DEFI MONEY CORE SECURITY AUDIT REPORT

Jun 19, 2024

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

Defi.Money is a fork of Curve's stablecoin crvUSD - users can deposit collateral and mint stablecoin as debt. In addition, users have the ability to exchange between different collaterals and stablecoin within the platform.

1.4 Project Dashboard

Project Summary

Title	Description
Client	DeFi Money
Project name	Core
Timeline	April 29 2024 - June 14 2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
29.04.2024	e22732083f79a5fb13bdb69132622017ebe79a59	Commit for the audit
13.05.2024	3a50cb6b38f5120707183549c1f443759e8fa538	Commit for the re-audit
04.06.2024	0bde9fb784400d73f434d585aab7d9dd0ad1c679	Commit for the diff audit
14.06.2024	90d43c77dc9e0240b010952c750494ebe19ca314	Commit for the re-audit 2

Project Scope

The audit covered the following files:

File name	Link
contracts/AggMonetaryPolicy2.vy	AggMonetaryPolicy2.vy
contracts/AMM.vy	AMM.vy

File name	Link
contracts/MainController.vy	MainController.vy
contracts/MarketOperator.vy	MarketOperator.vy
contracts/PegKeeperRegulator.vy	PegKeeperRegulator.vy
contracts/PegKeeper.vy	PegKeeper.vy
contracts/cdp/oracles/AggregateChainedOracle.sol	AggregateChainedOracle.sol
contracts/cdp/oracles/AggregateStablePrice.vy	AggregateStablePrice.vy
contracts/cdp/oracles/Layer2UptimeOracle.sol	Layer2UptimeOracle.sol
contracts/bridge/BridgeToken.sol	BridgeToken.sol
contracts/base/ProtocolCore.sol	ProtocolCore.sol
contracts/periphery/hooks/L2SequencerUptimeHook.sol	L2SequencerUptimeHook.sol
contracts/periphery/hooks/WhitelistHook.vy	WhitelistHook.vy
contracts/base/dependencies/DelegatedOps.sol	DelegatedOps.sol
contracts/cdp/AggMonetaryPolicy2.vy	AggMonetaryPolicy2.vy
contracts/cdp/AMM.vy	AMM.vy
contracts/cdp/MainController.vy	MainController.vy
contracts/cdp/MarketOperator.vy	MarketOperator.vy
contracts/cdp/PegKeeperRegulator.vy	PegKeeperRegulator.vy
contracts/cdp/PegKeeper.vy	PegKeeper.vy

Deployments

Deployment verification will be conducted later after contracts deployment.

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	1
Medium	8
Low	4

ID	Name	Severity	Status
H-1	Incorrect `last_tvl` state after price pair removal	High	Fixed
M-1	The AMM rate is not updated in certain cases	Medium	Fixed
M-2	Hooks may break some invariants	Medium	Fixed
M-3	Hooks may result in user debt exceeding the `global_market_debt_ceiling` and unbacked tokens being minted	Medium	Fixed
M-4	`CONTROLLER.total_debt()` may not be up to date	Medium	Fixed
M-5	Not reverting on the sequencer downtime	Medium	Acknowledged
M-6	`AggregateStablePrice` EMA can be manipulated	Medium	Acknowledged
M-7	`isMintEnabled` blocks extra functionality	Medium	Fixed
M-8	`BridgeToken.sendSimple()` does not call `msgInspector`	Medium	Fixed
L-1	Collateral price growth may cause hard liquidation	Low	Acknowledged

L-2	Stablecoin separate ownership	Low	Fixed
L-3	Restricting zero decimals in `AggregateChainedOracle`	Low	Fixed
L-4	Uncontrolled Slippage in `BridgeToken.sendSimple()`	Low	Acknowledged

1.6 Conclusion

DeFi Money enables users to mint stablecoins using multiple tokens as collateral and exchange between them. Users can create and manage positions, and these positions can also be passively managed. If the collateral price decreases, the loan can enter a soft-liquidation mode, meaning that positions can be restored if the collateral price rebounds. Throughout this process, the stablecoin remains overcollateralized. Additional mechanisms maintain the peg on exchanges by burning and minting stablecoins for liquidity provision and withdrawals.

This audit focused on the protocol's general architecture, bond math, interest rate calculations, user operations, liquidations, and peg-maintenance mechanisms. Special attention was given to handling edge cases during position creation, active position management, and passive position adjustments due to changes in oracle prices.

Key activities included:

- Checking the math in MarketOperator and AMM concerning bands, fees, and accrued interest
- Ensuring that positions with negative health cannot be created
- Checking edge cases with position creation across different sets of oracle returns and liquidity allocation
- Testing different input scenarios for `adjust_loan()`
- Verifying hook setting functions to manage hooks for Lending and AMM operations
- Testing the PegKeeper workflow to stabilize balances in Pools
- Ensuring correct role setup and function access to guarantee correct function callers
- Conducting architectural and functional reviews of PegKeeper management and PegKeeperRegulator migration
- Checking the Monetary Policy management workflow, the accuracy of Monetary Policy switching, and its impact on lending rates
- Conducting a flashmint impact analysis on the entire system
- Ensuring the current workflow for various debt ceiling setups and their anticipted changes, including special scenarios for "owed debt" accounting and netting

Key Observations and Recommendations:

- **Using hooks:** The owner can set arbitrary hooks called in key lending and exchange functions. These hooks adjust debt variables and can significantly impact the overall security of the system. However, the implementation of hooks was beyond the scope of this audit.
- **PegKeeperRegulator migration:** There are multiple scenarios to set a new PegKeeperregulator. Current smart contracts allow only automated transfers related to PegKeepers. But a full migration should also involve revoking the Minter role from the previous PegKeeperRegulator, setting for the new one.
- **Little AggregateChainedOracle validation:** This oracle is flexible, and any oracle path can be provided for the resulting oracle calculation. Individual returns from the formula are not validated for mistakes, have equal weights, are not checked for malicious outliers.

- **BridgeRelayer superpower:** BridgeRelay has permission to call `setPeer()` which is a strong power in the context of LayerZero. The BridgeRelay contract implementation was out-of-scope for this audit.

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

H-1	Incorrect `last_tvl` state after price pair removal
Severity	High
Status	Fixed in 90d43c77

Description

- AggregateStablePrice.vy#L90-L100

The `AggregateStablePrice` contract doesn't remove the corresponding entry in the `last_tvl` array during `remove_price_pair()` call.

This may result in inaccurate TVL calculations and incorrect price aggregations.

Recommendation

We recommend adjusting the `remove_price_pair()` function to remove the corresponding entry in the `last_tvl` array when a price pair is removed.

Client's commentary

> Fixed in `b4b72d3`

2.3 Medium

M-1	The AMM rate is not updated in certain cases
Severity	Medium
Status	Fixed in 14508882

Description

- MainController.vy#L1085-L1092
- MainController.vy#L1097-L1104

If the admin changes monetary policy via `MainController.change_existing_monetary_policy()` or `MainController.change_market_monetary_policy()`, the interest rate of the AMM is not set to the new one until someone interacts with AMM via the following operations:

- `create_loan()`
- `adjust_loan()`
- `close_loan()`
- `liquidate()`
- `collect_fees()`

Thus, if the new monetary policy rate for a market is high and users in that market are inactive, their positions will remain at the old low rate, potentially causing the protocol to miss out on additional profits. Conversely, if the new rate is low, positions in a low-activity market may continue at the old high rate with extra losses for borrowers.

Recommendation

We recommend calling `_update_rate()` when the monetary policy is changed. We also recommend implementing monitoring mechanisms to check that lenders and borrowers do not accumulate significant losses due to unsynchronized rates.

Client's commentary

> Addressed in 1450888:
> When calling `change_market_monetary_policy` the rate is now immediately updated.
> Lacking functionality to iterate markets using a given `mp_idx`, and due to the potential of gas exhaustion with enough markets, we choose to add a comment on `change_existing_monetary_policy` suggesting subsequent call(s) to `collect_fees` to force the rate refresh.

M-2	Hooks may break some invariants
Severity	Medium
Status	Fixed in 7b0cafb2

Description

- MainController.vy#L718
- MainController.vy#L768
- MainController.vy#L823
- MainController.vy#L861

If we suppose that hooks in the MainController can return arbitrary adjustments, the following invariants will break:

- `redeemed + total_debt >= minted`.

For example, in the initial state when `redeemed = total_debt = minted = 0`, if we create a new loan via `create_loan(debt_amount)` and the hook adjusts `debt_amount_final` to a smaller value, then `total_debt` will be increased by a smaller value than `minted` and the following will be true:

```
redeemed + total_debt < minted
```

Recommendation

We recommend checking safe boundaries for hooks.

Client's commentary

> Fixed in 7b0cafb, the total hook debt adjustment for each market is now tracked and cannot go below zero.

M-3	Hooks may result in user debt exceeding the `global_market_debt_ceiling` and unbacked tokens being minted
Severity	Medium
Status	Fixed in 3a50cb6b

Description

- MainController.vy#L718
- MainController.vy#L768-L794

If a user creates a loan and the hook returns a negative value, then `minted > debt_increase` and unbacked stablecoins will be minted and not accounted in `_assert_below_debt_ceiling(total_debt)`:

```
def create_loan(
  ...
  debt_amount: uint256,
  ...
):
  ...
  hook_adjust: int256 = self._call_hooks(
    ...
  )
  debt_amount_final: uint256 = self._uint_plus_int(debt_amount, hook_adjust)
  ...
  debt_increase: uint256 = MarketOperator(market).create_loan(account,
  coll_amount, debt_amount_final, n_bands)

  total_debt: uint256 = self.total_debt + debt_increase
  self._assert_below_debt_ceiling(total_debt)

  self.total_debt = total_debt
  self.minted += debt_amount
```

MainController.vy#L718

In another scenario, if a user provides a negative `debt_change` to the `adjust_loan()` and the hook adjusts it to a positive value, then the original `debt_change < 0` but `debt_adjustment > 0`. In that case the `total_debt` is increased, but we don't fall into the `if debt_change > 0` statement and do not check for the debt ceiling:

```
debt_change_final: int256 = self._call_hooks(...) + debt_change
...
debt_adjustment: int256 = MarketOperator(market).adjust_loan(account,
coll_change, debt_change_final, max_active_band)
...
total_debt: uint256 = self._uint_plus_int(self.total_debt, debt_adjustment)

if debt_change != 0:
    debt_change_abs: uint256 = convert(abs(debt_change), uint256)
    if debt_change > 0:
        self._assert_below_debt_ceiling(total_debt)
        self.minted += debt_change_abs
        STABLECOIN.mint(msg.sender, debt_change_abs)
    else:
        self.redeemed += debt_change_abs
        STABLECOIN.burn(msg.sender, debt_change_abs)
```

MainController.vy#L768-L794

Recommendation

We recommend checking the `total_debt` against the debt ceiling when the `debt_adjustment > 0` in the `adjust_loan()` function. We also recommend taking into account stablecoins which are not included in the `total_debt` and are not backed.

Client's commentary

> Fixed in e608e86, bounds checks are added for hook adjustments:
> On calls to `create_loan`, `close_loan`, and `liquidate` the hook is explicitly prevented from adjusting the debt change such that it would go negative. This was already implicitly prevented via bounds checks when type-casting to from `int256` to `uint256` however it now gives a meaningful revert message.
> On calls to `adjust_loan`, a hook cannot apply a debt adjustment if there is no initial debt change. A hook also cannot modify a debt change such that it's sign changes (an increase in debt cannot become a decrease, or vice-versa).
> Debt adjustment bounds are documented in the controller hook interface natspec

M-4	`CONTROLLER.total_debt()` may not be up to date
Severity	Medium
Status	Fixed in 57b7cb75

Description

- AggMonetaryPolicy2.vy#L171

The `MainController._update_rate()` calls the monetary policy rate calculation method which may use a not updated `total_debt()`.

For example, in the `adjust_loan()` method the `_update_rate()` is called before the `total_debt` update (MainController.vy#L781):

```
debt_adjustment: int256 = MarketOperator(market).adjust_loan(account,
coll_change, debt_change_final, max_active_band)

self._update_rate(market, c.amm, c.mp_idx) # self.total_debt is not updated

total_debt: uint256 = self._uint_plus_int(self.total_debt, debt_adjustment)
self.total_debt = total_debt
```

In case of large differences, `AggMonetaryPolicy2` may not calculate the new rate correctly.

Recommendation

We recommend calling `self._update_rate()` after updating the `MainController.total_debt`.

Client's commentary

> Fixed in 57b7cb7, `_update_rate` is always called after adjusting `total_debt`

M-5	Not reverting on the sequencer downtime
Severity	Medium
Status	Acknowledged

Description

`AggregateChainedOracle` uses stored prices during the sequencer downtime:

- AggregateChainedOracle.sol#L145-L154
- Layer2UptimeOracle.sol#L31-L38

This is a risky approach, potentially leading to the use of irrelevant prices.

The approach demonstrated in Chainlink's documentation is different:

- https://docs.chain.link/data-feeds/l2-sequencer-feeds
  The example, it reverts with `SequencerDown()` or `GracePeriodNotOver()`.

Recommendation

Consider reverting when the oracle reports sequencer downtime.

Client's commentary

> Acknowledged and accepted. In isolation we agree this is dangerous, however we will also apply the `L2SequencerUptimeHook` across all markets. In case of downtime this allows users to still repay debt or increase collateral.

M-6	`AggregateStablePrice` EMA can be manipulated
Severity	Medium
Status	Acknowledged

Description

- AggregateStablePrice.vy#L156-L157

If the `price_w()` function, which updates `last_timestamp`, is not called for a long time, the value of `alpha` decreases, leading to a risk of EMA manipulation by a hacker.

This occurs because the closer `alpha` gets to zero, the greater the influence of the new `totalSupply()` value, which can be manipulated within the current transaction:

```
alpha: uint256 = 10**18
if last_timestamp < block.timestamp:
    alpha = self.exp(- convert((block.timestamp-last_timestamp)*10**18/
                               TVL_MA_TIME, int256))

...
if alpha != 10**18:
    # alpha = 1.0 when dt = 0
    # alpha = 0.0 when dt = inf
    new_tvl: uint256 = self.price_pairs[i].pool.totalSupply()
    tvl = (new_tvl * (10**18 - alpha) + tvl * alpha) / 10**18
```

For example, after `10 * TVL_MA_TIME` seconds (which is about 5 days in the current implementation), if no one calls the function, alpha becomes 0.0000453999. Ultimately, if `alpha=0`, `new_tvl` will simply be equal to `totalSupply()`.

Recommendation

We recommend considering the possibility of manipulation with this aggregator and, for example, implementing monitoring that checks if the `price_w()` has not been called for a long time.

Client's commentary

> Acknowledged and accepted. We will monitor for this and if there are no interactions for several days, trigger one ourselves.

M-7	`isMintEnabled` blocks extra functionality
Severity	Medium
Status	Fixed in 90d43c77

Description

- BridgeToken.sol#L290

The `isMintEnabled` flag blocks the internal `_mint()` method. This can lead to user transactions getting stuck during cross-chain transfers. For example, if funds are burned on the first chain but cannot be minted on the second chain because the admin disables `isMintEnabled` midway.

Overall, the inability to mint internally blocks various functionalities:

- Creating and increasing debt.
- Adding liquidity from PegKeeperRegulator: PegKeeperRegulator.vy#L489
- The `collect_fees()` method: MainController.vy#L827
- There is also a risk of funds getting stuck between chains.

Recommendation

We recommend configuring the ability to emergency disable or limit LZ features separately from protocol features.

Client's commentary

> Fixed in `93e961c`, the scope of the functionality is limited to bridge out/in actions. We accept that this could still result in an undelivered transfer in case the functionality is disabled mid-message. This will only be used in a black swan situation.

M-8	`BridgeToken.sendSimple()` does not call `msgInspector`
Severity	Medium
Status	Fixed in 90d43c77

Description

- OFTCore.sol#L211
- BridgeToken.sol#L153

The `OFTCore.send()` method calls `msgInspector`. However, the new `BridgeToken.sendSimple()` method does not involve such a call.

Thus, if `msgInspector` is configured, the call to `msgInspector` will be ignored when `sendSimple()` is called.

Recommendation

We recommend that `msgInspector` is also called in `sendSimple()`.

Client's commentary

> Fixed in `fd4a888`.

2.4 Low

L-1	Collateral price growth may cause hard liquidation
Severity	Low
Status	Acknowledged

Description

When the oracle price changes, the AMM tick price changes by approximately 3.3 times more. If a user's position were replaced with stablecoins, its health would be determined by the market price of the collateral that can be purchased from the AMM with those stablecoins, and dynamic fees would not taken into account.

This may create a scenario where hard liquidations may occur even though the price of the collateral goes up:

1. A hacker buys the lenders' collateral from the AMM.
2. The hacker inflates the oracle price of the collateral by 2%.
3. The hacker can then `liquidate()` the lenders because their `health < 0`, as their positions, being replaced with stablecoins, can now purchase 3.3 times less collateral in AMM compared to the real market.

This situation may only arise if the hacker is able to inflate the oracle price between two trades within one or two blocks.

Recommendation

Oracle selection recommendations:

1. Ensure that the oracle's price cannot fluctuate mid-block. The price should be determined by the last transaction in the previous block. Most EMA oracles in Curve stable swap pools meet this criterion.
2. Note, that the `EmaPriceOracle.vy` file in the project repository is implemented **incorrectly** and **must not** be used: EmaPriceOracle.vy
3. Avoid integrating with low-liquidity tokens to reduce the likelihood of 2-block attacks.
4. Increase AMM fees, as recommended in the comment: test_oracle_attack.py.

Client's commentary

> Client: Addressed in 3954a2d, the contract has been moved to the `testing` subfolder and explicitly marked as "not for use in production". We look forward to a follow-on round of audit explicitly focused on oracles!
> MixBytes: The finding is marked as acknowledged because it is a fundamental feature of the AMM which is difficult to completely eliminate at the code level. With the correct choice of oracles, this feature is safe.

L-2	Stablecoin separate ownership
Severity	Low
Status	Fixed in 35bcf099

Description

Generally, each system contract follows the pattern indicating the Core contract on deployment.
E.g. `MainController`:

- MainController.vy#L280

As a result, a single system owner is stored on the `Core` contract allowing transferring the ownership of the whole system only on `Core`.

But this is not the case for `StableCoin` - it inherits from `Ownable`, so that a separate owner address is stored on the smart contract.

- StableCoin.sol#L14

Thus, there are two owners within the system. They are the same address on deployment.
But they can become different in case of the ownership transfer.

Recommendation

We recommend following the pattern with `Core` for `StableCoin`.

Client's commentary

> Implemented in 35bcf09, `StableCoin` has been promoted out of testing and employs the `CoreOwner` ownership pattern (along with flashloans and OFT functionality).

L-3	Restricting zero decimals in `AggregateChainedOracle`
Severity	Low
Status	Fixed in 90d43c77

Description

`AggregateChainedOracle.addCallPath()` does not allow zero decimals:

- AggregateChainedOracle.sol#L119

But in fact, zero decimals work well as an adjustment for the oracle return:

- AggregateChainedOracle.sol#L170

Zero `decimals` can be used as an indication that a a `10**18` adjustment of the returned oracle price is needed.

Recommendation

It's recommended to consider removing the line that restricts zero decimals.

Client's commentary

> Fixed in `ee6a470`

L-4	Uncontrolled Slippage in `BridgeToken.sendSimple()`
Severity	Low
Status	Acknowledged

Description

- BridgeToken.sol#L154

In the parent contracts, the `_debit()` method is called with the `_minAmountLD` parameter to control slippage. However, in `BridgeToken.sendSimple()`, `_minAmountLD` is passed as zero.

Recommendation

We recommend allowing the user to pass `_minAmountLD` when calling the `sendSimple()` method.

Client's commentary

> Acknowledge and accept. The slippage check is handling a a precision loss of `decimalConversionRate` (1e6) which in our case is 1e-12 tokens. It is economically illogical to bridge an amount so small that this precision rounding has any meaningful effect. Additionally, we require that `amountReceivedLD > 0` to ensure a user doesn't accidentally try to bridge such a small amount that the precision loss converts it to nothing.

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