PRISMA FINANCE SECURITY AUDIT REPORT

September 1, 2023

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

PRISMA is the governance token of the Prisma protocol. It is distributed to users of the protocol to perform a variety of actions, and can be locked to participate in protocol governance.

mkUSD is a collateralized stablecoin. A restricted list of highly liquid ERC20 tokens can be used as collateral.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Prisma Finance
Project name	mkUSD
Timeline	June 13 2023 - September 01 2023
Number of Auditors	3

Project Log

Date	Commit Hash	Note
13.06.2023	52b26b8a2f1904b048754d5443e08d2144610b92	Commit for the audit
18.07.2023	c0122d27677cd4e1aaee7f1e21f807ccadf46ac8	DAO commit for the audit
14.08.2023	7ada067d6cd0bd6761440a01e2ea81c09e5391f5	Commit for re-audit
18.08.2023	ac68167150040eddbeb453b861ed7480db8d5ad0	Commit for re-audit 2
23.08.2023	d9a2b2ba4d26e01115731fffadd8306c5955a660	Commit for diff audit

Date	Commit Hash	Note
31.08.2023	beda4525ae0995499bc77cb788fd933c24f3747f	Commit for deploy

Project Scope

The audit covered the following files:

File name	Link
BorrowerOperations.sol	BorrowerOperations.sol
DebtToken.sol	DebtToken.sol
Factory.sol	Factory.sol
GasPool.sol	GasPool.sol
LiquidationManager.sol	LiquidationManager.sol
PriceFeed.sol	PriceFeed.sol
PrismaCore.sol	PrismaCore.sol
SortedTroves.sol	SortedTroves.sol
StabilityPool.sol	StabilityPool.sol
TroveManager.sol	TroveManager.sol
DelegatedOps.sol	DelegatedOps.sol
PrismaBase.sol	PrismaBase.sol
PrismaMath.sol	PrismaMath.sol
PrismaOwnable.sol	PrismaOwnable.sol
SystemStart.sol	SystemStart.sol

File name	Link
AdminVoting.sol	AdminVoting.sol
AllocationVesting.sol	AllocationVesting.sol
BoostCalculator.sol	BoostCalculator.sol
EmissionSchedule.sol	EmissionSchedule.sol
FeeReceiver.sol	FeeReceiver.sol
IncentiveVoting.sol	IncentiveVoting.sol
PrismaToken.sol	PrismaToken.sol
TokenLocker.sol	TokenLocker.sol
Treasury.sol	Treasury.sol
ConvexDepositFactory.sol	ConvexDepositFactory.sol
ConvexDepositToken.sol	ConvexDepositToken.sol
CurveDepositFactory.sol	CurveDepositFactory.sol
CurveDepositToken.sol	CurveDepositToken.sol
CurveProxy.sol	CurveProxy.sol
AirdropDistributor.sol	AirdropDistributor.sol
Vault.sol	Vault.sol

Deployments

File name	Contract deployed on mainnet	Comment
PrismaCore.sol	0x5d17eA085F2FF5da3e6979D5d26F1dBaB664ccf8	
CurveProxy.sol	0x490b8C6007fFa5d3728A49c2ee199e51f05D2F7e	
PriceFeed.sol	0x5b0398D2A7EEb524C678bbE9f9a4C4104E864D38	
FeeReceiver.sol	0xfdCE0267803C6a0D209D3721d2f01Fd618e9CBF8	
GasPool.sol	0xE0598D793bAf7b4f49F4a003885E4180B28caB61	
Factory.sol	0x70b66E20766b775B2E9cE5B718bbD285Af59b7E1	
LiquidationManager.sol	0x5de309dfd7f94e9e2A18Cb6bA61CA305aBF8e9E2	
TroveManager.sol	0x4482BD395d78d36AF31a1d58fe86958707861Cf5	
SortedTroves.sol	0x3BaB3F90095c424b923D67F4bE1790935c8BBb50	
TokenLocker.sol	0x3f78544364c3eCcDCe4d9C89a630AEa26122829d	
IncentiveVoting.sol	0xfd8DF0Db401Ab7EC7a06a8465134FA32132e850C	
PrismaToken.sol	0xdA47862a83dac0c112BA89c6abC2159b95afd71C	
Vault.sol	0x06bDF212C290473dCACea9793890C5024c7Eb02c	
StabilityPool.sol	0xed8B26D99834540C5013701bB3715faFD39993Ba	
DebtToken.sol	0x4591DBfF62656E7859Afe5e45f6f47D3669fBB28	
BorrowerOperations.sol	0x72c590349535AD52e6953744cb2A36B409542719	
AdminVoting.sol	0xC5F87695cABBF16F81133CaBbc7CCC073E648139	
CurveDepositFactory.sol	0x60aF2b6eea2FDc2B6e2cb4A9668c80966D9759e9	
CurveDepositToken.sol	0x924eca29B9535ED43CdC12aAC6F8B5f6A08c7322	

File name	Contract deployed on mainnet	Comment
ConvexFactory.sol	0xD7BA3147F4C1563848fB760352c056D2C8465732	
ConvexDepositToken.sol	0xCF139DbdEfCF14E8BcBfC65d094cd1df8a744a96	

1.5 Summary of findings

Severity	# of Findings
Critical	2
High	3
Medium	6
Low	12

ID	Name	Severity	Status
C-1	Incorrect `defaultedDebt` interest accrual	Critical	Fixed
C-2	Mint awards can be manipulated	Critical	Fixed
H-1	An attacker can steal the StabilityPool depositors profit	High	Fixed
H-2	Whale governance attack on the protocol takes 1 day and cannot be cancelled by DAO	High	Fixed
H-3	Small amounts can be withdrawn without penalties from TockenLocker	High	Fixed
M-1	The implementation of contracts is mutable in Factory	Medium	Fixed
M-2	Users can lock tokens for a few seconds and receive the same weight as users with a one week lock	Medium	Fixed
M-3	A guardian cannot cancel a malicious proposal in AdminVoting	Medium	Fixed
M-4	No time to cancel a malicious proposal in AdminVoting	Medium	Fixed
M-5	`1e18-1` wei PRISMA can get lost in `AllocationVesting.lockFutureClaimsWithReceiver()`	Medium	Fixed

M-6	Using the `RESPONSE_TIMEOUT` constant for all oracles	Medium	Fixed
L-1	A flashmint max amount check is easily bypassed	Low	Fixed
L-2	The callerOrDelegated specification is not followed for CloseTrove	Low	Fixed
L-3	During the global pause users can increase and decrease TCR	Low	Acknowledged
L-4	TroveManager bytecodes are not recommended to be different	Low	Acknowledged
L-5	MCR and CCR management	Low	Fixed
L-6	Use `SafeTransfer`	Low	Fixed
L-7	A note about exotic tokens	Low	Acknowledged
L-8	Unused approval from Treasury to TokenLock	Low	Fixed
L-9	`CurveProxy` contract needs to be approved by Curve DAO	Low	Fixed
L-10	Events missing	Low	Fixed
L-11	Old votes can be called	Low	Fixed
L-12	`AllocationVesting.setAllocations()` allows setting a zero `numberOfWeeks`	Low	Fixed

1.6 Conclusion

During the audit process 2 CRITICAL, 3 HIGH, 6 MEDIUM and 12 LOW severity findings were spotted. After working on the reported findings, all of them were acknowledged or fixed by the client.

The current scope is very well written regarding the ease of understanding and reading the code. All functions can be easily comprehended.

To improve the overall security of the protocol and decrease the number of findings, we suggest increasing test coverage and using Fuzzing tests to find various complex attack vectors.

We recommend writing tests for the following files:

- PriceFeed.sol
- AllocationVesting.sol

The client provided the smart contracts for the deployment without:

- EmissionSchedule.sol
- BoostCalculator.sol
- AllocationVesting.sol
- AirdropDistributor.sol

To make sure that the deployed code hasn't been modified after the last audited commit, one should conduct their own investigation and deployment verification.

2. FINDINGS REPORT

2.1 Critical

C-1	Incorrect `defaultedDebt` interest accrual
Severity	Critical
Status	Fixed in 7ada067d

Description

TroveManager.sol#L1201

In Prisma, there are liquidations which distribute the debt and collateral from liquidated troves among the remaining ones. For example, these are liquidations when ICR < 100% or liquidations the size of which the StabilityPool cannot cover. Each liquidation of this kind increases the values of `defaultedDebt`, `L_debt`, and `L_collateral` in the relevant TroveManager. These variables are used to track the total debt for each trove.

The `_applyPendingRewards()` function is triggered with every user interaction with a trove and increases the `trove.debt` by the pending debt from the `defaultedDebt` pool, but only if `rewardSnapshots[trove].collateral < L_collateral`:

```
function _applyPendingRewards
    ...
    if (rewardSnapshots[_borrower].collateral < L_collateral) {
        ...
        // add pending interest
        debt = debt + pendingDebtReward;
        ...
    }
    ...
    // update trove debt
    t.debt = debt;
```

TroveManager.sol#L1201

Over time, the value of `L_debt` and the size of `defaultedDebt` increase:

```
function _redistributeDebtAndColl
    ...
    if (lastIndexUpdateCached < block.timestamp) {
        ...
        uint256 accruedInterest = Math.mulDiv(defaultedDebt, interestFactor,
        INTEREST_PRECISION);
        ...
        debtWithInterests += accruedInterest;
    }
    ...
    defaultedDebt += debtWithInterests;
```

TroveManager.sol#L1367

However, `L_collateral` can remain the same until the next bad liquidation (i.e., a liquidation with ICR < 100% or a liquidation which StabilityPool cannot cover).

Thus, if a hacker triggers `_applyPendingRewards()` for themselves after a bad liquidation (which increases `L_collateral` and `L_debt`), for example, using the `claimReward()` function, they will set their `rewardSnapshots[hacker].collateral` equal to the new `L_collateral`, and subsequent increases in the `L_debt` variable over time will be incorrectly tracked for them.

For example, the `getTroveCollAndDebt()` function will return the correct debt for the hacker, including the accumulated `defaultedDebt` interest. However, `_applyPendingRewards()` will not update the `trove.debt` variable because the condition `rewardSnapshots[hacker].collateral < L_collateral` is not satisfied as `L_collateral` remained unchanged. This allows the hacker to close their trove without paying the accrued `defaultedDebt` interest, which can be arbitrarily large. Upon trove closure, all their pending interest will be distributed among unsuspecting remaining troves.

This leads to instant losses for all troves, which will be forced to pay off the accumulated debt in `defaultedDebt` on behalf of the hacker, and it also leads to instability in the system when a multitude of troves can suddenly go underwater due to the sudden distribution of unpaid debt by the hacker.

The PoC was sent to the Prism team.

Recommendation

The issue with incorrect interest accrual could be addressed by expanding the check in the `_applyPendingRewards()` function as follows:

```
if (
    (rewardSnapshots[_borrower].collateral < L_collateral)
    ||
    (rewardSnapshots[_borrower].debt < L_debt)
)
```

However, it's worth noting that the `defaultedDebt` pool can only be zeroed out if after a bad liquidation, the `_applyPendingRewards()` function is called for absolutely all troves in the system, which transfers the debt from `defaultedDebt` to the `totalActiveDebt` pool:

```
_movePendingTroveRewardsToActiveBalance(pendingDebtReward,
pendingCollateralReward);
```

This is not practically possible. Therefore, `defaultedDebt` will remain positive because some accounts in the system will be inactive. Consequently, the interest accumulated over time in the `defaultedDebt` pool will be distributed among all other active accounts. This seems unfair to active accounts, who will have to pay for the debts of inactive users.

Therefore, the best solution seems to be a complete abandonment of interest accrual on the `defaultedDebt` pool.

Client's commentary

> Fixed in 2c9540b16690bb115501dff14cb9cebd29949350 - we no longer apply interest to defaulted debt.

C-2	Mint awards can be manipulated
Severity	Critical
Status	Fixed in 7ada067d

Description

- TroveManager.sol#L135

Mint awards can be manipulated. Next steps:

1. A hacker waits until TCR becomes 149% in TroveManager
2. The hacker takes the collateral flashloan and inside the transaction:

- Open a "huge" trove to get "accountLatestMint" (mint awards TroveManager.sol#L1018). No commission due to recovery mode (https://0xsydbs-organization.gitbook.io/prisma-finance/core-protocol-operations/recovery-mode#impact-on-fees).
- Open a "small" trove to increase TCR from 149% to 150% (by hacker_helper).
- In the end, recovery mode is more than 150, and the hacker can close the "huge" trove without a commission.

3. The hacker has a huge "accountLatestMint" after flashloan and they can claim prisma tokens.

The script has been provided.

Recommendation

This finding shows a way to open any trove without a commission. It is not recommended to give awards for minting directly (TroveManager.sol#L1018).

We recommend revising the architecture of rewards.

Client's commentary

> Fixed in 41d84cdf80c3717ab3074e6db674ef72c70b3e84 - emissions for minting are disabled while the system is in recovery mode.

2.2 High

H-1	An attacker can steal the StabilityPool depositors profit
Severity	High
Status	Fixed in 7ada067d

Description

The liquidation flow of the protocol is supposed to be as follows:

- users open troves and join `StabilityPool`
- anyone calls liquidation that iterates the given troves and liquidates them one by one
- in `StabilityPool` there are more collateral tokens

By using a flash loan, any user can bypass the provision of liquidity to the protocol for a long time and steal some of the `StabilityPool` provider's profit taking the following steps:

1. An attacker gets a flash mint (DebtToken.sol#L199).
2. The attacker makes a deposit `mkUSD` to `StabilityPool`.
3. The attacker calls the LiquidationManager.sol#L143 function to liquidate the troves.
4. The attacker calls StabilityPool.sol#L241
5. The attacker gets profit and returns the flash loan.

The attack's impact:

- loss of profit from liquidations by `StabilityPool` providers;
- decreased motivation to use the Stability Pool which may cause mkUSD to unpeg.

Recommendation

We recommend that you use the time factor to prevent flash loan attacks.

Client's commentary

> Fixed in 534b3576282d92848433f841ade8fe32d645ef06 - SP withdrawals are blocked in the same second that there has been a deposit.

H-2	Whale governance attack on the protocol takes 1 day and cannot be cancelled by DAO
Severity	High
Status	Fixed in 7ada067d

Description

The flow of the attack:

1. A whale prepares funds so that the lock for one week gives a weight to bypass the `passingPct`.
2. In the end of week [N], one minute before week [N+1], the whale calls `TokenLocker.lock(week=1)`. The token locker will register the weight for week [N]
3. Week [N+1] begins in one minute after Step 2. The funds for the attack are unlocked and can be withdrawn.
4. The whale calls `AdminVoting.createNewProposal()`
   The payload is the following:
   – `PrismaCore.setGuardian()`
   – `Treasury.transferTokens(token=prisma, receiver=whale, amount=everything)`
5. The whale calls `AdminVoting.voteForProposal()` and votes for the proposal from Step 4.
   AdminVoting uses weights from the previous week [N] which is already finalized and the whale was the largest voter at the end of the week.
   The whale bypasses `passingPct` for this proposal, so the proposal can be executed in 1 day.
6. In one day, the whale calls `AdminVoting.executeProposal()` and withdraws all funds from the Treasury.

The guardian cannot cancel this attack. Because the first call in the payload is `PrismaCore.setGuardian()` which is the only type of call that cannot be cancelled, thus the whole payload cannot be cancelled.

- AdminVoting.sol#L193-L196

Recommendation

We recommend that:

- the guardian restriction to cancel `PrismaCore.setGuardian()` should be fixed. For example, the guardian is not allowed to cancel `PrismaCore.setGuardian()` only if `payload.length == 1`. In this case proposers must have only one call in the payload. Otherwise, the guardian can cancel.

- when a proposal changes a guardian the protocol can require higher `passingPct`.

Client's commentary

> Payload check fixed in 7ada067d6cd0bd6761440a01e2ea81c09e5391f5. Higher `passingPct` fixed in 82467aa42af815093cdcd48978d6c74879205599.

H-3	Small amounts can be withdrawn without penalties from TockenLocker
Severity	High
Status	Fixed in 7ada067d

Description

- TokenLocker.sol#L806

There's a rounding error in the penalty calculation:

```
uint256 penaltyOnAmount = (lockAmount * weeksToUnlock) / MAX_LOCK_WEEKS;
```

The penalty becomes zero if `lockAmount * weeksToUnlock < MAX_LOCK_WEEKS`. For example, if `lockToTokenRatio=1e18`, then 1e18 PRISM is locked for 51 weeks (or alternatively, 51e18 PRISMA is locked for 1 week) can be withdrawn without penalties.

One example of an attack that allows you to withdraw 23% of the tokens from the `AllocationVesting` contract:

- We send a small amount of tokens using `allocation_vesting.transferPoints` to any of the addresses (AllocationVesting.sol#L83)
- Lock these tokens `allocation_vesting.lockFutureClaims` releasing 23% of future transfers (AllocationVesting.sol#L116)
- Call `locker.withdrawWithPenalty` in a loop (TokenLocker.sol#L769). In this case, the commission is not taken
- Get PRISMA tokens without blocking for 12 weeks

PoC has been sent to the customer.

Recommendation

We recommended that you improve the precision of the penalty calculation or prohibit the early withdrawal of small amounts.

Client's commentary

> Fixed in e1e26c62d672e00be3b25babf4cd0240479c34b2 by improving precision / rounding the penalty amount up during early withdrawals.

2.3 Medium

M-1	The implementation of contracts is mutable in Factory
Severity	Medium
Status	Fixed in 7ada067d

Description

- Factory.sol#L131

`troveManagerImpl` can be changed (Factory.sol#L124).

In `Factory` `cloneDeterministic` (`create2`) `troveManagerImpl` is used as a base contract for copying contracts, so the `getTroveManager` method may return incorrect results if `troveManagerImpl` is changed.

Recommendation

We recommend revising `predictDeterministicAddress` and getting the trove manager using a collateral.

Client's commentary

> Fixed in 079e39fa0004679faa5eefbdd8e4af1b497fa28d

M-2	Users can lock tokens for a few seconds and receive the same weight as users with a one week lock
Severity	Medium
Status	Fixed in ac681671

Description

The weight received right after the lock is stored in the `accountWeeklyWeights[currentWeek]`.

- TokenLocker.sol#L441

Imagine two users:

1. the first one locks tokens at the **first** seconds of week 6, duration is 1 week;
2. the second one locks tokens at the **last** seconds of week 6, duration is 1 week.

Both of them will receive the weight of `tokenAmount*1` and both of them will have all their tokens unlocked in week 7. They can withdraw their unlocked tokens in the first seconds of week 7.

It means that the protocol treats these two users identically though the first user locked PRISMA tokens for 1 week and the second one in fact locked tokens for only a few seconds.

Recommendation

We recommend requiring the minimum length of the lock for more than 1 week. It would allow the protocol to have a liquidity lock for at least 1 week even in case of malicius locks in last seconds.

Client's commentary

> Fixed in 2180af34c0eab5a73441c972d3ede802f28db7f4. We have disallowed 1 week locks within the final 3 days of each epoch week.

M-3	A guardian cannot cancel a malicious proposal in AdminVoting
Severity	Medium
Status	Fixed in 7ada067d

Description

AdminVoting.sol#L186

A user can create a new proposal via `createNewProposal()` with a payload of actions. A guardian, however, is prohibited from using `cancelProposal()` on a proposal, if the first action is a call to `IPrismaCore.setGuardian()`. Therefore, if the proposal contains a malicious action, the guardian is unable to cancel it.

Recommendation

Our recommendation is to allow the cancellation of a proposal if its `payload.length > 1`.

Client's commentary

> Fixed in 7ada067d6cd0bd6761440a01e2ea81c09e5391f5

M-4	No time to cancel a malicious proposal in AdminVoting
Severity	Medium
Status	Fixed in 7ada067d

Description

- AdminVoting.sol#L211

A malicious whale can create a proposal, wait for `MIN_TIME_TO_EXECUTION + 1` (currently 24 hours), and then vote and execute it in the same block, leaving no time for a guardian to cancel it.

It becomes possible because of `MIN_TIME_TO_EXECUTION < VOTING_PERIOD` and there are only two requirements to execute a proposal:

```
require(proposal.currentWeight >= proposal.requiredWeight, "Not passed");
require(proposal.createdAt + MIN_TIME_TO_EXECUTION < block.timestamp,
"MIN_TIME_TO_EXECUTION");
```

Recommendation

We recommend adding a delay between the point in time when the proposal gains enough votes for execution and the execution itself.

Client's commentary

> Fixed in c3b19625700434fc1a8e1c10b00ce909680ba663

M-5	`1e18-1` wei PRISMA can get lost in `AllocationVesting.lockFutureClaimsWithReceiver()`
Severity	Medium
Status	Fixed in beda4525

Description

- AllocationVesting.sol#L172-L173

In function `AllocationVesting.lockFutureClaimsWithReceiver()`, an `amount` of tokens is debited from the user. However, a number with a rounding error is locked:

```
tokenLocker.lock(receiver, amount / lockToTokenRatio, 52);
```

If `lockToTokenRatio==1e18`, then the dust up to 1e18 wei PRISMA will be lost by the user and will accumulate in the `AllocationVesting` contract.

Recommendation

It seems that the loss of `1e18-1` wei PRISMA would be unpleasant for the user, so it's recommended to prevent this.

One way to do this is to discard the dust at the beginning of the function:

```
function lockFutureClaimsWithReceiver(
    address account,
    address receiver,
    uint256 amount
) public callerOrDelegated(account) {
    amount = (amount / lockToTokenRatio) * lockToTokenRatio;
    // truncating the dust
    ...
```

Client's commentary

> Fixed in ee3349d69ca0c861312766645788ff634aaad4b7

M-6	Using the `RESPONSE_TIMEOUT` constant for all oracles
Severity	Medium
Status	Fixed in beda4525

Description

- PriceFeed.sol#L58

Chainlink uses the heartbeat when checking if the data feed is fresh.

Currently, the `25 hours` constant is used for the definition of staling. However, there are feeds that update the data faster (for example, https://data.chain.link/ethereum/mainnet/crypto-usd/eth-usd).

Without this, data can be considered fresh for a long time.

Recommendation

It is recommended not to use the `RESPONSE_TIMEOUT` constant. Each oracle can contain this variable.

Client's commentary

> Fixed in bead4a8a3b713c9e363520b79356401768ae7721

2.4 Low

L-1	A flashmint max amount check is easily bypassed
Severity	Low
Status	Fixed in 7ada067d

Description

Users can take two or more flashloans and bypass the requirement of the `maxFlashLoan()` size.

- DebtToken.sol#L155

The current cap for flashloans is set to `2**128`.
If this check aims to limit the flashloan amount in order not to exceed max totalSupply, take into account that OpenZeppelin´s ERC20 will not allow new totalSupply overflow `2**256` limit.

- ERC20.sol#L243

Recommendation

We recommend either removing this check or introducing a reentrancy protection for the flashloan function only.

Client's commentary

> Fixed in 49ee49f39584c35c05eff17db214fcc487ea3ecf - the restriction was removed.

L-2	The callerOrDelegated specification is not followed for CloseTrove
Severity	Low
Status	Fixed in 7ada067d

Description

For all key functions in `BorrowerOperations` users can choose their privileged address to delegate an operation.

- DelegatedOps.sol#L22-L29

Comment there states that:

```
In executing the call,
all internal state updates
should be applied for `account`
and all value transfers
should occur to or from the caller.
```

It is not the case for `closeTrove()`.

- BorrowerOperations.sol#L393-L398
  Here, the contract closes the trove taking debtToken from `msg.sender` (as it should), but the remaining collateral is sent to `account`, so the specification is not followed.

Recommendation

We recommend either sending collateral to `msg.sender` or fixing the comment describing the desired behavior.

Client's commentary

> Fixed in 594d9fc9de4148fcb5bb137bac8fc4884ffcecec

L-3	During the global pause users can increase and decrease TCR
Severity	Low
Status	Acknowledged

Description

We have two contracts that check the global pause:

- BorrowerOperations.sol (BO)
- StabilityPool.sol (SP)

These contracts have the following functions for users and sometimes they are allowed to be called during the global pause.

function	allowed on pause ?
BO.openTrove()	no
BO.addColl()	no
BO.withdrawCall()	yes
BO.withdrawDebt()	no
BO.repayDebt()	yes
BO.adjustTrove()	allowed if adjusted to [no more collateral] & [less debt]
BO.closeTrove()	yes
SP.provideToSP()	no
SP.withdrawFromSP()	yes

Withdrawing collateral makes TCR lower.
Repaying debt makes TCR higher.
Closing troves makes TCR lower, likely.

As a result, the "pause" does not serve as a true global pause and does not stop all operations.

Recommendation

We recommend ensuring that this pause behavior is enough and complete. If the project needs a true global pause, we recommend introducing a new type of pause that stops all functions.

Client's commentary

> We have updated our documentation to better explain the intent of the pause functionality.

L-4	TroveManager bytecodes are not recommended to be different
Severity	Low
Status	Acknowledged

Description

`Factory` allows selecting any implementation addresses for `TroveManager` and `SortedTroves`.

- Factory.sol#L83-L84

The problem is that TroveManagers are not completely separated and the TCR calculation is made via iterating through each TroveManager. Thus, the issue in one TroveManager can break the whole system. That is why it is risky that potentially not audited implementations are allowed.

Recommendation

We recommend ensuring that all connected TroveManagers share the same code. It is better to have new features separated from the audited codebase, e.g. in v2 of the protocol.

Client's commentary

> We understand and accept this risk.

L-5	MCR and CCR management
Severity	Low
Status	Fixed in 7ada067d

Description

`MCR` and `CCR` are constants:

- PrismaBase.sol#L13-L17

Over time, the value and volatility of the collateral token may change. It will require to adapt `MCR` and `CCR` to new market behavior.

Recommendation

We recommend that you add methods for changing `MCR` and `CCR` parameters.

Client's commentary

> Dynamic MCR implemented in c0122d27677cd4e1aaee7f1e21f807ccadf46ac8. We have decided not to implement a dynamic CCR.

L-6	Use `SafeTransfer`
Severity	Low
Status	Fixed in ac681671

Description

When transfer tokens, there are checks for returned value:

- TroveManager.sol#L850
- TroveManager.sol#L1381
- StabilityPool.sol#L679

This is the right way. But when working with specific tokens, we can use `SafeERC20`.

Recommendation

We recommend using SafeERC20.sol.

`SafeERC20` helps:

- check the boolean return values of ERC20 operations and revert the transaction if they fail;
- at the same time allowing you to support some non-standard ERC20 tokens that don't have boolean return values.

Client's commentary

> Implemented in 039cc86ac7a5e14ec7b067488b39a6f08a914f05 - `SafeERC20` is used for collaterals. We have not implemented it for transfers of our own protocol tokens, CRV, CVX or Curve LP tokens, as all of these tokens return True or revert.

L-7	A note about exotic tokens
Severity	Low
Status	Acknowledged

Description

The project's current implementation does not account for the unique behaviors associated with rebaseable tokens, fee-on-transfer tokens, ERC-777 (callback) tokens, or non-18 decimal tokens, exposing the system to potential risks and undesirable consequences, if such a token is used as a collateral.

Recommendation

It is necessary to manage the collateral registration process to ensure that such exotic tokens are not added as part of the protocol.

Client's commentary

> We understand and accept this risk.

L-8	Unused approval from Treasury to TokenLock
Severity	Low
Status	Fixed in 7ada067d

Description

In the constructor of Treasury we have this line:

```
_token.approve(address(_locker), type(uint256).max);
```

- Treasury.sol#L111

But TokenLocker never takes PrismaTokens from the Treasury. Moreover, TokenLocker does not store Treasury address.

Recommendation

This approval can be removed.

Client's commentary

> Fixed in 16b23cbb2c26ec76ac803706a61541cd1aea1111

L-9	`CurveProxy` contract needs to be approved by Curve DAO
Severity	Low
Status	Fixed in 7ada067d

Description

There is the following line in the methods for blocking CRV tokens (in Curve vesting contract):

```
self.assert_not_contract(msg.sender)
```

- CurveProxy.sol#L307

In order to make`CurveProxy` work, you need to get an approval from the Curve DAO.

Recommendation

We recommend documenting the CurveProxy deployment process.

Client's commentary

> Fixed in fee32677c2bf418d8a3ab89f345d615bfd8ebe0f

L-10	Events missing
Severity	Low
Status	Fixed in 7ada067d

Description

The function should emit an event for better UX in possible integrations:

- Treasury.sol#L173.

Recommendation

We recommend emitting events.

Client's commentary

> Fixed in ae366131d58611eece788fd71f736968d36428ce

L-11	Old votes can be called
Severity	Low
Status	Fixed in 7ada067d

Description

- AdminVoting.sol#L207

Old approved proposals may be called after a long time.

One example of using this loophole:

1. A hacker generates a duplicate proposal from the PRISMA team and motivates users to vote for it.
2. The proposal gets enough votes but doesn't execute `executeProposal`.
3. After a long time, the hacker returns to this proposal and calls it for their own purposes.

Recommendation

We recommend making the maximum lifetime for the proposal.

Client's commentary

> Fixed in f3c1d6f0a6d28807331300569c043a0c97d2d62b

L-12	`AllocationVesting.setAllocations()` allows setting a zero `numberOfWeeks`
Severity	Low
Status	Fixed in beda4525

Description

- AllocationVesting.sol#L94

`setAllocations()` allows an admin to create an allocation with zero `numberOfWeeks`, leading to unexpected behavior. For example, the `_vestedAt()` function for allocations with zero `numberOfWeeks` always returns zero and thus the `claim()` function reverts, so a user can't withdraw funds from such an allocation.

However, `transferPoints()` assumes that if an allocation has zero `numberOfWeeks`, then this allocation is empty and therefore rewrites `numberOfWeeks` with the sender's value:

```
if (numberOfWeeksTo == 0) {
    allocations[to].numberOfWeeks = numberOfWeeksFrom;
}
```

- AllocationVesting.sol#L130-L133

Thus, after transferring additional points, this allocation becomes available for withdrawal.

Recommendation

We recommend adding a check in the `AllocationVesting.setAllocations()` function to ensure that `numberOfWeeks` cannot be zero.

Client's commentary

> Fixed in 722161301ee077abdd5e922a783e4cae5063602a

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts


https://github.com/mixbytes/audits_public


https://mixbytes.io/


hello@mixbytes.io


https://twitter.com/mixbytes