MANTLE NETWORK METH SECURITY AUDIT REPORT

November 21, 2023

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

METH is a liquid staking protocol that allows users to stake their ETH and receive staking rewards generated from validators' activity in the consensus layer and additional rewards from the execution layer. When users deposit their ETH, they receive METH as proof of deposit. The balance of METH remains the same during staking, but its exchange rate to ETH increases with new rewards and decreases with slashings.
The protocol logic for unstake requests works in the following way. If a user unstake some METH and after that, a slashing event occurs, then their's unstake request will be canceled, and accrued rewards for the time that the user was waiting for unstake will be accounted on their balance.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Mantle Network
Project name	METH
Timeline	October 04 2023 - November 21 2023
Number of Auditors	3

Project Log

Date	Commit Hash	Note
04.10.2023	bd15a96aee7df0fc0566d662df0cf50ff2619d31	Commit for the audit (mntEth)
26.10.2023	93a55d8a3e9dfdb40ebd95e1e350af9d0c821883	Commit for the reaudit (mntEth)
06.11.2023	042cddc6f7dfdd1d76036b2c50edb3d859769a98	Commit for the audit (contracts)

Date	Commit Hash	Note
16.11.2023	41cefefb4acce2b3763ba32d76b95518594f9653	Commit for the audit (METHL2)
21.11.2023	b05cf1aa4c206d9c0c65559915a9199bd509c009	Commit with update (METHL2)

Project Scope

The audit covered the following files:

File name	Link
src/METH.sol	METH.sol
src/OracleQuorumManager.sol	OracleQuorumManager.sol
src/Oracle.sol	Oracle.sol
src/Pauser.sol	Pauser.sol
src/ReturnsAggregator.sol	ReturnsAggregator.sol
src/ReturnsReceiver.sol	ReturnsReceiver.sol
src/Staking.sol	Staking.sol
src/UnstakeRequestsManager.sol	UnstakeRequestsManager.sol
src/METH.sol	METH.sol
src/OracleQuorumManager.sol	OracleQuorumManager.sol
src/Oracle.sol	Oracle.sol
src/Pauser.sol	Pauser.sol
src/ReturnsAggregator.sol	ReturnsAggregator.sol
src/ReturnsReceiver.sol	ReturnsReceiver.sol
src/Staking.sol	Staking.sol
src/UnstakeRequestsManager.sol	UnstakeRequestsManager.sol
src/METHL2.sol	METHL2.sol

Deployments

File name	Contract deployed on mainnet	Comment
TimelockController.sol	0xc26016f1166bE7b6c5611AAB104122E0f6c2aCE2	OpenZeppelin v4.8.2
TransparentUpgradeableProxy.sol	0xe3cBd06D7dadB3F4e6557bAb7EdD924CD1489E8f	OpenZeppelin v4.8.2
Staking.sol	0xdecaCC56fc347274D3Df2b709602632845611D39	
TransparentUpgradeableProxy.sol	0xd5F7838F5C461fefF7FE49ea5ebaF7728bB0ADfa	OpenZeppelin v4.8.2
METH.sol	0xc9173Bf8Bd5c1b071B5CaE4122202A347b7EeFab	
TransparentUpgradeableProxy.sol	0x8735049F496727f824Cc0f2B174d826f5c408192	OpenZeppelin v4.8.2
Oracle.sol	0x7a6c874db238D7FdC84516cD940E97032271af69	
TransparentUpgradeableProxy.sol	0x92e56d2146D54d5AEcB25CA36c89D027a6ea0D90	OpenZeppelin v4.8.2
OracleQuorumManager.sol	0x54c23E0D89DA943165c969d1AbDb65f0D64174b4	
TransparentUpgradeableProxy.sol	0x38fDF7b489316e03eD8754ad339cb5c4483FDcf9	OpenZeppelin v4.8.2
UnstakeRequestsManager.sol	0x5A7b3CDe8aC8d780AF4797BF1517464aC54Ca033	
TransparentUpgradeableProxy.sol	0xD4e11C28E04c0c2bf370b7a9989498B7eA02493f	OpenZeppelin v4.8.2
ReturnsReceiver.sol	0x2d6A67a5FA23B2C4fd0243142694A6F046f1791d	
TransparentUpgradeableProxy.sol	0xD6E4aA932147A3FE5311dA1b67D9e73da06F9cEf	OpenZeppelin v4.8.2

File name	Contract deployed on mainnet	Comment
ReturnsReceiver.sol	0x1980B341cB31cb436Ffe0620a01a4Cb6D4d09A27	
TransparentUpgradeableProxy.sol	0x1766be66fBb0a1883d41B4cfB0a533c5249D3b82	OpenZeppelin v4.8.2
ReturnsAggregator.sol	0xf2Bc410fAd9Fc3140c4CDED7C6E5Bd56AC292c93	
TransparentUpgradeableProxy.sol	0x29Ab878aEd032e2e2c86FF4A9a9B05e3276cf1f8	OpenZeppelin v4.8.2
Pauser.sol	0x9D624DF2a423cc3f0425827FdDDFe053D9A0FDf3	
TransparentUpgradeableProxy.sol	0xcDA86A272531e8640cD7F1a92c01839911B90bb0	OpenZeppelin v4.8.2
METHL2.sol	0xa1f06B96F082C470E9759D1090d281b2493c6A2C	

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	3
Medium	4
Low	7

ID	Name	Severity	Status
H-1	Lack of sanity checks on the Oracle report update	High	Acknowledged
H-2	Report from a malicious oracle will be accounted for in the quorum	High	Acknowledged
H-3	The `latestCumulativeETHRequested` state variable can be zeroed inside the `cancelUnfinalized` function	High	Fixed
M-1	It is possible that some unstake requests can get uncancellable	Medium	Acknowledged
M-2	Lack of upper bound for `numberOfBlocksToFinalize` in `setNumberOfBlocksToFinalize()`	Medium	Acknowledged
M-3	`withdrawalWallet` cannot be upgraded in the `ReturnsAggregator`	Medium	Acknowledged
M-4	Request cancelling can be DoSed	Medium	Acknowledged
L-1	Execution layer rewards should be monitored	Low	Fixed
L-2	Events with the same requestId can be emitted	Low	Acknowledged

L-3	A `topUp` call can be front-run	Low	Acknowledged
L-4	`latestCumulativeETHRequested` is still updated if there were no requests canceled	Low	Fixed
L-5	Oracle service doesn't process dangerous case	Low	Acknowledged
L-6	Oracle can send reports for any block	Low	Fixed
L-7	Record updates cannot lead to rewards decreasing	Low	Acknowledged

1.6 Conclusion

During the audit process 3 HIGH, 4 MEDIUM, and 7 LOW severity findings were spotted. After working on the reported findings, all of them were acknowledged or fixed by the client.
The client acknowledged a big chunk of the findings because the modifications that admins will make will be thoroughly checked before applying them. The Mantle team clarified that all updates will be made via multisigs with an appropriate amount of checks, which can be suitable for the protocol growing stage (when the TVL of the protocol is not very high and some business logic can be updated to add new necessary functionalities). But for the cases when protocol holds a huge TVL, we recommend minimizing protocol updates and using an architecture that requires minimum input from trusted parties (e.g., ZK-oracles for reporting protocol state from the beacon chain, additional checks in modification function that guarantees modification correctness, etc.).

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

H-1	Lack of sanity checks on the Oracle report update
Severity	High
Status	Acknowledged

Description

Lack of sanity checks after record update can lead to a situation where new oracle reports will not pass checks because of incorrect values in previously updated reports Oracle.sol#L290.
The thing is that the modifier can update the last record with incorrect values, so new oracle reports will not pass sanity checks in the `receiveRecord` function. Additionally, the `ORACLE_MODIFIER_ROLE` holder can change the last record, bypassing the `sanityCheckUpdate()`. As a result, the last record can become incorrect, with an arbitrary value of `currentTotalValidatorBalance` and, to some degree, `cumulativeProcessedDepositAmount`. Consequently, the holder of the `ORACLE_MODIFIER_ROLE` can control `Staking.totalControlled()`: Staking.sol#L574

So, mETH exchange rate can be changed in one transaction. mETH can be minted at a reduced rate and sent to unstaking at a heightened rate.
The finding is classified as HIGH severity since after an incorrect update, only the admin can fix the contract to continue receiving new reports.

Recommendation

We recommend adding sanity checks on record modification:

```
sanityCheckUpdate(_records[idx-1], record);
sanityCheckUpdate(record, _records[idx+1]);
```

Client's commentary

> This is by design. It is important that the admin can modify records, and the records may have values which are out of the bounds of the checks in recovery cases. There are validity checks for strict invariants of the protocol which cannot be violated, even by the modification function.

H-2	Report from a malicious oracle will be accounted for in the quorum
Severity	High
Status	Acknowledged

Description

There is a chance that some of the oracle addresses will be compromised. After key leakage, the oracle address should be removed from the quorum, but if the malicious user sends a report before being excluded from the quorum, then their report will be accounted for in the quorum OracleQuorumManager.sol#L184-L186.
This vulnerability is classified as HIGH because after quorum reduction, it is easier for the malicious user to reach quorum (OracleQuorumManager.sol#L132-L133) and the compromised record will be valid.

Recommendation

We recommend setting `reporterRecordHashesByBlock` to 0 after the `SERVICE_ORACLE_REPORTER` role modification in the contract.

Client's commentary

> In practice, the absolute threshold parameter will require a majority of honest oracles to report the same hash.

H-3	The `latestCumulativeETHRequested` state variable can be zeroed inside the `cancelUnfinalized` function
Severity	High
Status	Fixed in 93a55d8a

Description

There is an issue at the line: UnstakeRequestsManager.sol#L269. It is possible that `latestRemainingRequest` was a claimed request, so that `latestRemainingRequest.cumulativeETHRequested` equals to zero. In such a case the `latestCumulativeETHRequested` value is changed to zero, which leads to two problems. The `allocatedETHSurplus` function defined at line UnstakeRequestsManager.sol#L332 would return a bigger value than it should (as `allocatedETHForClaims` is a cumulative variable), which will lead to revert inside the `withdrawAllocatedETHSurplus` function as it will attempt to transfer more ETH than it is stored on the contract.
The second issue is related to the unstake requests queue. There is a check at line UnstakeRequestsManager.sol#L199.
It ensures that users can claim in a first-come-first-serve manner, but if there are new requests created after zeroed `latestCumulativeETHRequested`, then users who joined the queue later would be able to claim before someone who had joined the queue earlier.
This finding is classified as HIGH since it leads to funds stuck on the `UnstakeRequestsManager` (`withdrawAllocatedETHSurplus` would be locked) and unfair claims of user requests. Those issues can be resolved only by redeploying the contract.

Recommendation

We recommend reducing the `latestCumulativeETHRequested` value only by the sum of the `ethRequested` fields of cancelled requests.

Client's commentary

> Great find.

2.3 Medium

M-1	It is possible that some unstake requests can get uncancellable
Severity	Medium
Status	Acknowledged

Description

There is an issue at line UnstakeRequestsManager.sol#L361. It is possible for an attacker to front-run `numberOfBlocksToFinalize` extension and claim an unstake request which would have gotten unfinalized after such extension. If there are some unfinalized requests before that claimed request, then they would get uncancellable due to the check at line UnstakeRequestsManager.sol#L243 (which will stop at the request claimed by an attacker).
This issue is classified as MEDIUM as it can lead to the inability to cancel unfinalized requests.

Recommendation

We recommend adding a special argument to the `cancelUnfinalizedRequests` function which would help to skip such claimed unstake requests if unfinalized requests are lying before them in the queue.

Client's commentary

> The issue presupposes that the setter for this field is intended to allow an admin to change finalization of requests which were already made, but this is not the case. Once requests become final, there is no intention to ever cancel them. Hence, a user front-running this transaction has no effect on the protocol. This function should only be used in normal operation, not when the protocol has experienced an event which requires cancellation.

M-2	Lack of upper bound for `numberOfBlocksToFinalize` in `setNumberOfBlocksToFinalize()`
Severity	Medium
Status	Acknowledged

Description

If `numberOfBlocksToFinalize` is set higher than the end block number of the latest record in oracle, then `UnstakeRequestsManager._isFinalized()` returns false for each claimed request UnstakeRequestsManager.sol#L377.
As a result, `UnstakeRequestsManager.cancelUnfinalizedRequests()` will revert on such requests (due to an attempt to make a transfer to `address(0)`):
UnstakeRequestsManager.sol#L275.
Furthermore, this can lead to other breaches in the logic of `UnstakeRequestsManager`.

Recommendation

We recommend setting a reasonable upper bound for `numberOfBlocksToFinalize` in `UnstakeRequestsManager.setNumberOfBlocksToFinalize()`. We also recommend not completely deleting the claimed request but only marking it as claimed.

Client's commentary

> We assume a competent and sane operator that does not set this number to millions of blocks. If that did happen, the fix would simply be to set it back.

M-3	`withdrawalWallet` cannot be upgraded in the `ReturnsAggregator`
Severity	Medium
Status	Acknowledged

Description

`withdrawalWallet` can be updated in the `Staking` contract Staking.sol#L709 but it cannot be updated in the `ReturnsAggregator` contract. Moreover, if the admin decides to update `withdrawalWallet`, it can break the protocol and require updating implementation for the `ReturnsAggregator` contract.

Recommendation

We recommend updating the architecture of the `ReturnsAggregator` contract so it can process with 2 different withdrawal wallet addresses.

Client's commentary

> We chose to initially implement support for a single withdrawal wallet for simplicity. In the rare case which this needs to change, the operator will need to upgrade the ReturnsAggregator contract to add support for multiple withdrawal wallets.

M-4	Request cancelling can be DoSed
Severity	Medium
Status	Acknowledged

Description

A user that wants to receive unstaked ETH without losses can DoS `cancelUnfinalizedRequests` UnstakeRequestsManager.sol#L240 by simply creating dozens of unstake requests. This will require only additional gas costs for sending transactions, all METH that will be blocked on the `UnstakeManager` during DoS will be returned to the user. Additional unstake requests will require admin to remove more requests than they expected, leading to out-of-gas cases or not all requests will be removed.

Recommendation

We recommend calling `cancelUnfinalizedRequests` only via private pools so users cannot front-run this call.

Client's commentary

> This is mitigated by the ability to pause unstake requests when cancelling. In practice, the contract will always be in a paused state for this function execution, as cancellation is an exceptional case which is only needed when there is a major slashing event (which pauses the protocol automatically). A private mempool can also be used.

2.4 Low

L-1	Execution layer rewards should be monitored
Severity	Low
Status	Fixed in 93a55d8a

Description

Execution layer rewards can be transferred to the ReturnsReceiver. Thus, they should be monitored by an external service to ensure that all EL rewards were sent by validator ReturnsAggregator.sol#L124.

Recommendation

We recommend using an external service to monitor EL rewards.

Client's commentary

> This check is implemented by our guardian service to ensure that rewards are received.

L-2	Events with the same requestId can be emitted
Severity	Low
Status	Acknowledged

Description

There is a chance that the `UnstakeRequestCreated` event will be emitted several times with the same `requestId` UnstakeRequestsManager.sol#L175-L177.

Recommendation

We recommend accounting for this fact in off-chain services or deleting requests during cancellation and not removing them from the list.

Client's commentary

> This is by design. Off-chain indexing services account for this.

L-3	A `topUp` call can be front-run
Severity	Low
Status	Acknowledged

Description

A `topUp` call can dramatically increase the exchange rate, so users that will front-run this call will "steal" some rewards Staking.sol#L523-L525.

Recommendation

We recommend using private mem pools for this call.

Client's commentary

> The operator can choose to pause staking or use a private mempool as suggested.

L-4	`latestCumulativeETHRequested` is still updated if there were no requests canceled
Severity	Low
Status	Fixed in 93a55d8a

Description

There is an issue at the line: UnstakeRequestsManager.sol#L269. `latestCumulativeETHRequested` gets updated even if there were no canceled requests and the `numCancelled` variable equals to zero.

Recommendation

We recommend adding a check for the `numCancelled` value before updating the `latestCumulativeETHRequested` variable.

Client's commentary

> comment here

L-5	Oracle service doesn't process dangerous case
Severity	Low
Status	Acknowledged

Description

Currently, the oracle service doesn't process the case when a different report was sent by a malicious user from the oracle address to the `OracleQuorum` contract. It is better to catch the situation when OracleQuorum already has non-zero hash for the same endBlock. In this case, it is better to alert immediately and pause the Oracle contract.

Recommendation

We recommend adding logging and alerting for this case.

Client's commentary

> It is expected that oracles may need to resubmit reports in some cases. Due to the off-chain services being designed to be stateless, it is difficult to detect cases where a re-report is due to compromise rather than re-computation. We agree that these cases should be monitored and investigated.

L-6	Oracle can send reports for any block
Severity	Low
Status	Fixed in 93a55d8a

Description

An oracle can send a report for any block range, including old blocks and new blocks OracleQuorumManager.sol#L182.

Recommendation

We recommend adding an alert if the quorum was reached more than once for a specific block.

Client's commentary

> Reaching quorum twice may be needed in some cases, but we agree that this is extremely unlikely and so there should be alerting for this case.

L-7	Record updates cannot lead to rewards decreasing
Severity	Low
Status	Acknowledged

Description

Record modification cannot lead to rewards decreasing because it is impossible to receive more rewards than ReturnsReceivers have Oracle.sol#L298-L303. If rewards were decreased during modification, it should be fixed by the admin, which will lead to new rewards that will not be on the contract. If the admin makes a mistake during modification, then the protocol will need to send additional funds to the Receivers' contracts to modify the record back.

Recommendation

We recommend thoroughly checking record modification because incorrect updating will require admins to top-up the `Return Aggregator` contract.

Client's commentary

> When needed, all reward modification is computed automatically by code.

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