ASPIDA NETWORK CONTRACTS SECURITY AUDIT REPORT

January 31, 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

This project is intended to provide liquidity staking for ETH2.0. Additionally, a portion of users' deposits is reserved for `strategies`, the code for which was not provided at the time of the audit. According to the developers' statements, these `strategies` are to be implemented for pegging maintenance in external DEXes.

The project exhibits a degree of centralization, similar to other projects in this domain, which is attributable to the architectural features of Ethereum Staking itself. However, some excess centralization, not necessitated by the specifics of the project, has been observed.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Aspida Network
Project name	Contracts
Timeline	26.10.2023 - 29.01.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
26.10.2023	a94928e27a72baabb8c73b42634350ca934b3353	initial commit for the audit
22.12.2023	d86baafd6cda73eb16b65ac5418cf8adc0ed9346	commit for the 1st re-audit
15.01.2024	eff32fce69e0e5ab5afc1bdb503ddfddbf94e867	commit for the 2nd re-audit

Project Scope

The audit covered the following files:

File name	Link
CorePrimary.sol	CorePrimary.sol
aETH.sol	dETH.sol (renamed to aETH.sol during the re-audit)

File name	Link
RewardOracle.sol	RewardOracle.sol
saETH.sol	sdETH.sol (renamed to saETH.sol during the re-audit)
core/ActionControl.sol	ActionControl.sol
core/CoreStrategy.sol	CoreStrategy.sol
core/CoreTreasury.sol	CoreTreasury.sol
core/StakingModel.sol	StakingModel.sol
core/Submit.sol	Submit.sol
core/WithdrawalQueue.sol	WithdrawalQueue.sol
library/Manable.sol	Manable.sol
library/Minter.sol	Minter.sol
library/PauseGuardian.sol	PauseGuardian.sol

Deployments

File name	Contract deployed on mainnet
TransparentUpgradeableProxy (aETH)	0xFC87753Df5Ef5C368b5FBA8D4C5043b77e8C5b39
aETH	0x5f898DC62d699ecBeD578E4A9bEf46009EA8424b
TransparentUpgradeableProxy (saETH)	0xF1617882A71467534D14EEe865922de1395c9E89
saETH	0xD9F64Ee3DD6F552c1BcfC8862dbD130bc6697a66

File name	Contract deployed on mainnet
TransparentUpgradeableProxy (CorePrimary)	0x5341864D99B50155F782C562Bd15Ac4a0A3C117e
CorePrimary	0x55b6aF0e89eAd974a80b70C5B30589B088113e24
TransparentUpgradeableProxy (RewardOracle)	0xD691b1c47a578f51aDa825A8565cAfceB401EdaC
RewardOracle	0xD3aFE58031998EAf2b0cCeE76dBd8ca50B19DCCa
TransparentUpgradeableProxy (StETHMinter)	0x25a01dBde45cc5Bb7071EB3c3b2F983ea923bec5
StETHMinter	0x76a444fa85d8DA2209D45c6f89D7f51b54FcdDF9

1.5 Summary of findings

Severity	# of Findings
Critical	1
High	4
Medium	3
Low	4

ID	Name	Severity	Status
C-1	Staking deposit can be stolen by a 3rd party	Critical	Fixed
H-1	Slashing and strategy losses have not been taken into design	High	Fixed
H-2	Excessive minting `aETH` permissions for strategies	High	Fixed
H-3	`RewardOracle.submitEpochReward` accepts arbitrary `epochId`	High	Fixed
H-4	Actions limits manipulation	High	Fixed
M-1	Centralization risks	Medium	Acknowledged
M-2	The `supplyReward()` function doesn't check the balance increase	Medium	Fixed
M-3	Race condition in the `submitEpochReward()` start epoch ID	Medium	Fixed
L-1	Redundant `require` statement due to non-decreasing `queueId`	Low	Fixed
L-2	The `Claim.amount` field can be omitted	Low	Fixed

L-3	Constructors may use `_disableInitializers()` instead of initialization routine	Low	Fixed
L-4	The public burn function	Low	Fixed

1.6 Conclusion

The project allows users to participate in ETH 2.0 staking with various deposit sizes. The project's architecture is flexible, supports integrations with external DeFi protocols through strategies, and effectively utilizes funds to obtain the desired financial result with acceptable risk.

The project is centralized, and the security of funds and efficiency depend on the owner's actions, the chosen strategies, and accounting of profits and losses. It is expected that all actions of the owner will be executed through the government, which implements voting and time delay mechanics, thereby allowing users to withdraw funds if they consider a particular proposal unsafe.

The project can compensate for losses of strategies only to the extent of funds available on the treasury balance. In the case of the larger losses, a bank run scenario is possible.

We have performed an investigation on potential attack vectors, listed below, and found out that the project isn't vulnerable to most of them. Any identified vulnerabilities have been fixed during the audit process.

1. **Reentrancy (cross-reentrancy and read-only reentrancy)**

   - We investigated reentrancy and cross-reentrancy attack vectors via any ETH transfers in the `CorePrimary`, `Strategy` cotracts.

2. **Balance manipulation and inflation attacks**

   - We investigated that using donations to manipulate balances would not break the logic of the contracts.
   - We are convinced that an attacker is not able to manipulate rates and shares in an empty saETH vault using donations.

3. **Front-running attacks**

   - We researched the possibility of a front-running attack with the ETH 2.0 deposit contract and found this vector to be feasible (see C1 finding).
   - We verified that an attacker is unable to execute a sandwich attack on the submitting yields action. This type of attack, involving depositing a large amount of underlying tokens prior to yield submission and withdrawing them afterwards to steal rewards (known as a yield stealing attack), is safeguarded because of the time-locking logic.

4. **Data validation**

   - We investigated data validation of all the functions. There have been found important issues in submitting the ETH 2.0 deposit contract and in reward submitting.

5. **Centralization risks**

   - We reviewed the logic of contracts from the perspective of an owner intending to make a rug pull and found it feasible. The government is needed to mitigate this risk

6. **Deployment attacks**

   - We researched that an attacker cannot interrupt the deployment process to make the state of the contracts invalid. Additionally, they are unable to call initialize functions or other functions that could result in loss of the ownership.

2.FINDINGS REPORT

2.1 Critical

C-1	Staking deposit can be stolen by a 3rd party
Severity	Critical
Status	Fixed in 6cc6180d

Description

The specification of ETH2.0 staking allows for two types of deposits: the initial deposit and the top-up deposit, which increases the balance of a previously made initial deposit. Unfortunately, the current implementation of the mainnet deposit contract does not sufficiently distinguish between these types of deposits. This oversight allows an attacker to front-run Aspida's deposit with their own initial deposit, causing Aspida's deposit to be treated as a top-up of the attacker's deposit. Consequently, Aspida's withdrawal credentials will be ignored, and the assets will be accounted for on behalf of the attacker.

This issue is classified as `critial` since it can lead to permanent loss of project liquidity.

Recommendation

Despite this being an architectural issue of ETH2.0 itself, we recommend applying a workaround fix for this issue.

```
        bytes32 actualRoot = depositContract.get_deposit_root();
        if (expectedDepositRoot != actualRoot) {
            revert InvalidDepositRoot(actualRoot);
        }
```

The `expected deposit root` can be calculated using appropriate offchain mechanics, ensuring that no initial deposit was made to the given pubkey at the time of the `expected deposit root` calculation.

Client's commentary

> We will implement the check of deposit root. Our first batches of validators are divided into fully trusted, whitelisted operators and third-party operators. For fully trusted validators, the existing

> solution will be used for depositing so it's more gas optimized. For whitelisted operators, a check of deposit root is added to ensure the security of deposit and mitigate frontrun risk.
>
> In addition, for other third-party node operators, we plan to integrate with DVT platform such as SSV to mitigate the risks.

2.2 High

H-1	Slashing and strategy losses have not been taken into design
Severity	High
Status	Fixed in 6d83f79f

Description

Currently, if a validator is slashed or losses occur within a strategy, there is no established mechanism to distribute these losses among all users. As a result, the supply of `aETH` may fall below the amount of `ETH` locked into the project, preventing some users from redeeming their tokens. Furthermore, users who contribute `ETH` to the project after such events inadvertently assume these risks, which is a deviation from an optimal economic model.

This issue is classified as `high` due to the inequitable distribution of losses, affecting both directly and indirectly involved users.

Recommendation

We recommend implementing a mechanism that equitably distributes the consequences of slashing among existing users, while safeguarding new users from bearing the losses incurred prior to their participation.

Client's commentary

> In case of slashing, the user's losses will be compensated by the protocol's income (the actual income amount at the time of the incident) and the node operator's income (based on the actual income amount). Additionally, a node operator penalty mechanism will be established to manage whitelisted node operators. Subsequently, using DVT technology similar to the SSV network, the risks of slashing will be mitigated.

H-2	Excessive minting `aETH` permissions for strategies
Severity	High
Status	Fixed in d86baafd

Description

The issue is identified in the `CorePrimary.strategyMinting` function.
Currently, the function, which can only be invoked by a `strategy`, permits the minting of an unrestricted amounts of `AETH` to the arbitrary address. It presents a significant risk of overinflating the circulating supply of `AETH`, potentially leading to a scenario where `AETH` is no longer backed on a one-to-one basis with the `ETH` locked in the project.

The issue is classified as `high` due to the excessive authority granted to the `strategy` in regulating the circulating supply of `AETH`.

Related code - strategyMinting: CorePrimary.sol#L257

Recommendation

We recommend removing this function or adding the constraints to ensure that the minted `AETH` amounts are backed by an equivalent `ETH` collateral on a one-to-one basis.

Client's commentary

> The minting permission for aETH is authorized based on DAO governance, which will be subject to community evaluation and voting.

H-3	`RewardOracle.submitEpochReward` accepts arbitrary `epochId`
Severity	High
Status	Fixed in 6cc6180d

Description

This vulnerability is identified in the `RewardOracle._updateEpochReward` function.
The function accepts `epochId` as a parameter from an authorized `manager`. This `epochId` is crucial for imposing restrictions on the rewards issued by the oracle. However, the current implementation lacks validation checks for the `epochId` submitted by the `manager`. Consequently, the `manager` can input any `epochId`, compromising the integrity of reward limit checks. This can lead to erroneous reporting of disproportionately high rewards by the `RewardOracle`, or potentially a DoS attack on the `RewardOracle` contract if the `epochId` is set to the maximum value of the `uint256` type.

This issue is categorized as `high` due to the potential risks of generating invalid, inflated reward reports from `RewardOracle`.

Related code - _updateEpochReward: RewardOracle.sol#L196

Recommendation

We recommend adding constraints to ensure that the `epochId` provided to the `submit` function correlates with the current `block.timestamp`.

Client's commentary

> Combining with 2.3.3, we associate the startEpochId with the corresponding block.timestamp to verify the actual timestamp of the epochId against the block.timestamp, ensuring the authenticity of the epochId.

H-4	Actions limits manipulation
Severity	High
Status	Fixed in d86baafd

Description

An attacker is able to manipulate submit/withdraw limits by doing cycles of submit
CorePrimary.sol#L275
and withdraw
CorePrimary.sol#L291
which leads to disabling these functions for the next users.

Recommendation

We recommend redesigning the logic to mitigate the risk of action lock-ups.

Client's commentary

> The contract has a daily limit mechanism. If the above issues occur, we can set a larger daily limit or cancel the daily limit.

2.3 Medium

M-1	Centralization risks
Severity	Medium
Status	Acknowledged

Description

The project exhibits a centralized structure, primarily under the control of the `owner`, who possesses the ability to withdraw the project's liquidity in a single transaction. Additionally, other components within the project possess excessive control, posing further risks to users' funds. Notably:

- `Strategies` can mint arbitrary amounts of the `aETH` tokens.
- Tokens deposited in `strategies` may not be returned.
- `RewardOracle` features a centralized design, increasing the risk of inaccurate reporting.
- The `aETH` token may have multiple `managers`, implying that entities other than the `CorePrimary` contract might have minting rights.

This issue is rated as `medium` since there are multiple centralized points of vulnerability within the project, each of which requires precise management and security measures.

Recommendation

To mitigate these risks, it is advised to implement `Multisig` accounts for each governance-related address, ensuring a more distributed and secure control mechanism.

Client's commentary

> MixBytes: The problem is partially solved: the strategy minting is removed from the code.

M-2	The `supplyReward()` function doesn't check the balance increase
Severity	Medium
Status	Fixed in 6cc6180d

Description

There is no verification that the specified reward amounts exist in the current implementation of reward delivery.
RewardOracle.sol#L240
In this way, there is a risk of an uncontrolled increase in the overall supply.

Recommendation

We recommend adding checks to ensure that the difference between the contract's balance and the reported rewards is not excessively large.

Client's commentary

> 1. Strategies can mint arbitrary amounts of the aETH tokens.— To ensure the protocol's flexibility and upgradability, we have set up a strategy contract with restricted minting permissions. The specific minting cap will be controlled by the Owner and executed based on the strategy (which will be determined through community voting).
> 2. Tokens deposited in strategies may not be returned.— we understand the risk of funds deposited into various stratigies.
> 3. RewardOracle features a centralized design, increasing the risk of inaccurate reportin — We have added threshold checks to ensure that the reported Reward amount is within a reasonable range.
> 4. The aETH token may have multiple managers, implying that entities other than the CorePrimary contract might have minting rights.—.The current mechanism involves the Owner configuring minting addresses (currently only one CorePrimary), and then managing Owner permissions through a multi-signature approach.
>
> The supplyReward function can only be called by the RewardOracle, which has certain restrictions. The reported reward amount from the RewardOracle will later include earnings from multiple strategies. In this system, the RewardOracle is considered trustworthy.

M-3	Race condition in the `submitEpochReward()` start epoch ID
Severity	Medium
Status	Fixed in eff32fce

Description

The `submitEpochReward()` function is designed to report the reward amount over the interval of one or several epochs. The arguments of this function are the end epoch ID and the reward amount. The start epoch ID is calculated onchain. Due to a potential race condition in the offchain oracle mechanics, there's a possibility that the start epoch ID will be miscalculated. This could lead to reporting extra rewards.

Related code - _updateEpochReward: RewardOracle.sol#L196

Recommendation

We recommend introducing an additional argument, `startEpochId`, and reverting reward reports with mismatching start epoch ID values.

Client's commentary

> In combination with the modification in section 2.2.3.

2.4 Low

L-1	Redundant `require` statement due to non-decreasing `queueId`
Severity	Low
Status	Fixed in 6cc6180d

Description

The issue is identified in the `WithdrawalQueue._withdrawalQueue` function.
The condition checks if `userQueueIds_[_receiver].add(_queueId)` already exists, which is intended to prevent duplicate `queueId` entries for a receiver. However, this check is unnecessary because the `queueId` is non-decreasing by design. This means that each new `queueId` is guaranteed to be unique and larger than the previous ones, rendering the check for an existing `queueId` redundant.

Related code - _withdrawalQueue: WithdrawalQueue.sol#L96

Recommendation

We recommend removing the `require` statement that checks for the existence of a `_queueId`.

Client's Commentary

> In combination with the modification in section 2.2.3.

L-2	The `Claim.amount` field can be omitted
Severity	Low
Status	Fixed in 6cc6180d

Description

An optimization opportunity is identified in the `WithdrawalQueue._withdrawalQueue` function. Currently, the function utilizes `claimData.amount`, which is redundant as its value is always equivalent to the difference between `claims_[id].accumulated` and `claims_[id - 1].accumulated`. Eliminating the use of `Claim.amount` can optimize the function by reducing the number of storage writes, thereby saving gas.

Related code - _withdrawalQueue: WithdrawalQueue.sol#L99

Recommendation

We recommend omitting the `Claim.amount` field and directly calculating the amount as the difference between `claims_[id].accumulated` and `claims_[id - 1].accumulated`.

Client's commentary

> Accept the suggested modification.

L-3	Constructors may use `_disableInitializers()` instead of initialization routine
Severity	Low
Status	Fixed in 6cc6180d

Description

The constructors of the contracts listed below use a call to their initialization routine to avoid a problem known as unauthorized initialization of the implementation. Generally, this is a usable solution; however, the OpenZeppelin project offers a more elegant way to disable initialization.

- CorePrimary: CorePrimary.sol#L63
- dETH: dETH.sol#L31
- RewardOracle: RewardOracle.sol#L61
- StETHMinter: StETHMinter.sol#L22

Recommendation

We recommend using `_disableInitializers()` in the constructors instead of calling the initialization routine.

Client's commentary

> Accept the suggested modification.

L-4	The public burn function
Severity	Low
Status	Fixed in eff32fce

Description

The `aETH` contract is inherited from OpenZeppelin `ERC20BurnableUpgradeable` and has the public `burn()` method that just burns the given amount of tokens on the caller balance without returning the deposited ETH. As a result, the total supply of ETH will be different from the deposited ETH amount. In addition to that, potentially in some cases this issue can be used for total supply manipulation.

Recommendation

We recommend overriding and disabling the `burn()` function.

Client's commentary

> Accept the suggested modification.

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