P2P.ORG STAKING FEE DEPOSITOR 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

The audited scope contains several smart contracts designed to deposit ETH to validators and to distribute staking rewards among the depositor, the service, and the referrer who attracted the depositor to the project.

1.4 Project Dashboard

Project Summary

Title	Description
Client	P2P
Project name	Staking Fee Distributor
Timeline	July 6 2023 - July 02 2024
Number of Auditors	4

Project Log

Date	Commit Hash	Note
06.07.2023	30a7ff78e8285f2eae4ae552efb390aa4453a083	Commit for the audit
01.08.2023	58bb08d59ece418bdf15b7023935c963c43c7682	Commit for the reaudit
19.06.2024	a0e88faf2583972a19b67e4b4728d98410a26e7a	Commit for the diff audit
28.06.2024	6db547f2a4387343205a0b1a4f9bc6b982ecbce4	Commit for the reaudit #2

Project Scope

The audit covered the following files:

File name	Link
Ownable2Step.sol	Ownable2Step.sol
OwnableBase.sol	OwnableBase.sol
Ownable.sol	Ownable.sol
OwnableWithOperator.sol	OwnableWithOperator.sol
AssetRecoverer.sol	AssetRecoverer.sol
OwnableAssetRecoverer.sol	OwnableAssetRecoverer.sol
OwnableTokenRecoverer.sol	OwnableTokenRecoverer.sol
TokenRecoverer.sol	TokenRecoverer.sol
BaseFeeDistributor.sol	BaseFeeDistributor.sol
ContractWcFeeDistributor.sol	ContractWcFeeDistributor.sol
ElOnlyFeeDistributor.sol	ElOnlyFeeDistributor.sol
Erc4337Account.sol	Erc4337Account.sol
OracleFeeDistributor.sol	OracleFeeDistributor.sol
FeeDistributorFactory.sol	FeeDistributorFactory.sol
P2pAddressLib.sol	P2pAddressLib.sol
Oracle.sol	Oracle.sol
P2pOrgUnlimitedEthDepositor.sol	P2pOrgUnlimitedEthDepositor.sol

Deployments

Commit 58bb08d59ece418bdf15b7023935c963c43c7682* (previos version)

File name	Contract deployed on mainnet	Comment
FeeDistributorFactory.sol	0x86a9f3...4212E1bb	
Oracle.sol	0x4E67Df...4FE38803	
ContractWcFeeDistributor.sol	0xf6aa12...85AB91ed	
ElOnlyFeeDistributor.sol	0x609176...16728d97	
OracleFeeDistributor.sol	0x7109De...6C05163f	
P2pOrgUnlimitedEthDepositor.sol	0xb3C7b1...ddA7E256	

Commit 6db547f2a4387343205a0b1a4f9bc6b982ecbce4 (latest version)

File name	Contract deployed on mainnet	Comment
Oracle	0xfC64A5...4e6E2051	
FeeDistributorFactory	0xa36420...EE77e136	
OracleFeeDistributor	0x02AC59...0239d778	
P2pOrgUnlimitedEthDepositor	0x7c58c0...4F949f1F	

1.5 Summary of findings

Severity	# of Findings
Critical	2

High	6
Medium	14
Low	9

ID	Name	Severity	Status
C-1	Rewards can be accounted as collateral	Critical	Fixed
C-2	The slashed validator's stake gets split between `client`, `referrer`, and `service`	Critical	Fixed
H-1	Null basis points will lead to the lock of funds	High	Fixed
H-2	Client's collateral can be distributed as rewards	High	Fixed
H-3	Possible ddos of an expiration time	High	Fixed
H-4	ERC-4337 call to `_payPrefund` may lead to the validator stake being split	High	Fixed
H-5	`referrer` and `service` can receive less rewards than expected in some cases	High	Acknowledged
H-6	`s_clientOnlyClRewards` can potentially exceed the `amountInGwei` value from `Oracle`, leading to DoS of `withdraw` and `recoverEther`	High	Acknowledged
M-1	`voluntaryExit` shouldn't be public	Medium	Acknowledged
M-2	An incorrect check can lead to revert	Medium	Fixed
M-3	An incorrect distribution of rewards in case of `_sendValue` revert	Medium	Fixed
M-4	`s_defaultClientBasisPoints` should be restricted	Medium	Fixed

M-5	Non-existing `_feeDistributorInstance` can be rejected	Medium	Fixed
M-6	Poisoned `feeDistributor` can lock or steal service rewards	Medium	Acknowledged
M-7	A user can call the `voluntaryExit` function with invalid/non-existent validator pubkeys	Medium	Acknowledged
M-8	The `recoverEther` function shouldn't be called with gas restrictions	Medium	Fixed
M-9	Incorrectly setting `s_referrerConfig.basisPoints` will lead to the lock of funds	Medium	Fixed
M-10	`clientConfig.basisPoints` lower than 50% triggers a constant revert in `withdraw` and `emergencyEtherRecoveryWithoutOracleData` of `OracleFeeDistributor`	Medium	Fixed
M-11	A failed send to `to` address in `recoverEther` leads to over-distribution to non-failing parties	Medium	Fixed
M-12	The `ServiceRejected` status is ignored in the `addEth` and `makeBeaconDeposit` functions of the `PwpOrgUnlimitedEthDepositor` contract	Medium	Fixed
M-13	`ClientConfig.basisPoints` equal to 100% at `OracleFeeDistributor` could lead to DoS of `emergencyEtherRecoveryWithoutOracleData`	Medium	Fixed
M-14	`P2pOrgUnlimitedEthDepositor.refundAll()` can be blocked for any client	Medium	Fixed
L-1	A `newOwner` can be the current owner	Low	Fixed
L-2	Zero address can be valid	Low	Fixed
L-3	ERC721 `safeTransferFrom` is not safe	Low	Fixed
L-4	`increaseDepositedCount` shouldn't be `external`	Low	Acknowledged
L-5	`chainId` can replace the flag	Low	Fixed

L-6	A missing zero address check	Low	Fixed
L-7	`referrerConfig.basisPoints` can be set to 0, even if `referrerConfig.recipient` is not `address(0)`	Low	Fixed
L-8	`Unchecked` in `for-loops` is the default Solidity behavior since version 0.8.22	Low	Fixed
L-9	Custom `require` errors are supported by Solidity since version 0.8.26	Low	Acknowledged

1.6 Conclusion

During the audit process, 2 critical, 6 high, 14 medium, and 7 low severity findings have been spotted. All these findings have been fixed or acknowledged by the client.
To improve the overall security of the protocol and decrease the number of findings, we suggest increasing test coverage for future scopes. The current scope is very well written when it comes to the ease of understanding and reading the code. All functions can be easily comprehended. Our recommendation is to add some specific corner case tests to cover some complex attack vectors, along with general test coverage for future development. Also, it is good to have tests that cover the most callable parts of the protocol (e.g., all modifiers and requires should be tested for positive and negative scenarios).

During the additional diff audit (comparing commit `a0e88faf2583972a19b67e4b4728d98410a26e7a` with base commit `58bb08d59ece418bdf15b7023935c963c43c7682`), the following attack vectors have been checked:

1. Security of the deposits data structure with the use of depositId hash

   - The code review focusing on the implementation of the deposits data structure and the usage of depositId hashes demonstrated that the use of depositId hashes enhances the security and integrity of the deposit data. The implementation follows best practices for secure data handling. No vulnerabilities were found.

2. Checking the amount range for deposits (32-2048 ETH):

   - The code analysis to verify the validation of deposit amounts confirmed that the code includes proper checks to ensure that deposit amounts are within the 32-2048 ETH range. The validation logic is robust and secure.

3. Deposit to the ETH Deposit Contract:

   - The code review to ensure proper handling of deposits into the ETH Deposit Contract showed that the code correctly processes deposits, ensuring security and adherence to the specified contract. No security issues were identified.

4. Reward Distribution through OracleFeeDistributor

   - The code analysis to understand the reward calculation mechanism in case of ETH transfer reverts confirmed that the code includes logic to handle reverts and ensure accurate reward distribution. No vulnerabilities were found.
   - The review of the emergency exit functionality in the code demonstrated that the emergency exit logic is implemented correctly, allowing the system to maintain stability and security in the absence of oracle updates. No issues were detected.

- The code review focusing on the slashing logic and its impact on the system showed that the code handles validator slashing appropriately, ensuring overall system security. No vulnerabilities were identified.

5. Reentrancy in ETH transfer

   - The review of the code to check for potential reentrancy vulnerabilities during ETH transfers demonstrated that the code includes measures to prevent reentrancy attacks. The implementation follows best practices for secure ETH transfers. No vulnerabilities were detected.

6. Correctness of Access Rights Handling

   - The code review focused on the handling of access rights across various functions and modules. The analysis confirmed that access control mechanisms are correctly implemented, with appropriate checks to ensure that only authorized entities can perform specific actions.

The code review during the **diff audit** confirms that the examined components and integrations are secure and implemented correctly according to the specified standards. No critical, high or medium severity vulnerabilities were identified during the analysis. Only findings affecting code optimization and readability, but not security, were found.

2.FINDINGS REPORT

2.1 Critical

C-1	Rewards can be accounted as collateral
Severity	Critical
Status	Fixed in ee36451d

Description

If the sum of the execution layer and consensus layer rewards reaches 32 ether, the rewards will be accounted as collateral, and the service will not collect some fees ContractWcFeeDistributor.sol#L114-L135. The main problem here is that there is no check that `collateralsCountToReturn <= (s_validatorData.exitedCount - s_validatorData.collateralReturnedCount)`. A malicious client can always front-run the `withdraw()` call and transfer additional eth to the contract, so EL + CL rewards always will be divisible by 32 eth.

Recommendation

We recommend adding the following check:
`collateralsCountToReturn <= (s_validatorData.exitedCount - s_validatorData.collateralReturnedCount)`.

Client's commentary

> Agree that the issue exists.
> Don't think the check will be sufficient since `collateralsCountToReturn` will still be incorrect and returning will not happen at all.
> Instead, it might be better to define `collateralsCountToReturn` differently in the first place:
> `collateralsCountToReturn = s_validatorData.exitedCount - s_validatorData.collateralReturnedCount`

C-2	The slashed validator's stake gets split between `client`, `referrer`, and `service`
Severity	Critical
Status	Fixed in ee36451d

Description

There is an issue at line ContractWcFeeDistributor.sol#L114.
If the user's stake deposited by `client` gets slashed in ETH2 staking and is withdrawn to the `ContractWcFeeDistributor` contract (and all previous rewards were withdrawn), the `balance >= COLLATERAL` check will not pass. In that case, execution after the `if` block continues, and the users' deposit gets split as if it were rewarded.
A CRITICAL severity level was assigned to that issue because the user receives less ETH than they initially deposited minus the slashing penalty. Their stake is also split into `referrer` and `service`, which shouldn't happen in a normal case.

Recommendation

We recommend replacing the `balance >= COLLATERAL` check at line ContractWcFeeDistributor.sol#L114 and introduce a special registry of initially deposited validators public keys. It will help protocol owners to approve validator exiting (initially triggered by the user). Also, it would be useful to mark the current contract balance at the time of the validator exit being started. At the time when the contract receives a stake, withdrawal can be marked as finished, and the user is able to withdraw up to `address(this).balance - markedBalance`.

Client's commentary

> The issue seems to be valid.
> The approach with markedBalance won't work since the validator has to continue working even after the voluntary exit message having been broadcasted while in the Exit Queue, until the actual exit at some point in the future. So, the rewards need to continue being split.
> Having a large registry for each public key doesn't seem to make much sense since the whole purpose of the protocol is to support batching for deposits/withdrawals. There can be more than 1000 validators per a client.
> Our new proposed solution is to keep track of the returned ETH value for collaterals instead of the collaterals count. The client will be able to withdraw without splitting up to 32 ETH per exited validator, no matter if slashings have happened or not.
> In case slashings have happened, the client may be reimbursed from EL rewards until 32 ETH per

> exited validator is reached. The client will need to call the `voluntaryExit` function in order to increase the `exitedCount`.

2.2 High

H-1	Null basis points will lead to the lock of funds
Severity	High
Status	Fixed in ee36451d

Description

The `predictFeeDistributorAddress` function doesn't account for null basis points FeeDistributorFactory.sol#L131-L133. However, when creating a fee distributor, null basis points will be converted to the default value FeeDistributorFactory.sol#L92-L94. This will cause a storage update for the incorrect address P2pOrgUnlimitedEthDepositor.sol#L90-L95, so user funds will be lost in the contract.

Recommendation

We recommend accounting for possible null client basis points.

H-2	Client's collateral can be distributed as rewards
Severity	High
Status	Fixed in ee36451d

Description

If `_sendValue` reverts here ContractWcFeeDistributor.sol#L128-L131, then `s_validatorData.collateralReturnedCount` will be updated incorrectly, and the client will lose some collateral. Moreover, `balance` here ContractWcFeeDistributor.sol#L134 will account for the client's collateral and distribute it as rewards.

Recommendation

We recommend adding a check for this specific `_sendValue` and giving the client one more chance to receive collateral if the first attempt is reverted. Also, it is necessary to add some cooldown after the first attempt so no one can ddos it.

Client's commentary

> This scenario looks very hypothetical. But it's not hard to add some cooldown.

H-3	Possible ddos of an expiration time
Severity	High
Status	Fixed in ee36451d

Description

The `addEth` function can be called by anyone with 1 wei for any client which will lead to an increased `expiration` for the client P2pOrgUnlimitedEthDepositor.sol#L88. This can be used by a malicious user to postpone the `refund` call by the client. Also, `expiration` can be reset after the reject P2pOrgUnlimitedEthDepositor.sol#L113.

Recommendation

We recommend reconsidering access rights for the `addEth` function and allowing it to be called only by the client. Also, we recommend blocking new deposits after the reject call.

Client's commentary

> Limiting `addEth` to client will be inconvenient. The ability to deposit from multiple addresses is a feature. To prevert DOS, we can set the minimal deposit to, say, 1 ETH. It will become prohibitively expensive to DOS.
> Resetting `expiration` after the reject is a feature as well. 0 `expiration` means "can be refunded immediately", which is good for the client.

H-4	ERC-4337 call to `_payPrefund` may lead to the validator stake being split
Severity	High
Status	Fixed in ee36451d

Description

There is an issue at line ContractWcFeeDistributor.sol#L114 and Erc4337Account.sol#L103.
If a user voluntarily exits staking, then `ContractWcFeeDistributor` will receive a 32 ETH stake (in case if there were no slashings). If all previous rewards were withdrawn from the contract (or they are quite low), then a call to the `withdraw()` function via `ERC-4337` account abstraction logic may lead to paying for the transaction fee using that stake funds (in case if there were no deposit or paymaster is not used). Then this `balance >= COLLATERAL` check will not pass and the users' stake would get split also to `referrer` and `service`.
The HIGH severity level was assigned to that issue since a user receives less ETH than they initially deposited. Their stake is split also to `referrer` and `service` what shouldn't happen in a normal case.

Recommendation

We recommend replacing the call to `_payPrefund()` here Erc4337Account.sol#L52. This replacement will prevent contract funds from being used to pay for transaction gas.

Client's commentary

> The line ContractWcFeeDistributor.sol#L114 has been removed in the new version. The client will be receiving non-splittable share until the full 32 ETH per validator has been received.
> Having `_payPrefund()` is considered convenient since it allows to pay for gas from rewards (by design). If the EntryPoint collected more than necessary ETH from a FeeDistributor, the funds can be returned via the `withdrawFromEntryPoint()` function.

H-5	`referrer` and `service` can receive less rewards than expected in some cases
Severity	High
Status	Acknowledged

Description

There is an issue at line OracleFeeDistributor.sol#L107.
If a user provided a merkle tree proof for the first time, then `client` rewards are reduced taking into account the validated `amount` value and `referrer` and `service` receive more rewards (assuming that the following branch will be executed - OracleFeeDistributor.sol#L147). If there are new rewards deposited to the contract before the oracle pushes a new merkle root (or the user frontruns that transaction), then the user is able to deposit just 1 wei (directly to the contract) to bypass the following check OracleFeeDistributor.sol#L112 and withdraw new rewards without taking into account a new `amount` of CL rewards pushed by the oracle.
The HIGH severity level was assigned to that issue because a user receives more ETH than expected and `referrer` and `service` receive less.

Recommendation

We recommend introducing a special registry of the used merkle proofs. Those proofs can be stored together with the timestamp of when the oracle pushed a merkle root for the last time. It will prevent using the same proof to withdraw new rewards without consideration to the new CL rewards.

Client's commentary

> Client: I think there is a misunderstanding of the contract intended logic. The goal of each `withdraw` is not to split the contract's balance according to the basis points. The contract's balance is just EL rewards. The goal is to have split the total rewards (EL + CL) after each `withdraw` according to the basis points.
> An example:
> Suppose, initially, we have:
> `s_clientOnlyClRewards == 0`
> `balance == 10` (balance is EL rewards here)
> `amount == 20` (amount is CL rewards here)
> `clientBp == 9000`
> `serviceBp == 1000` (let's ignore the referrer for simplicity)
> After the first call of `withdraw`:
> `totalAmountToSplit` was 30
> Client got 7

> Service got 3
> `s_clientOnlyClRewards` updated to 20
> Now, imagine, the contract received 10 more as EL rewards.
> After the second call of `withdraw`:
> `totalAmountToSplit` was 10
> Client got 9
> Service got 1
> `s_clientOnlyClRewards` remained 20
> Thus, we see that `s_clientOnlyClRewards` is not increasing no matter how many times `withdraw` is called.
> Effectively, it's the same if the contract initially had `balance == 20`. Then, after the first call of `withdraw`:
> `totalAmountToSplit` was 40
> Client got 16
> Service got 4
> `s_clientOnlyClRewards` updated to 20
> There is no difference on the timing of receiving EL rewards by the contract.
>
> MixBytes: Those calculations are correct, but a new oracle report with the info about CL rewards is not taken into account. In that case the `amount` value increases and `client` should receive rewards according to that value too. For example, if a new `amount` equals to 40 and the user frontruns that oracle report, then their rewards would be calculated using an old value (`amount = 20`). Especially, the user can do that when they withdraw the last portion of their rewards. As an example: after one withdrawal we have `s_clientOnlyClRewards == 20`, then the contract receives EL rewards and the balance increases to `balance == 10`. There is an old oracle report with `amount == 20`, but there is new info on CL rewards and the oracle pushes new data with `amount == 40`. The user frontruns that update and as a result receives 9 as reward (service gets 1). If the user doesn't frontrun the oracle report, then they would only get 7 as a reward (and service would receive 3).
>
> Client: We agree that the possibility for frontrunning exists. But a registry of used proofs won't solve it because if no one calls `withdraw` until the very last moment before the next oracle report, the amounts will still be the same and the proof will be unused. The client can always wait for some "large" CL rewards that have not been accounted for by the oracle. So, they can always have some temporary advantage. But it will be compensated for with the next report and more EL rewards. We think it's acceptable as long as the accounting is balanced eventually in the long run.

H-6	`s_clientOnlyClRewards` can potentially exceed the `amountInGwei` value from `Oracle`, leading to DoS of `withdraw` and `recoverEther`
Severity	High
Status	Acknowledged

Description

OracleFeeDistributor.sol#L230
OracleFeeDistributor.sol#L121
OracleFeeDistributor.sol#L151
The issue is that if a `client` calls the `emergencyEtherRecoverWithoutOracleData` function, the value of `s_clientOnlyClRewards` can exceed the `amountInGwei` value from `Oracle`. In the `withdraw` function, `totalAmountToSplit` is calculated as `balance + amount - s_clientOnlyClRewards` in line OracleFeeDistributor.sol#L121. In this context, the `serviceAmount` will always be lower than `halfBalance` unless the `client.basisPoints` fall below 50%.

Following this, `s_clientOnlyClRewards` is calculated as `totalAmountToSplit - balance` in line OracleFeeDistributor.sol#L151, but this leads to an underflow if `s_clientOnlyClRewards` is less than `amount`, resulting in a revert of the whole `withdraw` function. The funds can only be withdrawn by altering the `amountInGwei` value in `Oracle` by the `owner` or by calling `emergencyEtherRecoverWithoutOracleData`, forcing the `client` to pay additional fees to `service` and `referrer`.

This vulnerability is classified as `high` due to the potential of funds being locked until the `owner` intervenes.

Recommendation

We recommend eliminating the update of the `s_clientOnlyClRewards` accumulator in the `emergencyEtherRecoveryWithoutOracleData` function and consequently prohibiting the possibility of its value to exceed the `amountInGwei` value.

Client's commentary

> The `s_clientOnlyClRewards` accumulator needs to be updated to keep track of what amount of CL rewards the client has claimed during `emergencyEtherRecoveryWithoutOracleData`, even if the actual `amountInGwei` value hasn't caught up.
> You suggested a possible solution for the underflow issue yourself in M2. Just restrict the check for `OracleFeeDistributor__WaitForEnoughRewardsToWithdraw` to `if (amount <`

> `s_clientOnlyClRewards)`.
> It's OK to wait for more CL rewards to unlock the normal `withdraw`. The client has no incentive to DOS in such a way since they split the contract balance the worst possible way for themselves. (ClientBp cannot be less than 5000).

2.3 Medium

M-1	`voluntaryExit` shouldn't be public
Severity	Medium
Status	Acknowledged

Description

It should be impossible to call the `voluntaryExit` function from the `ElOnlyFeeDistributor` and `OracleFeeDistributor`, otherwise there is a chance that a client will generate some random events that can be somehow interpreted by off-chain services.

Recommendation

We recommend changing the function visibility to `internal`.

Client's commentary

> Client: Exposing the `voluntaryExit` function from any type of FeeDistributor is intentional. It's a way for the client to signal us to broadcast a VEM. Which other random events can be there?
> MixBytes: A user can generate any amount of this event with random parameters. It is okay to leave it like this if you have enough checks in off-chain services triggered by these events.

M-2	An incorrect check can lead to revert
Severity	Medium
Status	Fixed in ee36451d

Description

The check here OracleFeeDistributor.sol#L112 should be changed to `amount < s_clientOnlyClRewards` otherwise the call can be reverted on this line OracleFeeDistributor.sol#L151 because of the incorrectly set `s_clientOnlyClRewards`.

Recommendation

We recommend changing the check to `amount < s_clientOnlyClRewards`.

M-3	An incorrect distribution of rewards in case of `_sendValue` revert
Severity	Medium
Status	Fixed in ee36451d

Description

Because of a possible revert of the `_sendValue` call updated value of the `s_clientOnlyClRewards`, OracleFeeDistributor.sol#L151 will lead to an incorrect distribution of the rewards.

Recommendation

It's recommended not to update `s_clientOnlyClRewards` if all `_sendValue` calls were reverted.

Client's commentary

> We consider this scenario to be extremely unlikely. Especially, considering that non-accepting ETH is discouraged economically. However, we agree to add this extra check.

M-4	`s_defaultClientBasisPoints` should be restricted
Severity	Medium
Status	Fixed in ee36451d

Description

`s_defaultClientBasisPoints` should be less than 10_000 FeeDistributorFactory.sol#L52 FeeDistributorFactory.sol#L71 otherwise the intialization of the deployed feeDistributors will revert.

Recommendation

We recommend adding a check that `s_defaultClientBasisPoints < 10_000`.

M-5	Non-existing `_feeDistributorInstance` can be rejected
Severity	Medium
Status	Fixed in ee36451d

Description

There is no check that a specific `_feeDistributorInstance` exists here P2pOrgUnlimitedEthDepositor.sol#L112-L113, so the mapping value can be updated for non-existing address.

Recommendation

We recommend adding a check that `_feeDistributorInstance` was initialized in the mapping.

M-6	Poisoned `feeDistributor` can lock or steal service rewards
Severity	Medium
Status	Acknowledged

Description

`FeeDistributorFactory` allows using any implementation of `feeDistributor` for creating clones from it FeeDistributorFactory.sol#L78. This can leave the P2P service without rewards if a malicious user deploys their version of `ContractWcFeeDistributor` where all service rewards will be transferred to the hacker.

Recommendation

We recommend adding a whitelist for the `feeDistributor` implementations.

Client's commentary

> It's a feature that the client can suggest their `feeDistributor` implementations. We will have a whitelist off-chain and check against it before proceeding to `makeBeaconDeposit`. If it's not whitelisted, we can review the code manually and either agree with it by calling `makeBeaconDeposit` or disagree by calling `rejectService`.

M-7	A user can call the `voluntaryExit` function with invalid/non-existent validator pubkeys
Severity	Medium
Status	Acknowledged

Description

There is an issue at line BaseFeeDistributor.sol#L151 and ContractWcFeeDistributor.sol#L71.
A user is able to pass an arbitrary `_pubkeys` array to the function and then the corresponding event will be emitted. Also, `s_validatorData.exitedCount` is being increased in the `ContractWcFeeDistributor` contract. It will not signal the P2P protocol to start the withdrawal for some specific validator, but future calls to `voluntaryExit` may be impossible due to the check at line ContractWcFeeDistributor.sol#L75.
The MEDIUM severity level was assigned to that issue as it may lead to errors in case there are off-chain listeners to those events.

Recommendation

We recommend introducing validator public keys registry which will store validator public keys and flag if they have been deposited. It can be done via `mapping(address => bool) depositedValidators`.
Then the `pubkeys` array can be checked using such mapping.

Client's commentary

> This registry will be too large and gas expensive since we can have thousands of validators. In case of error, as you noticed correctly, the user will only be able to withdraw their collateral once. Since this event is only informational for us and doesn't have any direct consequences, the user will be able to contact our support to resolve the exiting issue.

M-8	The `recoverEther` function shouldn't be called with gas restrictions
Severity	Medium
Status	Fixed in 58bb08d5

Description

There is an issue at line ContractWcFeeDistributor.sol#L188, ElOnlyFeeDistributor.sol#L90 and OracleFeeDistributor.sol#L198.
It is possible to have an issue during a call to `this.withdraw(...)` which will be spending a bigger amount of gas passed with a transaction.
The MEDIUM severity level was assigned to that issue as it may lead to inability to send ether here - P2pAddressLib.sol#L12 due to `gasleft() / 4` passed as a gas. There is no need in such a restriction for a function with a restricted access.

Recommendation

We recommend implementing a separate `_sendValue` function without gas restrictions. That function can be called during all ether transfers initiated by "owner".

M-9	Incorrectly setting `s_referrerConfig.basisPoints` will lead to the lock of funds
Severity	Medium
Status	Fixed in ee36451d

Description

If `_referrerConfig.basisPoints` is greater than `_clientConfig.basisPoints` BaseFeeDistributor.sol#L99, then the `withdraw` call will always revert due to an underflow ContractWcFeeDistributor.sol#L150. This can happen if a user sets `_referrerConfig.basisPoints` greater than `_clientConfig.basisPoints` by mistake which is possible due to the lack of checks in the contract.

Recommendation

We recommend adding a check that `_referrerConfig.basisPoints < _clientConfig.basisPoints` and `_referrerConfig.basisPoints < (10_000 - _clientConfig.basisPoints)`.

M-10	`clientConfig.basisPoints` lower than 50% triggers a constant revert in `withdraw` and `emergencyEtherRecoveryWithoutOracleData` of `OracleFeeDistributor`
Severity	Medium
Status	Fixed in 00167351

Description

OracleFeeDistributor.sol#L144
OracleFeeDistributor.sol#L151
OracleFeeDistributor.sol#L227
OracleFeeDistributor.sol#L230
An issue has been identified in the `OracleFeeDistributor` where, if the `clientConfig.basisPoints` is less than 5000 (representing 50% of the total), the `totalAmountToSplit` calculated from half the balance at lines OracleFeeDistributor.sol#L144 and OracleFeeDistributor.sol#L227 will be less than the `balance`. It leads to an underflow of `s_clientOnlyClRewards`, triggering a revert at lines OracleFeeDistributor.sol#L151 and OracleFeeDistributor.sol#L230, respectively.

The severity of this vulnerability is classified as `medium` due to the potential for the funds to become irretrievable within the contract. This situation arises when the `clientConfig.basisPoints` is less than 50%, implying that the service basis points exceed 50%. Consequently, the rewards distributed to the `service` will always surpass the `halfBalance` value.

Recommendation

We recommend adding the following check to the `initialize` function of the `OracleFeeDistributor`: `require(clientConfig.basisPoints >= 5000)`.

M-11	A failed send to `to` address in `recoverEther` leads to over-distribution to non-failing parties
Severity	Medium
Status	Fixed in ee36451d

Description

ContractWcFeeDistributor.sol#L193
ElOnlyFeeDistributor.sol#L95
OracleFeeDistributor.sol#L203
An issue has been identified where, if the `owner` calls `recoverEther` with the `to` address failing to receive, they will need to call `recoverEther` again. This leads to the over-distribution of the remaining funds to non-failing parties. This scenario requires the `owner` to make an error. This vulnerability is classified as `medium` because the `owner`'s mistake results in losses for the failing parties of the contract.

Recommendation

We recommend adding a revert statement if an attempt to send Ether to the `to` address fails.

M-12	The `ServiceRejected` status is ignored in the `addEth` and `makeBeaconDeposit` functions of the `PwpOrgUnlimitedEthDepositor` contract
Severity	Medium
Status	Fixed in ee36451d

Description

P2pOrgUnlimitedEthDepositor.sol#L112
P2pOrgUnlimitedEthDepositor.sol#L55
P2pOrgUnlimitedEthDepositor.sol#L180

An issue has been identified where, following the rejection of a service by the `owner`, there is no provision to prevent the addition of new ether and beacon deposits for that service. The current implementation lacks these checks. This vulnerability is classified as `medium` since it permits the calling of `addEth` for an invalid service and the invoking of the `makeBeaconDeposit` function due to either a mistake or malicious action by the `owner`.

Recommendation

We recommend adding a requirement check, `require(deposit.status != ServiceRejected);`, to the appropriate functions.

M-13	`ClientConfig.basisPoints` equal to 100% at `OracleFeeDistributor` could lead to DoS of `emergencyEtherRecoveryWithoutOracleData`
Severity	Medium
Status	Fixed in ee36451d

Description

OracleFeeDistributor.sol#L230
An issue arises when `clientConfig.basisPoints` are set to 10000, as the `emergencyEtherRecoverWithoutOracleData` function will constantly revert at line OracleFeeDistributor.sol#L230 due to a division by zero occurrence. Conversely, the `withdraw` function is safeguarded by the condition that the `serviceAmount` will always be less than `halfBalance` in this case. This vulnerability is classified as `medium` because it requires the `owner` to mistakenly create an `OracleFeeDistributor` with 100% basis points assigned to the `client` for this scenario to take place. Moreover, the `withdraw` function remains unaffected.

Recommendation

We recommend adding a check in the `initialize` function of `OracleFeeDistributor` to ensure that `clientConfig.basisPoints` are less than 10000.

Client's commentary

> Restricted all fee distributors to have ClientConfig.basisPoints less than 10000. It doesn't make sense for any fee distributor to have 100% for the client since there is nothing to split in that case. We can use the client address directly if we ever need 100%.

M-14	`P2pOrgUnlimitedEthDepositor.refundAll()` can be blocked for any client
Severity	Medium
Status	Fixed in ee36451d

Description

P2pOrgUnlimitedEthDepositor.sol#L151
A hacker can create their malicious depositor-prototype contract with a `client()` function that always reverts. Suppose the hacker calls `P2pOrgUnlimitedEthDepositor.addEth()` with a client's address, 1 wei, and this malicious depositor. In that case, this client is unable to call `refundAll()` anymore because it will always revert on the malicious depositor here:
P2pOrgUnlimitedEthDepositor.sol#L122
This problem was marked as Medium because anyone can block `P2pOrgUnlimitedEthDepositor.refundAll()` function for any client. At the same time, this function is not important because `P2pOrgUnlimitedEthDepositor.refundAll(address[] calldata _allClientFeeDistributors)` can be used instead.

Recommendation

We recommend removing `P2pOrgUnlimitedEthDepositor.refundAll()`.

2.4 Low

L-1	A `newOwner` can be the current owner
Severity	Low
Status	Fixed in ee36451d

Description

There is no check that a `newOwner` differs from the current owner Ownable2Step.sol#L45-L48 which can lead to meaningless update and event emitting.

Recommendation

We recommend adding a check that `newOwner` is not equal to the current owner.

L-2	Zero address can be valid
Severity	Low
Status	Fixed in ee36451d

Description

After the operator dismisses, the zero address will become a valid address OwnableWithOperator.sol#L63-L65.

Recommendation

We recommend adding a check that after the operator dismissal it is not used in the `checkOperatorOrOwner` function.

L-3	ERC721 `safeTransferFrom` is not safe
Severity	Low
Status	Fixed in ee36451d

Description

ERC721 `safeTransferFrom` requires the receiver to implement the `onERC721Received` function if the receiver is a contract TokenRecoverer.sol#L72. This can lead to a tx revert, so it is better to use a simple `transferFrom` here.

Recommendation

We recommend changing `safeTransferFrom` to `transferFrom` for ERC721 recovering.

L-4	`increaseDepositedCount` shouldn't be `external`
Severity	Low
Status	Acknowledged

Description

The `increaseDepositedCount` function can be called by anyone in the `ElOnlyFeeDistributor` and `OracleFeeDistributor` implementations BaseFeeDistributor.sol#L146-L148.

Recommendation

We recommend changing the visibility of the `increaseDepositedCount` to `internal`.

Client's commentary

> Client: Not clear how we can make it `internal` in `BaseFeeDistributor` and `external` in `ContractWcFeeDistributor`. It needs to be a part of the `IFeeDistributor` interface for P2pOrgUnlimitedEthDepositor.sol#L238
>
> MixBytes: You can make an internal `_increaseDepositedCount` function and use it only in the `ContractWcFeeDistributor` contract in the `increaseDepositedCount` function.
>
> Client: We can do a separate internal `_increaseDepositedCount` but it won't help with removing the external `increaseDepositedCount` since it's still in the interface and must be implemented somehow. We cannot remove it from the interface due to the reason indicated above.

L-5	`chainId` can replace the flag
Severity	Low
Status	Fixed in ee36451d

Description

The `_mainnet` can be replaced with `chainId()` P2pOrgUnlimitedEthDepositor.sol#L44-L46.

Recommendation

We recommend using the `chainId()` instead of the `_mainnet` flag.

L-6	A missing zero address check
Severity	Low
Status	Fixed in ee36451d

Description

There is an issue in the following functions - ContractWcFeeDistributor.sol#L181, ElOnlyFeeDistributor.sol#L83 and OracleFeeDistributor.sol#L187.
There is a missing check for the `_to` address value.
The LOW severity level was assigned to that issue because it is possible to send funds to zero address.

Recommendation

We recommend adding the following check `require(_to != address(0))` to the mentioned functions.

L-7	`referrerConfig.basisPoints` can be set to 0, even if `referrerConfig.recipient` is not `address(0)`
Severity	Low
Status	Fixed in ee36451d

Description

BaseFeeDistributor.sol#L92
An issue has been identified where there is no check in place to ensure `referrerConfig.basisPoints` can't be set to 0 if `referrerConfig.recipient != address(0)`. This oversight leads to unnecessary `sendValue` calls to the `referrer` during the `withdraw` execution.

Recommendation

We recommend adding a condition check, `require(referrerConfig.basisPoints != 0);`, when the recipient of the referrer is set to a non-zero address.

L-8	`Unchecked` in `for-loops` is the default Solidity behavior since version 0.8.22
Severity	Low
Status	Fixed in 6db547f2

Description

While the project is using Solidity version 0.8.24, it is redundant to use `unchecked` incrementing of the `for-loop` variables since Solidity implements such behavior by default starting from version 0.8.22.

Recommendation

We recommend using a standard `for-loop` increment notation for better code readability.

Client's commentary

> Fixed in 6db547f2

L-9	Custom `require` errors are supported by Solidity since version 0.8.26
Severity	Low
Status	Acknowledged

Description

Currently, custom errors are implemented using the `if(condition) revert(CustomError)` flow, which is the only way to implement custom errors in Solidity versions before 0.8.26. However, the latest version of Solidity already supports the `require(bool, CustomError)` flow.

Recommendation

We recommend considering upgrading Solidity to the latest version and replacing the `if - revert` flow with the `require` flow when it is beneficial for code readability.

Client's commentary

> We prefer staying on 0.8.24 to match SSV Solidity version exactly. Switched to evm_version = 'cancun' for the same reason.

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