INSTADAPP FLUID DEX SECURITY AUDIT REPORT

Dec 04, 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

Fluid DEX is a decentralized exchange (DEX) protocol that is built on top of the Liquidity Layer. The protocol introduces Smart Debt and Smart Collateral mechanisms to optimize capital efficiency. By leveraging these features, Fluid DEX aims to increase liquidity provision and potentially enhance trading efficiency. The platform combines elements of Uniswap v2 and v3 to offer flexible pool configurations.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Instadapp Fluid
Project name	Fluid Dex
Timeline	09.10.2024 - 21.10.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
09.10.2024	9382e3c35b8a5c60984f529dca680974e7f340ac	Commit for the audit

Project Scope

The audit covered the following files:

File name	Link
poolT1/common/variables.sol	variables.sol
poolT1/common/constantVariables.sol	constantVariables.sol
poolT1/coreModule/helpers/userHelpers.sol	userHelpers.sol
poolT1/coreModule/helpers/coreHelpers.sol	coreHelpers.sol

File name	Link
poolT1/coreModule/helpers/secondaryHelpers.sol	secondaryHelpers.sol
poolT1/coreModule/events.sol	events.sol
poolT1/coreModule/immutableVariables.sol	immutableVariables.sol
poolT1/coreModule/interfaces.sol	interfaces.sol
poolT1/coreModule/structs.sol	structs.sol
poolT1/coreModule/core/perfectOperationsAndSwapOut.sol	perfectOperationsAndSwapOut.sol
poolT1/coreModule/core/shift.sol	shift.sol
poolT1/coreModule/core/main.sol	main.sol
poolT1/coreModule/core/colOperations.sol	colOperations.sol
poolT1/coreModule/core/debtOperations.sol	debtOperations.sol

Deployments

File name	Contract deployed on mainnet	Comment
colOperations.sol	0x2F9B396255e681574d26Fe466DE93A9dff2567a6	wstETH_ETH Pool
debtOperations.sol	0xF7c62a231088c2bABB32282bCf14e63DB3484b82	wstETH_ETH Pool
perfectOperationsAndSwapOut.sol	0xA512bDD83F9A81e2fbC4e24b54B9f5c642D5e025	wstETH_ETH Pool
colOperations.sol	0xfE34b33c6c8f2b44B12C18d88F4DfA7f4Cd7f074	USDC_USDT Pool

File name	Contract deployed on mainnet	Comment
debtOperations.sol	0x7c4ec0af359D8bc36565d91CdED7EB45F697ef8d	USDC_USDT Pool
perfectOperationsAndSwapOut.sol	0xf36Cb3226a40897eE98Eb5C77C347eeC1e726E19	USDC_USDT Pool
colOperations.sol	0xC1559fF463cfB75A09D7c09A62421e05D892609F	WBTC_cbBTC Pool
debtOperations.sol	0xA53eE17947590c4ce0A3C1C6ec66aEd9174F3fEa	WBTC_cbBTC Pool
perfectOperationsAndSwapOut.sol	0x8942beD2E25AA383178731c94C74Aa44Bd22d859	WBTC_cbBTC Pool
colOperations.sol	0x54759D4101F56e38E08a1406A08e35DD07dfdAD2	GHO_USDC Pool
debtOperations.sol	0x810148454056Cd8fB0AEd5157De1f4a6A73DDCcC	GHO_USDC Pool
perfectOperationsAndSwapOut.sol	0x32C2C4DDA4aE1620891ebDb72c1c90363012A60C	GHO_USDC Pool
colOperations.sol	0x4697eB7C234469AcE7EaE4c1c5d5Ad08C8104bdc	USDC_ETH Pool
debtOperations.sol	0x05FED1069A92ED377E1521050B7954bFa8fA7B00	USDC_ETH Pool
perfectOperationsAndSwapOut.sol	0x45316860c990De706A87Ca25106eA45FFd10b146	USDC_ETH Pool
colOperations.sol	0x2997C259a97d1474B0477C9478B09FF628812A55	WBTC_ETH Pool
debtOperations.sol	0xD766a81D376f3c1607D66c9086BBC1074B3B2e81	WBTC_ETH Pool
perfectOperationsAndSwapOut.sol	0xF3b79e312A3ba8F36B69515Eeae55a3e63197e93	WBTC_ETH Pool
colOperations.sol	0x397a7397eb4E60B1D38C6C6428Aa3a446Ea78631	cbBTC_ETH Pool

File name	Contract deployed on mainnet	Comment
debtOperations.sol	0x31e5ce17754Dd7d2547faaB7C148d8FacAa465e2	cbBTC_ETH Pool
perfectOperationsAndSwapOut.sol	0x47b7C021CDc90F7f2661a398669a33527AcCdD50	cbBTC_ETH Pool
colOperations.sol	0xA37713E827B7859f1909254dd0e1C685ADD123dC	USDe_USDC Pool
debtOperations.sol	0xc3BB75BB2d599E23308a9a9Ea9B7d9E99ADc4854	USDe_USDC Pool
perfectOperationsAndSwapOut.sol	0x2d7C3AA440f61f0Ec479A626B799D0FedecB7943	USDe_USDC Pool
colOperations.sol	0x838BF9de6A9890f8Bc46c1DF0C221c0ae5c8ef06	weETH_ETH Pool
debtOperations.sol	0x49675179133A8dbF04A7F0A6f3c69B2420D606cA	weETH_ETH Pool
perfectOperationsAndSwapOut.sol	0x4DC5249f0DC7B6E230bCF7F56c1d4Ce2Fe08d6FE	weETH_ETH Pool
colOperations.sol	0xB687B7bf8Ee13dB379757C46CE2bfF3D0164cd24	INST_ETH Pool
debtOperations.sol	0x1CB7BAffC349Bf1367766182bb04169655200D5a	INST_ETH Pool
perfectOperationsAndSwapOut.sol	0xA8d4491A89dbC8AbFd1e63fe1d272809833DdCD7	INST_ETH Pool

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	0
Low	3

ID	Name	Severity	Status
L-1	Revert without custom error	Low	Acknowledged
L-2	Misleading comment	Low	Acknowledged
L-3	Revert for `revenueCut>1%`	Low	Acknowledged

1.6 Conclusion

Fluid DEX is a protocol that combines the features of a decentralized exchange (DEX) and a lending platform, built on top of a liquidity layer.

The following attack vectors were analyzed:

- Extensive attention was given to checking various invariants. We did not find any user operations that would allow significant deviations from the expected equality within one block: `supply pool token0 / token1 = debt pool token1 / token0`. The invariant may diverge over time, providing an arbitrage opportunity for a swap, which immediately restores the invariant.
- Users can only withdraw up to the withdrawal limit value and borrow up to the borrowing limit value. These limits are updated correctly after each user action related to lending functionality.
- Users cannot perform an inflation attack, as the admin mints virtual shares during pool deployment.
- Functions accessible by users are protected against reentrancy, preventing malicious actions through multiple entries.
- The DEX implements rounding operations in a way that prevents user exploitation by ensuring that calculations consistently favor the platform. This rounding approach protects the DEX from potential losses due to rounding errors and ensures that users cannot manipulate token amounts to gain an unfair advantage.
- Users cannot swap repeatedly to extract value or gain additional profit from the liquidity layer.
- Users cannot repay less than the amount they borrowed.
- When users call `FluidDexT1PerfectOperationsAndSwapOut.swapOut()` with ETH and send more value than required, the excess amount is correctly refunded to them.

We also followed our detailed checklist, covering other aspects such as business logic, common ERC20 issues, interactions with external contracts, integer overflows, reentrancy attacks, access control, typecasting pitfalls, share calculations, rounding errors, and other potential vulnerabilities.

No significant vulnerabilities were identified.

The contracts are of high quality, gas-optimized, and well-documented. The implementation demonstrates a solid understanding of best practices, although there are a few areas where improvements could be made for consistency and security.

Key notes and recommendations:

- In some parts of the code, specific constants are used, but in others, magic numbers appear. For clarity, readability, and ease of maintenance, we recommend consistently using named constants for all numeric values instead of hardcoding them directly into the logic.
- Some sections of the code are susceptible to underflow or overflow errors due to the lack of proper sanity checks. For example, if a user tries to repay with one token when liquidity is insufficient, an underflow can occur. Implementing more comprehensive input validation and boundary checks, along

with named error messages, would improve user experience and help clarify why a specific revert occurred.

- The logic for calculating the health factor and handling liquidations is not present within the DEX itself but is managed by the vault, which is out of scope for this audit.
- Oracle data is accurately written to storage. When a user calls `FluidDexT1.oraclePrice()`, the price is calculated based on this stored data.
- The pool price is updated correctly after each swap, maintaining the protocol's main invariant (that collateral and debt pool prices remain synchronized).
- The center price cannot exceed the limits set by the admin. The shifting mechanism used to update ranges is implemented correctly.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Revert without custom error
Severity	Low
Status	Acknowledged

Description

A revert occurs in `UserHelpers._depositOrPaybackInLiquidity()` when both amounts (**depositAmt_** and **paybackAmt_**) are greater than zero, but no custom error message is provided to indicate the cause of the revert.

userHelpers.sol#L126

Recommendation

We recommend adding a custom error message for this specific revert case. This will improve debugging and user experience by providing clear feedback on why the operation failed. It will also allow developers to trace and handle this issue more effectively in case of future updates or expansions of the codebase.

Client's commentary

> Make sense. Although, this revert will never happen as we never sent both amounts together here but better alert or removing it all together would have been better.

L-2	Misleading comment
Severity	Low
Status	Acknowledged

Description

In the function `FluidDexT1PerfectOperationsAndSwapOut.depositPerfect()`, a comment is repeated, and in one of the two instances, it is misleading.

```
// Adding + 1, to keep protocol on the winning side
```

1. perfectOperationsAndSwapOut.sol#L430

2. perfectOperationsAndSwapOut.sol#L439

Recommendation

We recommend removing the misleading comment.

Client's commentary

> Agreed!

L-3	Revert for `revenueCut>1%`
Severity	Low
Status	Acknowledged

Description

Setting the `revenueCut` to a value >1% by the admin causes the `swapIn()` and `swapOut()` functions to revert due to a check in the liquidity layer's `operate()` function:

```
memVar_ > (memVar2_ *
    (FOUR_DECIMALS + MAX_INPUT_AMOUNT_EXCESS))
    / FOUR_DECIMALS
```

main.sol#L378

For example, when both `swapFee` and `revenueCut` are set to 10%, the `revenueCut` reduces the `temp_` value, which is then passed to liquidity layer's `operate()` as `supplyAmount`. If the amount the contract receives differs from `supplyAmount` by more than 1%, the transaction reverts.

Steps to Reproduce:

1. Add the code below to the `pool.t.sol` file.
2. Run `forge t --mt test_swap_RevenueCutRevert -vvvv`.

```
function test_swap_RevenueCutRevert() external {
    deal(address(DAI), alice, 1_000_000e18);
    deal(address(USDC), alice, 1_000_000e6);

    deal(address(DAI), bob, 10_000_000e18);
    deal(address(USDC), bob, 10_000_000e6);
    _makeUserContract(alice, true);
    _makeUserContract(bob, true);

    DexParams memory dexPool_ = DAI_USDC;
    DexType dexType_ = DexType.SmartCol;
    FluidDexT1 dex_ = _getDexType(dexPool_, dexType_, false);

    vm.prank(admin);
    FluidDexT1Admin(address(dex_)).updateFeeAndRevenueCut(100_000, 100_000);
    DexVariables2Data memory d2_ = _getDexVariables2Data(dex_);
    assertEq(d2_.fee, 100000); // 10%
    assertEq(d2_.revenueCut, 10); // 10%

    vm.startPrank(alice);

    vm.expectRevert();
    dex_.swapIn(true, 100e18, 0, alice);

    vm.expectRevert();
    dex_.swapOut(true, 90e6, 105e18, alice);

    vm.stopPrank();
}
```

Recommendation

We recommend accounting for the `revenueCut` in `operate()`.

Client's commentary

> This is expected as we have added a 1% check on Liquidity Layer that amount should not be more than 1%. So we cannot charge a revenue more than 1%.
>
> However, if there is a need to charge more than 1% then we can update Liquidity Layer and increase the percent to let's say 2%.
>
> We think 1% should be more than enough and we will never require to increase it more.

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