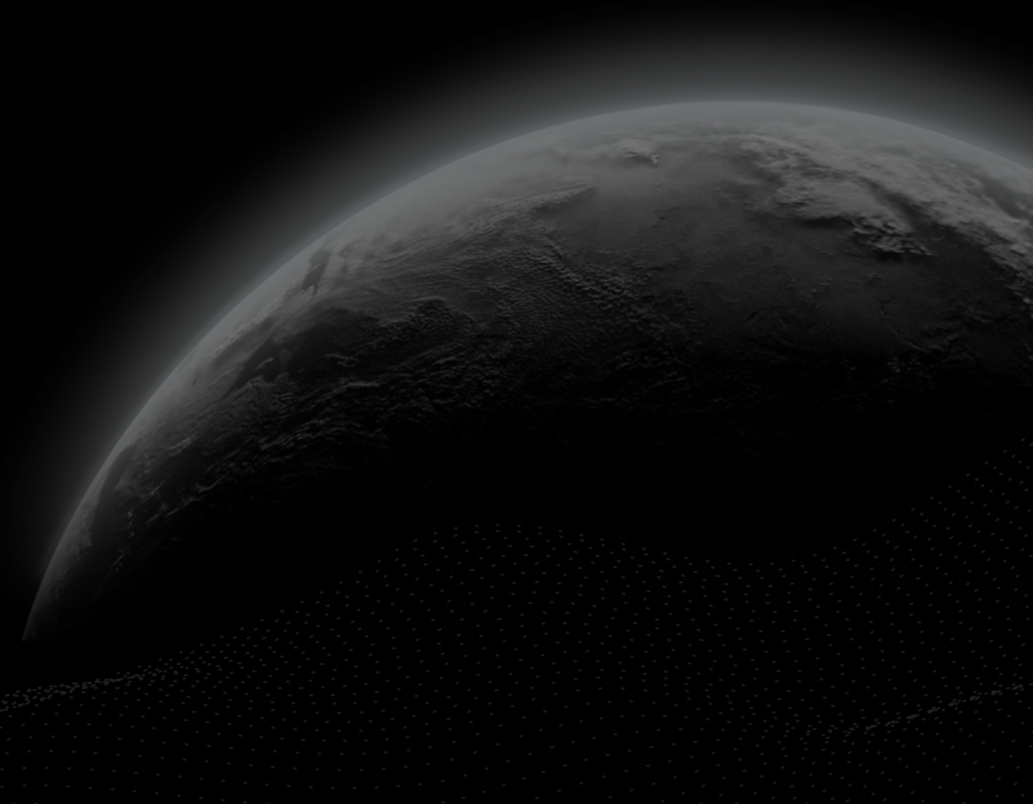
CERTIK

Security Assessment

Bytemasons - Stablecoin

CertiK Verified on Nov 28th, 2022



CertiK Verified on Nov 28th, 2022

Bytemasons - Stablecoin

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Fantom	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 11/28/2022	N/A

CODEBASE

https://github.com/Byte-Masons/liquity-dev

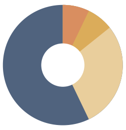
...View All

COMMITS

- 7086d2aa437420d4a8c8a6073bed258d874766c9
- e47f8f2a5f8e60e1b04339511f90f8481ab26995

...View All

Vulnerability Summary



14	5	0	4	5	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

🟥	0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧	1	Major	1 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨	1	Medium	1 Partially Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	4	Minor	2 Resolved, 2 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦	8	Informational	3 Resolved, 3 Partially Resolved, 2 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | BYTEMASONS - STABLECOIN

CODEBASE | BYTEMASONS - STABLECOIN

Repository

https://github.com/Byte-Masons/liquity-dev

Commit

- 7086d2aa437420d4a8c8a6073bed258d874766c9
- e47f8f2a5f8e60e1b04339511f90f8481ab26995

AUDIT SCOPE | BYTEMASONS - STABLECOIN

60 files audited ● 14 files with Acknowledged findings ● 46 files without findings

ID	Repo	Commit	File	SHA256 Checksum
● LBD	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/LiquityBase.sol	1458e40af0331a0a54f4d3dab9868acd4d5947137bc4feca6792a94d5b1b7933
● TCD	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/TellorCaller.sol	c4695396044b0a68d7b7577027ee69ce3348b75b454c5078c885d7d945a70644
● CIL	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/LQTY/CommunityIssuance.sol	1fc12d7125f2f1f7355b45da8e1158f307b6eff870f4b80f8d4d902e18ca22b6
● LQY	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/LQTY/LQTYStaking.sol	345a66d88133cc1fe8a5a5e4922dacca2bb982206b5b3a19b26f5e214f238286
● APB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/ActivePool.sol	cab87d196613f4a72c8de2ddb6e3fbfb7c94a794806d6c777c76dc1f83f53dd8
● BOB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/BorrowerOperations.sol	6f171e7ceb02d6443d5cf2225b85fa65951bc9e209dd63bf2e79f1f9e86d6f14
● CSP	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/CollSurplusPool.sol	0de226fbce27d0831ce6dfb050b3b2fa1db2136e5935f7f93df3410213ec36d2
● DPB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/DefaultPool.sol	a18ad14334ff9911b8886d96d54fa4600017358c3ec2569ce9aaa08b64e58ba3
● HHB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/HintHelpers.sol	ec8d8f2fc601b1a40bd7febf06f6765635e1cf161213967d542e8ebc8ef6bea1

ID	Repo	Commit	File	SHA256 Checksum
LUS	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/LUSDToken.sol	d51c34e6b5b779da4ec2016fac2261d93432b8ea67cf76d31fbf677acb659969
PFB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/PriceFeed.sol	b460d78c70a48e69a0dd0aed5d533c9942327af702d6a4d69ed315ad0647e936
STB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/SortedTroves.sol	ed1bbed4701f8b7f1e857bb7aa4e1edac246235ec82b4de43ca374a6e3b26828
SPB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/StabilityPool.sol	fc0bf68b2ed964893a8b72d7fcf1dea6c7eb44699d1f27215983fe14fbdd766e
TMB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/TroveManager.sol	2107ec0d547ee1b21f753c0ce2b5c4bb344fd232cacba20958d7c27510140220
ADB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/Address.sol	05a6a49cf9cc82c283f36d65e20f1e16fbf850588cb3312ad3c52f15eb4b6a12
AVI	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/AggregatorV3Interface.sol	b2ebef44df63d7e32405e0eb398a5868123c5f1800005c02d44d53cec38a28df
BMD	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/BaseMath.sol	fada1c3c95dcfa780a6d03bab5e1f329201a30ec94140b54818559f9e720cbc4
CCD	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/CheckContract.sol	aa32079b9f38a1669beb9fefe2e0af95fc543e1b717617713f00a648f0dc4b66
IER	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/IERC20.sol	64f3e9f771f7ba660ba11cf966318da692834288126f5b58601d8ba3ffc1a3fa
IEC	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/IERC2612.sol	83a6ac8c0f185342c500ada926f0dbe734a959382bbcbe8e1c69bcf7faa2454b

ID	Repo	Commit	File	SHA256 Checksum
IED	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/IERC4626.sol	51b6a0541758e81e0a7528d0f25279fa41dea20ec24e520f575b3b959a470dbc
ITD	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/ITellor.sol	01e59ebf4bb055a20905ea2d223d364d0e0ad8dcf4e3584dab245a9a3e4cacca
LMD	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/LiquityMath.sol	0d418289d7ba0ab052d1fff29b3c832bd809175d9bd69b8cd5552184a5e1bf17
LSM	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/LiquitySafeMath128.sol	93aa3e470a6e581ea7e515d2e34737f5e619af229637054ac09f0d303c0bddd2
ODB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/Ownable.sol	645208d3053f1ee614b73776e9c638f4529062bfc3333fa06ba5663d9193405b
SER	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/SafeERC20.sol	c8415a0fc3e2468a9875150199538a8861a4f242036981dc5bf963784c42ec04
SMD	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/SafeMath.sol	caa5397440fd9a0988eb40c136bd7a58baad05012edcf244f6b586e167e531f6
CON	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Dependencies/console.sol	fe7de02fbe78bf1af499331c9a5a404299a7141f0800e942e29b55c8c64029dc
IAP	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/IActivePool.sol	6234a3f23243411ec38aefdc2b05ad7310c360202e28b5064c32f61a334002ac
IBO	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/IBorrowerOperations.sol	5871184e47b1915b0a1ea746c895ac5ed6e0b6a6d7339bccb75fa73454022b97
ICS	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/ICollSurplusPool.sol	e2468140313d3222388f964d1128d3be47abd0814f0dac5ce708b866cee26c80

ID	Repo	Commit	File	SHA256 Checksum
ICI	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/ICommunityIssuance.sol	1b25e623b3db2a2d18dd30d347823e8640f601747a3632fb941d9bcdf84fd898
IDP	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/IDefaultPool.sol	e39e723083a938120f1d3725eb6c4baaac4f903b35ecec88f6799c72bc8bc432
ILQ	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/ILQTYStaking.sol	946b32e1d3177acb9d93143f589199292eb207bf9482dbcb44afca45cf1ca623
ILT	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/ILQTYToken.sol	6d54ecced315fda5a33ddeaf729f178fd55cb5d0990fa7157272fdf1efa2b9df
ILU	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/ILUSDToken.sol	30ce23ad28599dbf8c1273ff7d622a804297194fb0c3be89da25548f0b7c67a1
ILB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/ILiquityBase.sol	48697f434db39ab90174b90dd36ce654ec78e3e8ed169efb2dad119761fdab4a
ILC	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/ILockupContractFactory.sol	7e7c6a8d9f4dc43a6f02b1de70175a5964f7ef6e835492d426f8aa1854e20358
IPI	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/IPool.sol	faabec3b0e076b7e0082fbdaf616afdad9bc6d9ee024a4b22895127d7fb959a1
IPF	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/IPriceFeed.sol	65e2a0fa0a29f8574cec5d574383bbe780897b96313d718f9cbd376ac9ce3319
IST	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/ISortedTroves.sol	426ac4fe18c04834b64d068a4c310e8271969cebf020e4ed885bc97a08919393
ISP	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/IStabilityPool.sol	0efa7deae6057b70d2aa25e3c90af13e99709f5a83d562022faa0116131c1a10

ID	Repo	Commit	File	SHA256 Checksum
● ITC	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/ITellorCaller.sol	6c2a05ec4fabf91aee0a66145ec8d0566a34d54fcf9a0c38db9e808d553fbf7b
● ITM	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Interfaces/ITroveManager.sol	25d656df184c9e87059023a1b0cee0117e28e45f7bf61a8713521d165efa3d5b
● ADL	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/LPRewards/Dependencies/Address.sol	05a6a49cf9cc82c283f36d65e20f1e16fbf850588cb3312ad3c52f15eb4b6a12
● SEC	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/LPRewards/Dependencies/SafeERC20.sol	2bd09642c108993133303aa419f9edef8e94bcbab44411208e7e9e3da014c639
● ILW	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/LPRewards/Interfaces/ILPTokenWrapper.sol	bcfedabf6b5ae1487f11d40856510c1464068c720bb4ff42c22f6ea0bf311b6a
● IUI	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/LPRewards/Interfaces/IUnipool.sol	a0a344fec8abc86cfd17606b31d333fc9fd45d038faed69ef78b60eebf6d0d0e
● ERC	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/LPRewards/TestContracts/ERC20Mock.sol	98294836397df21cc98a3a196ef28935f8c267f7658b22da0edd442cfa969186
● LCL	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/LQTY/LockupContract.sol	cabc456a0dfd3b1f02b71129cc9dba655fe1c09ebfe5027c28331d594efbed9d
● GPB	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/GasPool.sol	9aab938a8b7985e223e5e0d13bbd720d2a0e365706dd789668968d67ed8e9581
● MBM	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Migrations.sol	8ccccd7b7cda827b7a839a8dae6fac75c85e4e8431d2ce8624470ae303aa281e
● MTG	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/MultiTroveGetter.sol	38effa9b3ea156805b18655641bfabb5cd9fe762bd3e25651a3f444931a23de1

ID	Repo	Commit	File	SHA256 Checksum
● BOS	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Proxy/BorrowerOperationsScript.sol	05a207d8612abb8f75eab48257f90cf6c197bb904e4a4733077ff8f294d9b325
● BWS	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Proxy/BorrowerWrappersScript.sol	a7a41ac90f3b03ecdfa8c279a03ebf825d6ce3d511ca197168b2620728d4e4d2
● ERT	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Proxy/ERC20TransferScript.sol	b8d6fe5614585beb85474c112e4715c1d4ee487f27829b4a17bc4fb79411c87b
● LQS	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Proxy/LQTYStakingScript.sol	b914f73e833ba81e98449d88e71ffee1588ef6ec3fde5ffa90fc8ff15dc935fb
● SPS	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Proxy/StabilityPoolScript.sol	1203a9e156482b50b487e551acd34d26ebaf4b864332d344dbc560fa4890103e
● TSP	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Proxy/TokenScript.sol	60436ad5cab76665145175d0329dcd10451fff19d3715930a94c12e3871c27f7
● TMS	Byte-Masons/liquity-dev	7086d2a	packages/contracts/contracts/Proxy/TroveManagerScript.sol	a413eccfd9da6551ef8467f4e49311bc18b09b5e2038d8bdfa63e7f1582e3223

APPROACH & METHODS | BYTEMASONS - STABLECOIN

This report has been prepared for Bytemasons to discover issues and vulnerabilities in the source code of the Bytemasons - Stablecoin project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis 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:

- Testing the smart contracts against both common and uncommon attack vectors;
- 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.

REVIEW NOTES | BYTEMASONS - STABLECOIN

Overview

The **Byte Masons** is a development collective pursuing open, secure, and reliable systems focused on helping users navigate the new web and Decentralized Finance (DeFi).

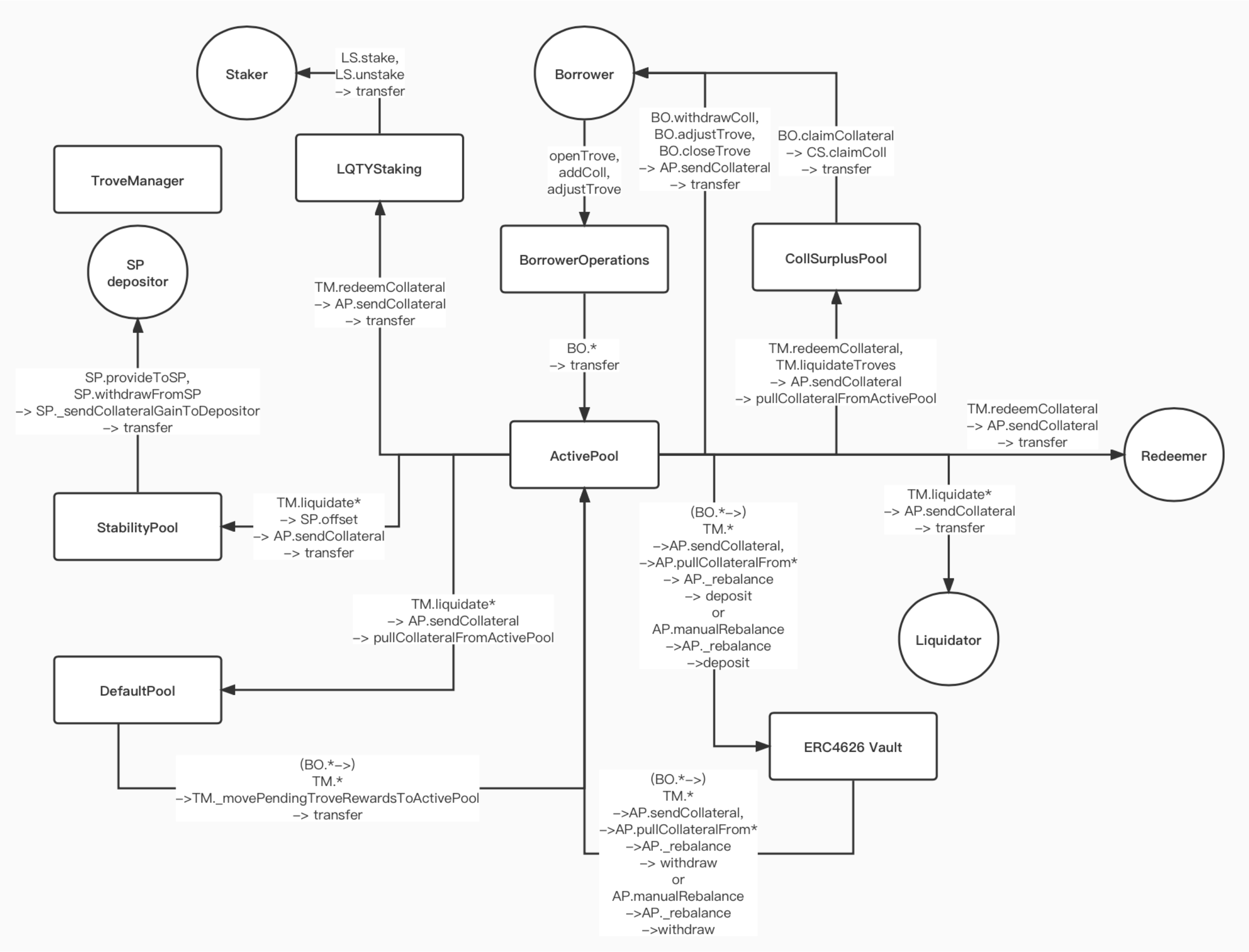
Byte Masons, in the pursuit of public good, seek to improve access to financial tools, transparency of financial organizations, and ethics in business operations.

The **Stablecoin** is a decentralized protocol that allows ERC20 token holders to obtain maximum liquidity against their collateral without paying interest. After locking up ERC20 tokens as collateral in a smart contract and creating an individual position called a "trove", the user can get instant liquidity by minting LUSD, a USD-pegged stablecoin. Each trove is required to be collateralized at a minimum of 110%. Any owner of LUSD can redeem their stablecoins for the underlying collateral at any time. The redemption mechanism along with algorithmically adjusted fees guarantees a minimum stablecoin value of 1 USD.

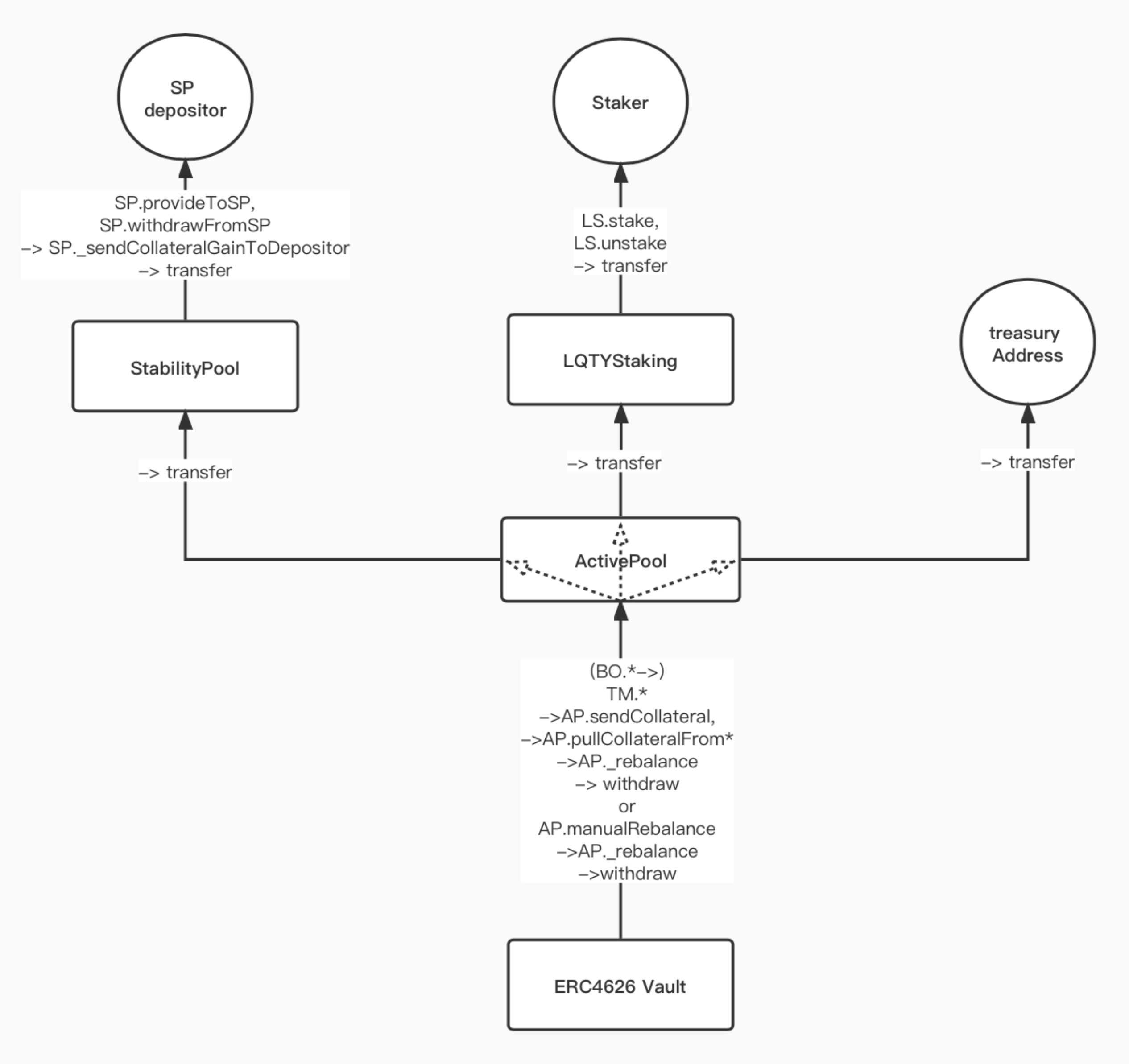
The main user-facing components in the **Stablecoin** platform include:

- `BorrowerOperations.sol` : contains the basic operations by which borrowers interact with their trove, such as trove creation, collateral top-up/withdrawal, and stablecoin issuance and repayment.
- `TroveManager.sol` : contains functionality for liquidations and redemptions.
- `StabilityPool.sol` : contains functionality for making deposits and withdrawing compounded deposits, accumulated collaterals, and LQTY gains.

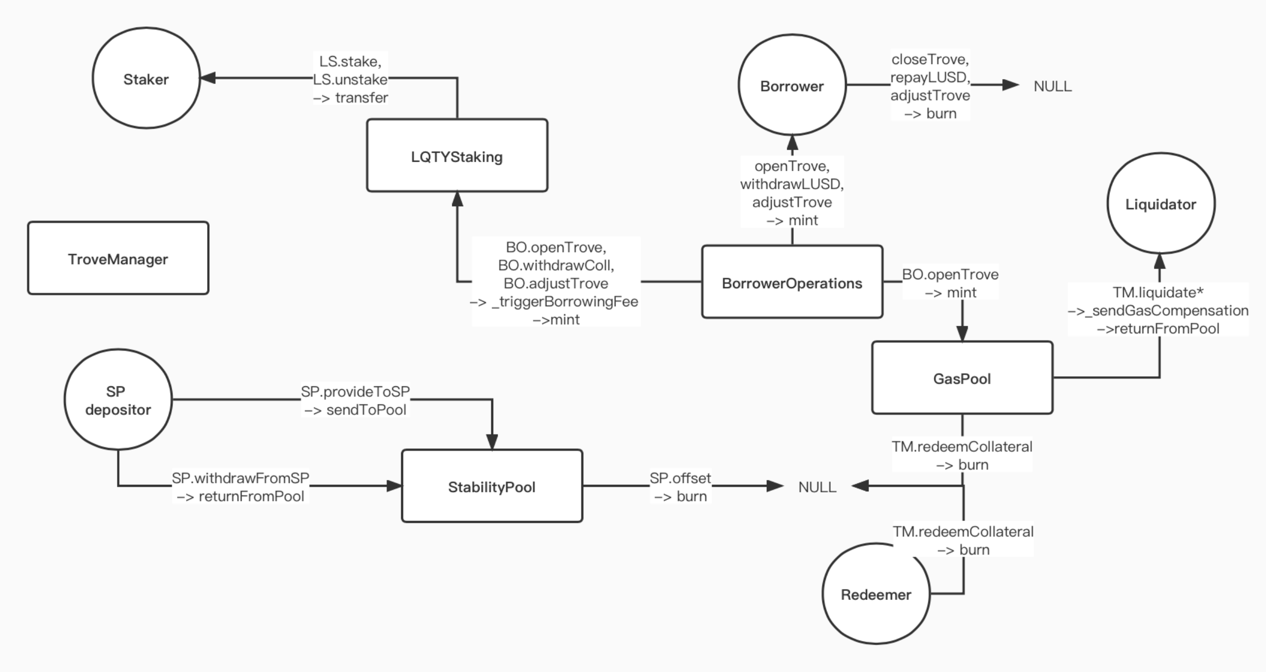
Flow of Collateral Capital in Byte Masons


Staker
LS.stake, LS.unstake –> transfer
Borrower
BO.withdrawColl, BO.adjustTrove, BO.closeTrove –> AP.sendCollateral –> transfer
BO.claimCollateral –> CS.claimColl –> transfer
TroveManager
LQTYStaking
openTrove, addColl, adjustTrove
SP depositor
BorrowerOperations
CollSurplusPool
TM.redeemCollateral –> AP.sendCollateral –> transfer
BO.* –> transfer
TM.redeemCollateral, TM.liquidateTroves –> AP.sendCollateral –> pullCollateralFromActivePool
SP.provideToSP, SP.withdrawFromSP –> SP._sendCollateralGainToDepositor –> transfer
TM.redeemCollateral –> AP.sendCollateral –> transfer
ActivePool
Redeemer
TM.liquidate* –> SP.offset –> AP.sendCollateral –> transfer
StabilityPool
(BO.*–>) TM.* –>AP.sendCollateral, –>AP.pullCollateralFrom* –> AP._rebalance –> deposit or AP.manualRebalance –>AP._rebalance –>deposit
TM.liquidate* –> AP.sendCollateral –> transfer
Liquidator
TM.liquidate* –> AP.sendCollateral –> pullCollateralFromActivePool
DefaultPool
ERC4626 Vault
(BO.*–>) TM.* –>TM._movePendingTroveRewardsToActivePool –> transfer
(BO.*–>) TM.* –>AP.sendCollateral, –>AP.pullCollateralFrom* –>AP._rebalance –> withdraw or AP.manualRebalance –>AP._rebalance –>withdraw

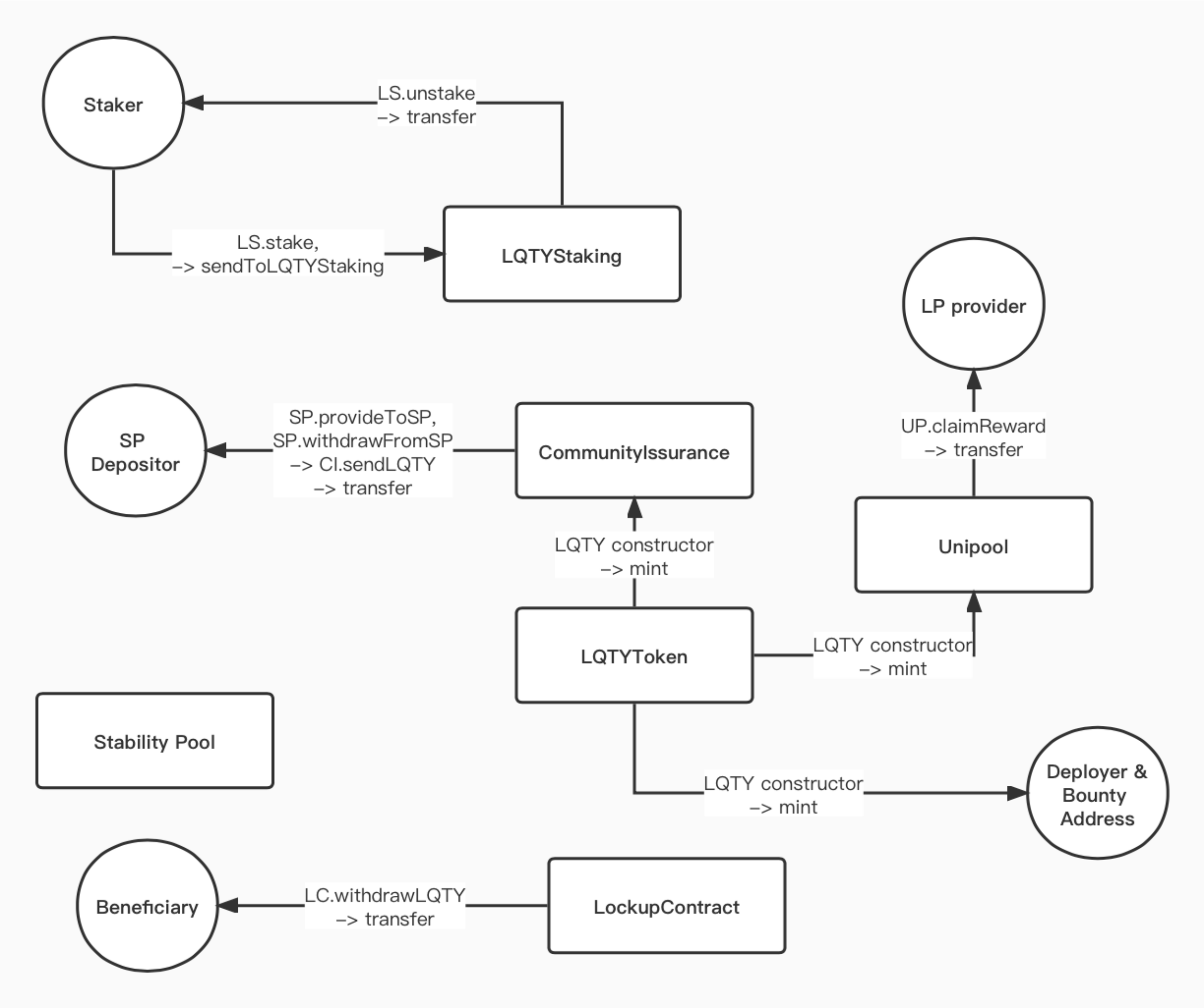
Flow of Collateral Profit in Byte Masons



Flow of LUSD in Byte Masons



Flow of LQTY TOKEN in Byte Masons



External Dependencies

In **Byte Masons**, the system inherits or uses a few depending injection contracts or addresses to fulfill the need of its business logic that is defined below:

Contracts

The project uses Openzeppelin libraries and contracts for contract format and functionality.

The following contracts are referenced in various files of the codebase:

- `@openzeppelin/contracts/token/ERC20/ERC20.sol`
- `@openzeppelin/contracts/token/ERC20/SafeERC20.sol`

In addition to previous Openzeppelin libraries, the following external component interfaces are declared:

- An ERC4626 vault interface
- A Chainlink contract interface
- A Tellor contract interface

The `ActivePool.sol` interacts with the third-party ERC4626 vault to allocate a certain (configurable) percentage of the assets, which are kept by the ActivePool, for earning yield.

The `BorrowOperation.sol` , `TroveManager.sol` , and other contracts depend on the third-party Chainlink and Tellor protocols to query the collateral price to do the business logic, such as open/close a trove, liquidations, etc.

The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

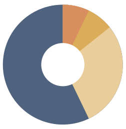
Privileged Functions

In the **Byte Masons** project, multiple privileged roles are adopted to ensure a good runtime behavior in the project, which were specified in the finding *GLOBAL-01* | *Centralization Related Risks*.

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the vault settings and configuration according to the runtime required to best serve the community. It is also worthy to note the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

FINDINGS | BYTEMASONS - STABLECOIN



14	0	1	1	4	8
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Bytemasons - Stablecoin. Through this audit, we have uncovered 14 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GLOBAL-01	Third Party Dependencies	Volatile Code	Minor	● Acknowledged
BMB-01	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	● **Acknowledged**
BMB-02	Potential Attacks When Price Oracle Goes Down	Logical Issue	Medium	● Partially Resolved
BMB-03	Susceptible To Signature Malleability	Volatile Code	Minor	● Resolved
BMB-04	Economic Model - Gas Compensation	Logical Issue	Minor	● Acknowledged
SPB-01	Event Is Not Emitted	Compiler Error	Minor	● Resolved
GLOBAL-02	Whitelist For The Collateral Tokens	Logical Issue	Informational	● Partially Resolved
GLOBAL-03	Governance Model - New Feature Of Pause	Logical Issue	Informational	● Resolved
BMB-05	The Tellor Protocol Does Not Support The Fantom Chain For Now	Volatile Code	Informational	● Acknowledged
BMB-06	Missing Error Messages	Coding Style	Informational	● Partially Resolved

ID	Title	Category	Severity	Status
BMB-07	Missing Emit Events	Coding Style	Informational	● Partially Resolved
BMB-08	Economic Model - Minimum Net Debt	Logical Issue	Informational	● Acknowledged
BMB-09	The Same MCR And CCR For All The Allowed Collaterals	Logical Issue	Informational	● Resolved
BOB-01	Missing A Check For Collateral On `adjustTrove()`	Inconsistency	Informational	● Resolved

GLOBAL-01 | THIRD PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	● Minor		● Acknowledged

Description

The ActivePool interacts with the third-party ERC4626 vault to allocate a certain (configurable) percentage of the assets, which are kept by the ActivePool, for earning yield.

The BorrowOperation, TroveManager, and other contracts depend on the third-party Chainlink and Tellor protocols to query the collateral price to do business logic, such as opening/closing a trove, liquidations, etc.

The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of this project requires interaction with ERC4626 vault, Chainlink, Tellor, etc. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Bytemasons - Stablecoin]: The team acknowledged this issue and will monitor the status internally.

BMB-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	**packages/contracts/contracts/ActivePool.sol: 62, 112, 117, 122, 126, 189, 279, 286, 294; packages/contracts/contracts/BorrowerOperations.sol: 104~116, 597; packages/contracts/contracts/CollSurplusPool.sol: 38~42, 106, 112, 118; packages/contracts/contracts/DefaultPool.sol: 37~40, 106, 110; packages/contracts/contracts/HintHelpers.sol: 24~27; packages/contracts/contracts/LQTY/CommunityIssuance.sol: 66~70, 129; packages/contracts/contracts/LQTY/LQTYStaking.sol: 64~72, 253, 257; packages/contracts/contracts/LQTY/LockupContractFactory.sol: 45; packages/contracts/contracts/LUSDToken.sol: 262, 266, 275, 279; packages/contracts/contracts/PriceFeed.sol: 86~91; packages/contracts/contracts/SortedTroves.sol: 80, 402, 406; packages/contracts/contracts/StabilityPool.sol: 259~268, 846, 850; packages/contracts/contracts/TroveManager.sol: 244~256, 1508**	● **Acknowledged**

Description

In the contracts listed below, the owner always controls the contract and the contract does not have a `public/external` function to renounce or transfer the ownership.

`ActivePool.sol`

The active Pool holds the collateral and LUSD debt for each collateral (but not LUSD tokens) for all active troves.

- `setAddresses()` : set all the addresses related to the business logic of this contract, can only be called once;
- `setYieldingPercentage()` : manage the state variable `yieldingPercentage` ;
- `setYieldingPercentageDrift()` : manage the state variable `yieldingPercentageDrift` ;
- `setYieldClaimThreshold()` : manage the state variable `yieldClaimThreshold` ;
- `setYieldDistributionParams()` : manage the state variables `yieldSplitTreasury` , `yieldSplitSP` and `yieldSplitStaking` ;
- `manualRebalance()` : manually rebalance the collateral between the active pool and the corresponding ERC4626 vault.

Any compromise to the owner may allow a hacker to take advantage of this authority and change the configurations of the contract.

In the contracts listed below, the owner will renounce the ownership of the contract after calling the function that initializes configurations.

- `BorrowerOperations.sol` : contains the basic operations by which borrowers interact with their Trove: Trove creation, ETH top-up/withdrawal, stablecoin issuance, and repayment;
- `CollSurplusPool.sol` : holds the ERC20 token surplus from troves that have been fully redeemed as well as from troves with an ICR > MCR that were liquidated in Recovery Mode;
- `DefaultPool.sol` : holds the total ERC20 token balance and records the total stablecoin debt of the liquidated troves that are pending redistribution to active troves;
- `HintHelpers.sol` : helper contract, containing the read-only functionality for calculation of accurate hints to be supplied to borrower operations and redemptions;
- `PriceFeed.sol` : contains functionality for obtaining the current `Collateral:USD` price, which the system uses for calculating collateralization ratios;
- `SortedTroves.sol` : a doubly linked list that stores addresses of Trove owners, sorted by their individual collateralization ratio (ICR);
- `StabilityPool.sol` : contains functionality for Stability Pool operations: making deposits, and withdrawing compounded deposits and accumulated ERC20 and LQTY gains;
- `TroveManager.sol` : contains functionality for liquidations and redemptions;
- `CommunityIssuance.sol` : handles the issuance of LQTY tokens to Stability Providers as a function of time;
- `LockupContractFactory.sol` : used to deploy LockupContracts;
- `LQTYStaking.sol` : contains stake and unstake functionality for LQTY holders.

It is noticed that the logic of the project needs the contracts to call each other, thus, many contracts in the **Stablecoin** have functions that require the caller to be a specified contract. Hence all contracts should be configured properly to ensure the correctness of the project. The related contracts are listed below:

`ActivePool.sol`

- `_requireCallerIsBorrowerOperationsOrDefaultPool()` : require the `msg.sender` must be `BorrowerOperations.sol` , or `DefaultPool.sol` contracts, used in the `pullCollateralFromBorrowerOperationsOrDefaultPool()` function;
- `_requireCallerIsBOorTroveMorSP()` : require the `msg.sender` must be `BorrowerOperations.sol` , or `TroveManager.sol` , or `StabilityPool.sol` contracts, used in the `sendCollateral()` , `decreaseLUSDDebt()` functions;
- `_requireCallerIsBOorTroveM()` : require the `msg.sender` must be `BorrowerOperations.sol` , or `TroveManager.sol` contracts, used in the `increaseLUSDDebt()` function.

`BorrowerOperations.sol`

- `_requireCallerIsStabilityPool()` : require the `msg.sender` must be `StabilityPool.sol` contract, used in the `moveCollateralGainToTrove()` function.

`CollSurplusPool.sol`

- `_requireCallerIsBorrowerOperations()` : require the `msg.sender` must be `BorrowerOperations.sol` contract, used in the `claimColl()` function;
- `_requireCallerIsTroveManager()` : require the `msg.sender` must be `TroveManager.sol` contract, used in the `accountSurplus()` function;
- `_requireCallerIsActivePool()` : require the `msg.sender` must be `ActivePool.sol` contract, used in the `pullCollateralFromActivePool()` function.

`DefaultPool.sol`

- `_requireCallerIsActivePool()` : require the `msg.sender` must be `ActivePool.sol` contract, used in the `pullCollateralFromActivePool()` function;
- `_requireCallerIsTroveManager()` : require the `msg.sender` must be `TroveManager.sol` contract, used in the `sendCollateralToActivePool()` , `increaseLUSDDebt()` , `decreaseLUSDDebt()` functions.

`LUSDToken.sol`

- `_requireCallerIsBorrowerOperations()` : require `msg.sender` must be `BorrowerOperations.sol` contract, used in the `mint()` function;
- `_requireCallerIsBOorTroveMorSP()` : require `msg.sender` must be `BorrowerOperations.sol` , or `TroveManager.sol` , or `StabilityPool.sol` contracts, used in the `burn()` function;
- `_requireCallerIsStabilityPool()` : require `msg.sender` must be `StabilityPool.sol` contract, used in the `sendToPool()` function;
- `_requireCallerIsTroveMorSP()` : require `msg.sender` must be `TroveManager.sol` , or `StabilityPool.sol` contracts, used in the `returnFromPool()` function.

`SortedTroves.sol`

- `_requireCallerIsTroveManager()` : require `msg.sender` must be `TroveManager.sol` contract, used in the `remove()` function;
- `_requireCallerIsBOorTroveM()` : require `msg.sender` must be `BorrowerOperations.sol` , or `TroveManager.sol` contracts, used in the `insert()` , `reInsert()` functions.

`StabilityPool.sol`

- `_requireCallerIsActivePool()` : require `msg.sender` must be `ActivePool.sol` contract, used in the `updateRewardSum()` function;
- `_requireCallerIsTroveManager()` : require `msg.sender` must be `TroveManager.sol` contract, used in the `offset()` function.

`TroveManager.sol`

- `_requireCallerIsBorrowerOperations()` : require `msg.sender` must be `BorrowerOperations.sol` contract, used in the `setTroveStatus()` , `increaseTroveColl()` , `decreaseTroveColl()` , `increaseTroveDebt()` , `decreaseTroveDebt()` , `applyPendingRewards()` , `updateTroveRewardSnapshots()` , `removeStake()` , `updateStakeAndTotalStakes()` , `closeTrove()` , `addTroveOwnerToArray()` , `decayBaseRateFromBorrowing()` functions.

`CommunityIssurance.sol`

- `_requireCallerIsStabilityPool()` : require `msg.sender` must be `StabilityPool.sol` contract, used in the `issueLQTY()` , `sendLQTY()` functions.

`LQTYStaking.sol`

- `_requireCallerIsTroveManagerOrActivePool()` : require `msg.sender` must be `TroveManager.sol` contract, used in the `increaseF_Collateral()` function;
- `_requireCallerIsBorrowerOperations()` : require `msg.sender` must be `BorrowerOperations.sol` contract, used in the `increaseF_LUSD()` function.

In the `ActivePool.sol` contract, the `treasuryAddress` address looks like an EOA (Externally Owned Account) because this value is not checked by `checkContract()` when the owner calls the `setAddresses()` function. If this address is indeed an EOA, it will be vulnerable to loss of assets due to private key compromise or other circumstances.

The following content is based on commit e47f8f2a5f8e60e1b04339511f90f8481ab26995.

`LUSDToken.sol`

- `_requireCallerIsGovernance()` : require `msg.sender` must be a governance address, which is a contract address and used in the `updateGovernance()` , `updateGuardian()` , `upgradeProtocol()` , `unpauseMinting()` functions;
- `pauseMinting()` : pause the mint of the LUSD token, controlled by `guardianAddress` and `governanceAddress` contracts.

`CollateralConfig.sol`

- `updateCollateralRatios()` : the `owner` of the contract can lower the collateralization requirements for a particular collateral.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[Bytemasons - Stablecoin]: Inside of ActivePool, we will retain ownership so we can configure variables that pertain to collateral farming. However, the addresses can be set only once--which ensures that owner cannot alter any other properties. Moreover, ownership will be transferred to a multisig after deployment.

All other contracts renounce ownership at the end of the setAddresses/initializer function.

BMB-02 POTENTIAL ATTACKS WHEN PRICE ORACLE GOES DOWN

Category	Severity	Location	Status
Logical Issue	● Medium	packages/contracts/contracts/BorrowerOperations.sol: 178, 273; packages/contracts/contracts/PriceFeed.sol: 146	● Partially Resolved

Description

Prices for collateral are fetched from Chainlink/Tellor in various calculations, most importantly the CR of a trove. The current timeout for the price uses the constant variable `TIMEOUT` (4 hours).

Chainlink and Tellor may stop updating the price of an asset in certain cases. During these scenarios, untimely price reactions could lead to an attack on stablecoin. The attacker can get the collateral at a lower price on other platforms and use these collaterals to borrow the stablecoin LUSD, which is more than the actual value.

In addition, there is no mint cap on the stablecoin LUSD and no pause function. As a result, the attacker can repeatedly arbitrage through the above method.

For example, the Venus protocol, a Compound-forked lending platform, was attacked in a similar case.

Reference:

- https://www.tronweekly.com/terra-causes-11m-in-losses-for-venus-protocol
- https://cointelegraph.com/news/defi-protocols-declare-losses-as-attackers-exploit-luna-price-feed-discrepancy

Recommendation

The auditing team recommends monitoring the status of Chainlink/Tellor, adding the pause/unpause function for LUSD, or adding corresponding logic to avoid this.

Alleviation

[Bytemasons - Stablecoin]: The "extra-mile" solution: read from an on-chain TWAP (for LUSD-stable pairing perhaps). If TWAP is reporting price that's higher than the oracle reported price by 10%, pause redemptions. On the other hand, if TWAP is reporting price that's lower than the oracle reported price by 10%, pause liquidations and minting.

The "extra-inch" solution: case where price goes up isn't as bad as case where price goes down (since redemption fees scales with the amount being redeemed, but borrow fee doesn't). So if we solely focus on price down case, we need to guard against minting additional LUSD.

This could perhaps be achieved by just adding a pause functionality in the mint() function of LUSD and we won't have to touch other contracts of the system.

We are currently leaning towards the latter.

The team partially resolved this issue by adding the pause/unpause function for LUSD in commit e47f8f2a5f8e60e1b04339511f90f8481ab26995, but still need to regularly monitor the oracles.

BMB-03 | SUSCEPTIBLE TO SIGNATURE MALLEABILITY

Category	Severity	Location	Status
Volatile Code	● Minor	packages/contracts/contracts/LQTY/LQTYToken.sol: 257; packages/contracts/contracts/LUSDToken.sol: 189	● Resolved

Description

The signature malleability is possible within the Elliptic Curve cryptographic system. An Elliptic Curve is symmetric on the X-axis, meaning two points can exist with the same `x` value. In the `r`, `s` and `v` representation this permits us to carefully adjust `s` to produce a second valid signature for the same `r`, thus breaking the assumption that a signature cannot be replayed in what is known as a replay-attack.

Recommendation

To fix this we would recommend adding the check from EIP-2, point 2 (https://eips.ethereum.org/EIPS/eip-2), and also check for the v value to ensure the off-chain library is properly used. For example, the `ecrecoverFromSig` function from SWC-117 (https://swcregistry.io/docs/SWC-117).

OpenZeppelin's ECDSA library contract contains proper implementation for recovering the address from the signature that is not prone to signature malleability. We suggest importing that and using it in the contract.

Alleviation

[Bytemasons - Stablecoin]: The LQTYToken will not be used by us. However, we will make the recommended change (using OZ's ECDSA library) within the LUSD token code since we will be using that to deploy the stablecoin contract. We will fix this in the future.

The team partially resolved this issue by adding the check in the `LUSDToken.sol` in commit e47f8f2a5f8e60e1b04339511f90f8481ab26995.

BMB-04 | ECONOMIC MODEL - GAS COMPENSATION

Category	Severity	Location	Status
Logical Issue	● Minor	packages/contracts/contracts/BorrowerOperations.sol: 194; packages/contracts/contracts/Dependencies/LiquityBase.sol: 28, 47~49; packages/contracts/contracts/TroveManager.sol: 552	● Acknowledged

Description

The protocol directly compensates liquidators for their gas costs to incentivize prompt liquidations in both normal and extreme periods of high gas prices.

The gas compensation formula is shown below: Gas compensation = 200 LUSD + 0.5% of trove's collateral

To ensure that larger Troves are liquidated promptly even in extreme high gas price periods. The larger the Trove, the stronger the incentive to liquidate it.

200 LUSD of gas compensation makes sense for the gas cost on Ethereum, but not necessarily for other chains, like the Fantom chain, because the prices of different chains' native tokens are different. The price of ETH is 1519 USD, and the price of FTM is 0.27 USD at the time of writing.

Unreasonable gas compensation is also bad for the capital usage of borrowers, and will also expose the chances to the liquidators to arbitrage on the gas compensation of troves, which are under-collateralized.

Here is a detailed blog published on Nov 2022, comparing gas costs between Ethereum and Fantom:
Comparing fees on Fantom/Ethereum

Here are gas trackers for the two chains:
Ethereum Gas Tracker
Fantom Gas Tracker

Recommendation

Consider setting a reasonable gas compensation for the target chain to increase the capital utilization and avoid the arbitrage on the gas compensation of the liquidated troves.

Alleviation

[Bytemasons - Stablecoin]: Chain is TBD. However if we do launch on a chain with cheap TX/gas, we will definitely be tweaking the gas compensation parameter.

SPB-01 | EVENT IS NOT EMITTED

Category	Severity	Location	Status
Compiler Error	● Minor	packages/contracts/contracts/StabilityPool.sol: 498, 608	● Resolved

Description

The `StabilityPool` contract contains an event `StabilityPoolCollateralBalanceUpdated` that is used in the functions `updateRewardSum()` and `_moveOffsetCollAndDebt()` .

```
 498          StabilityPoolCollateralBalanceUpdated(_collateral,
collAmounts[_collateral]);
```

However, this event is not emitted using the `emit` keyword, causing a compiler error.

Recommendation

We recommend emitting the event by using the `emit` keyword.

Alleviation

[Bytemason - Stablecoin]: The team resolved this issue by adding the `emit` keyword in commit 1029e4fb0e5b22248874c6ef229ae6bc12e2371f.

GLOBAL-02 | WHITELIST FOR THE COLLATERAL TOKENS

Category	Severity	Location	Status
Logical Issue	● Informational		● Partially Resolved

Description

The Bytemasons - Stablecoin protocol support ERC20-compatible collateral type addresses. To reduce potential risk, the collateral candidates should be strictly selected and obtain the community's consensus.

The following advice is provided to select collateral candidates:

- Deflationary tokens should not be used as collateral as it is not supported by this protocol.
- Tokens that could be arbitrarily minted by a centralized project owner are not suitable for candidacy.
- Tokens related to an algorithm, like LUNC/USTC, should be deeply considered before allowing them to be used as collateral.

Recommendation

Consider strictly selecting the collateral tokens to avoid potential risks and acquire the community's consensus on these tokens.

Alleviation

[Bytemasons - Stablecoin]: We have thought long and hard about the collateral and have decided to use no more than 2-3 highly liquid and decentralized ERC20 tokens that play well with the system like WETH and WBTC.

GLOBAL-03 | GOVERNANCE MODEL - NEW FEATURE OF PAUSE

Category	Severity	Location	Status
Logical Issue	● Informational		● Resolved

Description

The protocol specifies the allowed collateral during initialization and can not modify them afterward. The strategy of initializing the allowed collateral makes the protocol more decentralized.

However, the potential risk still comes with the allowed collateral. For example, are the collateral tokens good capital? Would any hackers compromise them?

Considering the above worries, a pause feature would be useful to protect this protocol from losing their capital and propagating the risk in extreme cases. An example strategy is pausing the functions of opening a trove and adding collateral, but still allowing functions to redeeming LUSD and closing troves, to protect the LUSD from being maliciously minted by a compromised collateral.

However, we need to keep in mind that the pause action should gain the consensus of the community and be under multiple people's control, requiring a governance strategy, e.g., a multi-signature should at least should be applied for it.

Recommendation

Consider adding the pause feature on specified functions to protect the protocol and not propagate any risk. At the same, apply an appropriate governance strategy on the pause feature.

Alleviation

[Bytemasons - Stablecoin]: The team resolved this issue by the pause/unpause function for LUSD, which is controlled by the governance contract in commit e47f8f2a5f8e60e1b04339511f90f8481ab26995.

BMB-05 | THE TELLOR PROTOCOL DOES NOT SUPPORT THE FANTOM CHAIN FOR NOW

Category	Severity	Location	Status
Volatile Code	● Informational	packages/contracts/contracts/Dependencies/TellorCaller.sol: 18; packages/contracts/contracts/PriceFeed.sol: 214	● Acknowledged

Description

The most current Collateral:USD price is important to the Bytemasons - Stablecoin protocol to decide which strategy to be adopted. Thus, two price oracles are applied in the protocol to ensure the current Collateral:USD price is available. The PriceFeed contract provides the protocol for the Collateral:USD price and will use the latest price from Tellor when the Chainlink Oracle is down.

Tellor is a decentralized oracle protocol that incentivizes an open, permissionless network of data reporting and data validation, ensuring that data can be provided by anyone and checked by everyone.

However, the Bytemasons - Stablecoin protocol intends to be deployed on the Fantom chain and Tellor does not support the Fantom chain for now per their official documentation contracts-reference. As a result, the functionality of the `TellorCaller` within `PriceFeed` will fail.

Recommendation

Consider applying an available price oracle on the target chain.

Alleviation

[Bytemasons - Stablecoin]: Chain is still TBD. However, if we decide to launch on Fantom, we are in contact with the Tellor team to have a deployment ready for us should we decide to.

BMB-06 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	● Informational	packages/contracts/contracts/ActivePool.sol: 77, 84, 96, 101, 113, 118, 127; packages/contracts/contracts/PriceFeed.sol: 97, 98, 99, 111; packages/contracts/contracts/TroveManager.sol: 260, 535, 659, 679, 1029, 1436, 1509, 1513, 1517, 1521, 1525, 1529, 1534, 1538	● Partially Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

[Bytemasons - Stablecoin]: Error strings were removed from TroveManager due to contract size issues. However, for the rest of the entries here (ActivePool and PriceFeed), we can go back and add appropriate error messages. We will fix them in the future.

The team partially resolved this issue by adding part of the missing error messages in commit e47f8f2a5f8e60e1b04339511f90f8481ab26995.

BMB-07 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	● Informational	packages/contracts/contracts/ActivePool.sol: 112, 117, 122 , 126, 189; packages/contracts/contracts/PriceFeed.sol: 86	● Partially Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[Bytemasons - Stablecoin]: The team partially resolved this issue by adding part of the missing events in commit e47f8f2a5f8e60e1b04339511f90f8481ab26995.

BMB-08 | ECONOMIC MODEL - MINIMUM NET DEBT

Category	Severity	Location	Status
Logical Issue	● Informational	packages/contracts/contracts/BorrowerOperations.sol: 191; packages/contracts/contracts/Dependencies/LiquityBase.sol: 31	● Acknowledged

Description

Note that the minimum net debt is 1800 LUSD, which makes sense if the protocol is deployed on the Ethereum chain since the gas fee and gas compensation are high there. However, the gas fee is cheaper for the Fantom chain.

To lower the threshold for allowing people to join this protocol, reducing the value for the `MIN_NET_DEBT` on the Fantom chain could make more sense.

Recommendation

Consider setting a reasonable minimum net debt for the target chain.

Alleviation

[Bytemasons - Stablecoin]: Chain is TBD. However if we do launch on a chain with cheap TX/gas, we will definitely be tweaking the gas compensation parameter.

BMB-09 | THE SAME MCR AND CCR FOR ALL THE ALLOWED COLLATERALS

Category	Severity	Location	Status
Logical Issue	● Informational	packages/contracts/contracts/Dependencies/LiquityBase.sol: 22~25 ; packages/contracts/contracts/TroveManager.sol: 201, 203, 732, 784	● Resolved

Description

Borrowers can open a separate trove for each collateral type they would like to mint stablecoins against. This allows the protocol to isolate the various markets.

However, we notice that the two fundamental factors, MCR and CCR, are consistent across all allowed collaterals. These are widely used in opening/closing/adjusting a Trove, liquidations, and so on.

The value of the allowed collaterals might vary widely due to the differences in their liquidity, stability, community, and other factors. Thus, the two fixed values of MCR and CCR probably cannot reflect and be consistent with the actual value of the allowed ERC20 tokens.

Please check if the current design meets the business requirements and check whether specifying different MCRs and CCRs for different collaterals are more suitable.

It is also worth noting that the MCR reflects the borrowing power for the specified collateral, is the liquidation threshold, and determines the rate of collateral being liquidated when the trove is under-collateralized. Both the MCR and CCR are protocol-level parameters, and therefore should have deep consideration when trying to adjust them.

Recommendation

Recommend checking whether different MCRs and CCRs are required for each collateral and doing the adjust if needed.

Alleviation

[Bytemason - Stablecoin]: The team resolved this issue by querying MCR and CCR from a config contract for each collateral in commit 1029e4fb0e5b22248874c6ef229ae6bc12e2371f.

BOB-01 | MISSING A CHECK FOR COLLATERAL ON `adjustTrove()`

Category	Severity	Location	Status
Inconsistency	● Informational	packages/contracts/contracts/BorrowerOperations.sol: 259	● Resolved

Description

It is noticed that the function `adjustTrove()` is very flexible and is a combination of `addColl()` , `withdrawColl()` , `withdrawLUSD()` and `repayLUSD()` .

However, when `_collTopUp` is greater than 0, the behavior of this function is similar to the function `addColl()` , which requires a check for collateral by calling `_requireSufficientCollateralBalanceAndAllowance()` before actually adjusting troves.

Thus, in this case, `adjustTrove()` misses a check for the user's collateral balance.

Recommendation

Consider adding the missing check for the user's collateral balance in the `adjustTrove()` function when the argument `_collTopUp` greater than zero.

Alleviation

[Bytemason - Stablecoin]: The team resolved this issue by adding the check for the user's collateral balance, in commit 1029e4fb0e5b22248874c6ef229ae6bc12e2371f.

OPTIMIZATIONS | BYTEMASONS - STABLECOIN

ID	Title	Category	Severity	Status
BMB-10	Useless Statement And Variables	Gas Optimization	Optimization	● Partially Resolved
BMB-11	Redundant Code Components	Volatile Code	Optimization	● Resolved

BMB-10 | USELESS STATEMENT AND VARIABLES

Category	Severity	Location	Status
Gas Optimization	● Optimization	packages/contracts/contracts/BorrowerOperations.sol: 184; packages/contracts/contracts/PriceFeed.sol: 33~34; packages/contracts/contracts/StabilityPool.sol: 341, 385	● Partially Resolved

Description

In the contract `BorrowerOperations.sol`, the linked statement does nothing.

```
184         vars.LUSDFee;
```

The state variables `borrowerOperationsAddress` and `troveManagerAddress`, in the `PriceFeed.sol` contract, are never used.

The variable `LUSDLoss`, in the `StabilityPool.sol` contract, is declared and assigned but never used.

Recommendation

Consider removing the useless statement and variables.

Alleviation

[Bytemasons - Stablecoin]: The team partially resolved this issue by removing a redundant statement and state variables, in the commit 1029e4fb0e5b22248874c6ef229ae6bc12e2371f.

BMB-11 | REDUNDANT CODE COMPONENTS

Category	Severity	Location	Status
Volatile Code	● Optimization	packages/contracts/contracts/BorrowerOperations.sol: 506; packages/contracts/contracts/StabilityPool.sol: 869	● Resolved

Description

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation

We advise to remove the redundant statements for production environments.

Alleviation

[Bytemasons - Stablecoin]: The team resolved this issue by removing the redundant functions in commit 1029e4fb0e5b22248874c6ef229ae6bc12e2371f.

FORMAL VERIFICATION | BYTEMASONS - STABLECOIN

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

Considered Functions And Scope

Verification of ERC-20 compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply` , which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-revert-zero	Function `transfer` Prevents Transfers to the Zero Address
erc20-transfer-succeed-normal	Function `transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-correct-amount	Function `transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-succeed-self	Function `transfer` Succeeds on Admissible Self Transfers
erc20-transfer-correct-amount-self	Function `transfer` Transfers the Correct Amount in Self Transfers
erc20-transfer-change-state	Function `transfer` Has No Unexpected State Changes
erc20-transfer-exceed-balance	Function `transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-recipient-overflow	Function `transfer` Prevents Overflows in the Recipient's Balance
erc20-transfer-false	If Function `transfer` Returns `false` , the Contract State Has Not Been Changed
erc20-transfer-never-return-false	Function `transfer` Never Returns `false`

Property Name	Title
erc20-transferfrom-revert-from-zero	Function `transferFrom` Fails for Transfers From the Zero Address
erc20-transferfrom-revert-to-zero	Function `transferFrom` Fails for Transfers To the Zero Address
erc20-transferfrom-succeed-normal	Function `transferFrom` Succeeds on Admissible Non-self Transfers
erc20-transferfrom-correct-amount-self	Function `transferFrom` Performs Self Transfers Correctly
erc20-transferfrom-succeed-self	Function `transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-correct-amount	Function `transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-allowance	Function `transferFrom` Updated the Allowance Correctly
erc20-transferfrom-fail-exceed-balance	Function `transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-change-state	Function `transferFrom` Has No Unexpected State Changes
erc20-transferfrom-fail-exceed-allowance	Function `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-totalsupply-succeed-always	Function `totalSupply` Always Succeeds
erc20-transferfrom-fail-recipient-overflow	Function `transferFrom` Prevents Overflows in the Recipient's Balance
erc20-transferfrom-false	If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed
erc20-transferfrom-never-return-false	Function `transferFrom` Never Returns `false`
erc20-totalsupply-change-state	Function `totalSupply` Does Not Change the Contract's State
erc20-totalsupply-correct-value	Function `totalSupply` Returns the Value of the Corresponding State Variable
erc20-balanceof-succeed-always	Function `balanceOf` Always Succeeds
erc20-balanceof-correct-value	Function `balanceOf` Returns the Correct Value
erc20-allowance-succeed-always	Function `allowance` Always Succeeds
erc20-balanceof-change-state	Function `balanceOf` Does Not Change the Contract's State
erc20-allowance-correct-value	Function `allowance` Returns Correct Value

Property Name	Title
erc20-allowance-change-state	Function `allowance` Does Not Change the Contract's State
erc20-approve-revert-zero	Function `approve` Prevents Giving Approvals For the Zero Address
erc20-approve-succeed-normal	Function `approve` Succeeds for Admissible Inputs
erc20-approve-correct-amount	Function `approve` Updates the Approval Mapping Correctly
erc20-approve-change-state	Function `approve` Has No Unexpected State Changes
erc20-approve-false	If Function `approve` Returns `false`, the Contract's State Has Not Been Changed
erc20-approve-never-return-false	Function `approve` Never Returns `false`

Verification Results

For the following contracts, model checking established that each of the 38 properties that were in scope of this audit (see scope) are valid:

Contract ERC20Mock (Source File packages/contracts/contracts/LPRewards/TestContracts/ERC20Mock.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-succeed-normal	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-succeed-self	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-change-state	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-recipient-overflow	● True	
erc20-transfer-false	● True	
erc20-transfer-never-return-false	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-succeed-normal	● True	
erc20-transferfrom-correct-amount-self	● True	
erc20-transferfrom-succeed-self	● True	
erc20-transferfrom-correct-amount	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-fail-recipient-overflow	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-never-return-false	● True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-change-state	● True	
erc20-totalsupply-correct-value	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample,this occurs if
    - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".

  - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a correspond finding is reported separately in the Findings section of this report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.

- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
  - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.
  - The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Contract LUSDToken (Source File packages/contracts/contracts/LUSDToken.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-change-state	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-false	● True	
erc20-transfer-recipient-overflow	● True	
erc20-transfer-never-return-false	● True	
erc20-transfer-succeed-normal	● Inapplicable	
erc20-transfer-succeed-self	● Inapplicable	Intended behavior

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-correct-amount	● True	
erc20-transferfrom-correct-amount-self	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-fail-recipient-overflow	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-never-return-false	● True	
erc20-transferfrom-succeed-normal	● Inapplicable	Intended behavior
erc20-transferfrom-succeed-self	● Inapplicable	Intended behavior

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

APPENDIX | BYTEMASONS - STABLECOIN

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
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.
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.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
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.
Compiler Error	Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

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 | CERTIK

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.

CertiK | Securing the Web3 World

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.