CERTIK

Security Assessment

Aircarbon Pte. Ltd. - Security Token Master (STM)

Aug 23rd, 2021

Table of Contents

Summary

This report has been prepared for AirCarbon to discover issues and vulnerabilities in the source code of the Aircarbon Pte. Ltd. - Security Token Master (STM) project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Aircarbon Pte. Ltd. - Security Token Master (STM)
Description	A multi-asset, multi-collateral on-chain ledger implementation facilitating atomic spot trading as well as future derivatives with built-in position management, settlement and liquidation.
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/aircarbon/stm
Commit	537a084fb996d69f7ce0f34d1036797c3f575b76 55e29990079257acd6fa3dd8232a4ef6409e20b8 b16d33df790b6afb40554dc4950951e19f983b02

Audit Summary

Delivery Date	Aug 23, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	Library Composability, Fund Flows, Sane Accounting System

Vulnerability Summary

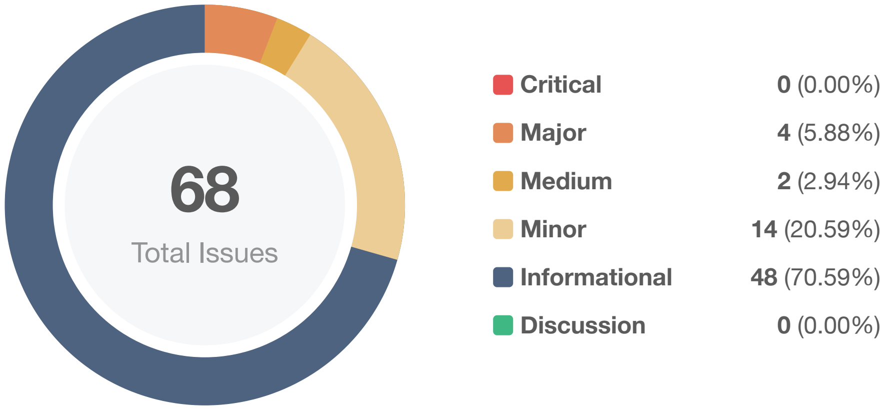
Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● Major	4	0	0	0	0	4
● Medium	2	0	0	0	0	2
● Minor	14	0	0	6	0	8
● Informational	48	0	3	6	0	39
● Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
ICA	Interfaces/IChainlinkAggregator.sol	317329c4886d4c2c2b390e5f04802421fa3be6f999caacbd5a348795a41d9bf0
SLI	Interfaces/StructLib.sol	4260240fdb6823d17e92355c6c63e8b831eeb7f7159eae1173985c928dcbd514
CLL	Libs/CcyLib.sol	fcb20ced301f55a410d01cd1305e37ff44eee208adaf30a7674039870c906389
ELL	Libs/Erc20Lib.sol	2a58de385c699eea10adeb123f2d1af46b50f8ebb1841ca78981269bac632dd4
LLL	Libs/LedgerLib.sol	582bd790b1477376d60da5e8e3aae653c6b05055a9a09de8b7714554f2e208cf
LOA	Libs/LoadLib.sol	12e584ae6c76359d4e02af93bab475fdf1a154861f6affb24a3762db4346f729
PLL	Libs/PayableLib.sol	9da0b03f3f8e409484ba8e66ede32eda646917a680add3d10f0fd0220dfdb4fa
SML	Libs/SafeMath.sol	a644e4558659f5d8d58f4dded8d284d5a48ea3a0f75ed34a6bb73237a4499665
SFL	Libs/SpotFeeLib.sol	6155b56a4cc1229fd222a2138b9c0768b4c52802f0676aa7ca70c5b36751921f
SLN	Libs/Strings.sol	e2850fdb43c8853560cda5df67ad8955ec1bc2720abe0e6186ce9a56d465ab28
TLL	Libs/TokenLib.sol	5f0034d52d8c9ec65edae57e71622518fe52b4c911963748fe2f0a8a583cc0e5
TRA	Libs/TransferLib.sol	4e99905d25161435dd381eda935d12272cd25b2b02c9477ef2d29c76464d6d03
CCS	StMaster/CcyCollateralizable.sol	319284e3c5e71e1466d23ef01b8a393971c39fc1488503b7d93cfc1bc6e7d07e
DLS	StMaster/DataLoadable.sol	79606e6aa106b0e6f5fba1581673b473d7003629461a7cc02959c4b0dc9b03f0
MSM	StMaster/Migrations.sol	f564e4cd46aca83e922c7c5ee436fcedd0ae80e6e94a0791b87d5ccf9010c638
OSM	StMaster/Owned.sol	35a6f095998f53ec1d94a25207e75218d25e33ded4fd217c57349c5e9c400803
SBS	StMaster/StBurnable.sol	04baa84b346519cbd2b53c63f55f580a393175b33d37c294abdece6e45f22547
SES	StMaster/StErc20.sol	3b099fa417c21ea560348d772adb136b045ffcc35306448bfa22e31f6c0a56e2
SFS	StMaster/StFees.sol	143789f1d45912657d3b054049258ead0e8888a3ebf6d8934a39fbfac3d3086e
SLS	StMaster/StLedger.sol	4c2a0e214798f910cf681a549e49c9537073638d893335c4b0cb5abe4dc35f71
SMS	StMaster/StMaster.sol	3235f5b7c614cd1269076e2447eab589df867bb918abce6469cbc01606c4b964
SMM	StMaster/StMintable.sol	42ff770678793f05f696f773c7a35953a792f8d6fa7549c777ccf7ee81d8f714

ID		File	SHA256 Checksum
SPS		StMaster/StPayable.sol	256d8ccface72d36cc91b1ed5bdf6a4f8a3a6e05e01b5cad9248fedeba348754
STS		StMaster/StTransferable.sol	4b8171d9cb30b4664081297adf8a02e2eaaae89ee5d6caf1373ccfc0b854f656

Findings



ID	Title	Category	Severity	Status
CCS-01	Unlocked Compiler Version	Language Specific	● Informational	✓ Resolved
CCS-02	Ineffectual inheriting of contract `Owned`	Language Specific	● Informational	✓ Resolved
CLL-01	Unlocked Compiler Version	Language Specific	● Informational	✓ Resolved
CLL-02	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
CLL-03	Inefficient code	Gas Optimization	● Informational	ⓘ Acknowledged
CLL-04	Explicitly returning local variable	Gas Optimization	● Informational	✓ Resolved
CLL-05	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
CLL-06	Comparison with literal `true`	Gas Optimization	● Informational	✓ Resolved
ELL-01	Unlocked Compiler Version	Language Specific	● Informational	✓ Resolved
ELL-02	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
ELL-03	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
ICA-01	Incorrect Chainlink Interface	Logical Issue	● Major	✓ Resolved
LLL-01	Unlocked Compiler Version	Language Specific	● Informational	✓ Resolved
LLL-02	Redundant Variable Initialization	Gas Optimization	● Informational	✓ Resolved
LLL-03	Inefficient use of `for` loop	Gas Optimization	● Informational	✓ Resolved
LLL-04	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved

ID	Title	Category	Severity	Status
LLL-05	Explicitly returning local variable	Gas Optimization	● Informational	✓ Resolved
LLL-06	Inefficient storage read	Gas Optimization	● Informational	ⓘ Acknowledged
LLL-07	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
LLL-08	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
LLL-09	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
LLL-10	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
LLL-11	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
LLL-12	Inefficient storage read	Gas Optimization	● Informational	ⓘ Acknowledged
LLL-13	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
LLL-14	Inefficient storage read	Gas Optimization	● Informational	ⓘ Acknowledged
LLL-15	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
LLL-16	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved
LOA-01	Potentially Disjoint Variable	Logical Issue	● Minor	ⓘ Acknowledged
LOA-02	Duplication of Code	Coding Style	● Informational	⊗ Declined
LOA-03	Inexistent Balance Sanitization	Logical Issue	● Minor	ⓘ Acknowledged
LOA-04	Lookup Optimizations	Gas Optimization	● Informational	✓ Resolved
LOA-05	Inexistent Duplicate Check	Logical Issue	● Minor	ⓘ Acknowledged
LOA-06	Inexistent Initializaiton Check	Logical Issue	● Minor	ⓘ Acknowledged
LOA-07	Inexistent Entry Check	Logical Issue	● Minor	ⓘ Acknowledged
OSM-01	Unlocked Compiler Version	Language Specific	● Informational	✓ Resolved
OSM-02	Inefficient storage layout	Gas Optimization	● Informational	✓ Resolved
OSM-03	Return Variable Utilization	Gas Optimization	● Informational	✓ Resolved
OSM-04	Redundant Variable Initialization	Gas Optimization	● Informational	✓ Resolved
OSM-05	Inefficient storage read	Gas Optimization	● Informational	✓ Resolved

ID	Title	Category	Severity	Status
OSM-06	Comparison with literal `false`	Gas Optimization	Informational	Resolved
OSM-07	Inexistent Management Functionality	Logical Issue	Minor	Resolved
OSM-08	Usage of `tx.origin`	Logical Issue	Minor	Resolved
PLL-01	Incorrect Chainlink Handling	Language Specific	Minor	Resolved
PLL-02	Inexistent Reentrancy Guard	Logical Issue	Medium	Resolved
PLL-03	Incorrect Limit Evaluation	Mathematical Operations	Medium	Resolved
SES-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
SES-02	Function Visibility Optimization	Gas Optimization	Informational	Resolved
SES-03	Return Variable Utilization	Gas Optimization	Informational	Resolved
SES-04	Insufficient `require` Check	Coding Style	Informational	Acknowledged
SFL-01	Potentially Incorrect Clauses	Logical Issue	Minor	Resolved
SFL-02	Inexistent Event	Coding Style	Informational	Acknowledged
SLI-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
SLI-02	Redundant Variable Initialization	Gas Optimization	Informational	Resolved
SLI-03	Inefficient storage read	Gas Optimization	Informational	Resolved
SLI-04	Variable Tight Packing	Gas Optimization	Informational	Declined
SLI-05	Unsafe Cast	Logical Issue	Major	Resolved
SLI-06	Non-Standard Decimal Representation	Language Specific	Informational	Declined
SLI-07	Loop Optimizations	Gas Optimization	Informational	Resolved
SLI-08	Unsafe Mathematical Operations	Mathematical Operations	Minor	Resolved
SLI-09	State Representation Inconsistency	Inconsistency	Minor	Acknowledged
SLS-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
SLS-02	Return Variable Utilization	Gas Optimization	Informational	Resolved
TLL-01	Inexplicable Constant	Logical Issue	Major	Resolved

ID	Title	Category	Severity	Status
TLL-02	Unsafe Addition	Mathematical Operations	● Major	✓ Resolved
TLL-03	Potentially Negative Quantities	Logical Issue	● Minor	✓ Resolved
TRA-01	Equal ID Transfers	Logical Issue	● Minor	✓ Resolved
TRA-02	Potentially Negative Quantity	Mathematical Operations	● Minor	✓ Resolved

CCS-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	StMaster/CcyCollateralizable.sol: 3	✓ Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for `^0.6.12`, the version can be locked at `0.6.12`.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

CCS-02 | Ineffectual inheriting of contract `Owned`

Category	Severity	Location	Status
Language Specific	● Informational	StMaster/CcyCollateralizable.sol: 6, 23	✓ Resolved

Description

The direct inheriting of contract `Owned` by contract `Collateralizable` is ineffectual as the contract `Owned` is already inherited by the contract `StLedger`.

Recommendation

We advise to remove the ineffectual inheriting of contract `Owned` by contract `Collateralizable` to increase the legibility of codebase.

Alleviation

Alleviations are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

CLL-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	Libs/CcyLib.sol: 3	✓ Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for `^0.6.12`, the version can be locked at `0.6.12`.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

CLL-02 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/CcyLib.sol: 26, 28	✓ Resolved

Description

The aforementioned line(s) read `ctd._ct_Count` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `ctd._ct_Count` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

CLL-03 | Inefficient code

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/CcyLib.sol: 28~30	ⓘ Acknowledged

Description

The `for` loop on the aforementioned lines check against the currency being added does not have an already registered name. The current implementation loops through all the existing currencies which results in significant gas cost.

Recommendation

We recommend to introduce a `mapping` within the struct `StructLib.CcyTypesStruct` that keeps track of the registered names of currencies and the `for` loop on the aforementioned can be replaced with the look-up of the introduced mapping which will have constant gas cost.

Alleviation

Author advises that the number of currencies in the contract is not expected to be large, so the looping construct's gas usage vs. mapping is probably acceptable. But noted as an improvement for a future version.

CLL-04 | Explicitly returning local variable

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/CcyLib.sol: 42	✓ Resolved

Description

The function on the aforementioned line explicitly returns local variable which increase overall cost of gas.

Recommendation

Since named return variables can be declared in the signature of a function, consider refactoring to remove the local variable declaration and explicit return statement in order to reduce the overall cost of gas.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

CLL-05 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/CcyLib.sol: 47, 49	✓ Resolved

Description

The aforementioned lines(s) read `ctd._ct_Count` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `ctd._ct_Count` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

CLL-06 | Comparison with literal `true`

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/CcyLib.sol: 123	✓ Resolved

Description

The aforementioned line performs comparison with literal `true` which can be substituted with the expression itself to save gas cost.

Recommendation

We advise to substitute the comparison with literal `true` with the expression itself.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

ELL-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	Libs/Erc20Lib.sol: 3	✓ Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for `^0.6.12`, the version can be locked at `0.6.12`.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

ELL-02 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/Erc20Lib.sol: 91	✓ Resolved

Description

The aforementioned line reads `std._tt_Count` from contract's storage upon each iteration of `for` loop which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `std._tt_Count` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

ELL-03 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/Erc20Lib.sol: 97~98	✓ Resolved

Description

The aforementioned line(s) read `ld._sts[tokenType_stIds[ndx]].currentQty` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `ld._sts[tokenType_stIds[ndx]].currentQty` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

ICA-01 | Incorrect Chainlink Interface

Category	Severity	Location	Status
Logical Issue	● Major	Interfaces/IChainlinkAggregator.sol: 9~18	✓ Resolved

Description

The Chainlink interface specified is incorrect as it is outdated and does not properly yield the extra variables Chainlink returns to sanitize the reading's validity.

Recommendation

We advise the interface and its contract integration to be updated accordingly to the Chainlink documentation.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	Libs/LedgerLib.sol: 3	✓ Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for `^0.6.12`, the version can be locked at `0.6.12`.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-02 | Redundant Variable Initialization

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 34, 54, 359	✓ Resolved

Description

The `uint256` type variables on the aforementioned lines are assigned `0` default value.

Recommendation

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-03 | Inefficient use of `for` loop

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 43~45	✓ Resolved

Description

The `for` loop on the aforementioned line is inefficient and can be substituted with a line incrementing `countAllSecTokens` by `baseLedgers[tokTypeId - 1].tokens.length` to save gas cost.

Recommendation

We advise to substitute the `for` loop on the aforementioned line with a line incrementing `countAllSecTokens` by `baseLedgers[tokTypeId - 1].tokens.length`.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-04 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 111	✓ Resolved

Description

The aforementioned line reads `ld._batches_currentMax_id` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `ld._batches_currentMax_id` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-05 | Explicitly returning local variable

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 23, 154, 358	✓ Resolved

Description

The functions on the aforementioned lines explicitly return local variable which increases overall cost of gas.

Recommendation

Since named return variables can be declared in the signature of a function, consider refactoring to remove the local variable declaration and explicit return statement in order to reduce the overall cost of gas.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-06 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 186~187	ⓘ Acknowledged

Description

The aforementioned lines read `globalFees.ccyType_Set[ccyTypeId]` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `globalFees.ccyType_Set[ccyTypeId]` and then utilize it to save gas cost.

Alleviation

No alleviations are considered and the team responded that "Noted on the gas cost of this function. The view is only intended to be run by the platform operator internally, and note that the gas cost is mitigated by the parameterized segmentation of the work (modulo pattern), such that the gas cost of each iteration can be controlled by the caller. Noted as a future improvement to pack more data into memory and reduce the number of keccak256 ops.".

LLL-07 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 192	✓ Resolved

Description

The aforementioned lines read `std._tt_Count` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `std._tt_Count` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-08 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 215	✓ Resolved

Description

The aforementioned lines read `erc20d._whitelist.length` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `erc20d._whitelist.length` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-09 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 231	✓ Resolved

Description

The aforementioned lines read `ld._batches_currentMax_id` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `ld._batches_currentMax_id` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-10 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 255	✓ Resolved

Description

The aforementioned line reads `ld._ledgerOwners.length` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `ld._ledgerOwners.length` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-11 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 266	✓ Resolved

Description

The aforementioned line reads `std._tt_Count` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `std._tt_Count` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-12 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 274~275	ⓘ Acknowledged

Description

The aforementioned lines read `entry.spot_customFees.tokType_Set[stTypeId]` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `entry.spot_customFees.tokType_Set[stTypeId]` and then utilize it to save gas cost.

Alleviation

No alleviations are considered and the team responded that "Noted on the gas cost of this function. The view is only intended to be run by the platform operator internally, and note that the gas cost is mitigated by the parameterized segmentation of the work (modulo pattern), such that the gas cost of each iteration can be controlled by the caller. Noted as a future improvement to pack more data into memory and reduce the number of keccak256 ops.".

LLL-13 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 284	✓ Resolved

Description

The aforementioned line reads `ctd._ct_Count` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `ctd._ct_Count` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-14 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 287~288	ⓘ Acknowledged

Description

The aforementioned lines read `entry.spot_customFees.ccyType_Set[ccyTypeId]` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `entry.spot_customFees.ccyType_Set[ccyTypeId]` and then utilize it to save gas cost.

Alleviation

No alleviations are considered and the team responded that "Noted on the gas cost of this function. The view is only intended to be run by the platform operator internally, and note that the gas cost is mitigated by the parameterized segmentation of the work (modulo pattern), such that the gas cost of each iteration can be controlled by the caller. Noted as a future improvement to pack more data into memory and reduce the number of keccak256 ops.".

LLL-15 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 301	✓ Resolved

Description

The aforementioned line reads `std._tt_Count` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `std._ct_Count` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LLL-16 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LedgerLib.sol: 310	✓ Resolved

Description

The aforementioned line reads `ld._tokens_currentMax_id` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `ld._tokens_currentMax_id` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LOA-01 | Potentially Disjoint Variable

Category	Severity	Location	Status
Logical Issue	● Minor	Libs/LoadLib.sol: 10~21	ⓘ Acknowledged

Description

The `_batches_currentMax_id` variable is set as an argument instead of calculated within the `for` loop that loads the tokens and is not sanitized.

Recommendation

We advise the new maximum to be calculated dynamically by first storing the current maximum in an in-memory variable and checking whether each upcoming `id` is greater than the currently set maximum, in which case it is stored to be later assigned to storage once the loop concludes.

Alleviation

No alleviations are considered and the team stated that the implementation is logically consistent with architectural design for contract upgrade".

LOA-02 | Duplication of Code

Category	Severity	Location	Status
Coding Style	● Informational	Libs/LoadLib.sol: 37~41	ⓧ Declined

Description

The linked statements are duplicated from the `StructLib` function's ledger initialization function.

Recommendation

We advise the `StructLib` implementation to be adjusted to accept the `spot_sumQtyMinted` and `spot_sumQtyBurned` variables as arguments, thereby preventing code duplication.

Alleviation

No alleviations are considered.

LOA-03 | Inexistent Balance Sanitization

Category	Severity	Location	Status
Logical Issue	● Minor	Libs/LoadLib.sol: 49~52	ⓘ Acknowledged

Description

The linked `for` loop does not sanitize the `reserve` member of `ccys[i]` to be less-than-or-equal-to the `balance` member.

Recommendation

We advise it to be done so to ensure proper state transitions.

Alleviation

No alleviations are considered and the team stated that it is logically consistent with architectural design for contract upgrade.

LOA-04 | Lookup Optimizations

Category	Severity	Location	Status
Gas Optimization	● Informational	Libs/LoadLib.sol: 50~51	✓ Resolved

Description

The linked statements perform `mapping` lookups that can be cached to memory.

Recommendation

We advise them to be done so to greatly optimize the contract's gas cost as each lookup performs a `keccak256` operation.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

LOA-05 | Inexistent Duplicate Check

Category	Severity	Location	Status
Logical Issue	● Minor	Libs/LoadLib.sol: 55~71	ⓘ Acknowledged

Description

The `addSecToken` can overwrite over a currently present security token ID as no sanitization is performed to ensure the security token hasn't already been added.

Recommendation

We advise such a check to be introduced to prevent malicious overwrites of storage.

Alleviation

No alleviations are considered and the team stated that it is logically consistent with architectural design for contract upgrade.

LOA-06 | Inexistent Initializaiton Check

Category	Severity	Location	Status
Logical Issue	● Minor	Libs/LoadLib.sol: 38~41	ⓘ Acknowledged

Description

The ledger that is initialized within `createLedgerEntry` isn't validated to not exist already, potentially allowing previously set `spot_sumQtyMinted` and `spot_sumQtyBurned` values to be overwritten.

Recommendation

We advise a corresponding check to be introduced here preventing this scenario from unfolding.

Alleviation

No alleviations are considered and the team stated that it is logically consistent with architectural design for contract upgrade.

LOA-07 | Inexistent Entry Check

Category	Severity	Location	Status
Logical Issue	● Minor	Libs/LoadLib.sol: 17~19	ⓘ Acknowledged

Description

The `loadSecTokenBatch` performs no input sanitization in the batch assignments it performs.

Recommendation

We advise a check to be introduced in the `for` loop to ensure that the batch `id` hasn't already been set to.

Alleviation

No alleviations are considered and the team stated that it is logically consistent with architectural design for contract upgrade.

OSM-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	StMaster/Owned.sol: 3	✓ Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for `^0.6.12`, the version can be locked at `0.6.12`.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

OSM-02 | Inefficient storage layout

Category	Severity	Location	Status
Gas Optimization	● Informational	StMaster/Owned.sol: 15	✓ Resolved

Description

The `bool` type storage variable on the aforementioned line can be placed next to `address` type variable on `L12` to tight pack both variables in a single storage slot.

Recommendation

We advise to place `bool` and `address` type variables next to each other.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

OSM-03 | Return Variable Utilization

Category	Severity	Location	Status
Gas Optimization	● Informational	StMaster/Owned.sol: 22	✓ Resolved

Description

The linked function declaration(s) contain explicitly named `return` variable(s) that are not utilized within the function's code block.

Recommendation

We advise that the linked variable(s) are either utilized or omitted from the declaration.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

OSM-04 | Redundant Variable Initialization

Category	Severity	Location	Status
Gas Optimization	● Informational	StMaster/Owned.sol: 26	✓ Resolved

Description

The `bool` type variable on the aforementioned line is assigned `false` default value.

Recommendation

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

OSM-05 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	StMaster/Owned.sol: 44	✓ Resolved

Description

The aforementioned line reads `owners.length` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `owners.length` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

OSM-06 | Comparison with literal `false`

Category	Severity	Location	Status
Gas Optimization	● Informational	StMaster/Owned.sol: 55	✓ Resolved

Description

The aforementioned line performs comparison with literal `false` which can be substituted with the negation of expression itself to save gas cost.

Recommendation

We advise to substitute the comparison with literal `false` with the negation of expression.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

OSM-07 | Inexistent Management Functionality

Category	Severity	Location	Status
Logical Issue	● Minor	StMaster/Owned.sol: 13	✓ Resolved

Description

The `Owned` contract implementation should be self-sufficient and possess adding and removing owners within it.

Recommendation

We advise such functionality to be introduced and utilized by inherited contracts instead of relying on the inherited contracts to implement proper ownership management processes.

Alleviation

Alleviations are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

OSM-08 | Usage of `tx.origin`

Category	Severity	Location	Status
Logical Issue	● Minor	StMaster/Owned.sol: 45	✓ Resolved

Description

The use of `tx.origin` should be avoided for ownership-based systems given that firstly it can be tricked on-chain and secondly it will change its functionality once EIP-3074 is integrated.

Recommendation

We advise a simple `msg.sender` invocation to be evaluated instead.

Alleviation

Alleviations are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

PLL-01 | Incorrect Chainlink Handling

Category	Severity	Location	Status
Language Specific	● Minor	Libs/PayableLib.sol: 26~31	✓ Resolved

Description

The Chainlink value reported should be handled for staleness as well as validity.

Recommendation

We advise it to be done so in accordance with the Chainlink documentation ensuring that the answer was submitted in a timeframe close to the latest one.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

PLL-02 | Inexistent Reentrancy Guard

Category	Severity	Location	Status
Logical Issue	● Medium	Libs/PayableLib.sol: 116~202	✓ Resolved

Description

The linked code performs an external native currency transfer but does not possess a guard against a re-entrancy. Although this issue is prevalent across multiple segments of the codebase, it is especially prevalent in this particular one due to the obvious attack vector.

Recommendation

We advise that the statements are either re-ordered to prevent a re-entrancy altogether or that the contracts that utilize this library are guarded by re-entrancy guards properly, as the current setup allows exceeding the minting cap by re-entering as the transfer of tokens to the payer is done after the native currency transfers.

Alleviation

Alleviations are applied by utilizing Re-entrancy guard in `StPayable` contract as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

PLL-03 | Incorrect Limit Evaluation

Category	Severity	Location	Status
Mathematical Operations	● Medium	Libs/PayableLib.sol: 256	✓ Resolved

Description

The limit of `msg.value` should be set to a strict `<` comparison as the value of `2**128` is not representable by `uint128` given that its maximum value is `2**128 - 1`.

Recommendation

We advise the `type(uint128).max` statement to be utilized instead to ensure such discrepancies do not arise in the future.

Alleviation

Alleviations are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

SES-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	StMaster/StErc20.sol: 3	✓ Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for `^0.6.12`, the version can be locked at `0.6.12`.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

SES-02 | Function Visibility Optimization

Category	Severity	Location	Status
Gas Optimization	● Informational	StMaster/StErc20.sol: 44	✓ Resolved

Description

The linked function is declared as `public`, contains array function arguments and is not invoked in any of the contract's contained within the project's scope.

Recommendation

We advise that the functions' visibility specifiers are set to `external` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

Alleviation

Alleviations are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

SES-03 | Return Variable Utilization

Category	Severity	Location	Status
Gas Optimization	● Informational	StMaster/StErc20.sol: 112, 124, 143, 154, 166	✓ Resolved

Description

The linked function declarations contain explicitly named `return` variables that are not utilized within the function's code block.

Recommendation

We advise that the linked variables are either utilized or omitted from the declaration.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

SES-04 | Insufficient `require` Check

Category	Severity	Location	Status
Coding Style	● Informational	StMaster/StErc20.sol: 124~132	ⓘ Acknowledged

Description

The `require` check imposed on a `transfer` may be insufficient as fees can be imposed at the library level.

Recommendation

We advise the evaluation to instead utilize an `Erc20Lib` function.

Alleviation

No alleviations are considered and the team responded that "Erc20Lib.transfer() internally calls TransferLib.transferOrTrade() which performs extensive validation of inputs, including sufficiency checks on quantities being transferred. The simple check being applied by StErc20.transfer() is a reasonable (and valid) preliminary check.".

SFL-01 | Potentially Incorrect Clauses

Category	Severity	Location	Status
Logical Issue	● Minor	Libs/SpotFeeLib.sol: 43, 47, 51, 55, 85, 89, 93, 97, 102	✓ Resolved

Description

The linked `if` clauses emit an event when the value is being set, however, they do so when the value is simply non-zero rendering the first conditional questionable.

Recommendation

We advise the clauses to be coupled with an and-clause (`&&`) instead of an or-clause (`||`) to properly conduct sanitization of whether a change occured. Additionally, we advise the non-zero check to be omitted as the value may be set to `0`.

Alleviation

Alleviation are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

SFL-02 | Inexistent Event

Category	Severity	Location	Status
Coding Style	● Informational	Libs/SpotFeeLib.sol: 107	ⓘ Acknowledged

Description

The contract emits an event whenever all variables are updated except `ccy_mirrorFee`.

Recommendation

We advise an event to also be emitted for this variable in accordance to the rest of the codebase.

Alleviation

No alleviations are considered and the team responded that "By design; the intention of the event emission in this function is to log whenever a numerical element of the fee structure changes. The bool field ccy_mirrorFee represents whether or not the numerical ccy fee is applied to each side; it doesn't affect the quantity of the fee itself.".

SLI-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	Interfaces/StructLib.sol: 3	✓ Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for `^0.6.12`, the version can be locked at `0.6.12`.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

SLI-02 | Redundant Variable Initialization

Category	Severity	Location	Status
Gas Optimization	● Informational	Interfaces/StructLib.sol: 423~424, 446	✓ Resolved

Description

The `uint256` type variables on the aforementioned lines are assigned `0` default value.

Recommendation

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

SLI-03 | Inefficient storage read

Category	Severity	Location	Status
Gas Optimization	● Informational	Interfaces/StructLib.sol: 447	✓ Resolved

Description

The aforementioned line reads `ld._ledger[ledger].tokenType_stIds[tokTypeId].length` from contract's storage repeatedly which can be optimized by storing it in a local variable and then utilizing it. This will reduce gas cost associated multiple read operations of contract's storage.

Recommendation

We advise to utilize a local variable to store `ld._ledger[ledger].tokenType_stIds[tokTypeId].length` and then utilize it to save gas cost.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

SLI-04 | Variable Tight Packing

Category	Severity	Location	Status
Gas Optimization	● Informational	Interfaces/StructLib.sol: 56~62, 153~161, 164~175	⊗ Declined

Description

The linked `struct` declarations can be tight-packed by re-ordering the variables' order of declaration.

Recommendation

We advise it to be done so to lower the storage slots involved in handling the struct.

Alleviation

No alleviations are considered by the team.

SLI-05 | Unsafe Cast

Category	Severity	Location	Status
Logical Issue	● Major	Interfaces/StructLib.sol: 68, 69	✓ Resolved

Description

The linked statements cast a `uint256` value to an `int256` without evaluating its bounds.

Recommendation

We advise a safe casting operation to be performed by ensuring the result is still positive as high `amount` numbers will cause an underflow to occur here, thereby causing the system to misbehave.

Alleviation

Alleviations are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

SLI-06 | Non-Standard Decimal Representation

Category	Severity	Location	Status
Language Specific	● Informational	Interfaces/StructLib.sol: 122	⊗ Declined

Description

Within the various ERC standards of Ethereum, decimals are represented by the `uint8` data type whereas here they are represented by the `uint16` data type.

Recommendation

As no form of tight-packing occurs, we advise the data type to also be set to `uint8` to ensure conformity in a potential future update of the contracts.

Alleviation

No alleviations are considered by the team.

SLI-07 | Loop Optimizations

Category	Severity	Location	Status
Gas Optimization	● Informational	Interfaces/StructLib.sol: 447~449, 480~484	✓ Resolved

Description

The linked loops can be significantly optimized by caching certain variables.

Recommendation

We advise their `length` loop limit to be cached to an in-memory variable preventing repetitive storage reads and we also advise the `ld._ledger[ledger].tokenType_stIds[tokTypeId]` lookup to be cached in memory further reducing the gas cost consumed by these loops.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

SLI-08 | Unsafe Mathematical Operations

Category	Severity	Location	Status
Mathematical Operations	● Minor	Interfaces/StructLib.sol: 448, 450, 465~467	✓ Resolved

Description

The linked statements perform unsafe mathematical operations between multiple arguments that would rely on caller sanitization, an ill-advised pattern.

Recommendation

We advise the operations to be performed safely by introducing an `int`-aware SafeMath implementation to the project.

Alleviation

Alleviations are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

SLI-09 | State Representation Inconsistency

Category	Severity	Location	Status
Inconsistency	● Minor	Interfaces/StructLib.sol: 114, 117, 145	ⓘ Acknowledged

Description

The `enum` declarations of the contract are inconsistent with regards to the default state. While the `ContractType` and `FundWithdrawType` enums have an actionable default state, the `SettlementType` enum has an `UNDEFINED` default state.

Recommendation

We advise the `SettlementType` paradigm to be applied to the other two declarations given that the default state is represented by `0` and is by default "valid" for zeroed-out data slots.

Alleviation

No alleviations are considered and the team stated that it is needed for ledger hash validation for our SC migration/upgrade.

SLS-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	StMaster/StLedger.sol: 3	✓ Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for `^0.6.12`, the version can be locked at `0.6.12`.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

SLS-02 | Return Variable Utilization

Category	Severity	Location	Status
Gas Optimization	● Informational	StMaster/StLedger.sol: 53, 60, 68, 76, 84, 94, 102, 112, 119, 127	✓ Resolved

Description

The linked function declarations contain explicitly named `return` variables that are not utilized within the function's code block.

Recommendation

We advise that the linked variables are either utilized or omitted from the declaration.

Alleviation

Alleviations are applied as of commit hash `b16d33df790b6afb40554dc4950951e19f983b02`.

TLL-01 | Inexplicable Constant

Category	Severity	Location	Status
Logical Issue	● Major	Libs/TokenLib.sol: 61	✓ Resolved

Description

The expiry timestamp evaluation set within the `addSecTokenType` function contains an old timestamp equivalent to April 1st of 2020. This permits expirations in the past in the current iteration.

Recommendation

We advise the timestamp to be marked as a value to be set as the current state misbehaves.

Alleviation

Alleviations are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

TLL-02 | Unsafe Addition

Category	Severity	Location	Status
Mathematical Operations	● Major	Libs/TokenLib.sol: 65	✓ Resolved

Description

The total margin evaluation adds up the `initMarginBips` and `varMarginBips` without evaluating their bounds thus permitting an overflow to circumvent this check.

Recommendation

We advise a safe addition to be performed here instead preventing this scenario from unfolding.

Alleviation

Alleviations are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8` by using Solidity v0.8.

TLL-03 | Potentially Negative Quantities

Category	Severity	Location	Status
Logical Issue	● Minor	Libs/TokenLib.sol: 350	✓ Resolved

Description

Negative quantities should be skipped by the `while` loop as the adidtion in L380 will lead to the remaining to burn increasing.

Recommendation

We advise an `if` conditional to be introduced that simply "continues" the `while` loop in such a case.

Alleviation

Alleviations are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

TRA-01 | Equal ID Transfers

Category	Severity	Location	Status
Logical Issue	● Minor	Libs/TransferLib.sol: 28	✓ Resolved

Description

The transfers of equal IDs are not prohibited in the `transferOrTrade` function.

Recommendation

We advise them to be properly prohibited as they can lead to misbehaviours arising.

Alleviation

Alleviations are applied as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

TRA-02 | Potentially Negative Quantity

Category	Severity	Location	Status
Mathematical Operations	● Minor	Libs/TransferLib.sol: 692	✓ Resolved

Description

The `v.stQty` value may be negative within the `transferSplitSecTokens` function.

Recommendation

We advise a check to be imposed whereby negative quantities are skipped. If negative quantities are not expected, the data types of the system should change to the unsigned representations.

Alleviation

Alleviations are applied by using Solidity 0.8 as of commit hash `55e29990079257acd6fa3dd8232a4ef6409e20b8`.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK’S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER’S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK’S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.