CERTIK

Security Assessment

Trustline

CertiK Verified on Dec 16th, 2022

CertiK Verified on Dec 16th, 2022

Trustline

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
Lending	Ethereum	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 12/16/2022	N/A

CODEBASE
https://github.com/trustline-inc/probity/tree/ebacbe8f8d812eecbefa938da99bc9c4b2fb0dc4
...View All

COMMITS
ebacbe8f8d812eecbefa938da99bc9c4b2fb0dc4
...View All

Vulnerability Summary

16	9	0	1	6	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.
2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
7	Minor	3 Resolved, 1 Partially Resolved, 3 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.
6	Informational	4 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 | TRUSTLINE

CODEBASE | TRUSTLINE

Repository

https://github.com/trustline-inc/probity/tree/ebacbe8f8d812eecbefa938da99bc9c4b2fb0dc4

Commit

ebacbe8f8d812eecbefa938da99bc9c4b2fb0dc4

AUDIT SCOPE | TRUSTLINE

25 files audited ● 7 files with Acknowledged findings ● 1 file with Partially Resolved findings
● 6 files with Resolved findings ● 11 files without findings

ID	File	SHA256 Checksum
ERC	contracts/probity/assets/ERC20AssetManager.sol	ccef722deadbbc1a9460a3c22d6b45edc76eeadaa01c09314c372ab99fb9077b
VPA	contracts/probity/assets/VPAssetManager.sol	30698a8b5976a81a710bda568b62364e5defae5f4cc74a2ed896cd4a6b81876a
LIQ	contracts/probity/Liquidator.sol	5ecc501c61fbdd988c075e0ec64ab66c659c05beb81933ef640d0f1b092e4f97
PFB	contracts/probity/PriceFeed.sol	df333fa4fa8749178b3b12f77ae3391545753ab6ae7f5b294bdb86f633b8502e
RPB	contracts/probity/ReservePool.sol	1a80f3221908dae4fbd579f70c150902ed43dcceaf7741cb6490022269d56e3c
VEB	contracts/probity/VaultEngine.sol	795225211bc504fb87468c535d7e0988bc8b61220746da782427c2a262bc9814
DEL	contracts/dependencies/Delegatable.sol	826fa400f5e74d4e7bc52db40ab550a3440578ab8344f951def247b1565324c4
BIB	contracts/probity/BondIssuer.sol	4350dee8ac80eb19bff83e5c96c0f3c6f04bc114912bb6a208b94502ad1eca6a
NAM	contracts/probity/assets/NativeAssetManager.sol	cb8fb1aefd1d8952191da8ff36ffab397a4423a2cdef513bf7fd868dd1ab8556
LDF	contracts/probity/priceFunction/LinearDecrease.sol	2771043fe43ee7f08697db9e09176e29032dc856be4a2f2280fac283bdb3114c
AUC	contracts/probity/Auctioneer.sol	9476e0a55fdd5c7f379ea898a4b5851af80ece6cda2fa7336f374d0d51f5a270
TEL	contracts/probity/Teller.sol	525b7bac2ca0ece7db8cf91d2bfa70738349af66f9f8eb9ee30bced4af6e307a
TRE	contracts/probity/Treasury.sol	0d390080ed9a265c2ff6326a33446eb69d00a93a970940fa907b9783c6c5765a

ID	File	SHA256 Checksum
ACB	contracts/dependencies/AccessControl.sol	bec5b47ca81d26fd70b1bc92e5f3d83908abbfbad54d6fccbb13f3c9529a5058
PBT	contracts/probity/tokens/Pbt.sol	f10436c8f895614f6eed069fd2ac2b98f2d48a87099cb35b9d8d0728cfc1c474
USD	contracts/probity/tokens/Usd.sol	09fca9f492257b85ecd300c90e34c87f3e7b343891984f6879c41abea94926bd
REG	contracts/probity/Registry.sol	cbb27b2524aab8b6f6337e6bcc4a40ed448aaa46ffe70245d0823ef05e823029
VEI	contracts/probity/VaultEngineIssuer.sol	e52eaa248a34cfa59f194a151fd48dc1e0e2d98273993cb1cf7b26e7de031c07
VEL	contracts/probity/VaultEngineLimited.sol	e515cf920e04e5749c133fe87277ad0f19c5741e97ecdda76b752cf4274e514c
VER	contracts/probity/VaultEngineRestricted.sol	02db0983bda70fe1c10ca662052445aaddf421daa735f1dcd1d98b7e1cd8206f
EVE	contracts/dependencies/Eventful.sol	3dfca0b279561b7716e0bc4ff7d27b8b9c56f67d2d5908c6c2f87f1d48363b66
HAP	contracts/dependencies/HighAPR.sol	42450767aedf456a8fd59926245eaef6d8135219ff9e3e06669d1679e0fda6d3
LAP	contracts/dependencies/LowAPR.sol	ce5051dd53cdf2385ffb558012db1814ec15b637bc36e707996d18fe28824cf3
MAT	contracts/dependencies/Math.sol	f5688b42f2db361818b1aaf440217231ea339e5b375a36b76022e858f18846b3
STA	contracts/dependencies/Stateful.sol	74a5990f7bc18d5c99b89a98fbf4abe157e4f34554f2f194d6999d3a6734c35f

APPROACH & METHODS | TRUSTLINE

This report has been prepared for Trustline to discover issues and vulnerabilities in the source code of the Trustline 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 | TRUSTLINE

The Trustline protocol is a financial system that allows for the extension of credit to borrowers.

Financial Models

From the user's aspect, there are two ways to participate in project investments.

1. Deposit the underlying tokens and get the governance token `PBT` and the `systemCurrency/USD` token as the interest.
2. Deposit the collateral tokens to borrow the `systemCurrency/USD` token.

The `systemCurrency/USD` token is a stablecoin created by the Trustline protocol and backed by the underlying tokens. The token is designed to be circulated as a stablecoin. The supply is determined by the number of loans taken out by all users.

From the project implementation, the `systemCurrency/USD` token can be backed with underlying assets, like normal ERC20 tokens, and platform native tokens(ETH/BTC). But according to the project team's description, the `systemCurrency/USD` token is intended to be backed by US dollars. Other assets are supported if the system governance wishes. However, there is only a PBT governance token in the current scope. There is no DAO voting governance logic. The role management and system privilege are relatively centralized currently.

Bond Sale and Liquidation are two ways of debt settlement. The primary way is Liquidation. Both lender and borrower positions can be liquidated. The lender positions will be only taken a small percentage. The liquidated assets will be sold in auctions, where bidders can use `systemCurrency/USD` to buy.

Financial models of blockchain protocols need to be resilient to attacks. They need to pass simulations and verifications to guarantee the security of the overall protocol.

The financial model of this protocol is not in the scope of this audit.

Centralized Risk

The role management of the current protocol is centralized and controlled by the `govAddress`.

Also, there are privileged functions like `modifyStandbyAmount()`, `moveAsset()`, `moveSystemCurrency()` in the contract `VaultEngine` that can modify account balances directly.

In addition, the price oracle `ftso` address can be changed by the `gov` role.

FINDINGS | TRUSTLINE

16	0	1	2	7	6
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Trustline. Through this audit, we have uncovered 16 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	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Acknowledged**
ASS-01	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Acknowledged
CON-01	Logic Issue About User Rewards	Logical Issue	Medium	Resolved
CON-02	Unsafe Type Conversion	Mathematical Operations	Minor	Resolved
CON-03	Third Party Dependencies	Volatile Code	Minor	Acknowledged
LIQ-01	Incorrect Assignment	Logical Issue	Medium	Resolved
NAM-01	Usage Of `transfer` / `send` For Sending Ether	Volatile Code	Minor	Resolved
PRO-01	Lack Of Reasonable Boundary	Volatile Code	Minor	Partially Resolved
VEB-01	Input Validation Of `equityAmount` And `debtAmount`	Volatile Code	Minor	Acknowledged
VEB-02	Missing Check For `assetId`	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
GLOBAL-02	Unlocked Compiler Version	Language Specific	Informational	Resolved
CON-04	Missing Inheritance	Language Specific	Informational	Resolved
DEL-01	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Informational	Resolved
LIQ-02	Logic Issue About Liquidation	Logical Issue	Informational	Acknowledged
RPB-01	Logic Issue About `increaseSystemDebt()`	Logical Issue	Informational	Acknowledged
TEL-01	Unused Event	Coding Style	Informational	Resolved

GLOBAL-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	**Major**		**Acknowledged**

Description

In the contract `VaultEngine`, the role `Probity` has authority over the following functions:

- function modifyStandbyAmount(): change the `standbyAmount` of an account which is the deposited amount of the given account;
- function moveAsset(): increase the `standbyAmount` of the `to` address and decrease the `standbyAmount` of the `from` address, to transfer the `standbyAmount` asset from one account to another;
- function moveSystemCurrency(): transfer the `systemCurrency` token from one account to another;
- function removeSystemCurrency(): decrease the `systemCurrency` balance of an account;
- function liquidateDebtPosition(): liquidate an debt position;
- function liquidateEquityPosition():liquidate an equity position;
- function settle(): use the `systemCurrency` balance of an account to pay its debt;
- function increaseSystemDebt(): increase the `systemCurrency` balance and `systemDebt` of an account;
- function updateAdjustedPrice(): update the `assets[assetId].adjustedPrice`.

Also, the role `gov` has authority over the following functions:

- function updateTreasuryAddress(): change the address of the contract `treasury`;
- function initAsset(): change the value of `assets[assetId].category` (`UNDERLYING`, `COLLATERAL`, `BOTH`);
- function updateCeiling(): change the value of `assets[assetId].ceiling`;
- function updateFloor(): change the value of `assets[assetId].floor`.

Also, the role `treasury` has authority over the following functions:

- function addSystemCurrency(): increase the `systemCurrency` balance of an account;
- function reducePbt(): decrease the `PBT` balance of an account.

Also, the role `teller` has authority over the following functions:

- function updateAccumulators(): update the value `debtAccumulator` and `equityAccumulator` and charge the protocol fee.

In the contract `VaultEngineIssuer`, the role `gov` has authority over the following functions:

- function modifySupply(): issue system currency to an account.

In the contract `VaultEngineLimited`, the role `gov` has authority over the following functions:

- function updateIndividualVaultLimit(): update the individual vault limit.

Also, in the contract `VaultEngineLimited` and `VaultEngineRestricted`, only the `whitelisted` role can call the function `modifyEquity()` and `modifyDebt()`.

In the contract `Teller`, the role `gov` has authority over the following functions:

- function setProtocolFee(): change the value of the variable `protocolFee`;
- function setReservePoolAddress(): change the address of the contract `reservePool`.

In the contract `ReservePool`, the role `gov` has authority over the following functions:

- function updateDebtThreshold(): change the value of the variable `debtThreshold`;
- function sendSystemCurrency(): transfer the `systemCurrency` token from the reserve pool to the `to` address.

Also, the role `liquidator` has authority over the following functions:

- function addAuctionDebt(): increase the value of `debtOnAuction`;
- function reduceAuctionDebt(): decrease the value of `debtOnAuction`.

Also, the role `Probity` has authority over the following functions:

- function settle(): use the `systemCurrency` balance of an account to pay its debt;
- function increaseSystemDebt(): increase the `systemCurrency` balance and `systemDebt` of an account;
- function startBondSale(): start a new bond sale.

In the contract `Registry`, the role `gov` has authority over the following functions:

- function setupAddress(): set an address to be a role in the system and make it has the `Probity` role;
- function removeAddress(): remove all the roles of an address.

In the contract `PriceFeed`, the role `gov` has authority over the following functions:

- function initAsset(): initialize a new asset and set the liquidation ratio and ftso addresss;
- function updateLiquidationRatio(): update the liquidation ratio of an asset;
- function updateFtso(): update the `FTSO` (price oracle) address of an asset.

In the contract `Liquidator`, the role `gov` has authority over the following functions:

- function initAsset(): initialize a new asset;

- function updatePenalties(): update the `debtPenaltyFee` and `equityPenaltyFee` of an asset;
- function updateAuctioneer(): update the `auctioneer` address of an asset.

Also, the role `auctioneer` has authority over the following functions:

- function reduceAuctionDebt(): decrease the value of `debtOnAuction`.

In the contract `Auctioneer`, the role `liquidator` has authority over the following functions:

- function startAuction(): start an asset auction.

Also, the role `Probity` has authority over the following functions:

- function cancelAuction(): cancel an auction.

In the contract `BondIssuer`, the role `gov` has authority over the following functions:

- function setReservePoolAddress(): change the address of the contract `reservePool`;
- function updateMaxDiscount(): update the maximum discount for an issue;
- function updateStepPeriod(): update the step period;
- function updateDiscountIncreasePerStep(): update the discount increase per step.

Also, the role `reservePool` has authority over the following functions:

- function newOffering(): start a new bond sale.

Also, the role `Probity` has authority over the following functions:

- function redeemBondTokensForUser(): redeem bondTokens for the `systemCurrency` token on behalf of users.

In the contract `AccessControl`, the role `gov` has authority over the following functions:

- function setRegistryAddress(): change the address of the contract `registry`.

In the contract `Delegatable`, the role `gov` has authority over the following functions:

- function changeDataProviders(): change the data providers by delegating a certain percentage.

Also, the role `auctioneer` has authority over the following functions:

- function collectRewardForUser(): allow the contract `auctioneer` to collect reward on behalf of users based on their locked-up token value.

In the contract `Stateful`, the role `gov` has authority over the following functions:

- function setState(): change the value of `states[name]`.

Any compromise to the privileged accounts may allow a hacker to take advantage of this authority and users' assets may suffer loss.

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 recommend carefully managing 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.

Alleviation

The team acknowledged this issue and they stated the following:

"They expect to use a smart contract-based account using pre-existing audited, open-source code, if and when deploying Probity. For example, they have already conducted minor testing with a Gnosis Safe (multi-sig) wallet contract. In the long run, they envision that any privileged accounts will use hardware security modules and approval workflows."

ASS-01 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/probity/assets/ERC20AssetManager.sol; contracts/probity/assets/VPAssetManager.sol	Acknowledged

Description

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrived at the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Reference: https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

The team acknowledged this issue and they will leave it as it is for now.

CON-01 | LOGIC ISSUE ABOUT USER REWARDS

Category	Severity	Location	Status
Logical Issue	Medium	contracts/dependencies/Delegatable.sol: 217~244; contracts/probity/assets/VPAssetManager.sol: 68~75	Resolved

Description

Based on the function `_userCollectReward()` of contract `Delegatable`, the `rewardableBalance` is the user balance in the current epoch. It is calculated by

```
currentBalance - recentTotalDeposit[user]
```

, which means `current balance - future changes`.

Also, the `recentTotalDeposit[user]` variable will then be updated by

```
recentTotalDeposit[user] -= recentDeposits[user][epochId]
```

to track the future user balance in and after the current epoch. This makes sense.

However, both of the variables `recentTotalDeposit[user]` and `recentDeposits[user][epochId]` are uint256, so they cannot be negative. Also, in the function `withdraw()` of contract `VPAssetManager`, the withdraw action only decreases the variables `recentTotalDeposit[user]` and `recentDeposits[user][epochId]`, when the current value of `recentDeposits[user][epochId]` is positive. As a result, these two values cannot track the actual user balance changes. Some of the user rewards may be missed out.

For example:

Let's assume that a user deposits 200 in epoch one, withdraws 100 in epoch two, and then claims rewards in epoch three.

- In epoch one, the variables recentTotalDeposit[user] and recentDeposits[user][epochId] are both updated to 200 by the deposit() function.
- But in epoch two, the variables recentTotalDeposit[user] and recentDeposits[user][epochId] have not been updated(recentTotalDeposit[user] is 200, recentDeposits[user][p2] is 0), because the current value of recentDeposits[msg.sender][currentRewardEpoch] is 0.
- Thus, when the user claims the rewards in epoch three, the current balance is 100 and the rewardableBalance is like the following:

epochId	`recentDeposits[user][epochId]`	`recentTotalDeposit[user]`	`rewardableBalance`

epochId	`recentDeposits[user][epochId]`	`recentTotalDeposit[user]`	`rewardableBalance`
1	200	200	0
2	0	0 (200 - 200)	100
3	0	0	100

- From the result sheet, we can see that the `rewardableBalance` for epoch two is 100, but it actually should be 200.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design. We recommend fixing the issue by using int256 for the variables `recentTotalDeposit[user]` and `recentDeposits[user][epochId]` and tracking the amount decrease value of the withdraw action.

The reasonable result should be:

epochId	`recentDeposits[user][epochId]`	`recentTotalDeposit[user]`	`rewardableBalance`
1	200	100	0
2	-100	-100 (100 - 200)	200
3	0	0	100

Alleviation

The team heeded our advice and resolved this issue in commit `c663d3adce107388722d9321b071d38feb3bff27`.

CON-02 | UNSAFE TYPE CONVERSION

Category	Severity	Location	Status
Mathematical Operations	Minor	contracts/dependencies/Delegatable.sol: 152; contracts/probity/Liquidator.sol: 244~246, 289~292; contracts/probity/assets/ERC20AssetManager.sol: 64, 70; contracts/probity/assets/NativeAssetManager.sol: 51, 56; contracts/probity/assets/VPAssetManager.sol: 51, 55, 63, 66	Resolved

Description

The linked statements cast between type `uint256` and `int256` without evaluating the bounds. If the most significant bit is 1, the casting result will be deducted by the Two's complement which may be not a correct result.

Recommendation

We recommend implementing a safe casting function to ensure the result is still positive. Alternatively, ensure the casting number is not too large and will not cause underflow/overflow.

Alleviation

The team heeded our advice and resolved this issue in commit `84b400486796229a378ecb02ceed5e96a23df16d`.

CON-03 | THIRD PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	Minor	contracts/dependencies/Delegatable.sol; contracts/probity/PriceFeed.sol; contracts/probity/assets/VPAssetManager.sol	Acknowledged

Description

The protocol is serving as the underlying entity to interact with 3rd party contracts such as `Ftso`, `FtsoRewardManager`, and `FtsoManager`. 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. Additionally, 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 the contract `Delegatable` and `PriceFeed` requires interaction with the `FTSO` system, etc. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

The team acknowledged this issue and they will leave it as it is for now.

LIQ-01 | INCORRECT ASSIGNMENT

Category	Severity	Location	Status
Logical Issue	Medium	contracts/probity/Liquidator.sol: 199	Resolved

Description

The variable `adjustedPrice` is assigned to `vaultEngine.assets(assetId)` whose type is a struct `VaultEngine.Asset` not uint256. `adjustedPrice` is one of the attributes of the struct `VaultEngine.Asset`.

Recommendation

We recommend fixing the assignment statement.

For example:

```
uint256 adjustedPrice = vaultEngine.assets(assetId).adjustedPrice;
```

Alleviation

The team heeded our advice and resolved this issue in commit `7c24595953eacb83b48ebaaab4188a12a9113cb2`.

NAM-01 | USAGE OF `transfer` / `send` FOR SENDING ETHER

Category	Severity	Location	Status
Volatile Code	Minor	contracts/probity/assets/NativeAssetManager.sol: 57	Resolved

Description

It is not recommended to use Solidity's `transfer()` and `send()` functions for transferring Ether, since some contracts may not be able to receive the funds. Those functions forward only a fixed amount of gas (2300 specifically) and the receiving contracts may run out of gas before finishing the transfer. Also, EVM instructions' gas costs may increase in the future. Thus, some contracts that can receive now may stop working in the future due to the gas limitation.

```
57        if (!payable(msg.sender).send(amount)) revert transferFailed();
```

- `NativeAssetManager.withdraw` uses `send()`.

Recommendation

We recommend using the `Address.sendValue()` function from OpenZeppelin.

Since `Address.sendValue()` may allow reentrancy, we also recommend guarding against reentrancy attacks by utilizing the Checks-Effects-Interactions Pattern or applying OpenZeppelin ReentrancyGuard.

Alleviation

The team heeded our advice and resolved this issue in commit `640eb3d392babcd81629d14ea82cd763a8375d5f`.

PRO-01 | LACK OF REASONABLE BOUNDARY

Category	Severity	Location	Status
Volatile Code	Minor	contracts/probity/BondIssuer.sol: 110, 119, 128, 140; contracts/probity/Liquidator.sol: 164~165; contracts/probity/PriceFeed.sol: 77, 89	Partially Resolved

Description

The variables such as `debtPenalty`, `equityPenalty`, `newMaxDiscount`, `stepPeriod`, `discountIncreasePerStep`, `liquidationRatio`, `offering.amount` do not have reasonable boundaries, so they can be given arbitrary values after deploying.

Recommendation

We recommend adding reasonable upper and lower boundaries to all the configuration variables.

Alleviation

The team heeded our advice and partially resolved this issue in commit `e82a829ae01d3f49e1e0ef2ad3ea47723cdec554`. Almost all the variables are added boundary checks except the variable `offering.amount`.

VEB-01 | INPUT VALIDATION OF `equityAmount` AND `debtAmount`

Category	Severity	Location	Status
Volatile Code	Minor	contracts/probity/VaultEngine.sol: 459, 511	Acknowledged

Description

In the functions `_modifyEquity()` and `_modifyDebt()`, the input parameters `equityAmount` and `debtAmount` are passed independently with `underlyingAmount` and `collAmount`. There are functions `_certifyEquityPosition()` and `_certifyDebtPosition()` to ensure their upper bound, but there is no lower bound check.

Recommendation

We recommend adding lower bound checks for these two parameters or just using the `assets[assetId].adjustedPrice` and `underlyingAmount/collAmount` to calculate their values.

Alleviation

The team acknowledged this issue and they stated the following:

"They do not have a minimum amount so that the user can specify whatever amount they desire, and their UI helps to ensure that the amount is entered correctly so that they do not make a mistake."

VEB-02 | MISSING CHECK FOR `assetId`

Category	Severity	Location	Status
Volatile Code	Minor	contracts/probity/VaultEngine.sol	Resolved

Description

The function `_modifyEquity()` requires the corresponding asset is not `COLLATERAL`. Also, the function `_modifyDebt()` requires the corresponding asset is not `UNDERLYING`. Both of them do not check whether the asset has been initialized by the function `initAsset()`. If an asset is not initialized, the default value of the `assets[assetId].category` is `UNDERLYING`, which may be not the desired value.

Recommendation

We recommend adding a new Category type `UNKNOWN` and making it to be the default enum value. Then check whether the `assets[assetId].category` has been initialized by comparing it with `Category.UNKNOWN`. `UNKNOWN` is just an example and you can choose any name you want.

Alleviation

The team heeded our advice and resolved this issue in commit `9c7d7371dbb8448591da42d425fe875c89509d54`.

GLOBAL-02 | UNLOCKED COMPILER VERSION

Category	Severity	Location	Status
Language Specific	Informational		Resolved

Description

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

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.8.0` the contract should contain the following line:

```
pragma solidity 0.8.0;
```

Alleviation

The team heeded our advice and resolved this issue in commit `91de89ff00e81b2b82909a42b20eb9ee9f4c9abf`.

CON-04 | MISSING INHERITANCE

Category	Severity	Location	Status
Language Specific	Informational	contracts/dependencies/AccessControl.sol: 4; contracts/dependencies/Delegatable.sol: 7, 17, 21; contracts/probity/Auctioneer.sol: 9, 24, 28, 32, 36; contracts/probity/BondIssuer.sol: 8; contracts/probity/Liquidator.sol: 50, 62; contracts/probity/PriceFeed.sol: 9, 13; contracts/probity/ReservePool.sol: 8, 24; contracts/probity/Treasury.sol: 7, 21; contracts/probity/priceFunction/LinearDecrease.sol: 7	Resolved

Description

The linked interfaces are used to refer to other contracts in the current project or third-party protocols. However, the interfaces are not inherited by their implementation contracts directly. For example, the contract `VaultEngine` does not inherit from the interface `VaultEngineLike`.

Recommendation

We recommend inheriting from the missing interface or contract. Also, ensure that the contract deployments are correct because the external calls may have potential reentrancy issues.

Alleviation

The team heeded our advice and resolved this issue in commit `c2298a3d8e5a6c62ade3a1158a512c145cdf3d66`.

DEL-01 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Informational	contracts/dependencies/Delegatable.sol: 243	Resolved

Description

The return value of the transfer()/transferFrom() call is not checked.

```
243         token.transfer(user, rewardBalance);
```

Recommendation

Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We advise using the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC-20 token implementations.

Alleviation

The team heeded our advice and resolved this issue in commit `fb01017158e51dfbbb709bcf7182156f452f590d`.

LIQ-02 | LOGIC ISSUE ABOUT LIQUIDATION

Category	Severity	Location	Status
Logical Issue	Informational	contracts/probity/Liquidator.sol	Acknowledged

Description

Bond Sale and Liquidation are two ways of debt settlement. The primary way is Liquidation, but the `Liquidator.liquidateVault()` function can be called by anyone. Also, the underlying/collateral tokens will be immediately liquidated if their value is not sufficient for the equity/debt position. Usually, the borrower's collateral assets are liquidated when the loan is overdue. We understand that the collateral tokens can be sold to liquidate the debt, but why do the underlying tokens need to be sold or auctioned?

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

The team acknowledged this issue and they stated the following:

"This was an intentional design decision. Linqto/Trustline Inc. will run a background process to liquidate vaults.

Investor/lender's equity positions can be liquidated if their equity position underlying assets are not meeting the underlying ratio. E.g., if the USD stablecoin is backed by ETH, they would have to maintain a 150% underlying ratio for example. They do not intend to back the stablecoin by ETH, only by USD, but it is still technically possible. Therefore, both lender and borrower positions can be liquidated.

In the original design, any asset can be used as an underlying asset to create stablecoin, which includes volatile assets such as BTC or ETH. The underlying asset's price can drop below the minimum ratio required. To discourage the equity position holders from allowing this to happen, they are penalizing them by taking away a small percentage (5% of the position in the current implementation) of the underlying asset and refunding the rest while simultaneously closing out the position. The asset seized as the penalty is then sold in auction with the reserve pool becoming beneficiary."

RPB-01 | LOGIC ISSUE ABOUT `increaseSystemDebt()`

Category	Severity	Location	Status
Logical Issue	Informational	contracts/probity/ReservePool.sol: 123	Acknowledged

Description

The amount of Bond Sale is determined by the system debt and liquidated debt amount. The system debt can be modified via the `increaseSystemDebt()` function. This function is marked as `onlyByProbity`. Also, the increase of the system debt can block the redeems of the `bondTokens` in the contract `BondIssuer`.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

The team acknowledged this issue and they stated the following:

"The idea behind `increaseSystemDebt()` is to allow the Probity system to solve potential cash flow issues in cases where the system doesn't have enough funds to operate as normal. It injects cash into the system by taking on additional system debt. In the current version of the codebase, the caller will only be by "gov" but in the next versions where there are incentivized actions within Probity, different modules will be able to call `increaseSystemDebt()` if the `reservePool` doesn't hold enough cash for the incentivized action."

TEL-01 | UNUSED EVENT

Category	Severity	Location	Status
Coding Style	Informational	contracts/probity/Teller.sol: 61, 62	Resolved

Description

```
61      event AssetInitialized(bytes32 indexed assetId, uint256 protocolFee);
```

- `AssetInitialized` is declared in `Teller` but never emitted.

```
62      event RatesUpdated(uint256 timestamp, uint256 debtAccumulator, uint256
equityAccumulator);
```

- `RatesUpdated` is declared in `Teller` but never emitted.

Recommendation

We advise removing the unused events or emitting them in the intended functions.

Alleviation

The team heeded our advice and resolved this issue in commit `90ff2e67339fc45dcbc4f5ab4b76894d0332ef8d`.

OPTIMIZATIONS | TRUSTLINE

ID	Title	Category	Severity	Status
VPA-01	Avoid Unnecessary External Call	Gas Optimization	Optimization	Resolved

VPA-01 | AVOID UNNECESSARY EXTERNAL CALL

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/probity/assets/VPAssetManager.sol: 52~53, 65~75	Resolved

Description

Since the result of `ftsoManager.getCurrentRewardEpoch()` has already been assigned to the variable `currentRewardEpoch`, further reference can use the variable `currentRewardEpoch` directly instead of performing the external call.

Recommendation

We recommend using the variable `currentRewardEpoch` instead of calling `ftsoManager.getCurrentRewardEpoch()` after the assignment to save gas.

Alleviation

The team heeded our advice and resolved this issue in commit `23066e0a15df03a6105ed008661b692cb9af69bb`.

APPENDIX | TRUSTLINE

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.
Mathematical Operations	Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

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.