CURVE LENDING SECURITY AUDIT REPORT

Sep 02, 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

Curve Lending allows users to borrow crvUSD against any collateral token or to borrow any token against crvUSD, while benefiting from the soft-liquidation mechanism provided by LLAMMA. This innovative approach to overcollateralized loans enhances risk management and user experience for borrowers. Additionally, Curve Lending allows users to generate interest through lending (supplying) their assets to be borrowed by others.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Curve Finance
Project name	Curve Lending
Timeline	06 February 2024 - 31 May 2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
30.05.2023	c5169a7eb687a9878b989696a5c813dfc737e377	Previous audit commit
06.02.2024	c3f7040960627f023a2098232658c49e74400d03	Commit for the audit
14.03.2024	9e20913fb46db6d3774c56b13ba17d6911cb2caa	Commit for the re-audit
10.05.2024	c08a3ab8eb29d7622eddf432cb518eeec6f88b63	Final commit
31.05.2024	25fb794f1acea1e1d498fab41f6cab9cbdc565e7	Commit for the re-audit 2
29.07.2024	cd8476a7d80fdc2c61a59f44885cda1fdfbbc4a3	Commit for the re-audit 3

Project Scope

The audit covered the following files:

File name	Link

File name	Link
AMM.vy	AMM.vy
ControllerFactory.vy	ControllerFactory.vy
Controller.vy	Controller.vy
Stablecoin.vy	Stablecoin.vy
Stableswap.vy	Stableswap.vy
OneWayLendingFactory.vy	OneWayLendingFactory.vy
TwoWayLendingFactory.vy	TwoWayLendingFactory.vy
Vault.vy	Vault.vy
AggMonetaryPolicy2.vy	AggMonetaryPolicy2.vy
AggMonetaryPolicy3.vy	AggMonetaryPolicy3.vy
SemilogMonetaryPolicy.vy	SemilogMonetaryPolicy.vy
AggregateStablePrice2.vy	AggregateStablePrice2.vy
CryptoWithStablePrice.vy	CryptoWithStablePrice.vy
CryptoFromPool.vy	CryptoFromPool.vy
CryptoWithStablePriceFrxethN.vy	CryptoWithStablePriceFrxethN.vy
CryptoWithStablePriceTBTC.vy	CryptoWithStablePriceTBTC.vy
CryptoWithStablePriceWBTC.vy	CryptoWithStablePriceWBTC.vy
CryptoWithStablePriceWstethN.vy	CryptoWithStablePriceWstethN.vy
CryptoFromPoolVault.vy	CryptoFromPoolVault.vy
OracleVaultWrapper.vy	OracleVaultWrapper.vy

File name	Link
PegKeeper.vy	PegKeeper.vy
OneWayLendingFactoryL2.vy	OneWayLendingFactoryL2.vy
CryptoFromPoolArbitrum.vy	CryptoFromPoolArbitrum.vy
CryptoFromPoolsRateArbitrum.vy	CryptoFromPoolsRateArbitrum.vy
CryptoFromPoolsRate.vy	CryptoFromPoolsRate.vy
AggMonetaryPolicy.vy	AggMonetaryPolicy.vy
SecondaryMonetaryPolicy.vy	SecondaryMonetaryPolicy.vy

Deployments

Ethereum:mainnet

File name	Contract	Comment
OneWayLendingFactory.vy	0xeA6876...783205E0	The Factory issues extra approvals to the AMM. This is safe
AMM.vy	0xDf41E2...b7FC1659	AMM implementation
Controller.vy	0x4c5d4F...d4504112	Controller implementation. Minor changes from the latest commit
Vault.vy	0xc014F3...77805085	Vault implementation. Unnecessary functions were removed from the interface
SemilogMonetaryPolicy.vy	0x4863c6...DC87f2d3	Monetary policy implementation
CryptoFromPool.vy	0xC455e6...6444D3F8	Oracle implementation
LiquidityGauge.vy	0x79D584...bd35874D	Gauge implementation

File name	Contract	Comment
Market 0: wstETH/crvUSD		
AMM.vy	0x847D7a...e5157E64	Dynamic fees are not fully implemented. Fixed by setting high fees
Controller.vy	0x1E0165...38114D71	Minor changes
Vault.vy	0x8cf1DE...9EC319b5	Unnecessary functions were removed from the interface
SemilogMonetaryPolicy.vy	0x112E37...838eAf02	wstETH Market
CryptoFromPool.vy	0x5e2406...4e9Ef181	wstETH Market
Market 1: WETH/crvUSD		
AMM.vy	0xb46aDc...e9a46EdF	Dynamic fees are not fully implemented. Fixed by setting high fees.
Controller.vy	0xaade92...1995267b	Minor changes.
Vault.vy	0x5AE28c...1D7A0FEF	Unnecessary functions were removed from the interface.
SemilogMonetaryPolicy.vy	0xa6c73D...c18Fcf1F	
CryptoFromPool.vy	0x6530B6...9c285BD8	
Market 2: tBTC/crvUSD		
AMM.vy	0x5338B1...faed7BE9	Dynamic fees are not fully implemented. Fixed by setting high fees.
Controller.vy	0x413FD2...138C29Fc	Minor changes.
Vault.vy	0xb2b23C...fDC0AAC9	Unnecessary functions were removed from the interface.
SemilogMonetaryPolicy.vy	0xde31c3...9961aeEF	
CryptoFromPool.vy	0xeF42b6...a942692B	

File name	Contract	Comment
Market 3: CRV/crvUSD		
AMM.vy	0xafca62...D2acF11b	Dynamic fees are not fully implemented. Fixed by setting high fees.
Controller.vy	0xEdA215...3295110A	Minor changes.
Vault.vy	0xCeA18a...15C0f2cA	Unnecessary functions were removed from the interface.
SemilogMonetaryPolicy.vy	0x8b6527...dce57d07	
CryptoFromPool.vy	0xE0a4C5...D5ee38B8	
Market 4: CRV/crvUSD		
AMM.vy	0xe7B1c8...16DAA8e6	Dynamic fees are not fully implemented. Fixed by setting high fees.
Controller.vy	0xC510d7...b50CF3Fc	Minor changes.
Vault.vy	0x4D2f44...85598450	Unnecessary functions were removed from the interface.
SemilogMonetaryPolicy.vy	0x40A442...F84Dd6DD	
CryptoFromPool.vy	0xD4Dc9D...801435A8	
Market 5: WETH/crvUSD		
AMM.vy	0x08Ba6D...0240F9Cf	Dynamic fees are not fully implemented. Fixed by setting high fees.
Controller.vy	0xa5D913...7503bac8	Minor changes.
Vault.vy	0x46196C...f8654A45	Unnecessary functions were removed from the interface.
SemilogMonetaryPolicy.vy	0xbDb065...275f8352	
CryptoFromPool.vy	0x4f4B89...12a20443	

File name	Contract	Comment
Market 6: tBTC/crvUSD		
AMM.vy	0xfcb53E...39ED8805	Dynamic fees are not fully implemented. Fixed by setting high fees.
Controller.vy	0xe43865...E00621b8	Minor changes.
Vault.vy	0x99Cff9...E894bd30	Unnecessary functions were removed from the interface.
SemilogMonetaryPolicy.vy	0x62cD08...928b7D3C	
CryptoFromPool.vy	0x33A95b...70bC06a6	
Market 7: sUSDe/crvUSD		
AMM.vy	0x9bBdb1...1aDE8f4B	Dynamic fees are not fully implemented. Fixed by setting high fees.
Controller.vy	0x98Fc28...ADd96907	Minor changes.
Vault.vy	0x520965...438ED23b	Unnecessary functions were removed from the interface.
SemilogMonetaryPolicy.vy	0xF82A5a...bDd1365F	
CryptoFromPool.vy	0x50c39E...66e9b477	
Market 8: UwU/crvUSD		
AMM.vy	0x6BE658...417507b1	
Controller.vy	0x09dBDE...716ce16B	Old commit, minor changes
Vault.vy	0x7586C5...c5dA0f7E	Unnecessary functions were removed from the interface.
SemilogMonetaryPolicy.vy	0x905823...7b6837da	
CryptoFromPool.vy	0xBcda2a...f930A9F1	

File name	Contract	Comment
Market 9: WBTC/crvUSD		
AMM.vy	0x8eeDE2...Ab03F0C8	
Controller.vy	0xcaD85b...Aa0cE617	Old commit, minor changes
Vault.vy	0xccd37E...0e0D0063	Unnecessary functions were removed from the interface.
SemilogMonetaryPolicy.vy	0xE53C7e...1A50bA95	
CryptoFromPool.vy	0xE3ee57...AC288E2C	
Market 10: pufETH/crvUSD		
AMM.vy	0xcd28cF...63919836	
Controller.vy	0x4f8715...1cE0817C	Old commit, minor changes
Vault.vy	0xff467c...c89Fbe0d	Unnecessary functions were removed from the interface.
SemilogMonetaryPolicy.vy	0x2e478D...43dA359F	
CryptoFromPool.vy	0xb08eB2...75602cd9	

Arbitrum:mainnet

File name	Contract	Comment
OneWayLendingFactoryL2.vy	0xcaEC11...75922DeA	
AMM.vy	0xaA2377...47C65a6A	AMM implementation
Controller.vy	0xd5DCcB...35f91B97	Controller implementation. Old commit. Minor changes
Vault.vy	0x104e15...23B04d7a	Vault implementation. Unnecessary functions were removed from the interface

File name	Contract	Comment
CryptoFromPool.vy	0x57390a...3Dd8DfF8	Oracle implementation
SemilogMonetaryPolicy.vy	0x0b3536...c2c8f5C1	Monetary policy implementation
Market 0: WETH/crvUSD		
AMM.vy	0x38EB8A...Eb1F2bF2	
Controller.vy	0xB5B6f0...3B51F0A4	Old commit. Minor changes
Vault.vy	0x49014A...7BF6Cc4d	Unnecessary functions were removed from the interface
SemilogMonetaryPolicy.vy	0xEB9c27...048BD597	
CryptoFromPool.vy	0x4B24b0...83E4cd26	
Market 1: WBTC/crvUSD		
AMM.vy	0x12D1c9...11355Db0	
Controller.vy	0x013be8...dFfe6B68	Old commit. Minor changes
Vault.vy	0x60D38b...8cC506B1	Unnecessary functions were removed from the interface
SemilogMonetaryPolicy.vy	0xEdbbD4...8C8a68F7	
CryptoFromPool.vy	0x772dc3...8f716188	
Market 2: WBTC/crvUSD		
AMM.vy	0x772B6F...53Ee5c41	
Controller.vy	0x28c205...94d8898E	Old commit. Minor changes
Vault.vy	0xB50409...1FE2fcf8	Unnecessary functions were removed from the interface
SemilogMonetaryPolicy.vy	0xBcAbDE...CeE79559	

File name	Contract	Comment
CryptoFromPool.vy	0x9B053d...D8fa46bd	
Market 3: CRV/crvUSD		
AMM.vy	0x742089...fe419424	
Controller.vy	0x88f88e...704e47Ab	Old commit. Minor changes
Vault.vy	0xeEaF2c...CfDe6A32	Unnecessary functions were removed from the interface
SemilogMonetaryPolicy.vy	0x1F56Fb...f3D27B21	
CryptoFromPool.vy	0x20ee73...5B6538C9	
Market 4: ARB/crvUSD		
AMM.vy	0x33e5ea...445aEE09	
Controller.vy	0x76709b...22AE98f5	Old commit. Minor changes
Vault.vy	0x65592b...64e950e4	Unnecessary functions were removed from the interface
SemilogMonetaryPolicy.vy	0xc4fFBf...0341eCA0	
CryptoFromPool.vy	0x6341D0...6F9e696C	
Market 5: FXN/crvUSD		
AMM.vy	0x27dd80...1D4Bf026	
Controller.vy	0xAe659C...27A7a2c7	Old commit. Minor changes
Vault.vy	0xb56369...8B8d62B0	Unnecessary functions were removed from the interface
SemilogMonetaryPolicy.vy	0x0914E7...f4b7ACA7	
CryptoFromPool.vy	0x6EFE6D...F98e8835	

File name	Contract	Comment
Market 6: FXN/crvUSD		
AMM.vy	0xbEAC2f...3D627063	
Controller.vy	0x7Adcc4...0d176F1f	Old commit. Minor changes
Vault.vy	0xebA51f...B32E1ae7	Unnecessary functions were removed from the interface
SemilogMonetaryPolicy.vy	0x1e4D74...9269c379	
CryptoFromPool.vy	0xbB82bf...AA97970E	
Market 7: AssetVaultUSDC/crvUSD		
AMM.vy	0x134477...b962BFD4	
Controller.vy	0x4064Ed...24668b5D	Old commit. Minor changes
Vault.vy	0x241574...0151e4F8	Unnecessary functions were removed from the interface
SemilogMonetaryPolicy.vy	0x6a94FB...70AfD23c	
CryptoFromPool.vy	0x9f4864...DB6ea79B	

1.5 Summary of findings

Severity	# of Findings
Critical	1
High	2
Medium	5
Low	7

ID	Name	Severity	Status
C-1	An inflation attack allows for hard liquidations	Critical	Fixed
H-1	An inflated fee in the AMM leads to a partial AMM DOS	High	Fixed
H-2	An incorrect `last_tvl` state after the price pair removal	High	Fixed
M-1	Breaking gauge creation in lending factories	Medium	Acknowledged
M-2	The use of `tx.origin`	Medium	Acknowledged
M-3	`TwoWayLendingFactory.create_from_pool` does not work	Medium	Fixed
M-4	TwoWayLendingFactory price manipulation	Medium	Fixed
M-5	`AggregateStablePrice` EMA can be manipulated	Medium	Acknowledged
L-1	The delay between the EMA and instant price can accumulate bad debt	Low	Acknowledged
L-2	Additional Chainlink validation	Low	Acknowledged

L-3	No way to modify oracle in the AMM	Low	Acknowledged
L-4	EmaPriceOracle manipulation	Low	Acknowledged
L-5	`TwoWayLendingFactory.exchange()` griefing	Low	Fixed
L-6	Incorrect initialization of `USE_RATES` in `CryptoFromPoolsRate`	Low	Acknowledged
L-7	`RATE MAX SPEED` in `CryptoFromPoolsRateArbitrum` may fail in case of sequencer downtime	Low	Acknowledged

1.6 Conclusion

In this audit, we have examined various security and functionality aspects to ensure the robustness and reliability of the system. Our primary focus was on the lending features built on the existing codebase of the Curve stablecoin, particularly the interactions between vaults in the TwoWayLendingFactory and the broader implications of permissionless pool deployment. Based on our findings, we offer several suggestions to enhance the project's security posture and user experience.

Key Audit Vectors:

- **Reentrancy Attacks**: We checked for reentrancy attacks, especially when combining two pools within the TwoWayLendingFactory. Thanks to the implementation of `check_lock()` mechanisms and non-reentrant flags, we found no vulnerabilities in this area.
- **Vault Share Attacks and Inflation**: We analyzed potential attacks on vault shares, including inflation attacks and rounding errors, ensuring the integrity of asset valuation within the system.
- **Oracle Price Manipulation**: Our examination extended to potential manipulations of oracle prices, particularly the impact on share prices in two-way lending and issues arising from price volatility.
- **Interest Rate Accrual Accuracy**: We verified the correctness of interest rate accruals crucial for maintaining fair and predictable lending and borrowing conditions.
- **Factory Invariants and Common Issues**: Attempts were made to disrupt or exploit factory invariants, alongside a comprehensive check for common security issues prevalent in DeFi protocols.

Recommendations:

- **Enhanced Documentation for Users**: Given the permissionless nature of pool deployment, it is vital to improve documentation for users. Clarifications on the risks associated with the delay between the EMA Oracle and instant prices, especially in scenarios where a token's price might surge by 20% leading to potential pool insolvency, would be beneficial. Including simulation examples that detail these parameters and warnings against using unconventional tokens (e.g., tokens with fees, ERC-777, or rebaseable tokens) could significantly enhance user understanding and safety.
- **Code Comment Revisions**: Some code comments require updates for clarity, such as specifying that the `use_eth` parameter in Controller is now unused.

Notes on Fixes

The final commit includes two fixes aimed at addressing the issue of hard liquidations in cases of oracle manipulation.

To address the issue of manipulations through Inflation Attack for some oracles, a limit on the rate of price growth was introduced. It should be noted that such a limit will create a lag between the real and actual price of the asset. Therefore additional analysis is needed to ensure that this lag does not affect the delay in the PegKeeper's operation. If the PegKeeper is late in normalizing the price, some borrowers may face unfair liquidations.

To protect against two-block market manipulations, special dynamic fees were introduced, making the attack more costly. It should be noted that for some older AMMs, this feature was not implemented, so fees were manually increased as a hotfix. Dynamic and increased fees do not fully protect against manipulations but they significantly raise the cost of a two-block attack and make this vector even less realistic.

Another note concerns the incorrect operation of the `remove_price_pair()` function: since old contracts are not upgradeable, a workaround is expected to be used. One can remove several items from the end of the list and then add them back again without the one to be removed, all in a single transaction.

2. FINDINGS REPORT

2.1 Critical

C-1	An inflation attack allows for hard liquidations
Severity	Critical
Status	Fixed in 9e20913f

Description

In `TwoWayLending`, the share price can easily inflate through a direct transfer of funds to the `Controller`. This can be used to trigger hard liquidations:

1. A hacker buys the victim's collateral shares in the `AMM` using `exchange()`.
2. The hacker inflates the collateral price per share (even a small increase of +1.1% may be sufficient).
3. The victim's health becomes negative, and the hacker can liquidate it for profit.

Multiple tests show that the attack have been forwarded to the client.

In order to understand why it works, we'll have to take two features of the protocol into consideration.

Feature 1: If the oracle price increases by N percent, then the price of the tick `p_current_down()` increases by ~3.3 times.

That is, for example, if the price in the oracle increased by 10%, then the price grid in the `AMM` shifted not by 10%, but on average somewhere around 33%.

Feature 2: The `AMM` uses the `limit_p_o()` function which limits price surges and increases fees when the oracle price fluctuates.

For example, if the oracle price increases by 10% in a transaction (for example, from 1 to 1.1), then in the same transaction, `limit_p_o()` will return 1.1 (small fluctuations are not thresholded), and the dynamic fee will be high.

Over time, approximately after 120 seconds, the dynamic fee will drop to its minimum value.

Example: Let's see what happens after these operations are performed:

1. A user creates a loan with `create_loan()`.
2. A hacker buys all collateral in the `AMM` with `exchange()`.
3. The oracle price inflates!

As soon as the oracle price increases, say by 10%, the price in ticks increases by 33%. However, if the hacker decides to buy back their stablecoins right after this, they will, surprisingly, spend more collateral than they initially bought. It's because of the dynamic commission returned by `limit_p_o()`. It's large and compensates for the tick price increase.

However, after 120 seconds the commission drops to the minimum, and after that the hacker can buy back their stablecoins not just at market price, but at a much more favorable price (because the tick price increases by 3.3 times compared to the oracle).

Herein lie two problems.

Problem 1: The `health()` function does not account for dynamic fees and, therefore, in this scenario, is negative. In reality, the user's position remains healthy, at least while the dynamic fees are high, because if the hacker makes the reverse trade, they would add even more collateral to the user than was initially present. The `get_x_down()` function does not take this into account.

In particular, because of this, a problem arises that we can now hard liquidate the user and extract profit. Whereas for the hard liquidation, we don't need to make the reverse trade and spend the dynamic fee.

Problem 2: As the tick price grows three times faster than the oracle, bad debt may accumulate.

Suppose:

- We brought 100 collateral and borrowed 88 stablecoins.
- We buy back our collateral for 105 stablecoins.
- Suppose then the price increases by 10%.
- 120 seconds pass for the `limit_p_o()` fees to fall.

How much of the collateral can we sell back in the `AMM` to retrieve our stables? The collateral now costs 10% more on the market, but we can sell it at a price 33% higher in the `AMM`! That is, we buy back our 105 stablecoins for about 75 collateral.

But on the market, 75 collateral are worth 75*1.10 = 82 stablecoins?

Thus, it turns out we borrowed 88 stablecoins in the `AMM`, but then put back collateral that is worth 82 stablecoins, which is less.

In conclusion, the fact that the price in ticks changes by 3.3 times more than the oracle, combined with the user's health not accounting for dynamic fees, and the ability of a hacker to inflate the price of collateral (which are simply vault shares in the case of TwoWayLending), enables the hacker to profitably liquidate users whose positions should otherwise be considered healthy.

Recommendation

We recommend revisiting the logic of the `health()` function, not relying on `balanceOf()` while calculating `pricePerShare()` in vault's shares, and thinking of a way to make it impossible for a hacker to perform attacks such as `exchange(forth)+inflate+liquidate()` or `exchange(forth)+inflate+exchange(back)`, as the current implementation of the protocol, specifically the tick price rising three times higher than the oracle, is highly susceptible to such attacks.

Client's commentary

> 1. Added dynamic fee which fixes very related problem - 2-block attacks with ANY market in a similar fashion
> 2. Limited growth of pricePerShare to 1% a minute in 0c373156fb58ae89b6a8234d6ed3ff82eda82d4f

2.2 High

H-1	An inflated fee in the AMM leads to a partial AMM DOS
Severity	High
Status	Fixed in 9e20913f

Description

The `admin_fees_x` variable is not divided by `BORROWED_PRECISION` in AMM `withdraw()`:

```
# If withdrawal is the last one - transfer dust to admin fees
if new_shares == 0:
    if x > 0:
        self.admin_fees_x += x
    if y > 0:
        self.admin_fees_y += unsafe_div(y, COLLATERAL_PRECISION)
```

AMM.vy#L794

If `borrowed_token` is WBTC (decimals=8), then the error in fee accrual will be on the order of 10 magnitudes. A hacker could perform an inflation attack on the nearest available tick to trigger this piece of code and inflate `admin_fees_x` to the total amount of the `borrowed_token` balance in the AMM, while losing 10 magnitudes less funds than the final inflation amount. If the hacker then calls `collect_fees()`, which is a public method, all borrowed tokens from the AMM will be sent to `FACTORY.fee_receiver()`.

This will lead to a partial DOS of the AMM, as users will lose the ability to withdraw borrowed tokens from ticks, as there simply won't be any funds in the AMM for this.

There are a few notes on this:

1. **This attack does not depend on `ADMIN_FEE`**; the code is always activated when there is dust in the tick. In order to execute the attack, the hacker needs to inflate the share price in the tick, causing the dust to have a large value.
2. AMM uses dead shares, but price inflation is still possible via the `exchange()` method or other means. It's just not profitable for the hacker.
3. Currently, `collect_fees()` reverts as `Vault` does not have a `fee_receiver()` method which is called by the `collect_fees()` method. Still, if the `collect_fees()` revert issue is addressed by introducing a `fee_receiver()` in the factory, then the fee inflation bug will arise.

Recommendation

We recommend adding the missing division by the `BORROWED_PRECISION`:

```
self.admin_fees_x += unsafe_div(x, BORROWED_PRECISION)
```

Client's commentary

> Fixed in 3336ed838f8ba90490155d7401c4c4eb96824b5c

H-2	An incorrect `last_tvl` state after the price pair removal
Severity	High
Status	Fixed in cd8476a7

Description

- AggregateStablePrice2.vy#L108

The `AggregateStablePrice` contract doesn't remove the corresponding entry in the `last_tvl` array during the `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

> Old contracts are not upgradeable, thus, a workaround will be used: one can remove several items from the end of the list and then add them back again without the one to be removed, all in a single transaction.

2.3 Medium

M-1	Breaking gauge creation in lending factories
Severity	Medium
Status	Acknowledged

Description

Lending factories have public `deploy_gauge()` to deploy a gauge for an chosen vault:

- OneWayLendingFactory.vy#L326-L343
- TwoWayLendingFactory.vy#L371-L388

There are two problems there:

1. One gauge per vault. Only the first `deploy_gauge()` caller can create a gauge for a vault.
2. The caller of `deploy_gauge()` receives rights in the created gauge. This caller as a `manager` can add 8 malicious reward tokens through `add_reward()`.

These two things together will permanently block adding new valid reward tokens and creating the correct gauge (even if a manager role is reset).

Recommendation

Consider a few options:

1. Allow multiple gauges for every vault and not revert if a gauge for a vault already exists. It will allow ignoring malicious gauges.
2. Create a gauge for a vault right after vault deployment inside `_create()`.
3. Ensure that users can create vaults and deploy gauges atomically in one transaction.

Client's commentary

> This is a known issue (or nonissue) in ALL the gauges. If ever is a problem - can create in a single tx

M-2	The use of `tx.origin`
Severity	Medium
Status	Acknowledged

Description

Liquidity gauge `manager` is set as a `tx.origin` of the gauge deploy transaction.

- LiquidityGauge.vy#L176

It is not recommended to have `tx.origin` in access control logic.
Some DeFi users are Multisigs or Account Abstraction wallets.
In such cases, `tx.origin` is not correct final user identification.

Recommendation

We recommend having `manager` as a customizable input for `factory.deploy_gauge()`.

Client's commentary

> This is a known issue (or nonissue) in ALL the gauges.

M-3	`TwoWayLendingFactory.create_from_pool` does not work
Severity	Medium
Status	Fixed in 9e20913f

Description

- TwoWayLendingFactory.vy#L249

The `TwoWayLendingFactory.create_from_pool` method always returns an error with `CryptoFromPoolVault`. It happens so because `vault.borrowed_token()` was not initialised at the time of validation:

```
assert pool.coins(collateral_ix) == vault.borrowed_token()
```

- CryptoFromPoolVault.vy#L37

Recommendation

We recommend passing an initialised Vault to the `CryptoFromPoolVault` or moving the check to Factory.

Client's commentary

> Indeed, and there is enough validation in the factory already regardless -> removing in 069b3886f90cb49903ab1bf0e1af80fd70ead710.

M-4	TwoWayLendingFactory price manipulation
Severity	Medium
Status	Fixed in 9e20913f

Description

- CryptoFromPoolVault.vy#L65
- OracleVaultWrapper.vy#L39

These oracles are used in `TwoWayLendingFactory`. If you try to influence the price via `controller_long`/`controller_short`, there will be a `check_lock` check (Vault.vy#L266).

However, an attacker still has the ability to influence the price via a transfer to the controller. For example:

```
print(amm_short.price_oracle()) # 1000000000000000
borrowed_token.transfer(controller_long, amount)
print(amm_short.price_oracle()) # 1250000000000000
```

This can be advantageous if there are few funds in the `Controller` and it is very easy to influence the price.

Recommendation

We recommend using a more stable oracle to price the LP Vault.

Client's commentary

> Limited growth of pricePerShare to 1% a minute in 0c373156fb58ae89b6a8234d6ed3ff82eda82d4f.

M-5	`AggregateStablePrice` EMA can be manipulated
Severity	Medium
Status	Acknowledged

Description

- AggregateStablePrice2.vy#L167-L168

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 to check if the `price_w()` function has not been called for a long time.

Client's commentary

> This is indeed true. However, MA time is really large, and the same aggregator is used for multiple markets. So price_w is called really often. Nevertheless, not a bad idea to do the activity checking, as recommended.

2.4 Low

L-1	The delay between the EMA and instant price can accumulate bad debt
Severity	Low
Status	Acknowledged

Description

The EMA does not follow the market instantly, resulting in a delay between the actual price and the oracle in Curve. Hypothetically, a situation could arise where the market price jumps so significantly that it becomes profitable to borrow this token at a lower price in the Curve lending and sell it at a higher price on the market. This can be done in a flash loan, instantly deleting the Vault and leaving the protocol with bad debt.

Recommendation

We recommend disallowing borrowing (using `create_loan()`) if the momentary price significantly differs from the EMA.

Client's commentary

> This is a design decision and fundamental to the algorithm.

L-2	Additional Chainlink validation
Severity	Low
Status	Acknowledged

Description

Chainlink feed data have edge cases that are recommended being covered. The current issues are:

- Stale thresholds are not implemented in some oracles (`CryptoWithStablePriceAndChainlink`, `CryptoWithStablePriceAndChainlinkFrxeth`);
- Stale threshold is 24 hours now. But the Chainlink price synchronizes for low volatile pairs like stETH/ETH can update with the stale threshold slightly above 24 hours. In this case, the feed is not stale;
- `updateTime != 0` is not checked (it means that the round is not complete);
- `answeredInRound >= roundId` is not checked (it can additionally indicate that the price is stale).

Recommendation

Consider the following improvements:

1. Implement stale price checks for `CryptoWithStablePriceAndChainlink` and `CryptoWithStablePriceAndChainlinkFrxeth`
2. Consider updating `CHAINLINK_STALE_THRESHOLD` to 24.5 hours in cases of low volatile feeds
3. Require that `updateTime != 0`
4. Require that `answeredInRound >= roundId`

Client's commentary

> That is absolutely true, however using chainlink introduce other issues, so we don't use it. It is, however, possible to make EMA oracle out of chainlink oracles. Gas costs for that are prohibitive on Ethereum mainnet, but totally acceptable on L2s

L-3	No way to modify oracle in the AMM
Severity	Low
Status	Acknowledged

Description

- AMM.vy#L127

There is currently no way to change `price_oracle` in the AMM.

```
price_oracle_contract: public(immutable(PriceOracle))
```

Recommendation

`AMM.price_oracle` is external, it may be worth keeping the ability to change it.

Client's commentary

> This is correct and design decision to not have the admin (even DAO) be able to rug everything

L-4	EmaPriceOracle manipulation
Severity	Low
Status	Acknowledged

Description

EMAPriceOracle is a wrapper that adds an Exponential Moving Average (EMA) to another price source.

When it is deployed or the protocol has not been used for some time, the EMA prefers new values from the price source over old ones:

```
alpha: uint256 = self.exp(- convert((block.timestamp - last_timestamp)
    * 10**18 / MA_EXP_TIME, int256))
return (current_price * (10**18 - alpha) + last_price * alpha) / 10**18
```

EmaPriceOracle.vy#L113-L115

At such times, a hacker can shift the price from the source for just one second and record a new price value in EMAPriceOracle. Although the real price source will return to its original value immediately, the EMAPriceOracle will need about `4 * MA_EXP_TIME` seconds to catch up. Until then, it will give a deviated price, which could be used for arbitrage and creating bad debt.

Recommendation

We recommend preferring old values over new ones when calculating the EMA.

Client's commentary

> Wrapper is not to be used: now we use "native" EMA oracle in pools which is not affected by such manipulations.

L-5	`TwoWayLendingFactory.exchange()` griefing
Severity	Low
Status	Fixed in c08a3ab8

Description

- TwoWayLendingFactory.vy#L589
- TwoWayLendingFactory.vy#L612

It is possible to cause a temporary DOS in the `exhange()` and `exchange_dy()` functions in TwoWayLendingFactory by directly transferring tokens to the contract.

For example, the following line reverts if the right side of subtraction is higher:

```
dxy[0] = max_in - self.transfer_out(vault, other_vault, i, receiver)
```

The attack scenario:

Suppose a user calls the function: `TwoWayLendingFactory.exchange_dy(vault_id, i=1, j=0, amount, max_in, receiver)`.

Under normal operation, an amount of collateral tokens will be deducted from the user, converted into shares:

```
_max_in: uint256 = self.transfer_in(
        vault, other_vault, i=1, msg.sender, max_in
    )
        assert token.transferFrom(_from, self, amount,
            default_return_value=True)
        ...
            return other_vault.deposit(amount)
```

Then exchanged for borrowed token in favor of the receiver:

```
    dxy: uint256[2] = self.amms[vault_id].exchange_dy(
        i, j, _amount, _max_in, _receiver
        )
```

And, finally, all the remaining unspent collateral will be returned:

```
dxy[0] = max_in - self.transfer_out(vault, other_vault, i=1, receiver)
```

In our case, the `transfer_out()` function will return:

```
other_vault.redeem(other_vault.balanceOf(self), _to)
```

The exploiter sandwiches the transaction:

1. Before the user's transaction, they deposit an extra amount of shares into TwoWayLendingFactory.
2. Then the user's transaction is executed, but it reverts at this line because the contract's balance now contains not only the user's trade remainder but also the extra amount of shares deposited by the exploiter:

```
max_in - self.transfer_out(vault, other_vault, i=1, receiver)
```

3. Afterwards, the exploiter withdraws their funds using an empty trade with `max_in=infinity`.

Thus, the exploiter can always revert the trader's transaction. No one loses money (except for gas).

Also, there's another attack variant without sandwiching: the exploiter simply deposits different tokens into TwoWayLendingFactory, say, worth $1-100. This reverts all trades with a small slippage, until a trade with a big possible slippage occurs.

Recommendation

We recommend taking into account a scenario where funds are already available on the contract before any trades.

L-6	Incorrect initialization of `USE_RATES` in `CryptoFromPoolsRate`
Severity	Low
Status	Acknowledged

Description

- CryptoFromPoolsRate.vy#L93-L97

Rates are excluded from future use if `stored_rates()` returns `10**18` during initialization:

```
u: bool = False
for r in stored_rates:
    if r != 10**18:
        u = True
use_rates.append(u)
```

This may be incorrect if `stored_rates()` changes over time.

Recommendation

We recommend using all rates, as they may deviate from `10**18` over time.

Client's commentary

> It is assumed that the deployer validates the oracle after deployment

L-7	`RATE_MAX_SPEED` in `CryptoFromPoolsRateArbitrum` may fail in case of sequencer downtime
Severity	Low
Status	Acknowledged

Description

- CryptoFromPoolsRateArbitrum.vy#L144-L150

If the sequencer is down and the difference `block.timestamp - self.cached_timestamp` gets too high, the protection against inflation attacks using `RATE_MAX_SPEED` may be broken.

Recommendation

We recommend taking into account the case of a non-functioning sequencer when using `RATE_MAX_SPEED`.

Client's commentary

> Good point, needs to be addressed

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