Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Libra Finance
Website: http://librafinance.xyz
Platform: Astar Network
Language: Solidity
Date: April 25th, 2022

Table of contents

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the Libra Finance team to perform the Security audit of the Libra Finance Protocol smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on April 25th, 2022.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

The Libra Finance Contracts have functions like stake, withdraw, earned, setTaxTiersRate, mint, burn, setPeriod, setEpoch, twap, safeLibraTransfer, etc. The Libra Finance contract inherits the IERC20, SafeERC20, Math, ReentrancyGuard, Ownable, SafeMath, ERC20Burnable, ERC20 standard smart contracts from the OpenZeppelin library. These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

Name	**Code Review and Security Analysis Report for Libra Finance Protocol Smart Contracts**
Platform	**Astar Network / Solidity**
File 1	RebateTreasury.sol
File 1 MD5 Hash	E47E3B0F229E0AEF5C189B155AE16BE3
File 2	Treasury.sol
File 2 MD5 Hash	9FC141FB230D3E477E8C59A4A8EACF68
File 3	Boardroom.sol
File 3 MD5 Hash	D5DB6E807C7AD5C7AEE42F7C6E57E6F5
File 4	LBond.sol

File 4 MD5 Hash	38AA3DEE2FCE8AAB9AC7101E2255E384
File 5	Libra.sol
File 5 MD5 Hash	DDC50DCB39D71472C423E3D20FA45428
File 6	LShare.sol
File 6 MD5 Hash	187A780003A79C87B459EA162E28CCA1
File 7	Oracle.sol
File 7 MD5 Hash	CEDE169255BB09E06B929403162CE52B
File 8	LibraGenesisRewardPool.sol
File 8 MD5 Hash	B3618BE29B29169AA85CAA2BA1EF7201
File 9	LibraRewardPool.sol
File 9 MD5 Hash	62BBC991E6B8DE5C1E29C8DE17C9F461
File 10	LShareRewardPool.sol
File 10 MD5 Hash	8858D9A5DA0E3E7B8C2D3E41CAF0D6B5
Audit Date	April 25th,2022

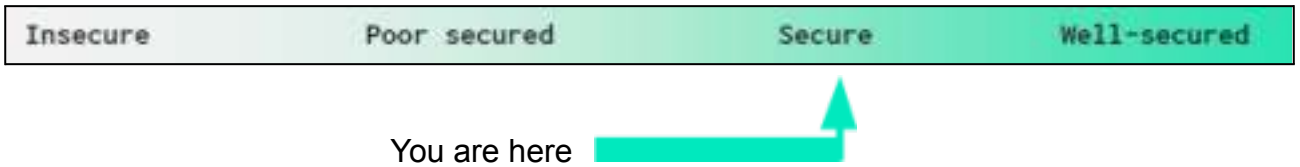
Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 Boardroom.sol • Withdraw Lockup Epochs: 6 epochs • Reward Lockup Epochs: 3 epochs • Decimals: 18	**YES, This is valid.**
File 2 LBond.sol • Name: Libra Finance BOND • Symbol: LBOND • Decimals: 18 • Max Minting Limit: No limit. An operator can mint unlimited tokens	**YES, This is valid.**
File 3 Libra.sol • Name: LIBRA • Symbol: LIBRA Token • Decimals: 18 • Initial Genesis Pool Distribution: 11,000 Tokens • Initial Libra Pool Distribution: 140,000 Tokens • Initial Airdrop Wallet Distribution: 10,000 Tokens • Initial Supply: 1 LIBRA • Burn Threshold: 1.1 • Max minting limit: No limit. An operator can mint unlimited tokens.	**YES, This is valid.**
File 4 LShare.sol • Name: Libra Finance SHARE • Symbol: LSHARE • Decimals: 18 • Farming Pool Reward Allocation: 45500 Tokens; • Treasury Fund Pool Allocation: 3500 Tokens • Community Fund Pool Allocation: 14000 Tokens	**YES, This is valid.**

• Dev Fund Pool Allocation: 7000 Tokens • Vesting Duration: 365 Days	
File 5 Oracle.sol • Oracle has functions like: update, consult, twap.	**YES, This is valid.**
File 6 RebateTreasury.sol • Bond Vesting: 3 days • Bound Factor: 800,000 • Bound threshold: 200,000 • Buy Back Amount: 100,000 • Denominator: 1 million • Secondary Factor: 150,000 • Secondary Threshold: 700,000	**YES, This is valid.**
File 7 Treasury.sol • Period: 6 hours • Treasury has functions like: nextEpochPoint, setOperator, setLibraOracle,etc.	**YES, This is valid.**
File 8 LibraGenesisRewardPool.sol • Libra Per Second: 0.11574 LIBRA • Running Time: 24 hours • Total Rewards: 10000 LIBRA	**YES, This is valid.**
File 9 LibraRewardPool.sol • Pool Start Time: 4 days • Pool End Time: 5 days after start time • LibraRewardPool has functions like: checkPoolDuplicate, getGeneratedReward, etc.	**YES, This is valid.**
File 10 LShareRewardPool.sol • LShare Per Second: 0.00187687 Lshare • Running Time: 370 days • Total Rewards: 60000 Lshare	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. Also, these contracts do contain owner control, which does not make them fully decentralized.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 2 low and some very low level issues.
These are fixed / acknowledged in the revised code.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Acknowledged
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 10 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the Libra Finance Protocol are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Libra Finance Protocol.

The Libra Finance team has not provided unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Some code parts are **not** well commented on smart contracts. We suggest using Ethereum's NatSpec style for the commenting.

Documentation

We were given a Libra Finance Protocol smart contract code in the form of a github link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **not well** commented. So it is not easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contracts infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Boardroom.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	checkSameOriginReentranted	internal	Passed	No Issue
3	checkSameSenderReentranted	internal	Passed	No Issue
4	onlyOneBlock	modifier	Passed	No Issue
5	totalSupply	read	Passed	No Issue
6	balanceOf	read	Passed	No Issue
7	stake	write	Passed	No Issue
8	withdraw	write	Passed	No Issue
9	onlyOperator	modifier	Passed	No Issue
10	memberExists	modifier	Passed	No Issue
11	updateReward	modifier	Passed	No Issue
12	notInitialized	modifier	Passed	No Issue
13	initialize	write	Passed	No Issue
14	setOperator	external	access only Operator	No Issue
15	setLockUp	external	access only Operator	No Issue
16	latestSnapshotIndex	read	Passed	No Issue
17	getLatestSnapshot	iinternal	Passed	No Issue
18	getLastSnapshotIndexOf	read	Passed	No Issue
19	getLastSnapshotOf	internal	Passed	No Issue
20	canWithdraw	external	Passed	No Issue
21	canClaimReward	external	Passed	No Issue
22	epoch	external	Passed	No Issue
23	nextEpochPoint	external	Passed	No Issue
24	getLibraPrice	external	Passed	No Issue
25	rewardPerShare	read	Passed	No Issue
26	earned	read	Passed	No Issue
27	stake	write	access only One Block	No Issue
28	withdraw	write	access only One Block	No Issue
29	exit	external	Passed	No Issue
30	claimReward	write	Passed	No Issue
31	allocateSeigniorage	external	access only One Block	No Issue
32	governanceRecoverUnsupported	external	access only Operator	No Issue

LBond.sol

Functions

Sl.	Functions	Type	Observation	Concluburnsion
1	constructor	write	Passed	No Issue
2	burn	write	Passed	No Issue
3	burnFrom	write	Passed	No Issue
4	operator	read	Passed	No Issue
5	onlyOperator	modifier	Passed	No Issue
6	isOperator	read	Passed	No Issue
7	transferOperator	write	access only Owner	No Issue
8	_transferOperator	internal	Passed	No Issue
9	mint	write	access only Operator	No Issue
10	burn	write	Passed	No Issue
11	burnFrom	write	access only Operator	No Issue

Libra.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	burn	write	Passed	No Issue
3	burnFrom	write	Passed	No Issue
4	operator	read	Passed	No Issue
5	onlyOperator	modifier	Passed	No Issue
6	isOperator	read	Passed	No Issue
7	transferOperator	write	access only Owner	No Issue
8	_transferOperator	internal	Passed	No Issue
9	onlyTaxOffice	modifier	Passed	No Issue
10	onlyOperatorOrTaxOffice	modifier	Passed	No Issue
11	getTaxTiersTwapsCount	read	Passed	No Issue
12	getTaxTiersRatesCount	read	Passed	No Issue
13	isAddressExcluded	read	Passed	No Issue
14	setTaxTiersTwap	write	access only TaxOffice	No Issue
15	setTaxTiersRate	write	access only TaxOffice	No Issue
16	setBurnThreshold	write	access only TaxOffice	No Issue
17	_getLibraPrice	internal	Passed	No Issue
18	_updateTaxRate	internal	Passed	No Issue
19	enableAutoCalculateTax	write	access only TaxOffice	No Issue

20	disableAutoCalculateTax	write	access only TaxOffice	No Issue
21	setLibraOracle	write	access only Operator Or TaxOffice	No Issue
22	setTaxOffice	write	access only Operator Or TaxOffice	No Issue
23	setTaxCollectorAddress	write	access only TaxOffice	No Issue
24	setTaxRate	write	access only TaxOffice	No Issue
25	excludeAddress	write	access only Operator Or TaxOffice	No Issue
26	includeAddress	write	access only Operator Or TaxOffice	No Issue
27	mint	write	access only Operator	No Issue
28	burn	write	Passed	No Issue
29	burnFrom	write	access only Operator	No Issue
30	transferFrom	write	Passed	No Issue
31	_transferWithTax	internal	Passed	No Issue
32	distributeReward	external	access only Operator	No Issue
33	governanceRecoverUnsupported	external	access only Operator	No Issue

LShare.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	burn	write	Passed	No Issue
3	burnFrom	write	Passed	No Issue
4	operator	read	Passed	No Issue
5	onlyOperator	modifier	Passed	No Issue
6	isOperator	read	Passed	No Issue
7	transferOperator	write	access only Owner	No Issue
8	_transferOperator	internal	Passed	No Issue
9	setCommunityFund	external	access only Operator	No Issue
10	setTreasuryFund	external	access only Operator	No Issue
11	setDevFund	external	Passed	No Issue

12	unclaimedCommunityFund	read	Passed	No Issue
13	unclaimedDevFund	read	Passed	No Issue
14	unclaimedTreasuryFund	read	Passed	No Issue
15	claimRewards	external	Passed	No Issue
16	distributeReward	external	access only Operator	No Issue
17	burn	write	Passed	No Issue
18	governanceRecoverUnsupported	write	access only Operator	No Issue

Oracle.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	checkStartTime	modifier	Passed	No Issue
3	checkEpoch	modifier	Passed	No Issue
4	getCurrentEpoch	read	Passed	No Issue
5	getPeriod	read	Passed	No Issue
6	getStartTime	read	Passed	No Issue
7	getLastEpochTime	read	Passed	No Issue
8	nextEpochPoint	read	Passed	No Issue
9	setPeriod	external	access only Operator	No Issue
10	setEpoch	external	access only Operator	No Issue
11	update	external	Passed	No Issue
12	consult	external	Passed	No Issue
13	twap	external	Passed	No Issue

RebateTreasury.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	onlyAsset	modifier	Passed	No Issue
8	bond	external	access only Asset	No Issue
9	claimRewards	external	Passed	No Issue
10	setLibra	external	access only Owner	No Issue
11	setLibraOracle	external	access only Owner	No Issue

12	setTreasury	external	access only Owner	No Issue
13	setAsset	external	access only Owner	No Issue
14	setBondParameters	external	access only Owner	No Issue
15	redeemAssetsForBuyback	external	access only Owner	No Issue
16	_claimVested	internal	Passed	No Issue
17	getLibraReturn	read	access only Asset	No Issue
18	getBondPremium	read	Passed	No Issue
19	getLibraPrice	read	Passed	No Issue
20	getTokenPrice	read	access only Asset	No Issue
21	claimableLibra	external		No Issue

Treasury.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	checkSameOriginReentranted	internal	Passed	No Issue
3	checkSameSenderReentranted	internal	Passed	No Issue
4	onlyOneBlock	modifier	Passed	No Issue
5	onlyOperator	modifier	Passed	No Issue
6	checkCondition	modifier	Passed	No Issue
7	checkEpoch	modifier	Passed	No Issue
8	checkOperator	modifier	Passed	No Issue
9	notInitialized	modifier	Passed	No Issue
10	isInitialized	read	Passed	No Issue
11	nextEpochPoint	read	Passed	No Issue
12	getLibraPrice	read	Passed	No Issue
13	getLibraUpdatedPrice	read	Passed	No Issue
14	getReserve	read	Passed	No Issue
15	getBurnableLibraLeft	read	Passed	No Issue
16	getRedeemableBonds	read	Passed	No Issue
17	getBondDiscountRate	read	Passed	No Issue
18	getBondPremiumRate	read	Passed	No Issue
19	initialize	write	Passed	No Issue
20	setOperator	external	access only Operator	No Issue
21	setBoardroom	external	access only Operator	No Issue
22	setBondTreasury	external	access only Operator	No Issue
23	setLibraOracle	external	access only Operator	No Issue
24	setLibraPriceCeiling	external	access only Operator	No Issue

25	setMaxSupplyExpansion Percents	external	access only Operator	No Issue
26	setSupplyTiersEntry	external	access only Operator	No Issue
27	setMaxExpansionTiersEntry	external	access only Operator	No Issue
28	setBondDepletionFloorPercent	external	access only Operator	No Issue
29	setMaxSupplyContraction Percent	external	access only Operator	No Issue
30	setMaxDebtRatioPercent	external	access only Operator	No Issue
31	setBootstrap	external	access only Operator	No Issue
32	setExtraFunds	external	access only Operator	No Issue
33	setMaxDiscountRate	external	access only Operator	No Issue
34	setMaxPremiumRate	external	access only Operator	No Issue
35	setDiscountPercent	external	access only Operator	No Issue
36	setPremiumThreshold	external	access only Operator	No Issue
37	setPremiumPercent	external	access only Operator	No Issue
38	setMintingFactorForPayingDebt	external	access only Operator	No Issue
39	setBondSupplyExpansion Percent	external	access only Operator	No Issue
40	_updateLibraPrice	internal	Passed	No Issue
41	getLibraCirculatingSupply	read	Passed	No Issue
42	buyBonds	external	access only One Block	No Issue
43	redeemBonds	external	access only One Block	No Issue
44	_sendToBoardroom	internal	Passed	No Issue
45	_sendToBondTreasury	internal	Passed	No Issue
46	_calculateMaxSupplyExpansionPercent	internal	Passed	No Issue
47	allocateSeigniorage	external	access only One Block	No Issue
48	governanceRecoverUnsupported	external	access only Operator	No Issue
49	boardroomSetOperator	external	access only Operator	No Issue
50	boardroomSetLockUp	external	access only Operator	No Issue

51	boardroomAllocateSeigniorage	external	access only Operator	No Issue
52	boardroomGovernanceRecoverUnsupported	external	access only Operator	No Issue

LibraGenesisRewardPool.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyOperator	modifier	Passed	No Issue
3	checkPoolDuplicate	internal	Passed	No Issue
4	add	write	access only Operator	No Issue
5	set	write	access only Operator	No Issue
6	getGeneratedReward	read	Passed	No Issue
7	pendingLIBRA	external	Passed	No Issue
8	massUpdatePools	write	Passed	No Issue
9	updatePool	write	Passed	No Issue
10	deposit	write	Passed	No Issue
11	withdraw	write	Passed	No Issue
12	emergencyWithdraw	write	Passed	No Issue
13	safeLibraTransfer	internal	Passed	No Issue
14	setOperator	external	access only Operator	No Issue
15	governanceRecoverUnsupported	external	access only Operator	No Issue

LibraRewardPool.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyOperator	modifier	Passed	No Issue
3	checkPoolDuplicate	internal	Passed	No Issue
4	add	write	access only Operator	No Issue
5	set	write	access only Operator	No Issue
6	getGeneratedReward	read	Passed	No Issue
7	pendingLIBRA	external	Passed	No Issue
8	massUpdatePools	write	Passed	No Issue
9	updatePool	write	Passed	No Issue
10	deposit	write	Passed	No Issue
11	withdraw	write	Passed	No Issue

12	emergencyWithdraw	write	Passed	No Issue
13	safeLibraTransfer	internal	Passed	No Issue
14	setOperator	external	access only Operator	No Issue
15	governanceRecoverUnsupported	external	access only Operator	No Issue

LShareRewardPool.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyOperator	modifier	Passed	No Issue
3	checkPoolDuplicate	internal	Passed	No Issue
4	add	write	access only Operator	No Issue
5	set	write	access only Operator	No Issue
6	getGeneratedReward	read	Passed	No Issue
7	pendingShare	external	Passed	No Issue
8	massUpdatePools	write	Passed	No Issue
9	updatePool	write	Passed	No Issue
10	deposit	write	Passed	No Issue
11	withdraw	write	Passed	No Issue
12	emergencyWithdraw	write	Passed	No Issue
13	safeLShareTransfer	internal	Passed	No Issue
14	setOperator	external	access only Operator	No Issue
15	governanceRecoverUnsupported	external	access only Operator	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Token max minting limits can be set

```
function mint(address recipient_, uint256 amount_) public onlyOperator returns (bool) {
    uint256 balanceBefore = balanceOf(recipient_);
    _mint(recipient_, amount_);
    uint256 balanceAfter = balanceOf(recipient_);

    return balanceAfter > balanceBefore;
```

It is advisable to set an upper minting limit in the LIBRA and LBOND tokens contracts. This is considered good for healthy tokenomics.

Resolution: We advise setting a max minting limit. On the other hand, this also can be acknowledged by the owner to mint it responsibly.

Status: Acknowledged

(2) Missing Event Log: There are some events missing in the LibraRewardPool.sol, LibraGenesisRewardPool and LShareRewardPool.sol smart contracts. It is recommended to fire appropriate events in case of important state change.

Some functions need an event log.

- add
- set
- updatePool
- governanceRecoverUnsupported

Resolution: We suggest adding an event log for above listed functions.

Status: Acknowledged

Very Low / Informational / Best practices:

(1) Function input parameters lack of check:

Input variable validation is not performed in the function of LibraGenesisRewardPool.sol, LibraRewardPool.sol, LShareRewardPool smart contracts.

Variable validation is not performed in below functions :

- governanceRecoverUnsupported

Resolution: We advise using validation like address type variables should not be

address(0). This helps protect the information to be added mistakenly.

Status: Acknowledged

(2) Variables should be made immutable:

Variables that are defined within the constructor but further remain unchanged should be marked as immutable to save gas and to ease the reviewing process of third-parties.

- **LibraGenesisRewardPool.sol**
  - libra, poolStartTime, poolEndTime
- **LibraRewardPool.sol**
  - libra, poolStartTime, epochEndTimes, epochLibraPerSecond
- **LShareRewardPool.sol**
  - lshare, poolStartTime, poolEndTime
- **Treasury.sol**
  - libra, lbond, lshare, libraPriceOne
- **LShare.sol**
  - startTime, endTime, communityFundRewardRate, devFundRewardRate, treasuryFundRewardRate

Resolution: We suggest setting these variables as immutable.

Status: Acknowledged

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- setOperator: Boardroom operator can set operator address.
- setLockUp: Boardroom operator can withdraw Lockup Epochs, reward Lockup Epochs values set.
- allocateSeigniorage: Boardroom operators can allocate seigniorage.
- governanceRecoverUnsupported: Boardroom operators can governance recover unsupported tokens.
- mint: LBond Operator mints basis bonds to a recipient.
- burnFrom: LBond Operator can burn from amount.
- governanceRecoverUnsupported: LBond operators can governance recover unsupported tokens.
- distributeReward: Libra operators can distribute to the reward pool.
- burnFrom: Libra operators can burn from amount.
- mint: Libra Operator mints LIBRA to a recipient.
- includeAddress: Libra operators can include addresses.
- setTaxOffice: Libra operators can set tax office addresses.
- setLibraOracle: Libra operators can set libra oracle addresses.
- governanceRecoverUnsupported: LShare operators can governance recover unsupported tokens.
- distributeReward: LShare operators can distribute to the reward pool.
- setTreasuryFund: LShare operators can set treasury fund addresses.
- setCommunityFund: LShare operators can set community fund addresses.
- setLibra: RebateTreasury owner can set libra token address.
- setLibraOracle: RebateTreasury owner can set libra oracle address.
- setTreasury: RebateTreasury owner can set Libra treasury address.
- setAsset: RebateTreasury owner can set bonding parameters of token.
- setBondParameters: RebateTreasury owners can set bond pricing parameters.
- redeemAssetsForBuyback: RebateTreasury owners can redeem assets for buyback under peg.

- add: LibraGenesisRewardPool owner can add a new token to the pool.
- set: LibraGenesisRewardPool owner can update the given pool's LIBRA allocation point.
- setOperator: LibraGenesisRewardPool owner can set operator address.
- governanceRecoverUnsupported: LibraGenesisRewardPool operators can governance recover unsupported tokens.
- add: LibraRewardPool owner can add a new token to the pool.
- set: LibraRewardPool owner can update the given pool's LIBRA allocation point.
- setOperator:LibraRewardPool owner can set operator address.
- governanceRecoverUnsupported: LibraRewardPool operators can governance recover unsupported tokens.
- governanceRecoverUnsupported: LShareRewardPool operators can governance recover unsupported tokens.
- setOperator: LShareRewardPool can set operator address.
- set: LShareRewardPool owner can update the given pool's lSHARE allocation point.
- add: LShareRewardPool owner can add a new lp to the pool.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of files. And we have used all possible tests based on given objects as files. We had observed some issues in the smart contracts, but they were resolved / acknowledged in the revised smart contract code. **So, the smart contracts are ready for the mainnet deployment**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on standard audit procedure scope, is **"Secured"**.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

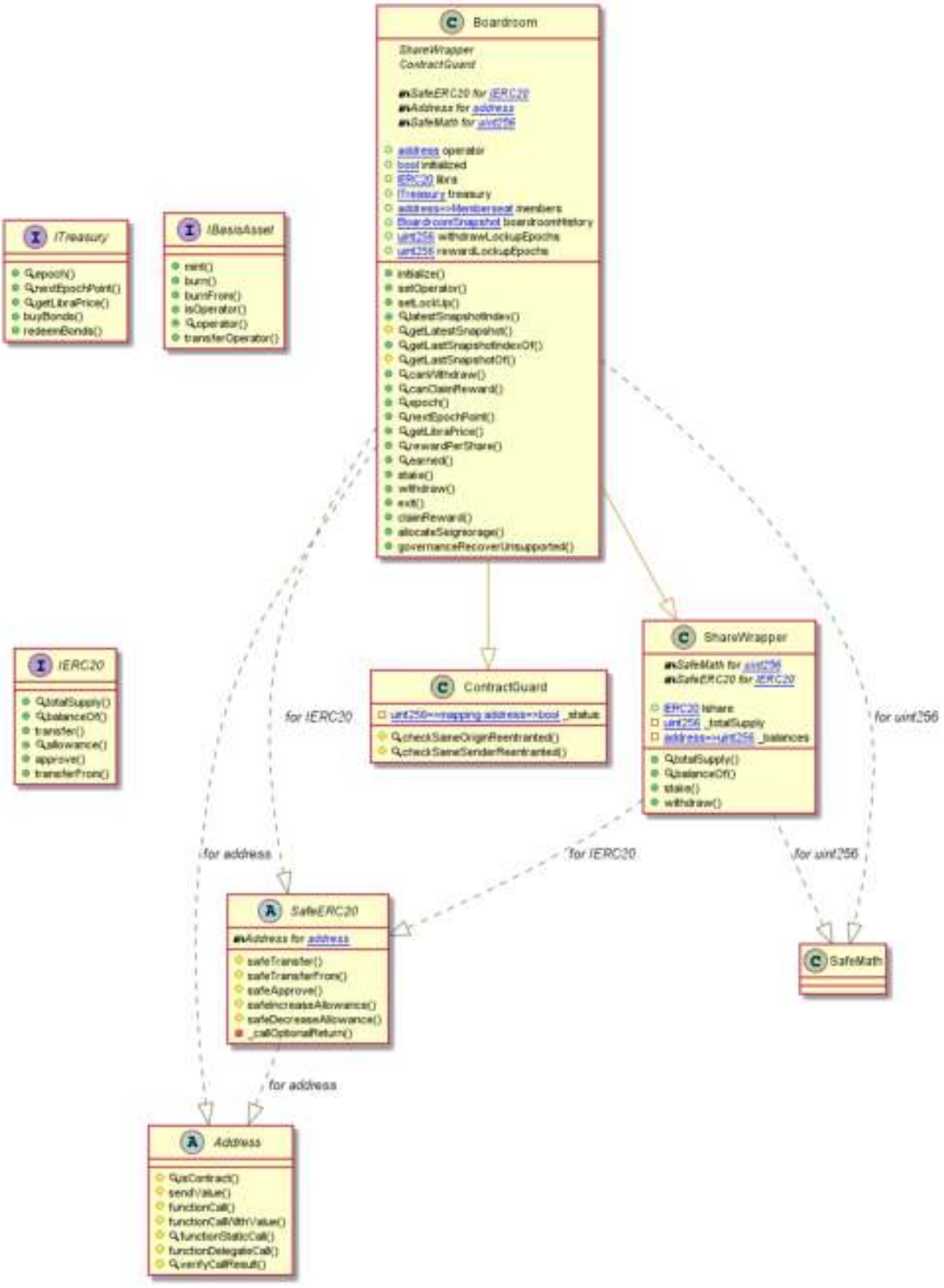
Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

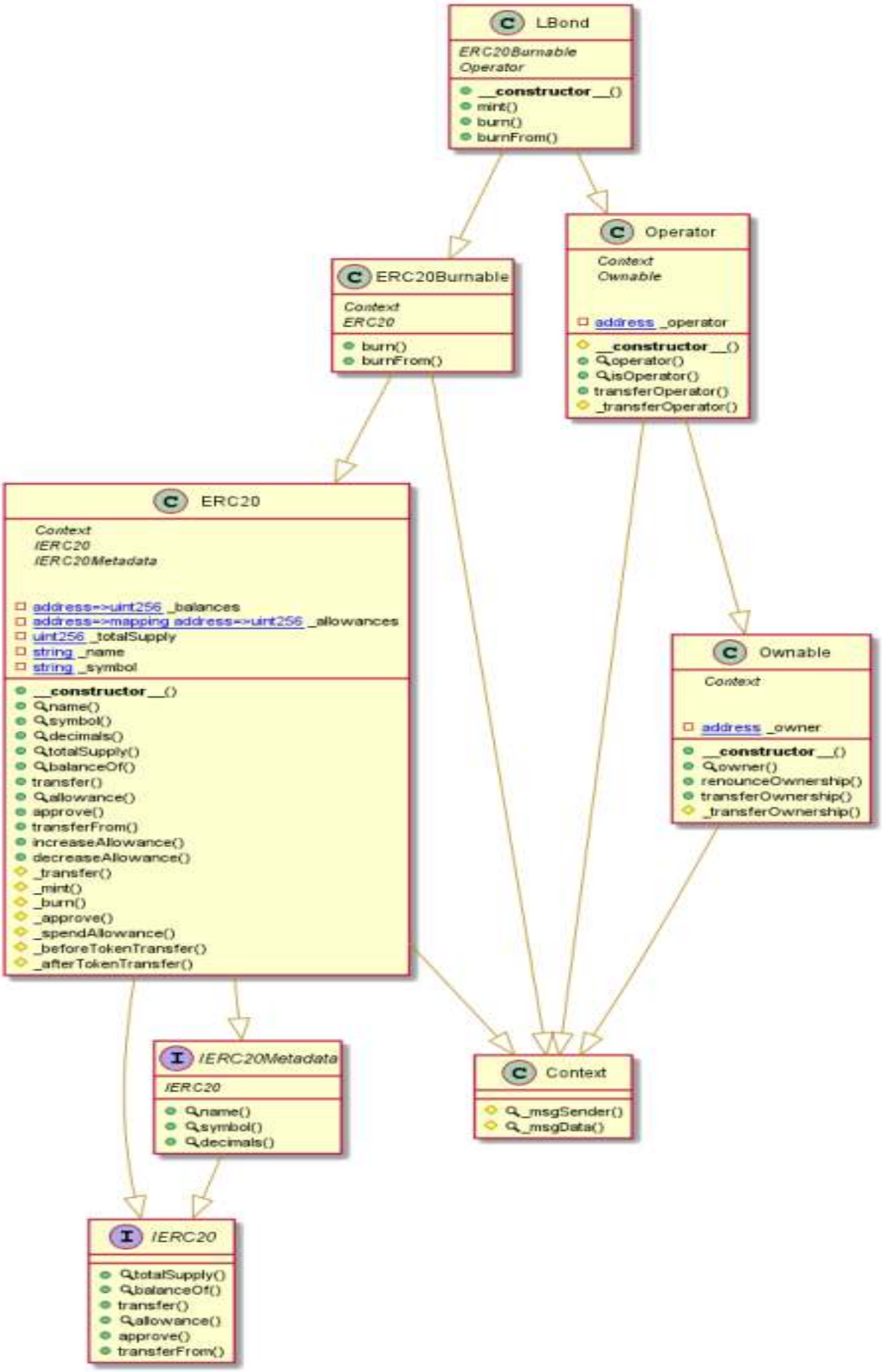
Appendix

Code Flow Diagram - Libra Finance Protocol

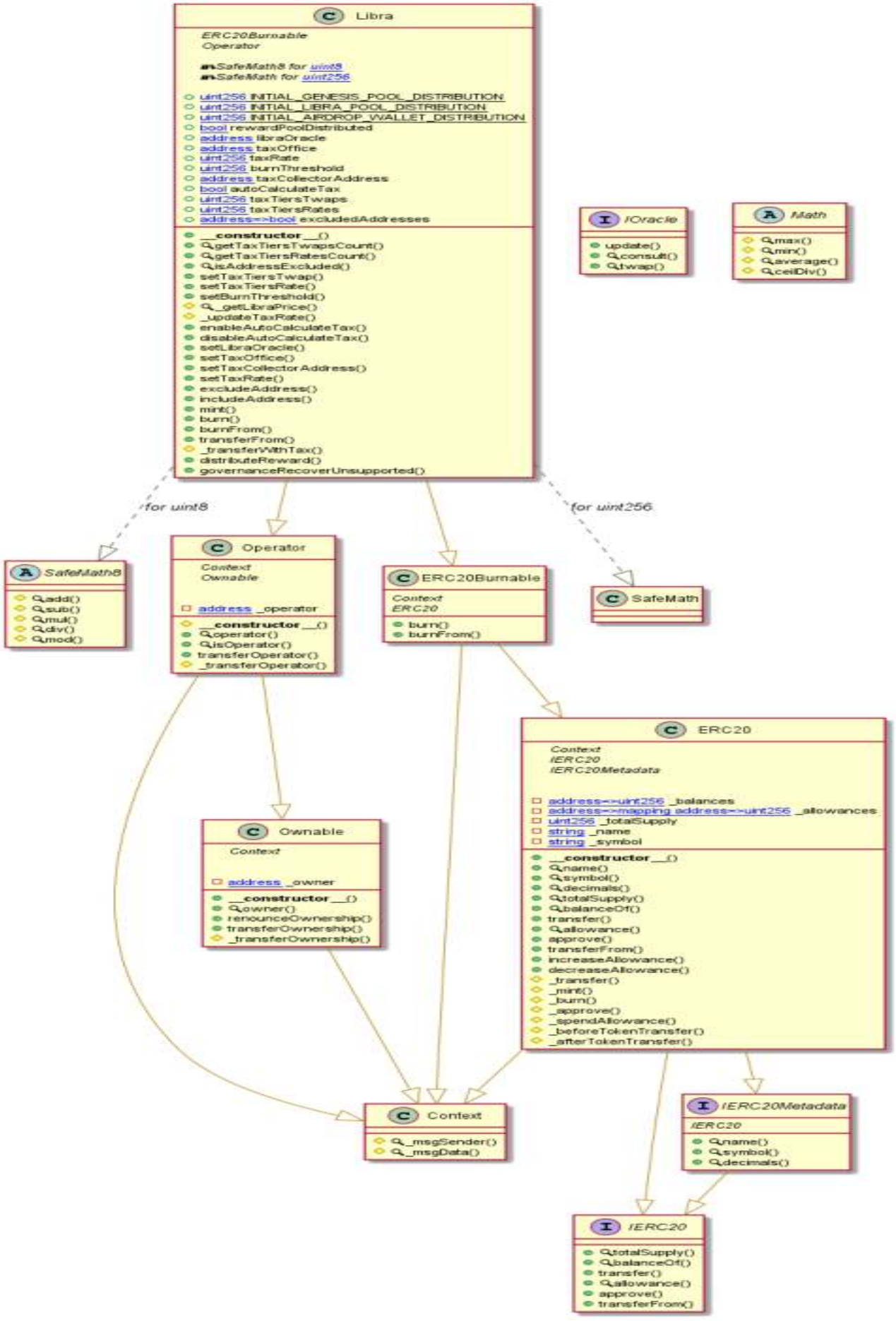
Boardroom Diagram



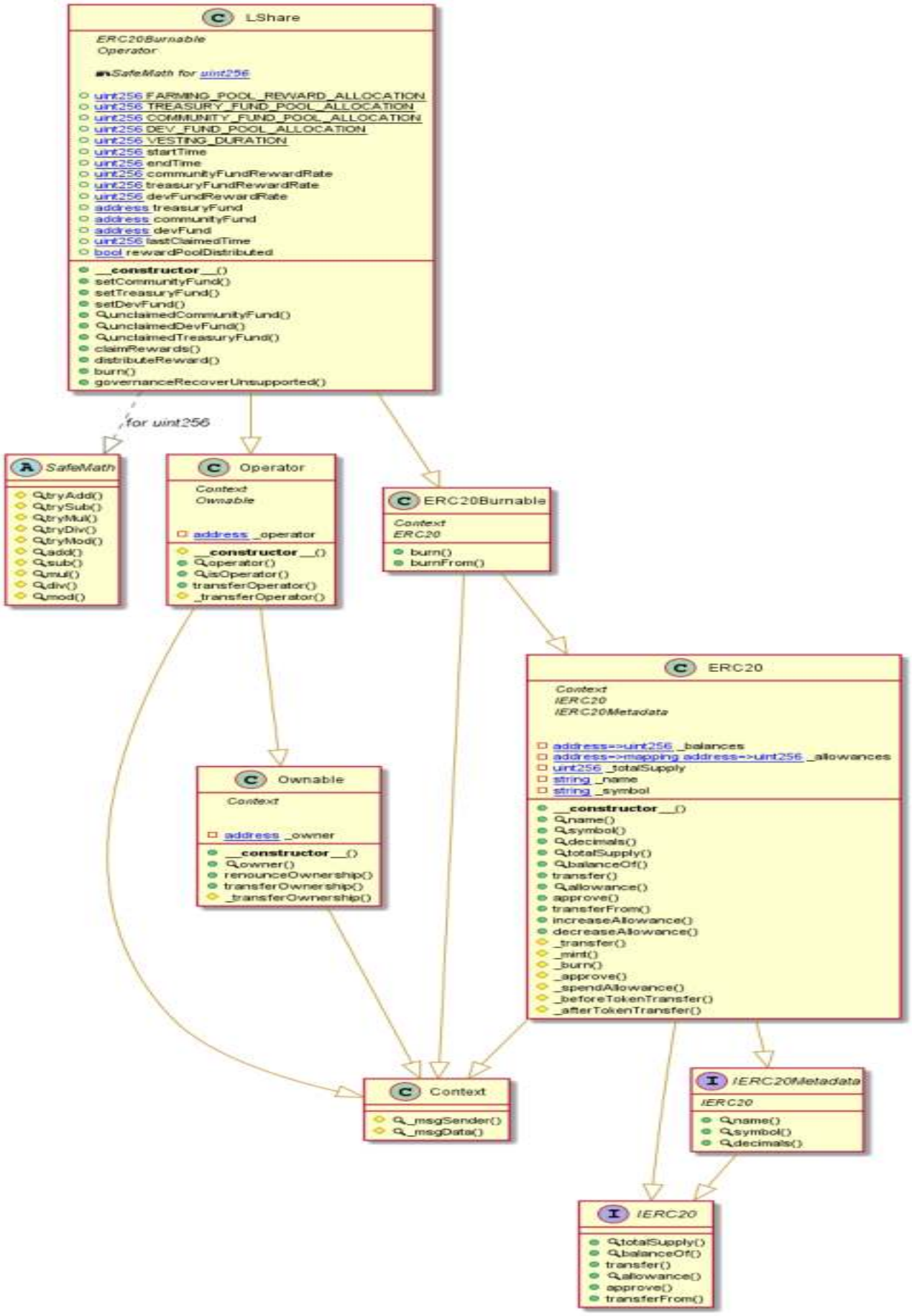
LBond Diagram



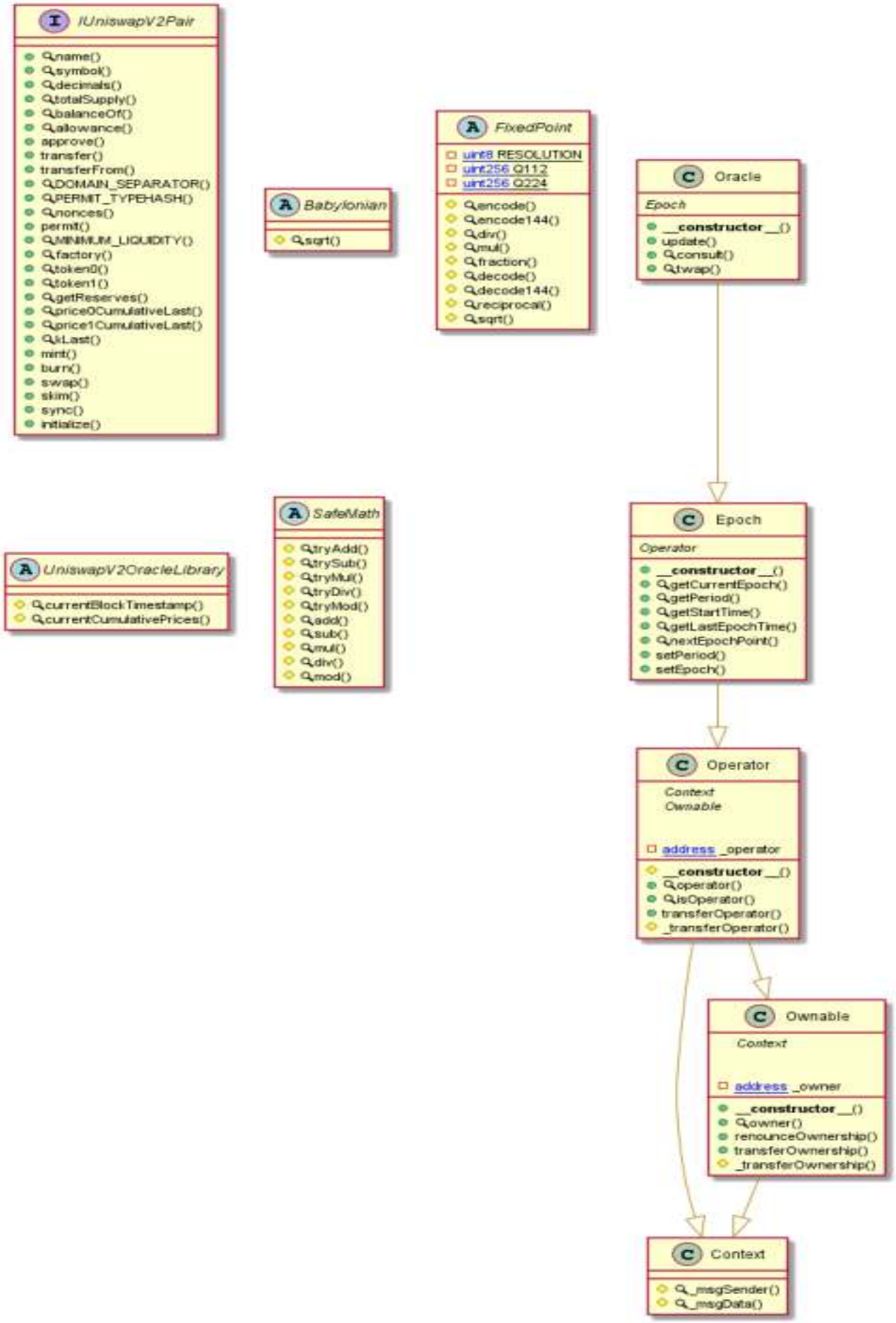
Libra Diagram



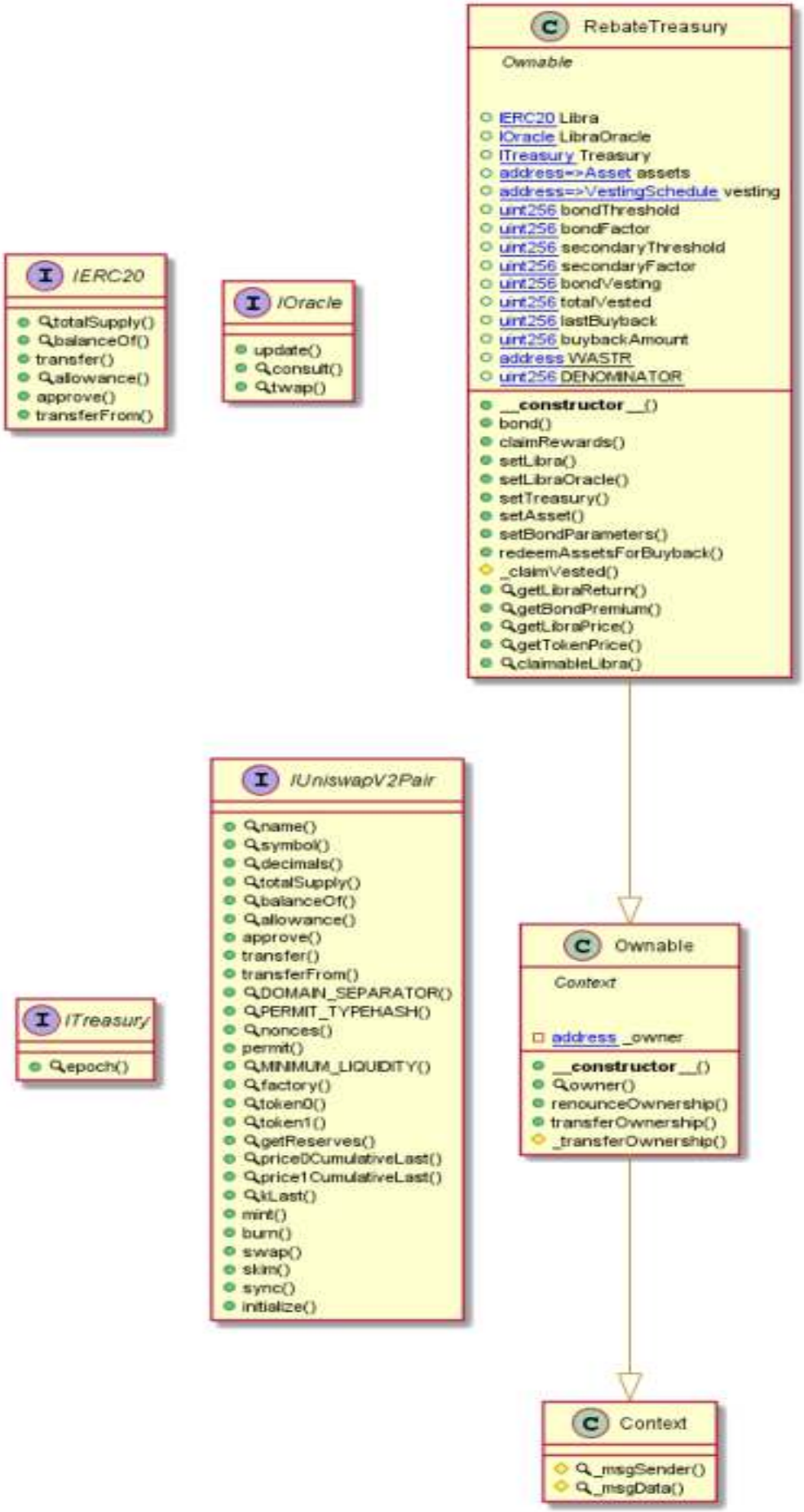
LShare Diagram



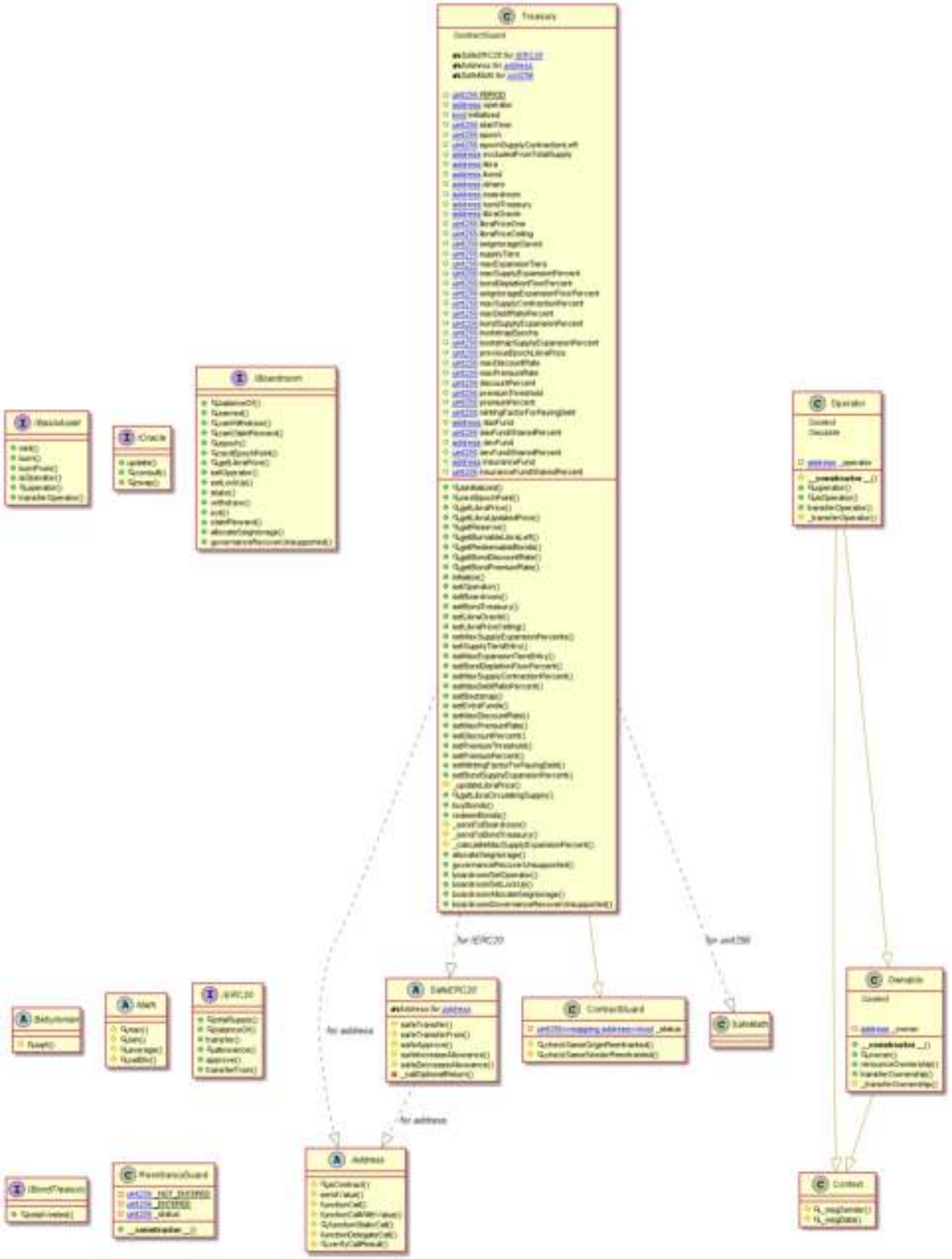
Oracle Diagram



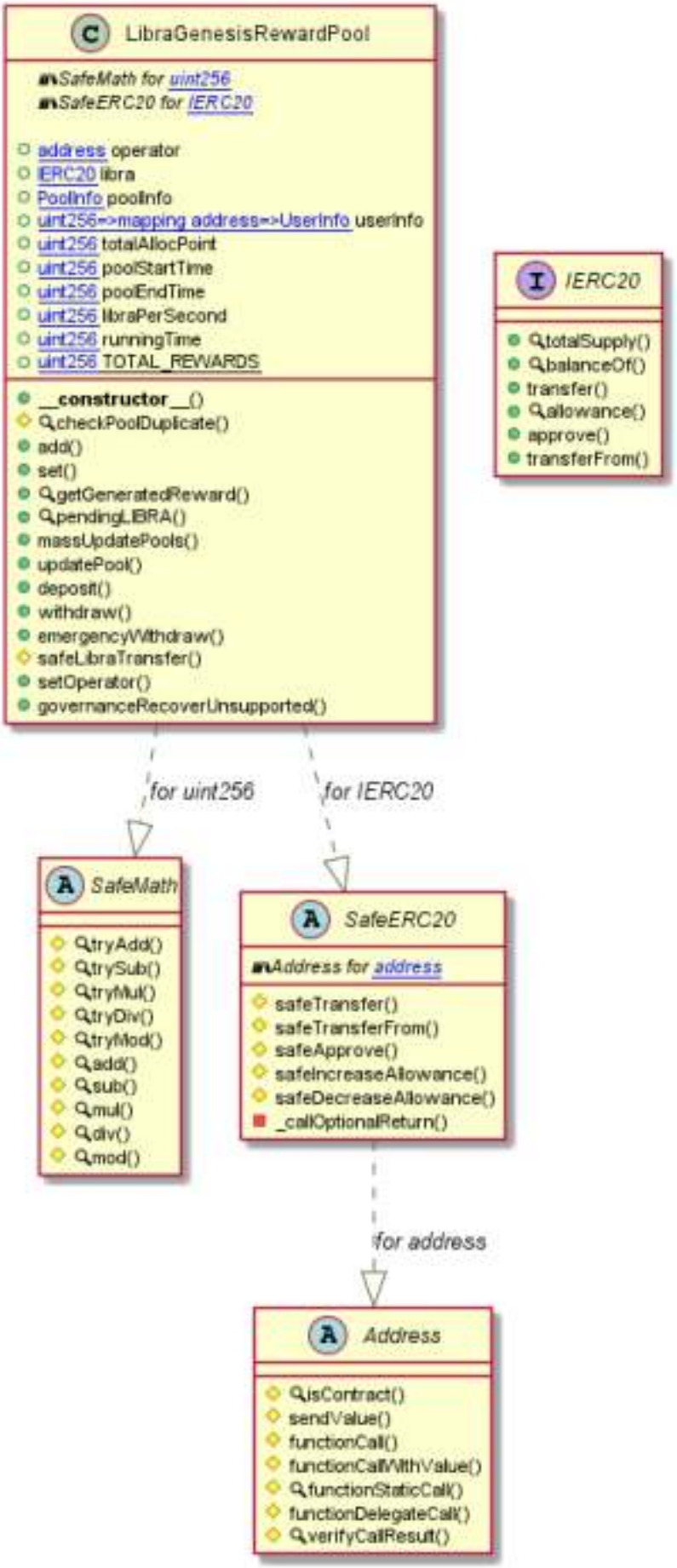
RebateTreasury Diagram



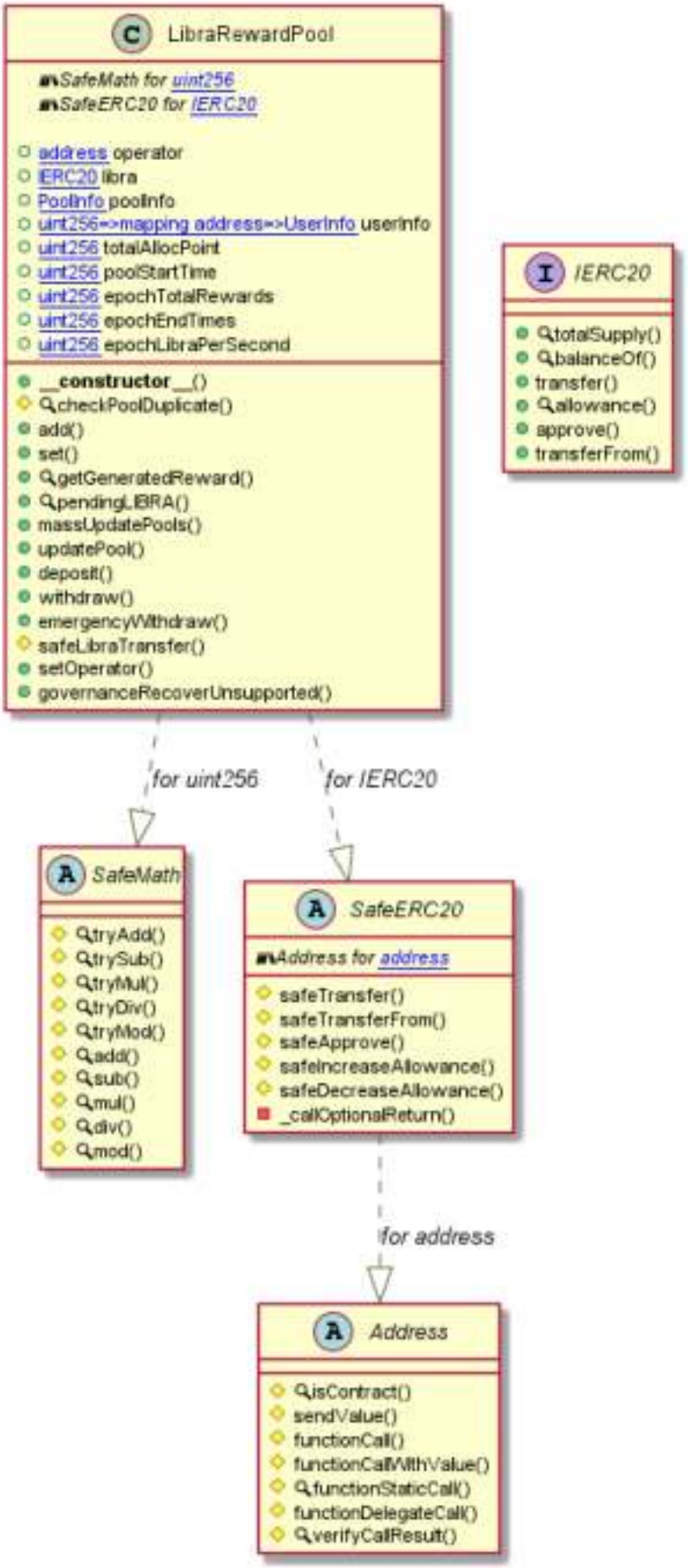
Treasury Diagram



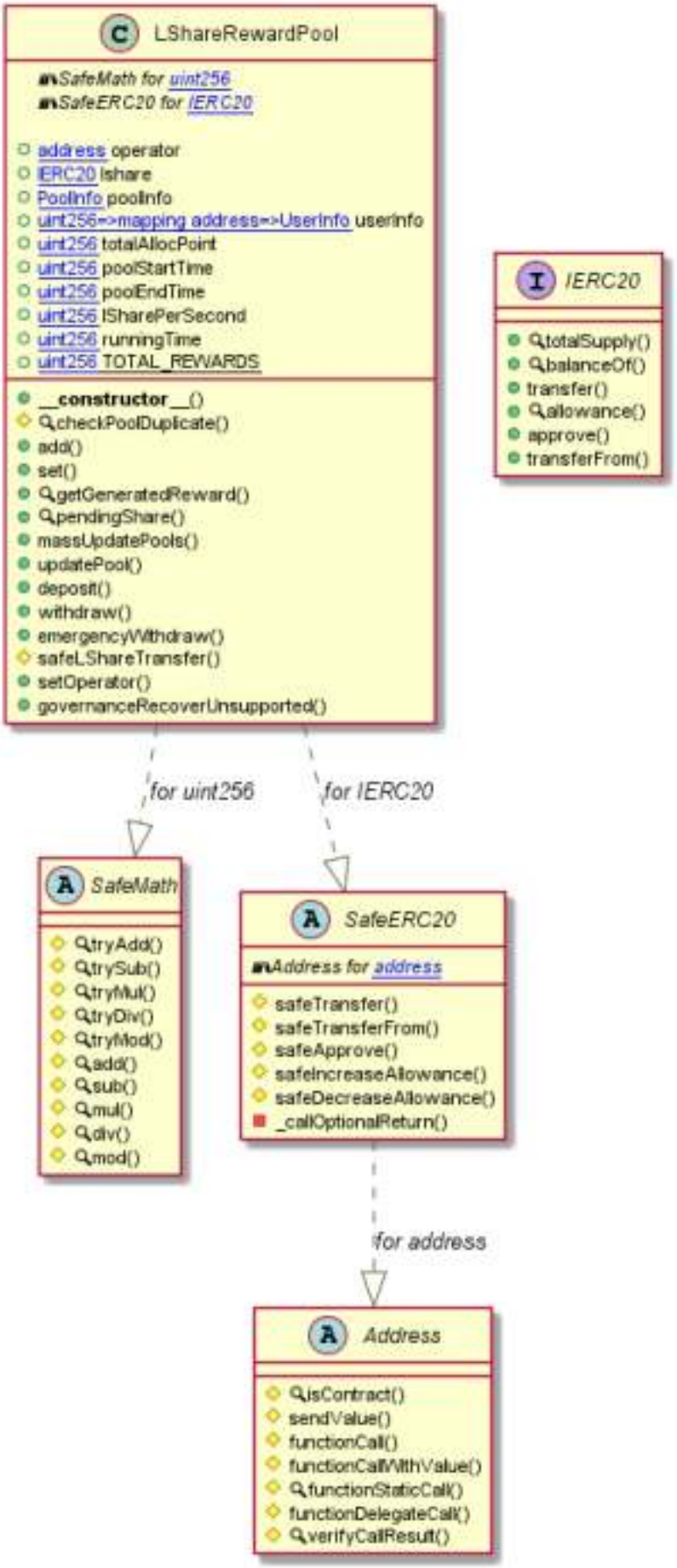
Libra Genesis Reward Pool Diagram



Libra Reward Pool Diagram



LShare Reward Pool Diagram



Slither Results Log

Slither log >> Boardroom.sol

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (Boardroom.sol#150-163):
        - (success) = recipient.call{value: amount}() (Boardroom.sol#161)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Boardroom.sol#220-237):
        - (success,returndata) = target.call{value: value}(data) (Boardroom.sol#235)
Low level call in Address.functionStaticCall(address,bytes,string) (Boardroom.sol#255-264):
        - (success,returndata) = target.staticcall(data) (Boardroom.sol#262)
Low level call in Address.functionDelegateCall(address,bytes,string) (Boardroom.sol#282-291):
        - (success,returndata) = target.delegatecall(data) (Boardroom.sol#289)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter Boardroom.initialize(IERC20,IERC20,ITreasury)._libra (Boardroom.sol#744) is not in mixedCase
Parameter Boardroom.initialize(IERC20,IERC20,ITreasury)._lshare (Boardroom.sol#745) is not in mixedCase
Parameter Boardroom.initialize(IERC20,IERC20,ITreasury)._treasury (Boardroom.sol#746) is not in mixedCase
Parameter Boardroom.setOperator(address)._operator (Boardroom.sol#763) is not in mixedCase
Parameter Boardroom.setLockUp(uint256,uint256)._withdrawLockupEpochs (Boardroom.sol#767) is not in mixedCase
Parameter Boardroom.setLockUp(uint256,uint256)._rewardLockupEpochs (Boardroom.sol#767) is not in mixedCase
Parameter Boardroom.governanceRecoverUnsupported(IERC20,uint256,address)._token (Boardroom.sol#878) is not in mixedCase
Parameter Boardroom.governanceRecoverUnsupported(IERC20,uint256,address)._amount (Boardroom.sol#879) is not in mixedCase
Parameter Boardroom.governanceRecoverUnsupported(IERC20,uint256,address)._to (Boardroom.sol#880) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
initialize(IERC20,IERC20,ITreasury) should be declared external:
        - Boardroom.initialize(IERC20,IERC20,ITreasury) (Boardroom.sol#743-761)
rewardPerShare() should be declared external:
        - Boardroom.rewardPerShare() (Boardroom.sol#815-817)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Boardroom.sol analyzed (9 contracts with 75 detectors), 44 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> LBond.sol

```
INFO:Detectors:
Context._msgData() (LBond.sol#101-103) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (LBond.sol#133-135)
symbol() should be declared external:
        - ERC20.symbol() (LBond.sol#141-143)
decimals() should be declared external:
        - ERC20.decimals() (LBond.sol#158-160)
totalSupply() should be declared external:
        - ERC20.totalSupply() (LBond.sol#165-167)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (LBond.sol#184-188)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (LBond.sol#207-211)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (LBond.sol#229-238)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (LBond.sol#252-256)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (LBond.sol#272-281)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (LBond.sol#517-519)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (LBond.sol#525-528)
operator() should be declared external:
        - Operator.operator() (LBond.sol#551-553)
isOperator() should be declared external:
        - Operator.isOperator() (LBond.sol#560-562)
transferOperator(address) should be declared external:
        - Operator.transferOperator(address) (LBond.sol#564-566)
mint(address,uint256) should be declared external:
        - LBond.mint(address,uint256) (LBond.sol#587-593)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:LBond.sol analyzed (8 contracts with 75 detectors), 16 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> Libra.sol

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Libra.sol#326-328)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Libra.sol#334-337)
operator() should be declared external:
        - Operator.operator() (Libra.sol#360-362)
transferOperator(address) should be declared external:
        - Operator.transferOperator(address) (Libra.sol#373-375)
name() should be declared external:
        - ERC20.name() (Libra.sol#411-413)
symbol() should be declared external:
        - ERC20.symbol() (Libra.sol#419-421)
decimals() should be declared external:
        - ERC20.decimals() (Libra.sol#436-438)
totalSupply() should be declared external:
        - ERC20.totalSupply() (Libra.sol#443-445)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (Libra.sol#462-466)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (Libra.sol#485-489)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (Libra.sol#507-516)
        - Libra.transferFrom(address,address,uint256) (Libra.sol#1160-1185)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (Libra.sol#530-534)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (Libra.sol#550-559)
isAddressExcluded(address) should be declared external:
        - Libra.isAddressExcluded(address) (Libra.sol#1048-1050)
setTaxTiersTwap(uint8,uint256) should be declared external:
        - Libra.setTaxTiersTwap(uint8,uint256) (Libra.sol#1052-1063)
setTaxTiersRate(uint8,uint256) should be declared external:
        - Libra.setTaxTiersRate(uint8,uint256) (Libra.sol#1065-1070)
setBurnThreshold(uint256) should be declared external:
        - Libra.setBurnThreshold(uint256) (Libra.sol#1072-1074)
```

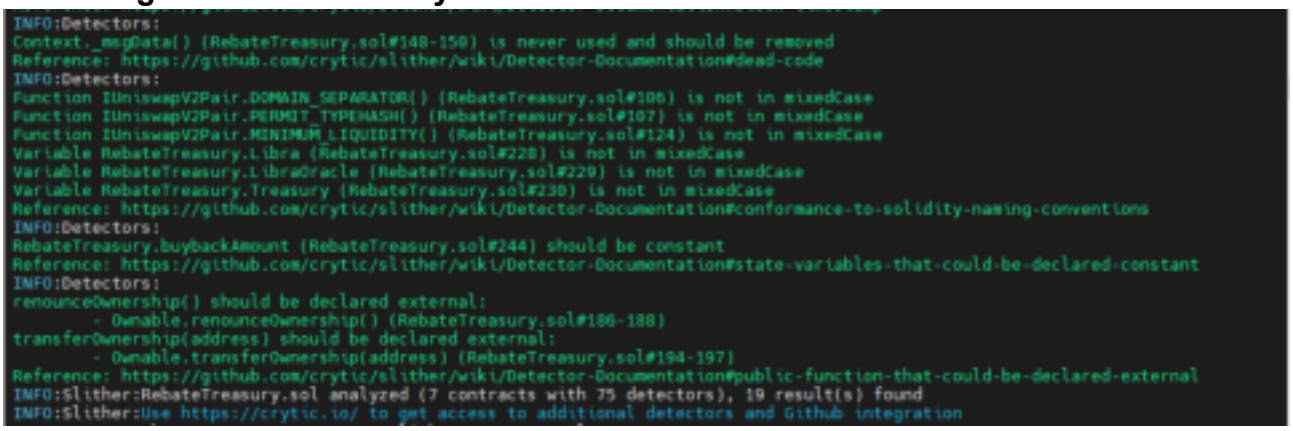
```
setBurnThreshold(uint256) should be declared external:
        - Libra.setBurnThreshold(uint256) (Libra.sol#1072-1074)
enableAutoCalculateTax() should be declared external:
        - Libra.enableAutoCalculateTax() (Libra.sol#1096-1098)
disableAutoCalculateTax() should be declared external:
        - Libra.disableAutoCalculateTax() (Libra.sol#1100-1102)
setLibraOracle(address) should be declared external:
        - Libra.setLibraOracle(address) (Libra.sol#1104-1107)
setTaxOffice(address) should be declared external:
        - Libra.setTaxOffice(address) (Libra.sol#1109-1113)
setTaxCollectorAddress(address) should be declared external:
        - Libra.setTaxCollectorAddress(address) (Libra.sol#1115-1118)
setTaxRate(uint256) should be declared external:
        - Libra.setTaxRate(uint256) (Libra.sol#1120-1124)
includeAddress(address) should be declared external:
        - Libra.includeAddress(address) (Libra.sol#1132-1136)
mint(address,uint256) should be declared external:
        - Libra.mint(address,uint256) (Libra.sol#1144-1150)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Libra.sol analyzed (12 contracts with 75 detectors), 74 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> LShare.sol

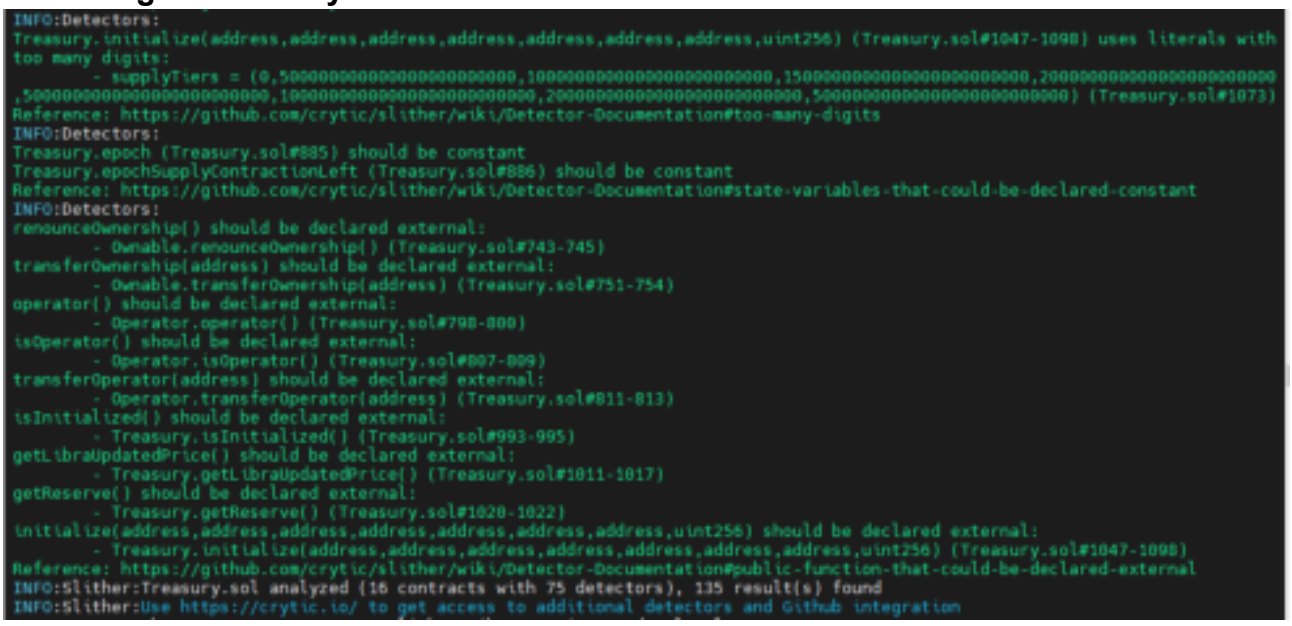
```
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (LShare.sol#352-354)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (LShare.sol#360-363)
operator() should be declared external:
        - Operator.operator() (LShare.sol#386-388)
isOperator() should be declared external:
        - Operator.isOperator() (LShare.sol#395-397)
transferOperator(address) should be declared external:
        - Operator.transferOperator(address) (LShare.sol#399-401)
name() should be declared external:
        - ERC20.name() (LShare.sol#437-439)
symbol() should be declared external:
        - ERC20.symbol() (LShare.sol#445-447)
decimals() should be declared external:
        - ERC20.decimals() (LShare.sol#462-464)
totalSupply() should be declared external:
        - ERC20.totalSupply() (LShare.sol#469-471)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (LShare.sol#476-478)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (LShare.sol#488-492)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (LShare.sol#511-515)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (LShare.sol#533-542)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (LShare.sol#556-560)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (LShare.sol#576-585)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (LShare.sol#781-784)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:LShare.sol analyzed (9 contracts with 75 detectors), 38 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> Oracle.sol

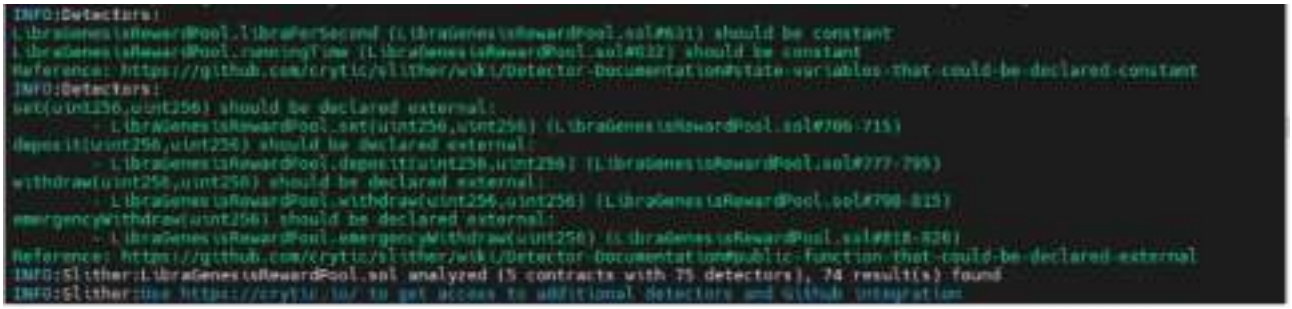
Slither log >> RebateTreasury.sol



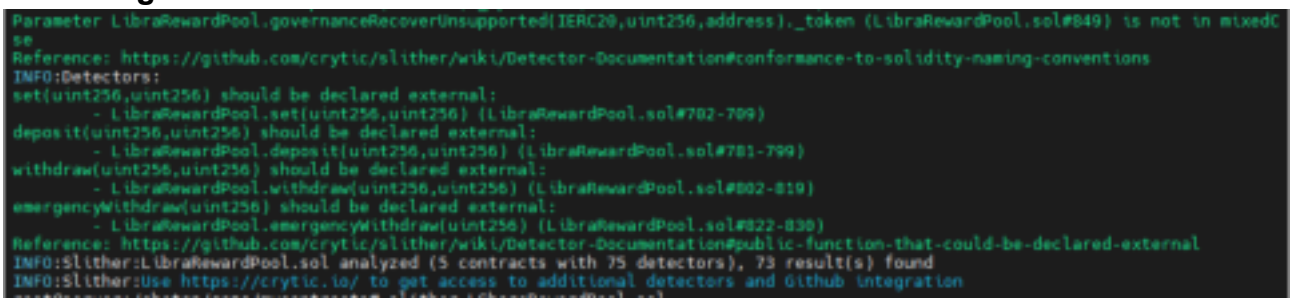
Slither log >> Treasury.sol



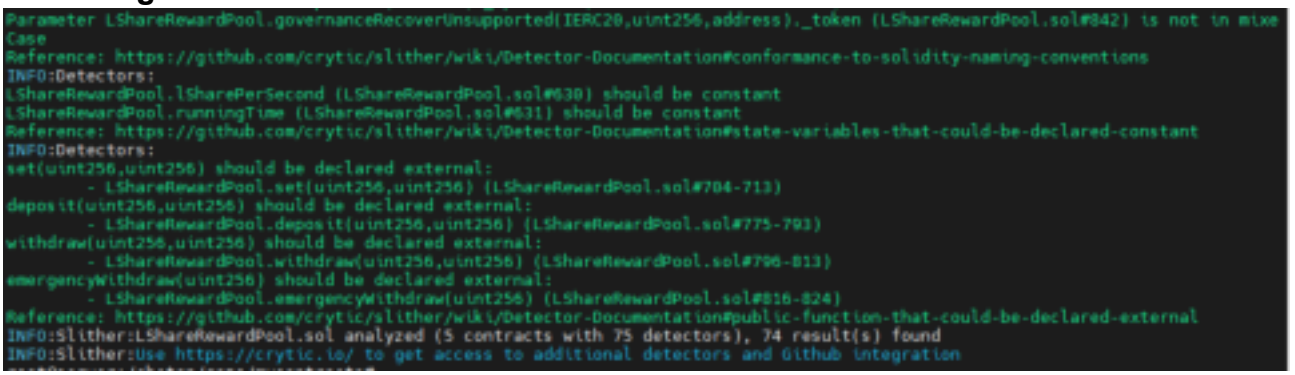
Slither log >> LibraGenesisRewardPool.sol



Slither log >> LibraRewardPool.sol



Slither log >> LShareRewardPool.sol



Solidity Static Analysis

Boardroom.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Boardroom.withdraw(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 835:4:

Low level calls:

Use of "delegatecall": should be avoided whenever possible. External code, that is called can change the state of the calling contract and send ether from the caller's balance. If this is wanted behaviour, use the Solidity library feature if possible.

more

Pos: 289:50:

Gas & Economy

Gas costs:

Gas requirement of function Boardroom.claimReward is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 847:4:

Gas costs:

Gas requirement of function Boardroom.allocateSeigniorage is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 858:4:

Miscellaneous

Constant/View/Pure functions:

Boardroom.governanceRecoverUnsupported(contract IERC20,uint256,address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 877:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 883:8:

LBond.sol

Gas & Economy

Gas costs:

Gas requirement of function LBond.burn is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 595:4:

Gas costs:

Gas requirement of function LBond.burnFrom is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 599:4:

Miscellaneous

Constant/View/Pure functions:

LBond.burnFrom(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 599:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 569:8:

Libra.sol

Gas & Economy

Gas costs:

Gas requirement of function Libra.distributeReward is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1213:4:

Miscellaneous

Constant/View/Pure functions:

Libra.burnFrom(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1156:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1221:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 943:19:

LShare.sol

Security

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 852:23:

Gas & Economy

Gas costs:

Gas requirement of function LShare.distributeReward is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 894:4:

Gas costs:

Gas requirement of function LShare.burn is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 901:4:

Miscellaneous

Similar variable names:

ERC20Burnable.burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 783:23:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 896:8:

Oracle.sol

Security

Block timestamp:

Use of "now": "now" does not mean current time. "now" is an alias for "block.timestamp". "block.timestamp" can be influenced by miners to a certain degree, be careful.
more
Pos: 576:20:

Gas & Economy

Gas costs:

Gas requirement of function Oracle.consult is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 678:4:

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type

more

Pos: 13:4:

Miscellaneous

Constant/View/Pure functions:

Oracle.twap(address,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 687:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 650:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 691:54:

RebateTreasury.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in RebateTreasury.bond(address,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 280:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 386:76:

Gas & Economy

Gas costs:

Gas requirement of function RebateTreasury.getTokenPrice is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 435:4:

Miscellaneous

Constant/View/Pure functions:

RebateTreasury.getLibraReturn(address,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 404:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 369:12:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 466:15:

Treasury.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Treasury._sendToBoardroom(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1288:4:

Block timestamp:

Use of "now": "now" does not mean current time. "now" is an alias for "block.timestamp". "block.timestamp" can be influenced by miners to a certain degree, be careful.
more
Pos: 1302:31:

Gas & Economy

Gas costs:

Gas requirement of function Treasury.getLibraPrice is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1003:4:

Miscellaneous

Constant/View/Pure functions:

Address.functionStaticCall(address,bytes) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 536:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1182:8:

LibraGenesisRewardPool.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in LibraGenesisRewardPool.updatePool(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 754:4:

Gas & Economy

Gas costs:

Gas requirement of function LibraGenesisRewardPool.set is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 706:4:

Miscellaneous

Constant/View/Pure functions:

Address.functionStaticCall(address,bytes) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 430:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 855:16:

LibraRewardPool.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in LibraRewardPool.updatePool(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 758:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 690:58:

Gas & Economy

Gas costs:

Gas requirement of function LibraRewardPool.set is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 702:4:

Miscellaneous

Constant/View/Pure functions:

Address.functionStaticCall(address,bytes) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 430:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 859:16:

LShareRewardPool.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in LShareRewardPool.updatePool(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 752:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 735:12:

Gas & Economy

Gas costs:

Gas requirement of function LShareRewardPool.set is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 704:4:

Miscellaneous

Constant/View/Pure functions:

Address.functionStaticCall(address,bytes) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 430:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 845:12:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 849:16:

Solhint Linter

Boardroom.sol

```
Boardroom.sol:3:1: Error: Compiler version 0.6.12 does not satisfy
the r semver requirement
Boardroom.sol:134:71: Error: Code contains empty
blocksBoardroom.sol:289:51: Error: Avoid using low level calls.
Boardroom.sol:311:17: Error: Avoid using inline assembly. It is
acceptable only in rare cases
Boardroom.sol:408:38: Error: Avoid to use tx.origin
Boardroom.sol:421:31: Error: Avoid to use tx.origin
```

LBond.sol

```
LBond.sol:3:1: Error: Compiler version 0.6.12 does not satisfy the r
semver requirement
LBond.sol:433:24: Error: Code contains empty blocks
LBond.sol:453:24: Error: Code contains empty blocks
LBond.sol:579:63: Error: Code contains empty blocks
```

Libra.sol

```
Libra.sol:3:1: Error: Compiler version 0.6.12 does not satisfy the r
semver requirement
Libra.sol:711:24: Error: Code contains empty blocks
Libra.sol:731:24: Error: Code contains empty blocks
```

LShare.sol

```
LShare.sol:3:1: Error: Compiler version 0.6.12 does not satisfy the r
semver requirement
LShare.sol:737:24: Error: Code contains empty blocks
LShare.sol:757:24: Error: Code contains empty
blocksLShare.sol:888:27: Error: Avoid to make time-based decisions in
your business logic
```

Oracle.sol

```
Oracle.sol:3:1: Error: Compiler version 0.6.12 does not satisfy the r
semver requirementOracle.sol:119:5: Error: Contract name must be in
CamelCase
```

```
Oracle.sol:567:47: Error: Use double quotes for string literals
Oracle.sol:576:21: Error: Avoid to make time-based decisions in your
business logic
Oracle.sol:606:60: Error: Use double quotes for string literals
```

RebateTreasury.sol

```
RebateTreasury.sol:2:1: Error: Compiler version ^0.6.12 does not
satisfy the r semver requirement
RebateTreasury.sol:465:29: Error: Avoid to make time-based decisions
in your business logic
RebateTreasury.sol:465:77: Error: Avoid to make time-based decisions
in your business logic
```

Treasury.sol

```
Treasury.sol:602:17: Error: Avoid using inline assembly. It is
acceptable only in rare cases
Treasury.sol:1310:30: Error: Avoid to make time-based decisions in
your business logic
Treasury.sol:1313:34: Error: Variable "_amount" is unused
```

LibraGenesisRewardPool.sol

```
LibraGenesisRewardPool.sol:3:1: Error: Compiler version 0.6.12 does
not satisfy the r semver requirement
LibraGenesisRewardPool.sol:474:51: Error: Avoid using low level
calls.
LibraGenesisRewardPool.sol:496:17: Error: Avoid using inline
assembly. It is acceptable only in rare cases
LibraGenesisRewardPool.sol:645:17: Error: Avoid to make time-based
decisions in your business logic
LibraGenesisRewardPool.sol:675:13: Error: Avoid to make time-based
decisions in your business logic
```

LibraRewardPool.sol

```
LibraRewardPool.sol:3:1: Error: Compiler version 0.6.12 does not
satisfy the r semver requirement
LibraRewardPool.sol:691:35: Error: Avoid to make time-based decisions
in your business logic
LibraRewardPool.sol:694:85: Error: Avoid to make time-based decisions
in your business logic
LibraRewardPool.sol:777:31: Error: Avoid to make time-based decisions
in your business logic
```

```
LibraRewardPool.sol:853:13: Error: Avoid to make time-based decisions
in your business logic
```

LShareRewardPool.sol

```
LShareRewardPool.sol:3:1: Error: Compiler version 0.6.12 does not
satisfy the r semver requirement
LShareRewardPool.sol:319:71: Error: Code contains empty blocks
LShareRewardPool.sol:771:31: Error: Avoid to make time-based
decisions in your business logic
LShareRewardPool.sol:843:13: Error: Avoid to make time-based
decisions in your business logic
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io