P2P.ORG LENDING PROXY SECURITY AUDIT REPORT

Jan 24, 2025

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

The Lending Proxy project streamlines permissioned interactions with lending protocols via lightweight, deterministic proxy contracts. The `P2pLendingProxyFactory` deploys proxies upon initial deposits, validating calldata against configurable rules, while the `P2pLendingProxy` manages deposits, withdrawals, and profit-sharing logic to distribute earnings between clients and a treasury. It also includes functionality for trusted reward claims from Morpho's Universal Reward Distributors, optimizing asset management.

1.4 Project Dashboard

Project Summary

Title	Description
Client	P2P.org
Project name	Lending Proxy
Timeline	09.01.2025 - 24.01.2025
Number of Auditors	3

Project Log

Date	Commit Hash	Note
09.01.2025	4d3c90ff8b49f2f4eda35337ff97ed71e0776274	p2p-lending-proxy initial commit
09.01.2025	7a70df614d7076a38c78381475c8b4223d50b038	eth-staking-fee-distributor-contracts initial commit
22.01.2025	1dd35b124655c999492fb6693cc70d3b6e16589f	p2p-lending-proxy reaudit commit
22.01.2025	dcea97b5294999ec9c28c50155669804bb6a2d01	eth-staking-fee-distributor-contracts reaudit commmit

Date	Commit Hash	Note
24.01.2025	e27cd40e4e9a0dba5d99b5c07b624b31686afa52	p2p-lending-proxy reaudit fix

Project Scope

The audit covered the following files:

File name	Link
src/common/AllowedCalldataChecker.sol	AllowedCalldataChecker.sol
src/common/P2pStructs.sol	P2pStructs.sol
src/p2pLendingProxyFactory/P2pLendingProxyFactory.sol	P2pLendingProxyFactory.sol
src/p2pLendingProxy/P2pLendingProxy.sol	P2pLendingProxy.sol
src/access/P2pOperator.sol	P2pOperator.sol
src/access/P2pOperator2Step.sol	P2pOperator2Step.sol
src/adapters/CalldataParser.sol	CalldataParser.sol
src/adapters/P2pMorphoProxy.sol	P2pMorphoProxy.sol
src/adapters/P2pMorphoProxyFactory.sol	P2pMorphoProxyFactory.sol

File name	Link
contracts/feeDistributor/DeoracleizedFeeDistributor.sol	DeoracleizedFeeDistributor.sol

Deployments

File name	Contracts deployed on Base
P2pMorphoProxyFactory.sol	0x917a23...78fd99db
P2pMorphoProxy.sol	0xa749Da...f0dc83D2

File name	Contracts deployed on Ethereum
P2pMorphoProxyFactory.sol	0x3fe77f...2e799568
P2pMorphoProxy.sol	0xD1E5ec...47b203a2

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	1
Medium	4
Low	5

ID	Name	Severity	Status
H-1	Unvalidated Variable Parameters Allow Fee Manipulation	High	Fixed
M-1	Missing `PERMIT2` Front-Run Protection	Medium	Fixed
M-2	Overstated Withdrawals When Existing Tokens Are Held	Medium	Fixed
M-3	Overestimated Fees if Tokens Are Already Present Before Claims	Medium	Fixed
M-4	Signature Replay Allows Unauthorized Use of Permits	Medium	Acknowledged
L-1	Rounding-Based Fee Bypass for Small Withdrawals	Low	Fixed

L-2	Strict Revert on `None` Rule Type Causes Unnecessary Failures	Low	Fixed
L-3	Centralization Risks	Low	Acknowledged
L-4	Redundant Zero-Balance Check in `withdraw`	Low	Fixed
L-5	Fee-on-Transfer Tokens Counted Incorrectly in `deposit`	Low	Acknowledged

1.6 Conclusion

During the audit, we examined the following attack vectors:

1. **Attacks on the calldata restriction mechanism for the deposit, withdraw, and callAnyFunction methods.** Users can provide arbitrary calldata to external protocols, which undergoes validation based on administrator-defined rules. We assessed whether the rule system is flexible enough to perform its intended functions effectively.
2. **Attack on EIP-1271.** We checked whether the contracts are vulnerable to the common isValidSignature issue, which allows an attacker to reuse (replay) a single signature across different contracts.
3. **PERMIT front-run DoS.** We evaluated whether the code is protected against a DoS attack that could prevent normal use of the permit function.
4. **Verification of fee calculation accuracy.** We examined whether the fee calculations are correct and whether malicious activity could impact their accuracy.
5. **Centralization risks.** We ensured that the project administrator does not have excessive privileges that are unjustified by the project's operational logic.
6. **Support for fee-on-transfer and rebaseable tokens.** We examined how the project handles non-standard ERC20 tokens.

The identified issues are listed below.

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

H-1	Unvalidated Variable Parameters Allow Fee Manipulation
Severity	High
Status	Fixed in `e27cd40e`

Description

This issue has been identified within the `deposit` and `withdraw` functions of the `P2pLendingProxy` contract.

It is impossible to validate the variable parameters (such as `asset`, `amount`, or `receiver`) in the calldata passed to `lendingContract` under the current rule types (`StartsWith`, `EndsWith`). Attackers can exploit this lack of parameter integrity checks to manipulate fee calculations or bypass them entirely.

For example:

1. A user can call `deposit` with a larger token amount than what is actually deposited into the lending protocol, thus inflating `s_totalDeposited`. Later, they can redeem tokens from proxy balance via Permit2, creating a discrepancy between the protocol's recorded deposit and the actual tokens deposited, leading to fee bypassing.
2. A user could specify a more valuable token (e.g., WETH) in the `deposit` function but actually deposit a cheaper one (e.g., USDC), causing `newProfit` to be undercounted and fees bypassed.
3. A user can specify an arbitrary `receiver` address during withdrawal, avoiding the intended fee logic by redirecting tokens elsewhere.
4. A user may pass one token as `vault` to the `withdraw` function but actually call `erc4626Redeem` on another token, again bypassing protocol fees.

These scenarios ultimately allow fee manipulation and value misreporting, resulting in potential financial losses to the protocol.

This issue is classified as **high** severity because it enables direct fee manipulation and significant misreporting of asset flows.

Recommendation

We recommend implementing an adapter pattern for each lending protocol integration. This approach ensures all relevant parameters (`token`, `amount`, `receiver`, `vault`, etc.) are validated via a protocol-specific adapter rather than relying on general calldata verification rules alone. Alternatively, existing rule types could be expanded to verify all non-static arguments, though this might overcomplicate the verification logic.

Client's Commentary

> Client: Fixed in 1dd35b12
> Mixbytes: In the `deposit` function, when verifying the `erc4626Deposit` call, the `vault.asset()` parameter is not checked to ensure it matches the `asset`.
> Client: Fixed in e27cd40e

2.3 Medium

M-1	Missing `PERMIT2` Front-Run Protection
Severity	Medium
Status	Fixed in 1dd35b12

Description

This issue has been identified within the `deposit` function of the `P2pLendingProxy` contract.

An attacker can front-run a legitimate user's transaction by preemptively calling `Permit2.permit`, causing reverts or unexpected over-allowances granted to the proxy.

The issue is classified as **medium** severity because while it might lead to transaction failure or confusion, it does not necessarily result in direct fund loss.

Recommendation

We recommend wrapping Permit2.permit into a try-catch pattern preventing unintended reverts due to invalidated nonce.

Client's Commentary

> Fixed in 1dd35b12

M-2	Overstated Withdrawals When Existing Tokens Are Held
Severity	Medium
Status	Fixed in 1dd35b12

Description

In the `withdraw` function of `P2pLendingProxy`, the contract's post-withdrawal token balance is used to calculate how many tokens were withdrawn. This can inflate the perceived withdrawn amount if the contract already held tokens. Consequently, users may pay more fees if they (or others) previously sent tokens directly to the proxy.

The issue is classified as **medium** severity because it causes inaccurate fee or profit calculations, although it does not enable direct theft.

Recommendation

We recommend capturing the balance before the withdrawal call, then computing the difference to determine the exact tokens retrieved from the external protocol. This ensures the correct number of tokens withdrawn is used for fee calculations.

Client's Commentary

> Fixed in 1dd35b12

M-3	Overestimated Fees if Tokens Are Already Present Before Claims
Severity	Medium
Status	Fixed in 1dd35b12

Description

In the `morphoUrdClaim` function of `P2pLendingProxy`, the `p2pAmount` is calculated as a fraction of the total balance after tokens are claimed. If tokens were already present in the contract, the fee might be over-allocated.

The issue is classified as **medium** severity because it distorts the fee distribution, possibly overcharging the client.

Recommendation

We recommend using the difference between the post-claim and pre-claim balances to determine the newly received tokens.

Client's Commentary

> Fixed in 1dd35b12

M-4	Signature Replay Allows Unauthorized Use of Permits
Severity	Medium
Status	Acknowledged

Description

This issue has been identified within the signature verification flow of the `isValidSignature` function in the `P2pLendingProxy` contract.

Currently, signatures are verified against `s_client`, but the contract's address or any unique identifier is not included in the signed data. An attacker can replay the same signature across multiple `P2pLendingProxy` instances owned by the same user. For example, the replayed signature could authorize unwanted or additional transfers.

In the context of `Permit2`, such reuse could allow multiple proxies to be drained using a single signature. Moreover, any message a client signs for themselves may inadvertently be valid for their proxies, and vice versa.

This issue is classified as **medium** severity because, while the proxy contracts are not designed to store tokens on balance, replay attacks still pose a risk of unauthorized operations.

Recommendation

We recommend incorporating the contract address (or a unique contract-specific field) into the signed data, such as via an EIP-712 domain separator. By doing so, signatures become valid exclusively for the intended `P2pLendingProxy` contract, mitigating replay across different addresses.

Client's Commentary

> Acknowledged. The existing implementation was recognized to be safe for Morpho exclusively because Morpho doesn't accept deposits via non-caller's signatures. However, we admit that it must be changed if we decide to use it for other lending protocols.

2.4 Low

L-1	Rounding-Based Fee Bypass for Small Withdrawals
Severity	Low
Status	Fixed in 1dd35b12

Description

This issue appears in the `withdraw` function of the `P2pLendingProxy` contract. When users make small withdrawals, the rounding process in fee calculation may yield zero fees, allowing them to avoid fees altogether for minimal amounts.

The issue is classified as **low** severity because it only affects small withdrawals and does not result in major fee bypassing.

Recommendation

We recommend calculating fees with rounding up strategy or enforcing a minimum fee threshold to ensure fees are applied even on minor withdrawals.

Client's Commentary

> Fixed in 1dd35b12

L-2	Strict Revert on `None` Rule Type Causes Unnecessary Failures
Severity	Low
Status	Fixed in 1dd35b12

Description

Within the `checkCalldata` function of `P2pLendingProxyFactory`, encountering a `None` rule type causes the transaction to revert outright rather than rejecting non-zero calldata as intended.

The issue is classified as **low** severity because it results in unexpected reverts and reduced flexibility without compromising assets.

Recommendation

We recommend modifying the logic so that any non-zero calldata for a `None` rule type is gracefully rejected rather than reverting the entire transaction.

Client's Commentary

> Fixed in 1dd35b12

L-3	Centralization Risks
Severity	Low
Status	Acknowledged

Description

This issue has been identified in the `setCalldataRules` and `removeCalldataRules` functions of the `P2pLendingProxyFactory` contract.

The P2P operator can update calldata rules crucial for the lending protocol integrations. Malicious or erroneous rule changes could disrupt user withdrawals.

The issue is classified as **low** severity because it grants significant control to a single entity, but does not directly enable theft of funds.

Recommendation

We recommend using a multi-sig and timelock mechanism for calldata rule changes and other critical functions, ensuring that modifications undergo review before becoming active.

Client's Commentary

> Acknowledged. We agree with using a multi-sig for P2P operator. However, it will be at the configuration level, not defined in p2p-lending-proxy code.

L-4	Redundant Zero-Balance Check in `withdraw`
Severity	Low
Status	Fixed in dcea97b5

Description

Within the `withdraw` function of `DeoracleizedFeeDistributor`, there is a check for `address(this).balance == 0` before verifying `(client + service + referrer) > address(this).balance`. The latter conditions already cover the case of zero balance, making the preliminary check unnecessary.

The issue is classified as **low** severity because it slightly affects code clarity and may add minor overhead without causing security flaws.

Recommendation

We recommend removing the `if (balance == 0)` condition.

Client's Commentary

> Fixed in dcea97b5

L-5	Fee-on-Transfer Tokens Counted Incorrectly in `deposit`
Severity	Low
Status	Acknowledged

Description

In the `deposit` function of `P2pLendingProxy`, the contract increments `s_totalDeposited` by the nominal `amount` from the permit without accounting for fee-on-transfer tokens that arrive in smaller amounts.

The issue is classified as **low** severity because it primarily leads to inaccurate accounting, rather than an exploit affecting conventional ERC20 tokens.

Recommendation

We recommend measuring the actual balance change before and after `transferFrom` and updating `s_totalDeposited` accordingly to account for fee-on-transfer tokens properly.

Client's Commentary

> Client: Fixed in 1dd35b12
> Mixbytes: Issue with fee-on-transfer tokens isn't completely resolved yet. Currently, deposits would revert if fees were incurred due to a mismatch between the tokens transferred to the proxy and those deposited to the MorphoBundler (since the proxy wouldn't have sufficient balance).
> Client: Acknowledged for fee-on-transfer tokens since they won't be a priority for us in the foreseeable future.

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