QuillAudits

Audit Report
April, 2023

For

SELFKEY

Table of Content

Executive Summary

Project Name Self Token by Selfkey Dao

Overview The SelfkeyiD Authorization contract provides users a means of validating privileges to make transactions. The Self tokens implemented here can only be minted when the said parameters (addresses, timestamps, and signers) match up with expectations.

Timeline 19 December, 2022 to 26 December, 2022

Method Manual Review, Functional Testing, Automated Testing etc.

Scope of Audit The scope of this audit was to analyze the Self Token codebase for quality, security, and correctness.

Fixed In *https://github.com/selfkey-dao/self-erc20-token/blob/main/contracts/SelfToken.sol*
Branch: master
Commit Hash: 1d1a9e7afc85cfeec17a2cd27a835af7a011719e

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	**1**	**1**	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	**3**	**1**	**5**

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - SelfToken.sol

High Severity Issues

No issues were found

Medium Severity Issues

A1. Uncapped minting / token supply

Description

The Self token does not have any caps to the total supply, or any check to the number of tokens that can be minted. If a transaction gets mistakenly approved to mint a large amount of tokens, it can lead to drastic price changes if the address approved to decides to swap these tokens thereby draining liquidity and causing high impact on the token price.

Remediation

Consider having limits to token amounts that can be minted / total supply of the token.

Status

Acknowledged

Self Token Team Comment - Acknowledged, this is intended as the Self token will be inflationary by design. Our future products' smart contracts will make use of the Self token's Burn function in order to mitigate the inflation. Users will need to Burn Self in order to perform transactions or other on-chain activities in our future products. Furthermore, Self token will be a part of the SelfkeyDAO, which will regulate the amount of Self that can be minted per person per day, which will act as an on-chain deterrent to any such incidents.

A2. Address Validation

Description

Some critical functions do not perform address validation. Any address can be passed in, including the dead address/zero address which will inhibit contract functionality. The affected functions are setAuthorizationContractAddress() and the constructor().

Remediation

Implement address checks before critical changes to the contract state occur.

Status

Resolved

A3. Privilege and Ownership transfer risks

Description

The setAuthorizationContractAddress( ) function in the contract allows the current contract owner to set another address as the authorization contract. This can lead to a case where the owner has set an incorrect address and wants to reverse the change.

The same vector applies to the transferOwnership( ) function from the inherited Ownable contract. Ownership can be transferred to an incorrect or inaccessible address and will in turn handicap the contract functionality.

Remediation

It is advised to make ownership and privilege transfer a two-step process OR override the transferOwnership( ) function if it will not be implemented in the scope of this contract. Also, ensure proper management on the owner private key to prevent compromise which could lead to setting a malicious AuthorizationContractAddress.

References

- *Link1*
- *Link2*

Status

Resolved

Low Severity Issues

No issues were found

Informational Issues

A4. Unlocked pragma (pragma solidity >=0.8.0)

Description

Contracts should be deployed with the same compiler version that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock all.

Status

Resolved

A5. Unused code

Description

The internal burn mechanism is defined in the ERC20.sol library and is overwritten in the SelfToken.sol contract but is not called or used in the scope of the SelfToken.sol contract.

Remediation

If there is no need for the above-mentioned function, consider removing the unused code to reduce the contract size.

Status

Resolved

A6. Wrong comments

Description

The comments made above the functions totalSupply() and balanceOf() are swapped, thus providing misleading information to anyone reading through the codebase.

Remediation

Adjust the comments describing both functions to enhance code readability and reduce possible queries from other code readers.

Status

Resolved

B. Contract - SelfkeyIdAuthorization.sol

High Severity Issues

No issues were found

Medium Severity Issues

B1. Address Validation

Description

Some critical functions do not perform address validation. Any address can be passed in, including the dead address/zero address which will inhibit contract functionality. The affected functions are changeSignerAddress() and initialize(). If the signer address gets changed / set to an incorrect address, the entire contract can become crippled.

Remediation

Implement address checks before critical changes to state occur.

Status

Resolved

B2. Assembly Usage

Description

The SelfkeyIdAuthorization.sol contract contains a block of assembly code which bypasses all checks of the solidity compiler and is not recommended for use. The assembly code attempts to retrieve the values of v, r and s from the signature.

Remediation

Consider alternatives to the assembly block implemented here as assembly code is not checked by the solidity compiler before it is run.

Status

Acknowledged

Self Token Team Comment: We acknowledge this, we think is a very contained function, that will not change in the future, and is also used elsewhere, we consider the risk to be minimal.

B3. Privilege transfer

Description

The changeSignerAddress() function in the contract allows any address with the CONTROLLER_ROLE role to set another address as the signer. This can lead to a case where an incorrect address is set there is a need to reverse the change. If this new authorizedSigner differs from the signer address passed in the authorize() function, none of the tries to authorize transactions will pass.
The attack scenario goes as follows: Malicious / Compromised Owner sets new Signer Address, the attacker gets the payload verified with new signer and mints tokens.

Remediation

It is advised to make privilege transfer a two-step process.

References

- *Link1*
- *Link2*

Status

Resolved

Informational Issues

B4. Unlocked pragma (pragma solidity >=0.8.0)

Description

Contracts should be deployed with the same compiler version that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock all.

Status

Resolved

B5. Recursive imports

Description

The imported and inherited “AccessControlUpgradeable.sol” OpenZeppelin contract also inherits the “Initializable.sol” OpenZeppelin contract thus extending its functions and usage into the SelfkeyIdAuthorization.sol contract. Importing the Initializable.sol contract will be unnecessary.

Remediation

Consider removing the import statement on line 5.

Status

Resolved

SelfToken Team Comment: AccessControl and Upgradability was removed.

Functional Testing

Some of the tests performed are mentioned below:

- ✓ should only accept transactions with proper signatures
- ✓ should fail on signature replay attacks
- ✓ should only accept valid payloads
- ✓ should only accept payloads with trusted signers
- ✓ should not accept payloads with a zero-address signer

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

```
SelfkeyIdAuthorization.authorize(address,address,uint256,string,uint256,address,bytes) (contracts/SelfkeyIdAuthorization.sol#40-53) uses tx.origin for authorization: require(bool,string)(_to == tx.origin,Invalid subject) (contrac
ts/SelfkeyIdAuthorization.sol#44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-usage-of-txorigin

LockToken.transfer(address,uint256).owner (contracts/LockToken.sol#75) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#43-45) (function)
LockToken.allowance(address,address).owner (contracts/LockToken.sol#83) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#43-45) (function)
LockToken.approve(address,uint256).owner (contracts/LockToken.sol#98) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#43-45) (function)
LockToken.increaseAllowance(address,uint256).owner (contracts/LockToken.sol#139) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#43-45) (function)
LockToken.decreaseAllowance(address,uint256).owner (contracts/LockToken.sol#159) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#43-45) (function)
LockToken._approve(address,address,uint256).owner (contracts/LockToken.sol#269) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#43-45) (function)
LockToken._spendAllowance(address,address,uint256).owner (contracts/LockToken.sol#285) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#43-45) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

LockToken.constructor(address)._authorizationContractAddress (contracts/LockToken.sol#29) lacks a zero-check on :
                - authorizationContractAddress = _authorizationContractAddress (contracts/LockToken.sol#30)
LockToken.setAuthorizationContractAddress(address)._authorizationContractAddress (contracts/LockToken.sol#34) lacks a zero-check on :
                - authorizationContractAddress = _authorizationContractAddress (contracts/LockToken.sol#35)
SelfkeyIdAuthorization.initialize(address)._signer (contracts/SelfkeyIdAuthorization.sol#23) lacks a zero-check on :
                - authorizedSigner = _signer (contracts/SelfkeyIdAuthorization.sol#28)
SelfkeyIdAuthorization.changeSignerAddress(address)._signer (contracts/SelfkeyIdAuthorization.sol#33) lacks a zero-check on :
                - authorizedSigner = _signer (contracts/SelfkeyIdAuthorization.sol#35)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in LockToken.mint(address,uint256,uint256,address,bytes) (contracts/LockToken.sol#43-48):
        External calls:
        - authorizationContract.authorize(address(this),_to,_amount,mint:lock,_timestamp,_signer,signature) (contracts/LockToken.sol#46)
        State variables written after the call(s):
        - _mint(msg.sender,_amount) (contracts/LockToken.sol#47)
                - _balances[account] += amount (contracts/LockToken.sol#220)
        - _mint(msg.sender,_amount) (contracts/LockToken.sol#47)
                - _totalSupply += amount (contracts/LockToken.sol#217)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in LockToken.mint(address,uint256,uint256,address,bytes) (contracts/LockToken.sol#43-48):
        External calls:
        - authorizationContract.authorize(address(this),_to,_amount,mint:lock,_timestamp,_signer,signature) (contracts/LockToken.sol#46)
        Event emitted after the call(s):
        - Transfer(address(0),account,amount) (contracts/LockToken.sol#222)
                - _mint(msg.sender,_amount) (contracts/LockToken.sol#47)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

SelfkeyIdAuthorization.authorize(address,address,uint256,string,uint256,address,bytes) (contracts/SelfkeyIdAuthorization.sol#40-53) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_timestamp > timeLimit,Invalid timestamp) (contracts/SelfkeyIdAuthorization.sol#42)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#174-194) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#186-189)
```

```
SelfkeyIdAuthorization.splitSignature(bytes) (contracts/SelfkeyIdAuthorization.sol#78-98) uses assembly
        - INLINE ASM (contracts/SelfkeyIdAuthorization.sol#80-96)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['>=0.8.0', '^0.8.0', '^0.8.1', '^0.8.2']
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - >=0.8.0 (contracts/ISelfkeyIdAuthorization.sol#2)
        - >=0.8.0 (contracts/LockToken.sol#3)
        - >=0.8.0 (contracts/SelfkeyIdAuthorization.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

LockToken._burn(address,uint256) (contracts/LockToken.sol#238-254) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version>=0.8.0 (contracts/ISelfkeyIdAuthorization.sol#2) allows old versions
Pragma version>=0.8.0 (contracts/LockToken.sol#3) allows old versions
Pragma version>=0.8.0 (contracts/SelfkeyIdAuthorization.sol#3) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

SelfkeyIdAuthorization (contracts/SelfkeyIdAuthorization.sol#9-99) should inherit from ISelfkeyIdAuthorization (contracts/ISelfkeyIdAuthorization.sol#4-17)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance
```

```
Function AccessControlUpgradeable.__AccessControl_init() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#51-52) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#54-55) is not in mixedCase
Variable AccessControlUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#259) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#24-25) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#27-28) is not in mixedCase
Variable ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#41) is not in mixedCase
Parameter LockToken.setAuthorizationContractAddress(address)._authorizationContractAddress (contracts/LockToken.sol#34) is not in mixedCase
Parameter LockToken.mint(address,uint256,uint256,address,bytes)._to (contracts/LockToken.sol#43) is not in mixedCase
Parameter LockToken.mint(address,uint256,uint256,address,bytes)._amount (contracts/LockToken.sol#43) is not in mixedCase
Parameter LockToken.mint(address,uint256,uint256,address,bytes)._timestamp (contracts/LockToken.sol#43) is not in mixedCase
Parameter LockToken.mint(address,uint256,uint256,address,bytes)._signer (contracts/LockToken.sol#43) is not in mixedCase
Parameter SelfkeyIdAuthorization.initialize(address)._signer (contracts/SelfkeyIdAuthorization.sol#23) is not in mixedCase
Parameter SelfkeyIdAuthorization.changeSignerAddress(address)._signer (contracts/SelfkeyIdAuthorization.sol#33) is not in mixedCase
Parameter SelfkeyIdAuthorization.authorize(address,address,uint256,string,uint256,address,bytes)._from (contracts/SelfkeyIdAuthorization.sol#40) is not in mixedCase
Parameter SelfkeyIdAuthorization.authorize(address,address,uint256,string,uint256,address,bytes)._to (contracts/SelfkeyIdAuthorization.sol#40) is not in mixedCase
Parameter SelfkeyIdAuthorization.authorize(address,address,uint256,string,uint256,address,bytes)._amount (contracts/SelfkeyIdAuthorization.sol#40) is not in mixedCase
Parameter SelfkeyIdAuthorization.authorize(address,address,uint256,string,uint256,address,bytes)._scope (contracts/SelfkeyIdAuthorization.sol#40) is not in mixedCase
Parameter SelfkeyIdAuthorization.authorize(address,address,uint256,string,uint256,address,bytes)._timestamp (contracts/SelfkeyIdAuthorization.sol#40) is not in mixedCase
Parameter SelfkeyIdAuthorization.authorize(address,address,uint256,string,uint256,address,bytes)._signer (contracts/SelfkeyIdAuthorization.sol#40) is not in mixedCase
Parameter SelfkeyIdAuthorization.authorize(address,address,uint256,string,uint256,address,bytes)._signature (contracts/SelfkeyIdAuthorization.sol#40) is not in mixedCase
Parameter SelfkeyIdAuthorization.getMessageHash(address,address,uint256,string,uint256)._from (contracts/SelfkeyIdAuthorization.sol#55) is not in mixedCase
Parameter SelfkeyIdAuthorization.getMessageHash(address,address,uint256,string,uint256)._to (contracts/SelfkeyIdAuthorization.sol#55) is not in mixedCase
Parameter SelfkeyIdAuthorization.getMessageHash(address,address,uint256,string,uint256)._amount (contracts/SelfkeyIdAuthorization.sol#55) is not in mixedCase
Parameter SelfkeyIdAuthorization.getMessageHash(address,address,uint256,string,uint256)._scope (contracts/SelfkeyIdAuthorization.sol#55) is not in mixedCase
Parameter SelfkeyIdAuthorization.getMessageHash(address,address,uint256,string,uint256)._timestamp (contracts/SelfkeyIdAuthorization.sol#55) is not in mixedCase
Parameter SelfkeyIdAuthorization.getEthSignedMessageHash(bytes32)._messageHash (contracts/SelfkeyIdAuthorization.sol#59) is not in mixedCase
Parameter SelfkeyIdAuthorization.verify(address,address,uint256,string,uint256,address,bytes)._from (contracts/SelfkeyIdAuthorization.sol#67) is not in mixedCase
Parameter SelfkeyIdAuthorization.verify(address,address,uint256,string,uint256,address,bytes)._to (contracts/SelfkeyIdAuthorization.sol#67) is not in mixedCase
Parameter SelfkeyIdAuthorization.verify(address,address,uint256,string,uint256,address,bytes)._amount (contracts/SelfkeyIdAuthorization.sol#67) is not in mixedCase
Parameter SelfkeyIdAuthorization.verify(address,address,uint256,string,uint256,address,bytes)._scope (contracts/SelfkeyIdAuthorization.sol#67) is not in mixedCase
Parameter SelfkeyIdAuthorization.verify(address,address,uint256,string,uint256,address,bytes)._timestamp (contracts/SelfkeyIdAuthorization.sol#67) is not in mixedCase
Parameter SelfkeyIdAuthorization.verify(address,address,uint256,string,uint256,address,bytes)._signer (contracts/SelfkeyIdAuthorization.sol#67) is not in mixedCase
Parameter SelfkeyIdAuthorization.recoverSigner(bytes32,bytes)._ethSignedMessageHash (contracts/SelfkeyIdAuthorization.sol#73) is not in mixedCase
Parameter SelfkeyIdAuthorization.recoverSigner(bytes32,bytes)._signature (contracts/SelfkeyIdAuthorization.sol#73) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

AccessControlUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#259) is never used in SelfkeyIdAuthorization (contracts/SelfkeyIdAuthorization.sol#9-99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

grantRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.grantRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#150-152)
revokeRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.revokeRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#165-167)
renounceRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.renounceRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#185-189)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#61-63)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#69-72)
name() should be declared external:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72)
```

```
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#87-89)
totalSupply() should be declared external:
        - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#94-96)
        - LockToken.totalSupply() (contracts/LockToken.sol#54-56)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#101-103)
        - LockToken.balanceOf(address) (contracts/LockToken.sol#62-64)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#113-117)
        - LockToken.transfer(address,uint256) (contracts/LockToken.sol#74-78)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#136-140)
        - LockToken.approve(address,uint256) (contracts/LockToken.sol#97-101)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#158-167)
        - LockToken.transferFrom(address,address,uint256) (contracts/LockToken.sol#119-124)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#181-185)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#201-210)
setAuthorizationContractAddress(address) should be declared external:
        - LockToken.setAuthorizationContractAddress(address) (contracts/LockToken.sol#34-37)
increaseAllowance(address,uint256) should be declared external:
        - LockToken.increaseAllowance(address,uint256) (contracts/LockToken.sol#138-142)
decreaseAllowance(address,uint256) should be declared external:
        - LockToken.decreaseAllowance(address,uint256) (contracts/LockToken.sol#158-167)
initialize(address) should be declared external:
        - SelfkeyIdAuthorization.initialize(address) (contracts/SelfkeyIdAuthorization.sol#23-29)
changeSignerAddress(address) should be declared external:
        - SelfkeyIdAuthorization.changeSignerAddress(address) (contracts/SelfkeyIdAuthorization.sol#33-37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
. analyzed (16 contracts with 78 detectors), 98 result(s) found
```

Closing Summary

In this report, we have considered the security of the Self Token. We performed our audit according to the procedure described above.

Some issues of Medium, Low and Informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Self Token. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Self Token Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2023

For

SELFKEY

QuillAudits

Canada, India, Singapore, United Kingdom

www.quillaudits.com

audits@quillhash.com