NAPIER V1 SECURITY AUDIT REPORT

Jul 03, 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

Napier is a fixed-rate DeFi protocol that allows to:

- Swap large positions on the AMMs for traders;
- Get higher APY for liquidity providers.

The audited scope Napier v1 consists of the core part, implementing deployment of `tranches` with a given DeFi adapter and maturity timestamp, `yield token` linked to the `tranche`, and some `adapters` establishing connection between the core part and external DeFi projects.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Napier
Project name	Napier v1
Timeline	27.11.2023 - 02.07.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
27.11.2023	3833f3965990e23f13df3c86584d1f9e2b575dfb	Initial commit for the audit
20.03.2024	5b344ab2c7fea509900959a325a2e8f78df96b5f	Some functionality added by the developers
12.04.2024	d9772f14daddd6731ea55c70b34e6015ac730be5	Fixes after the reaudit
19.04.2024	c4661d20270cd915b96278c5e0c0e02d4fb28e7b	Additional fixes
30.05.2024	f548e64514856fc6624f6c7dc3b235cfc8bff8dd	Additional fixes

Project Scope

The audit covered the following files:

File name	Link

File name	Link
src/Authorizable.sol	Authorizable.sol
src/BaseAdapter.sol	BaseAdapter.sol
src/BaseToken.sol	BaseToken.sol
src/Constants.sol	Constants.sol
src/Create2TrancheLib.sol	Create2TrancheLib.sol
src/TrancheFactory.sol	TrancheFactory.sol
src/Tranche.sol	Tranche.sol
src/YieldToken.sol	YieldToken.sol
src/adapters/aaveV3/AaveV3Adapter.sol	AaveV3Adapter.sol
src/adapters/compoundV2/CompoundV2BaseAdapter.sol	CompoundV2BaseAdapter.sol
src/adapters/compoundV2/WrappedCETHAdapter.sol	WrappedCETHAdapter.sol
src/adapters/morphoAaveV3/MA3WETHAdapter.sol	MA3WETHAdapter.sol
src/utils/DateTime.sol	DateTime.sol
src/utils/SafeERC20Namer.sol	SafeERC20Namer.sol
src/adapters/BaseLSTVault.sol	BaseLSTVault.sol
src/adapters/BaseLSTAdapter.sol	BaseLSTAdapter.sol
src/adapters/frax/SFrxETHAdapter.sol	SFrxETHAdapter.sol
src/adapters/lido/StEtherAdapter.sol	StEtherAdapter.sol
src/adapters/rocketPool/RETHAdapter.sol	RETHAdapter.sol
src/adapters/rocketPool/Swapper.sol	Swapper.sol

File name	Link
src/adapters/EETHAdapter.sol	EETHAdapter.sol

Deployments

File name	Contract deployed on mainnet	Comment
TrancheFactory	0x83CE9e...D4ee12e3	
Create2TrancheLib	0x09aaa7...0cb24e64	Tranche bytecode inside
StEtherAdapter	0x497E3B...16C7b312	
SFrxETHAdapter	0x3EC0AC...0Fae692d	
RETHAdapter	0x4ab76d...B6039584	
Swapper	0xe2166E...3255850F	
Tranche	0xdafff5...1dd2964b	ERC20.name=PT-eStETH@30-12-2024
Tranche	0x6fd958...6c186231	ERC20.name=PT-eFrxETH@30-12-2024
Tranche	0x79a585...17204857	ERC20.name=PT-erETH@30-12-2024

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	8
Low	7

ID	Name	Severity	Status
M-1	The owner is able to withdraw pool share tokens from specific adapters	Medium	Fixed
M-2	Revert with the 100% tilt value	Medium	Fixed
M-3	Lack of validation in the Tranche setter	Medium	Fixed
M-4	Lack of QoS on redemptions, potential DoS attacks	Medium	Acknowledged
M-5	Centralization risks	Medium	Fixed
M-6	A linear increase in the withdrawal wait time for Lido for requests exceeding 500 ETH	Medium	Fixed
M-7	A potential blocking issue in the `RETHAdapter.withdrawAll` function	Medium	Fixed
M-8	Incorrect handling of pending requests in the `requestWithdrawal` function	Medium	Fixed
L-1	Redudant inheritance of `Authorizable` in `BaseAdapter`	Low	Fixed
L-2	Lack of fee control	Low	Fixed

L-3	Gas optimization tips	Low	Fixed
L-4	Incorrect comments	Low	Fixed
L-5	A redundant function call in `SFrxETHAdapter`	Low	Fixed
L-6	Dynamic stake limit configuration in `BaseLSTAdapter`	Low	Fixed
L-7	The silent behavior on attempting withdraw of small amount of Ether	Low	Fixed

1.6 Conclusion

The project code is well-written. We have performed investigation on potential attack vectors (enlisted below) and found out that the project isn't vulnerable to the most of them:

1. Donation manipulation and inflation attack on `AaveV3Adapter`:

   - The use of the latest OpenZeppelin implementation of `ERC4626` in `AaveV3Adapter` effectively prevents theft through inflation attacks, as creating 1 virtual share does not let the deposits of other users to be stolen.
   - Increasing the exchange rate with a single share by donation is economically irrational meaning that the attacker's funds will not come back. This approach does not lead to user losses, and in the event of any issues, the adapter can be redeployed.
   - Until the fix of `M1` is applied, the tokens of the attacker remain redeemable by the `owner`.

2. Scale manipulation attacks:

   - To significantly alter the scale, an attacker would have to irreversibly transfer their tokens to `AaveV3Adapter`. Moreover, the transfers to the adapter result in the distribution of transferred tokens among other `YieldToken` holders as `unclaimedYield` value accumulation.

3. Solvency of mathematical model:

   - The solvency and consistency of the model have been confirmed. When issuing, a specified amount `y` of `yield` and `principal` tokens are minted for the user, and the submitted `underlying` tokens are converted to the `x` amount of `target` tokens. It was verified that during redemption, burning `y` `yield` tokens along with `y` `principal` tokens results in the return of the `x` amount of `target` tokens subtracting issuance fees and precision errors.
   - The adapter's scale does not affect this invariant.

4. Cross-contract reentrancy attacks:

   - `Tranche` methods being `nonReentrant` reduce the risk of reentrancy attacks. The potential for reentrancy exists only with the `ERC-777` `underlying` tokens.
   - Methods such as `redeem`, `withdraw`, and `redeemWithYT` are secured against reentrancy as they call `previewWithdraw` at the end of their executions, following state changes.
   - The `issue` method is identified as potentially vulnerable due to the state of `Tranche` which is left incomplete during the transfer. However, currently, no adapters support ERC-777, mitigating immediate risks.
   - Recommendations include either prohibiting the use of the `ERC-777` `underlying` tokens within the project or altering the `Tranche` contract implementation to perform deposits to the adapter before state changes in order to prevent future attack vectors with new adapter implementations.

5. Other attacks, also successfully addressed by the good quality of the code: insolvency during yield token transfers, interception of deposit/withdrawal of prefund-* functions, manipulation by `scale` and `maxScale`, typical deployment procedure attacks, typical proxy-related attacks.

As some issues have been discovered, they are enlisted below.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	The owner is able to withdraw pool share tokens from specific adapters
Severity	Medium
Status	Fixed in 0a705e2b

Description

A centralization concern has emerged within the `AaveV3Adapter`, `WrappedCETHAdapter`, and `MA3WETHAdapter` contracts. These adapters, inheriting from the `ERC-20` tokens, serve as `target` tokens assigned to `Tranche`. Contrarily, the adapters maintain on their balance the pool shares from connected protocols, into which the target tokens are converted. This architecture leads to a possibility inherited from the `BaseAdapter` logic, where the owner might have the capability to withdraw these pool shares from the adapters.

This issue is rated as `medium` severity as such a withdrawal could potentially disrupt the logic and financial stability of the specified adapters, but may be triggered only by the authorized account (owner).

Related code - the `RecoverERC20` function in `BaseAdapter`: BaseAdapter.sol#L34.

Recommendation

We recommend disabling the redemption of pool shares from the specified adapters.

M-2	Revert with the 100% tilt value
Severity	Medium
Status	Fixed in c3e0b04d

Description

According to the logic design, 100% tilt is the normal value and the Tranche contract should work with it TrancheFactory.sol#L62

but according to the codebase

Tranche.sol#L645,

it doesn't because of the division by zero in the `_computePrincipalTokenRedeemed` function which is used by the `withdraw` function.

We marked this issue as MEDIUM because in case of using the 100% tilt value, there is a risk of logic flow break but there are other options to withdraw assets.

Recommendation

We recommend you adding a check for the 100% tilt value.

M-3	Lack of validation in the Tranche setter
Severity	Medium
Status	Fixed in 16b9338c

Description

The BaseLSTVault.sol#L65 function allows setting the arbitrary Tranche address. However, changing that address will break system functionality. Additionally, the address of the Tranche could be validated using the `Tranche.adapter()` view function.

Recommendation

We recommend performing additional checks against the Tranche address in the `setTranche` function and disallowing to change the address once it has been set.

M-4	Lack of QoS on redemptions, potential DoS attacks
Severity	Medium
Status	Acknowledged

Description

The adapters inherited from the `BaseLSTAdapter` abstract contract are susceptible to a Denial-of-Service (DoS) attack that could significantly hinder user redemptions before the maturity timestamp. A malicious actor could exploit this vulnerability to block legitimate redemptions by repeatedly performing the following actions:

1. **Deposit Manipulation:** Deposit an amount of ETH equal to `(totalAssets * targetBufferPercentage) / (1 - targetBufferPercentage)`. This strategically chosen amount maximizes buffer drain.
2. **Immediate Redemption:** Immediately redeem the deposited ETH back from the adapter, effectively draining the buffer.

By continuously repeating these steps, an attacker could deplete the adapter's buffer, preventing other users from redeeming their funds until the buffer replenishes through new deposits or rebalancer withdrawals. While this attack incurs transaction fees for the malicious actor, it can still disrupt the functionality of the adapter for legitimate users.

Additionally, even if no malicious activity is performed against the adapters, users are forced to compete for the capacity of the withdrawal buffer without fair competition rules. Unsuccessful withdrawal requests are just reverted, forcing users to re-join the competition, leading to inefficient gas usage and user inconvenience.

Related code: `prefundedRedeem()` BaseLSTAdapter.sol#L168

Recommendation

We recommend implementing a redemption queue on the adapter. This queue would ensure a fair and orderly process for handling redemption requests, even when the current buffer size is insufficient. Here's how the queue could function:

- Users submitting redemption requests are placed in a queue based on the order of their transactions.
- The adapter fulfills redemption requests sequentially from the beginning of the queue.
- If a redemption request exceeds the available buffer, the user's position remains in the queue until enough buffer accumulates to fulfill the request.

Client's commentary

> The current implementation (with the buffer) and the queue-based implementation are not mutually exclusive. It's not trivial to implement such a queue on the adapter. We would monitor transactions to track situations when the current buffer size is insufficient.

M-5	Centralization risks
Severity	Medium
Status	Fixed in d9772f14

Description

The LST Adapter contracts implement a centralized governance model with two privileged roles: owner and rebalancer. The owner has extensive control, including:

- Modifying the `tranche` value of an adapter, potentially halting all `previewDeposit` and `previewRedeem` functions marked with the `onlyTranche` modifier.
- Pausing/unpausing staking, altering staking limits, and assigning a new rebalancer to the pool.

The rebalancer role, while less powerful, can also significantly impact user experience by:

- Initiating withdrawal requests to integrated LST projects.
- Adjusting the adapter's buffer percentage.

If the rebalancer fails to initiate withdrawal requests, users will be unable to redeem their funds until the owner intervenes. This centralization of control introduces a single point of failure and poses a risk to user trust.

Recommendation

We recommend implementing a mechanism to prevent the modification of the `tranche` value while the tranche holds a non-zero balance of adapter shares. This ensures the intended functionality of the `onlyTranche`-marked functions. Additionally, we suggest considering a more distributed mechanism for withdrawing funds from integrated projects to mitigate the reliance on a single rebalancer for user redemptions.

M-6	A linear increase in the withdrawal wait time for Lido for requests exceeding 500 ETH
Severity	Medium
Status	Fixed in 50fcf4d6

Description

The current implementation of Lido withdrawals in the `StEtherAdapter` contract exhibits a scalability issue. Withdrawal requests exceeding 500 ETH through the `StEtherAdapter` contract are forcibly trimmed to 500 ETH. Additionally, the `BaseLSTAdapter` design only processes a single request at a time.

This combination results in a linear increase in the waiting time for larger withdrawals. Each additional 500 ETH chunk incurs a further processing delay. Considering Lido's average withdrawal processing time of 3-5 days, a 5000 ETH withdrawal could take approximately 30-50 days to complete. This extended wait time can significantly impact users, especially during time-sensitive redemptions after maturity timestamps. Moreover, it hinders the project's future scalability.

Related code: `StEtherAdapter._requestWithdrawal()` StEtherAdapter.sol#L94

Recommendation

We recommend implementing the following changes:

1. **Batch Withdrawal Support:** Modify the adapters to support submitting multiple withdrawal requests concurrently. This functionality would allow users to initiate larger withdrawals by splitting them into smaller, parallel requests.
2. **Lido Request Composition:** For the `StEtherAdapter` specifically, introduce logic to automatically compose a series of 500 ETH withdrawal requests for Lido when the total withdrawal amount surpasses the 500 ETH threshold. This ensures efficient handling of larger Lido withdrawals without exceeding their processing.

M-7	A potential blocking issue in the `RETHAdapter.withdrawAll` function
Severity	Medium
Status	Fixed in 6e07621d

Description

The `RETHAdapter.withdrawAll` function in the `RETHAdapter` contract carries a potential risk of blocking user withdrawals. The function lacks a mechanism to verify the availability of collateral within the `RETH` contract before attempting a withdrawal. Consequently, if the requested withdrawal amount exceeds the current available collateral, the function call will revert.

This issue could impact the users attempting to redeem all their funds after the maturity timestamp. While the `withdraw` function with a `targetBufferPercentage` of 100% offers an alternative approach by checking available collateral, a blocked `withdrawAll` function can still cause inconvenience and delay. Moreover, users might be forced to wait for the RETH contract's collateral supply to replenish before successfully executing a withdrawal.

Related code: `withdrawAll()` RETHAdapter.sol#L173

Recommendation

We recommend implementing logic within `RETHAdapter.withdrawAll` to automatically adjust the withdrawal amount in case it exceeds the available collateral in the RETH contract.
Additionally, consider implementing a restricted emergency functionality for exiting ETH from RETH under critical circumstances. This could potentially involve leveraging a CurvePool as an alternative withdrawal mechanism. However, this recommendation requires cautious evaluation due to the potential security implications it introduces.

M-8	Incorrect handling of pending requests in the `requestWithdrawal` function
Severity	Medium
Status	Fixed in bc0cff80

Description

Invoking the `requestWithdrawal` function should calculate the amount required to refill the ETH buffer to its target value. However, if the function is called during the concurrent requests are already in progress, the amount is calculated improperly. This may cause the withdrawal of an undesired extra amount and other unexpected behavior.

Related code: `requestWithdrawal()` BaseLSTAdapter.sol#L202

Recommendation

We recommend taking into account the amount of the pending withdrawal during the calculation of the amount required to be withdrawn.

2.4 Low

L-1	Redudant inheritance of `Authorizable` in `BaseAdapter`
Severity	Low
Status	Fixed in 98630ecb

Description

The `BaseAdapter` contract is currently inheriting from the `Authorizable` contract. However, the functionalities of `Authorizable` are not being utilized in any part of the `BaseAdapter` implementation. This inheritance appears to be redundant and does not contribute to the functionality of `BaseAdapter`.

Related code - inheritance of `Authorizable` in `BaseAdapter`: BaseAdapter.sol#L16

Recommendation

We recommend removing the inheritance of `Authorizable` from `BaseAdapter`. Instead, this inheritance should be applied directly to those implementations where the functionalities of `Authorizable` will be required.

L-2	Lack of fee control
Severity	Low
Status	Fixed in d9772f14

Description

The fee value can be set by the owner up to 100%.
TrancheFactory.sol#L63

Recommendation

We recommend using a max limit for fees to mitigate passing wrong fee values to a tranche.

L-3	Gas optimization tips
Severity	Low
Status	Fixed in d9772f14

Description

The memory variable `oneSubTilt` is declared multiple times that increases the codebase size and gas consumption:
Tranche.sol#L621
Tranche.sol#L642
Tranche.sol#L658.

Recommendation

We recommend adding the `oneSubTilt` as the immutable variable.

L-4	Incorrect comments
Severity	Low
Status	Fixed in 6ce72eb3

Description

At line
Tranche.sol#L660
there is an incorrect formula that doesn't match the one below (which is correct).

Recommendation

We recommend changing the comment formula.

L-5	A redundant function call in `SFrxETHAdapter`
Severity	Low
Status	Fixed in 2a2e8d6b

Description

The `_requestWithdrawal` function within the `SFrxETHAdapter` contract exhibits a potential redundancy. This function calls the view function `REDEMPTION_QUEUE.redemptionQueueState()` to retrieve the next NFT ID. However, the same ID is also returned by the `REDEMPTION_QUEUE.enterRedemptionQueue` function upon entering the redemption queue. This results in an unnecessary extra call.

Related code: `_requestWithdrawal` SFrxETHAdapter.sol#L101

Recommendation

To optimize gas consumption, we recommend removing the redundant call to `REDEMPTION_QUEUE.redemptionQueueState().nextNftId` within `_requestWithdrawal`.

L-6	Dynamic stake limit configuration in `BaseLSTAdapter`
Severity	Low
Status	Fixed in 0be03561

Description

The `BaseLSTAdapter` contract employs two constant values to define the default stake limit values. These values are essential to control staking behavior within the adapter.

These predefined constants might not accurately reflect dynamic market conditions at the time of deployment. Therefore, adjusting these limits after deployment requires an additional function call `setStakingLimit`, which can be inconvenient for initial configuration.

Related code: `setStakingLimit` in `BaseLSTAdapter` BaseLSTAdapter.sol#L284

Recommendation

We propose modifying the `BaseLSTAdapter` constructor to accept arguments for the minimum and maximum stake limits. This allows for setting these values during deployment based on the prevailing market conditions.

L-7	The silent behavior on attempting withdraw of small amount of Ether
Severity	Low
Status	Fixed in c4661d20

Description

The `_requestWithdrawal` function may silently bypass the attempt to withdraw small amount of Ether (i.e. < 100 wei, depending on the implementation). No event nor revert will be generated. On the edge cases, this may mislead offchain services, causing their unexpected behavior.

Related code: StETHAdapter - StEtherAdapter.sol#L99

Recommendation

We recommend emitting event or reverting the transaction instead of silently ignoring this edge case.

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