shieldify

Ion Protocol

Vault and Core Upgrade

SECURITY REVIEW

Date: 10 June 2024

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Ion Protocol - Vault & Core Upgrade

Ion Protocol is a decentralized money market purpose-built for all types of staked and re-staked assets. Ion protocol unlocks capital efficiency for yield-bearing staking collaterals using reactive interest rates, collateral-specific utilization, and price-agnostic liquidations. Borrowers can collateralize their yield-bearing staking assets to borrow WETH, and lenders can gain exposure to the boosted staking yield generated by borrower collateral.

- Core Upgrade:

1. This upgrade introduces a few changes to the lending pool contract to improve UX and to introduce gas improvements.
2. Introducing transferrability of the `RewardToken` following ERC20 interface.
3. Creating a Lens contract for view functions and exposing the storage of IonPool used in upgradeability.

- Ion Lending Vault:

1. Users who want exposure to multiple markets on Ion can deposit into a vault managed by a third party. This third party can receive deposits and rebalance all deposits across multiple pairs. For example, if users deposit `100 wstETH` into the vault, the vault manager can distribute the asset by lending `20 wstETH` into the `weETH/wstETH` market, `50 wstETH` into the `rsETH/wstETH` market, and `30 wstETH` into the `rswETH` market.

Take a deep dive into ion's documentation here.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review was conducted in the span of 13 days and was part of Shieldify's Private Pool competitions. A total of 73 researchers participated and identified four issues of Medium and Low severity, around plausibility, bad UX, input sanitization and potential for DoS.

5.1 Protocol Summary

Project Name	**Ion protocol – Vault and Core Upgrade**
Repository	ion-protocol
Type of Project	DeFi, Assets Lending Platform - Vault and Core Upgrade
Audit Timeline	13 days
Review Commit Hash	c7bf309e9dba7e19eb81e456fa4cd3254e5e65b9
Fixes Review Commit Hash	a073854c96d718a8af3d6f1193e8d9e607006224

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
src/IonPool.sol	524
src/periphery/IonLens.sol	246
src/token/RewardToken.sol	275
src/vault/Vault.sol	434
src/vault/VaultFactory.sol	20
Total	**1499**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Critical** and **High** issues: **0**
- **Medium** issues: **1**
- **Low** issues: **3**

ID	Title	Severity	Status
[M-01]	If a Pool at the Beginning of `withdrawQueue()` Is Paused, the Depositor Won't Be Able to Redeem from Other Active Pools	Medium	Fixed
[L-01]	Attacker Can Make User's `createVault()` Call Fail Leading to a Bad User Experience	Low	Fixed
[L-02]	Missing `feePercentage` Check in the `constructor()`	Low	Acknowledged
[L-03]	Vault Owner Can DoS the `withdraw/redeem()` Functionality By Setting the `feeRecipient` to `address(0)`	Low	Acknowledged

7. Findings

[M-01] If a Pool at the Beginning of `withdrawQueue()` Is Paused, the Depositor Won't Be Able to Redeem from Other Active Pools

Severity

Medium Risk

Reported By

Egis Security

Description

When a user wants to withdraw his stakings, he returns his shares by calling `withdraw()` or `redeem()`, those two functions calls withdrawFromIonPool(), which is then iterating over

`withdrawQueue()` from the first added pool. The problem here is that if a pool is paused, the tx will revert here, because pool.withdraw() has whenNotPaused modifier and user won't be able to receive his funds, even if there is available liquidity in next pools in the queue. The impact is a temporary DoS.

NOTE The same is valid for `deposit()`.

Location of Affected Code

File src/vault/Vault.sol

Recommendation

Check if the pool is paused and if so, skip withdrawal from it.

Team Response

This is true. A paused IonPool will lead to a temporary DoS of the vault's `withdraw` or `redeem`. In general, the `pause` on the IonPool is used for incident response scenarios such as a potential exploit or upon observing contract upgrades of integrated protocols, and pausing vault interactions as a byproduct of the IonPool pause is intended behavior.

However, we see that it would be more accurate to return zero values for depositable or withdrawable amounts when the pool is paused rather than to allow reverts. The likelihood is low, but the impact could be high, so we believe this issue is a medium. Will fix.

[L-01] Attacker Can Make User's `createVault()` Call Fail Leading to a Bad User Experience

Severity

Low Risk

Reported By

djxploit

Description

In the contract `VaultFactory.sol`, the function `createVault()` is vulnerable to frontrunning, due to missing `msg.sender` parameter.

```
vault = new Vault{ salt: salt }(
  baseAsset, feeRecipient, feePercentage, name, symbol, initialDelay,
     initialDefaultAdmin
);
```

An attacker can front-run the victim's `createVault()` transaction call with their own `createVault()` call request, with the same `salt` parameters passed by the victim. This would create the exact address that would have been created by the victim's `createVault()` call since the `salt` and

other parameters are the same. But when the victim's transaction is executed, the address is non-empty so the EVM would revert the victim's call. This would result in a bad UX for a user, who thinks the vault creation did not succeed, as his call had reverted.

Impact

The front-running would make the victim's `createVault()` function fail. This would result in a bad UX for a user, who will think that the vault has not been created, due to the revert error. Though the vault is already deployed at that address by the attacker (due to front-running), as the victim's call had failed, so in the UX it would be a bad experience for the victim user. He wouldn't know that the contract is deployed but instead would think that the vault is not deployed as his call had failed. This would result in a bad user experience.

Attack Scenario

1. The victim calls `createVault()`.
2. The attacker monitors mempool, and front-runs it with the same parameters by calling the `createVault()` function.
3. The attacker's call gets executed, and a contract is deployed at the specific address.
4. Finally when the victim's `createVault()` call gets executed, it will revert, as a contract is already present in the specific address.
5. Due to this victim's call the tx will revert, making him believe that the contract has not been created.

Location of Affected Code

File src/vault/VaultFactory.sol#L38

```
function createVault(
  IERC20 baseAsset,
  address feeRecipient,
  uint256 feePercentage,
  string memory name,
  string memory symbol,
  uint48 initialDelay,
  address initialDefaultAdmin,
  bytes32 salt
)
  external
  returns (Vault vault)
{
// TODO use named args syntax
  vault = new Vault{ salt: salt }(
    baseAsset, feeRecipient, feePercentage, name, symbol, initialDelay,
       initialDefaultAdmin
  );

  emit CreateVault(address(vault), baseAsset, feeRecipient, feePercentage
     , name, symbol, initialDefaultAdmin);
}
```

Recommendation

Consider adding the `msg.sender` parameter while creating the vault.

Team Response

The recommendation for using the `msg.sender` parameter along with the `salt` is acknowledged and this change will be made. The bad UX is low impact, and the likelihood is also low due to the lack of incentivization for frontrunning vault creations.

[L-02] Missing `feePercentage` Check in the `constructor()`

Severity

Low Risk

Reported By

Silvermist

Description

Inside `updateFeePercentage()` there is a check if the `_feePercentage` is more than the `RAY` value. One `RAY` is 100% and the `_feePercentage` should not be more than this. However, there is no such check inside the constructor meaning when deploying the Vault, the `feePercentage` can be set to more than 100%.

Impact

The `feePercentage` can be set to more than 100%.

Location of Affected Code

File src/vault/Vault.sol#L117

Recommendation

Add this if inside the constructor.

```
if (_feePercentage > RAY) revert InvalidFeePercentage();
```

Team Response

Acknowledged.

[L-03] Vault Owner Can DoS the `withdraw/redeem` Functionality By Setting the `feeRecipient` to `address(0)`

Severity

Low Risk

Reported By

Egis Security

Description

A pool owner can set `feeRecipient`, which would receive a fee in form of shares on all ERC4626-related operations.

```
function _accrueFee() internal returns (uint256 newTotalAssets) {
  uint256 feeShares;
  (feeShares, newTotalAssets) = _accruedFeeShares();
  if (feeShares != 0) _mint(feeRecipient, feeShares);

  lastTotalAssets = newTotalAssets;

  emit FeeAccrued(feeShares, newTotalAssets);
}
```

He can set the address to `address(0)`, which would result in a revert each time a user tries to withdraw/redeem/deposit because we use OZ implementation of ERC4626, which uses OZ implementation of ERC20, which revert on the following line if `recipient` is `address(0)`.

Function flow:

1. withdraw() -> _accrueFee() -> ERC20::_mint()

Location of Affected Code

File src/vault/Vault.sol#L774

Recommendation

Check if `feeRecipient == address(0)` and if so, skip the line that mints him shares.

Team Response

Acknowledged.

shieldify

Thank you!