspired by the OWASP risk rating methodology[1]).

Likelihood represents the likelihood of a security vulnerability to be encountered or exploited in the wild.

Impact specifies the technical and business related consequences of an exploit.

Severity is derived based on the likelihood and the impact calculated previously.

We categorize the findings into 4 distinct categories, depending on their severities:

- L Low: can be considered as less important
- M Medium: should be fixed
- H High: we strongly suggest to fix it before release
- C Critical: needs to be fixed before release

These severities are derived from the likelihood and the impact using the following table, following a standard approach in risk assessment.

	IMPACT		
LIKELIHOOD	**High**	**Medium**	**Low**
High	C	H	M
Medium	H	M	L
Low	M	L	L

During the audit concerns might arise or tools might flag certain security issues. After careful inspection of the potential security impact, we assign the following labels:

- ✓ No Issue : no security impact
- ✓ Fixed : during the course of the audit process, the issue has been addressed technically
- ✓ Addressed : issue addressed otherwise by improving documentation or further specification
- ✓ Acknowledged : issue is meant to be fixed in the future without immediate changes to the code

Findings that are labelled as either ✓ Fixed or ✓ Addressed are resolved and therefore pose no security threat. Their severity is still listed, but just to give the reader a quick overview what kind of issues were found during the audit.

[1] https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

Limitations

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

In most cases, applications are either fully protected against a certain type of attack, or they lack protection against it completely. Some of the issues may affect the entire smart contract application, while some lack protection only in certain areas. We therefore carry out a source code review trying to determine all locations that need to be fixed. Within the customer-determined timeframe, ChainSecurity has performed auditing in order to discover as many vulnerabilities as possible.

System Overview

Token Name & Symbol	GLOBCOIN TOKEN, GLX
Decimals	6
Refund	At market rate
Token Supply	Unlimited
Token Type	Upgradeable ERC20
Token Generation	Mintable, Burnable
Pausable	Yes
Whitelist	Yes

Table 1: Facts about the GLX token and the Token Sale.

In the following we describe the GLOBCOIN TOKEN (GLX). Table gives the general overview.

Token Overview

The GLOBCOIN TOKENs will be pegged to the value of a basket including major currencies and gold. GLOBCOIN claims that there will be a strict correspondence between the amount of GLX tokens and the collateral basket, which will be backed by bank deposits and regularly audited. GLOBCOIN will mint tokens as users buy tokens on their platform with fiat payments. There are no restrictions on the amount of GLXs tokens to be minted and hence their supply is indefinite.

Token Sale Overview

Users will be able to buy GLX tokens on the GLOBCOIN Platform where GCP tokens are needed to interact with the services. Alternatively, the token is planned to be listed and traded on secondary markets.

Extra Token Features

Upgradability The functions of the GLX token are upgradeable. This is achieved with the usage of a proxy pattern, redirecting function calls on the token to different implementations. The GLOBCOIN TOKEN contract is the proxy, delegating calls such as transfers to a currently valid contract address, curated by GLOBCOIN.

Whitelist Only users with whitelisted addresses can hold GLX tokens and make use of them, by e.g. trading.

Best Practices in GLOBCOIN's project

Projects of good quality follow best practices. In doing so, they make audits more meaningful, by allowing efforts to be focused on subtle and project-specific issues rather than the fulfillment of general guidelines.

Avoiding code duplication is a good example of a good engineering practice which increases the potential of any security audit.

We now list a few points that should be enforced in any good project that aims to be deployed on the Ethereum blockchain. The corresponding box is ticked when GLOBCOIN's project fitted the criterion when the audit started.

Hard Requirements

These requirements ensure that the GLOBCOIN's project can be audited by CHAINSECURITY.

- ✓ The code is provided as a Git repository to allow the review of future code changes.
- ✓ Code duplication is minimal, or justified and documented.
- ✓ Libraries are properly referred to as package dependencies, including the specific version(s) that are compatible with GLOBCOIN's project. No library file is mixed with GLOBCOIN's own files.
- ✗ The code compiles with the latest Solidity compiler version. If GLOBCOIN uses an older version, the reasons are documented.
- ✗ There are no compiler warnings, or warnings are documented.

Soft Requirements

Although these requirements are not as important as the previous ones, they still help to make the audit more valuable to GLOBCOIN.

- ✓ There are migration scripts.
- ✓ There are tests.
- ✗ The tests are related to the migration scripts and a clear separation is made between the two.
- ✗ The tests are easy to run for CHAINSECURITY, using the documentation provided by GLOBCOIN.
- ✗ The test coverage is available or can be obtained easily.
- ✓ The output of the build process (including possible flattened files) is not committed to the Git repository.
- ✓ The project only contains audit-related files, or, if not possible, a meaningful separation is made between modules that have to be audited and modules that CHAINSECURITY should assume correct and out of scope.
- ✓ There is no dead code.
- ✗ The code is well documented.
- ✗ The high-level specification is thorough and allow a quick understanding of the project without looking at the code.
- ✗ Both the code documentation and the high-level specification are up to date with respect to the code version CHAINSECURITY audits.
- ✓ There are no getter functions for public variables, or the reason why these getters are in the code is given.
- ✓ Function are grouped together according either to the Solidity guidelines[2], or to their functionality.

[2] `https://solidity.readthedocs.io/en/latest/style-guide.html#order-of-functions`

Security Issues

In the following, we discuss our investigation into security issues. Therefore, we highlight whenever we found specific issues but also mention what vulnerability classes do not appear, if relevant.

Unrestricted writes to `KeyValueStorage` H ✓ Fixed

GLOBCOIN updated the storage contract after an initial security discussion about the project. Due to lack of documentation, deployment scripts and tests the initial system design and its intended usage were unclear.

It seemed that `PausableTokenDelegate` writes directly to storage. This would have been a fundamental problem as `DELEGATECALL` retains the original `msg.sender` and hence all users interacting with the contract would have written to different storage fields. However, the storage is an external contract and only accessed via calls to its functions from the proxy contract `PausableTokenDelegate`, ensuring that `msg.sender` remains the proxy contract. The original version would have worked, but is rather inefficient.

GLOBCOIN removed the initial two-dimensional addressing of the storage fields which in combination with lacking access control on storage writes introduced a severe vulnerability.

All functions in the `KeyValueStorage` are publicly accessible and anyone can write to any storage field. This is the case because the modifier `isAllowed` guarding read/write/delete operations to the storage allows everyone except the `0x0` address to perform these. Aside from this, CHAINSECURITY notes that the derivation of the keys of the storage fields is trivial.

```
modifier isAllowed {
    require(senderIsValid(), "sender␣not␣valid");
     _;
 }
...
function senderIsValid() private view returns (bool) {
    return msg.sender != 0x0;
}
```

KeyValueStorage.sol

The likely reason for this is that the code was based on a publicly available storage implementation[3]. One of these repositories provides multiple implementations of a key-value storage, some of which implement access control checks. The one GLOBCOIN actually uses does not restrict access, but the modifier seems to remain as a left-over.

Instead of going back to the original inefficient version, an improved solution might be to change the modifier to only allow write and delete operations to the proxy contract, as the `isAllowed` modifier already exists but does not enforce proper access control.

extbfLikelihood: high extbfImpact: high

Fixed: GLOBCOIN solved this by enforcing that only the proxy contract is able to call functions writing into the key value storage.

[3] `https://github.com/levelkdev/upgradability-blog-post/tree/b59db2d1dbce764b15051281482f4a3caa239aa8` or `https://github.com/mikec/smart-contract-upgradability/tree/f86939a3bf5d649d0221b6231b9ed5e213306eb7`

Trust Issues

The issues described in this section are not security issues but describe functionality which is not fixed inside the smart contract and hence requires additional trust into GLOBCOIN, including in GLOBCOIN's ability to deal with such powers appropriately.

Owner can wipe funds of any address H ✓ Acknowledged

GLOBCOIN acts as an administrator of the contracts and can wipe all GLOBCOIN TOKENs of any user address by using the `WhipableTokenDelegate` contract and its `whipeAddress` functions. This behavior is intended and defined by the GLOBCOIN specification, but still introduces a trust issue for fund holders.

CHAINSECURITY strongly remarks that there exists a possibility of the affected user front-running GLOBCOIN: A user that suspects that he is about to get wiped by the `admin` can observe the transaction pool and try to front-run this by transferring his tokens just before. CHAINSECURITY wants to make GLOBCOIN aware of this potential issue. Especially if GLOBCOIN intends to use this functionality to wipe tokens of someone who obtained them maliciously, they likely will have enough technical knowledge to evade this. GLOBCOIN should consider the pausing of the token in such extreme situations.

Acknowledged: GLOBCOIN replies:

"We understand the trust aspect of the Wipe process. However, with respect to the regulator, we have to be able to Wipe an address. We should also be able to Wipe an address in case a user looses his/her private key. In this case, after a strong identification process, we'll be able to wipe addresses of the users and mint new tokens on a new address."

Unrestricted change of GLX implementation H

Two different cases have to be considered, depending on if `OwnableProxy` or `PublicProxy` is used. The underlying issue can be expressed as follows: Using `DELEGATECALL`, the storage of the contract initiating the delegate-call is used, which is known to GLOBCOIN and necessary to implement the desired functionality. Nonetheless this entails noteworthy trust implications:

- Usage of `OwnableProxy`

  The owner may arbitrarily change the proxy's implementation contract to any address. As CHAINSECURITY understands, the proxy is planned to point to the `PausableTokenDelegate` contract, meaning the delegate-call will be executed on this contract. This should be explicitly documented. However the owner of the proxy contract may set the implementation, referring to the target address of the delegate-call, to an arbitrary address. Therefore the functionality provided by the token contract may change. The updated implementation could be set to any contract with malicious features.

- Usage of `PublicProxy`

  This implementation would allow each user to change the execution target address of the `DELEGATECALL` by himself. Depending on further system interactions, this can be highly dangerous as using a specially crafted contract a malicious participant could arbitrarily change the storage of the proxy contract.

Addressed: GLOBCOIN removed `PublicProxy` from the codebase. This removes the trust issue `Usage of Public Proxy`. Note that the case of `Usage of OwnableProxy` was not addressed.

Design Issues

The points listed here are general recommendations about the design and style of GLOBCOIN's project. They highlight possible ways for GLOBCOIN to further improve the code.

Outdated compiler version M ✓ Fixed

GLOBCOIN's code uses `pragma solidity ^0.4.24` instead of the most recent version. Without a documented reason, the newest compiler version should be used.

Fixed: GLOBCOIN updated all files to `pragma solidity 0.5.0` which was published less than 24h before the audit report was released and hence technically is the most recent version at the time. The updated code however fails to compile as the included files from `OpenZeppelin` are not compatible and require an older compiler version.

Solidity version 0.5.0, a major update with breaking changes was just released recently on 13/11/2018. This conflicts with CHAINSECURITY's recommendation to update to the most recent compiler version, which was version 0.4.25 at the time of the audit.

As solidity 0.5.0 contains some breaking changes and is not battle tested yet it may be prone to unexpected issues. Hence CHAINSECURITY does not recommend to use version 0.5.0 at this time.

Using `pragma solidity ^0.4.25` protects against the known bugs of solidity 0.4.24[4]

No technical specifications M

Due to missing technical specifications checking the correct behavior of the code introduces unnecessary burdens for CHAINSECURITY and any potential future reviewer. Documentation would help to understand what the code functionality is intended and to determine if it is actually doing so. More so, advanced users and future developers will heavily benefit from better documentation and additional material.

Unused contracts: `PublicProxy`, `TransferableStorage` L ✓ Fixed

The contracts `PublicProxy.sol` and `TransferableStorage.sol` are present in the code base, but never used. `TransferableStorage.sol` was already removed from the code base during the audit process, but the `PublicProxy` is still present. GLOBCOIN must clarify which contracts are intended to be deployed on main network.

Fixed: GLOBCOIN removed `TransferableStorage.sol` and `PublicProxy.sol` from the code base.

Unsafe contracts M ✓ Fixed

`PublicProxy` lacks any security measures such as basic access control and should not be deployed on main network.

For `PublicProxy` every `msg.sender` must set the address of the implementation manually, as opposed to the `OwnableProxy`, where the owner sets the address for everyone. As delegate-calls are used there, all storage changes will happen in the storage of the proxy contract. Here however, a user can specify the code which will be executed in the context of the proxy contract, accessing the storage of the proxy contract. This is highly dangerous and can have severe consequences.

As documentation is completely missing, CHAINSECURITY cannot be sure that GLOBCOIN does not intend to actually deploy these and how future interaction between the contracts are planned.

Fixed: GLOBCOIN removed `PublicProxy.sol` from the code base as this file is not used.

[4] https://blog.ethereum.org/2018/09/13/solidity-bugfix-release/

Limited test coverage M

The tests provided cover only a limited part of all supplied contracts and initially no tests were available. Without a good testing suite, neither GLOBCOIN nor CHAINSECURITY can ensure fully working functionality.

CHAINSECURITY strongly recommends to improve test coverage, as currently `solc` reports a very low coverage of 56 percent.

Libraries with `public` functions M ✓ Fixed

All functions in GLOBCOIN's libraries[5] are declared as `public`.

Library functions can only be called directly if they do not modify the state, because libraries are assumed to be stateless. Most of the GLOBCOIN's libraries modify state. If they were to be callable directly, this would have a severe security impact on the system. Although the `public` keyword is not unsafe in this case, its usage in libraries is generally considered bad practice and unnecessarily wastes gas.

If there is no explicit reason to make library functions public then their visibility should be restricted. This enables all internal types to be passed directly and types stored in memory can be passed by reference and do not have to be copied[6].

Fixed: GLOBCOIN changed all library functions from `public` to `internal`.

Unused function in `WhitelistableTokenDelegate` L

The function `getIfWhitelisted` is defined but never used:

```
function getIfWhitelisted(address who) public view returns (bool) {
        return WhitelistingTokenLib.getIfWhitelisted(_storage, who);
}
```

WhitelistableTokenDelegate.sol

CHAINSECURITY notes that all calls checking if an address is whitelisted, call the library function directly (e.g. in the modifier). Hence there seems to be no benefit to have this function. The library function `getIfWhitelisted` is directly callable by anyone, as it is a `view` function not modifying state[4].

`OwnableDelegate` is not used M

GLOBCOIN's code uses two different implementations for ownership control:

- `Ownable` by OpenZeppelin, which stores the owner in the field `address public owner`.
- `OwnableDelegate`, which stores the owner in a proprietary `KeyValueStorage`.

CHAINSECURITY is aware of the intention to store all information in the `KeyValueStorage`, which facilitates the upgradeability of the implementation contract. Therefore, `OwnableDelegate` is intended to be used for ownership control when accessing the token functionality, while `Ownable` of OpenZeppelin is intended to be used to control ownership of the proxy contract. However, only `Ownable` is ever used. It seems that GLOBCOIN is not aware of this.

The `GlobCoinToken` contract sets both, the owner in the `KeyValueStorage` and the `Ownable owner` in the contract's storage to the `msg.sender` of the initial deploying transaction. But when the `PausableTokenDelegate` contract is deployed, only the owner of `Ownable` is set to the sender of the contract creating transaction.

Note that when the `PausableTokenDelegate` contract is accessed via the proxy contract and a delegatecall, the `PausableTokenDelegate` code execution happens in the context of the `GlobCoinToken` contract, as opposed to when `PausableTokenDelegate` is called directly. One must be careful to review which address is actually stored in the field `address public owner` in the current storage context.

The `PausableTokenDelegate` inherits from both and both have the `onlyOwner` modifier and functionality to `transferOwnership`. The `Ownable` implementation is always the most derived one, so only this one is always used in the current implementation.

[5] `BasicTokenLib`, `OwnableLib`, `StandardTokenLib` and `WhitelistingTokenLib`.
[6] https://solidity.readthedocs.io/en/latest/contracts.html#libraries

CHAINSECURITY remarks that transferOwnership only updates the owner stored in address public owner of the current context and not the owner stored in the KeyValueStorage. Should the ownership get changed and later the proxy redirects to an updated implementation where the OwnableDelegate implementation happens to be the most derived one, unexpected behavior can occur as the very first owner will suddenly be relevant[7].

Addressed: This was addressed by removing the "unused" OwnableDelegate. However this contract is supposed to be used, but it was not due to the construction of the inheritance chain, as another function with the same name was prioritized. Removing the OwnableDelegate, which uses the owner stored in the KeyValueStorage introduces new issues.

GLOBCOIN's implementation uses this KeyValueStorage together with the proxy setup. It is important to understand why this setup was chosen and this should be clearly documented. Both, the proxy contract and the current implementation, PausableTokenDelegate, inherit from Ownable of OpenZeppelin. Thus both have a state variable address owner. One might think that this is the same variable. However, it is important to remember that all calls to the proxy contract are redirected to the current implementation by a DELEGATECALL. It is crucial to deeply understand how DELEGATECALL exactly works as it is highly dangerous and intricate to use correctly when dealing with storage.

DELEGATECALL executes code stored at the call target location in the context of the caller, but preserves the msg.sender. It hence only writes into the storage of the caller and loads EVM bytecode. The variable owner in contract A may be stored at a different location than the variable with the same name in a different contract. To clarify, during compilation of the proxy contract, the compiler decides where to store what storage variable.

In this example, the variable owner may be placed into storage slot 2 of the proxy contract. In the bytecode access to this variable will be represented as SLOAD(2), simply loading whatever data is stored in storage slot 2. When the PausableTokenDelegate contract gets compiled, the compiler again needs to decide where the state variables will be placed in the storage of this contract. address owner may or may not end up in the same storage slot. This highly depends on other state variables and compiler internals. No fixed slots are guaranteed and any arbitrary memory allocation can be chosen.

Assume for the PausableTokenDelegate the compiler decides to place address owner in storage slot 3. Consequently accesses to this variable will be encoded by SLOAD(3) in the bytecode of the PausableDelegateToken contract.

Now DELEGATECALL is used to execute this bytecode. A transaction calls the proxy which redirects it via DELEGATECALL to the actual implementation. Now the bytecode of the implementation is executed inside the context of the proxy contract. This bytecode however expects the address of the owner at storage slot 3 and simply accesses it via SLOAD(3).

During the execution, the variable owner will be whatever is stored in storage slot 3 of the proxy contract. This is the reason why the separate KeyValueStorage contract is used for storing data.

Documentation and specifications of the project and its implementation would help to understand the design, its consequences and likely avoid such issues.

If DELEGATECALL is used, be very careful when accessing and writing to storage. All implications of the use of DELEGATECALL must be understood in detail. CHAINSECURITY recommends to investigate more deeply the consequences and implications of the project's upgradeability requirements.

We highly recommend to review the following documents which provide a highly detailed introduction and overview about proxy contracts and contract migration.[8] [9]

Duplicate onlyOwner check in mint L ✓ Fixed

Upon calling the mint function, the most derived implementation is executed. In this case, this is mint() as inherited from the WhitelistableTokenDelegate contract. As this function calls super.mint(), the next most derived implementation is also executed, which is mint in the MintabletokenDelegate contract. Both functions have the onlyOwner modifier. Thus the onlyOwner modifier is checked twice.

Fixed: GLOBCOIN removed the duplicate onlyOwner check.

[7]As this role was previously stored in the KeyValueStorage.
[8]https://blog.trailofbits.com/2018/09/05/contract-upgrade-anti-patterns/
[9]https://blog.trailofbits.com/2018/10/29/how-contract-migration-works/

Recommendations / Suggestions

- ✗ GLOBCOIN's implementation does not follow the solidity best practices[10]:
  - Event names should be CapWords while mixedCase should be used for function names. This is not respected in GLOBCOIN's code, notably in BurnableTokenDelegate and mintableTokenDelegate.
  - Functions should be in the order: constructor, fallback, **external**, public, **internal**, private.
- ✓ GLOBCOIN's code contains several spelling mistakes, e.g.
  - WhipableTokenDelegate instead of WipeableTokenDelegate,
  - Several occurrences of adress instead of **address**.

Post-audit comment: GLOBCOIN has fixed some of the issues above and has is aware of all the implications of those points which were not addressed. Given this awareness, GLOBCOIN has to perform no more code changes with regards to these recommendations.

[10] https://solidity.readthedocs.io/en/latest/style-guide.html