Predeposit contract ethereum Security Review

Auditors

Desmond Ho, Lead Security Researcher

Rikard Hjort, Lead Security Researcher

Sujith Somraaj, Security Researcher

Report prepared by: Lucas Goiriz

January 31, 2025

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Byzantine is a trustless and efficient restaking layer with permissionless strategy creation that enables the deployment of minimal, individual, and isolated restaking strategy vaults.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Predeposit contract ethereum according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 2 days in total, Byzantine Finance engaged with Spearbit to review the predeposit-contract-ethereum protocol. In this period of time a total of **12** issues were found.

Summary

Project Name	Byzantine Finance
Repository	predeposit-contract-ethereum
Commit	8a8ab1c2
Type of Project	Vaults, Staking
Audit Timeline	Jan 20th to Jan 22nd

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	4	4	0
Gas Optimizations	3	3	0
Informational	3	3	0
Total	**12**	**12**	**0**

The Spearbit team reviewed Byzantine Finance's `predeposit-contract-ethereum` holistically on commit hash `ac5cbe7` and concluded that all issues were fixed and no new vulnerabilities were introduced.

5 Findings

5.1 Medium Risk

5.1.1 Limited user control over share allocation causes unexpected behavior during vault deposits

Severity: Medium Risk

Context: ByzantineDeposit.sol#L197

Description: In the `moveToVault()` function, users deposit tokens into Byzantine vaults but have no control over the minimum number of shares they receive in return. The function directly accepts whatever number of shares the vault provides without any slippage protection:

```
function moveToVault(
    IERC20 _token,
    address _vault,
    uint256 _amount,
    address _receiver
) external onlyWhenNotPaused(PAUSED_VAULTS_MOVES) nonReentrant {
    // ... validation checks ...

    if (_token == beaconChainETHToken) {
        IERC7535(_vault).deposit{value: amount}(amount, _receiver);
    } else {
        // No minimum shares check here
        IERC4626(_vault).deposit(amount, _receiver);
    }
}
```

This is problematic because:

- The exchange rate between tokens and vault shares can fluctuate.
- Users have no way to specify acceptable limits for the shares they receive.

This could lead to users receiving fewer shares than anticipated if the vault's share price is affected by other on-chain activities.

Proof of Concept: Demonstrated the issue using a vault that won't mint any shares in the `deposit()` call.

```
contract ERC7535MockZero is ERC7535Mock {
    constructor(string memory _name, string memory _symbol) ERC7535Mock(_name, _symbol) {}

    function deposit(uint256 amount, address receiver) payable public override returns (uint256 shares)
    ↪  {
        return 0;
    }
}

function test_moveToVault_ZeroShares() public {
        _depositETH(alice, 1 ether);
        _unpauseVaultMoves();
        _recordVaults();

        vm.startPrank(alice);
        deposit.moveToVault(
            beaconChainETHToken, address(vault7535ETH), 1 ether, alice
        );
}
```

Recommendation: Modify `moveToVault()` to accept a `minSharesOut` parameter and validate the received shares:

```
  function moveToVault(
      IERC20 _token,
      address _vault,
      uint256 _amount,
      address _receiver,
+     uint256 _minSharesOut
  ) external onlyWhenNotPaused(PAUSED_VAULTS_MOVES) nonReentrant {
      // ....
      if (_token == beaconChainETHToken) {
+         sharesBefore = IERC7535(_vault).balanceOf(_receiver);
          IERC7535(_vault).deposit{value: _amount}(_amount, _receiver);
+         sharesAfter = IERC7535(_vault).balanceOf(_receiver);
      } else {
          if (_token == stETHToken) {
              _amount = wstETH.unwrap(_amount);
          }
          _token.approve(_vault, _amount);
+         sharesBefore = IERC4626(_vault).balanceOf(_receiver);
          IERC4626(_vault).deposit(_amount, _receiver);
+         sharesAfter = IERC4626(_vault).balanceOf(_receiver);
      }

+     uint256 sharesReceived = sharesAfter - sharesBefore;
+     require(sharesReceived >= _minSharesOut, "ByzantineDeposit: insufficient shares received");
      // ....
  }
```

Byzantine Finance: Fixed in commit 781df3db.

Spearbit: Fix verified.

5.1.2 No user safeguards against malicious or unrecoverable pause

Severity: Medium Risk

Context: ByzantineDeposit.sol#L167

Description: In the event that a withdrawal unpauser or owner is malicious or permanently offline, user funds can be stuck indefinitely. There is hence no guarantee, once the funds have been deposited, that the user can retrieve the funds eventually in case they want to not move the funds to a vault. Hence they need to put full trust in the owner.

Recommendation: Implement a guaranteed withdrawal, with a delay. If a user requests a special withdrawal when withdrawals are paused, set a timestamp for e.g. 2 weeks, after which point the user can withdraw.

```
// Enables the sender to withdraw the full amount of the specified token after a certain delay, e.g. 2
↪  weeks.
function requestDelayedWithdrawal(IERC20 _token, address _recipient) external nonReentrant;
// Withdraw the specified amount of tokens, conditional on delay time having passed.
function executeDelayedWithdrawal(IERC20 _token, uint256 _amount,
```

Ensure that if the user has initiated a delayed withdrawal they do not receive any points from that point on, and cannot send the funds to the vault. Otherwise the feature can be misused by a user simply initiating such a withdrawal immediately, thus circumventing pauses while still reaping the benefits of the predeposit.

Byzantine Finance: Fixed in commit b4f732a5. We've decided to never pause the withdrawals (i.e remove `onlyWhenNotPaused(PAUSED_WITHDRAWALS)` from `withdraw()`).

Spearbit: Fixed.

5.2 Low Risk

5.2.1 Replace unsafe `approve()` usage with `forceApprove()`

Severity: Low Risk

Context: ByzantineDeposit.sol#L154, ByzantineDeposit.sol#L220

Description: The `moveToVault()` function is used by users to move their tokens from the `ByzantineDeposit` contract to an external ERC4626 / ERC7535 compliant vault. To finish the deposit process, the contract uses the standard `approve()` function to authorize the Byzantine vault to deposit tokens:

```
_token.approve(_vault, amount);
IERC4626(_vault).deposit(amount, _receiver);
```

This usage of `approve()` is unsafe and may fail with specific tokens that have non-standard approval behavior, such as USDT.

Recommendation: Consider using `forceApprove()` from OpenZeppelin's SafeERC20.sol library instead of the standard `approve()`:

```
_token.forceApprove(_vault, amount);
IERC4626(_vault).deposit(amount, _receiver);
```

Byzantine Finance: Fixed in commit 8acf17e2.

Spearbit: Fix verified.

5.2.2 User cannot specify a withdraw address

Severity: Low Risk

Context: ByzantineDeposit.sol#L167

Description: The withdraw function relies on `msg.sender` as the recipient of funds. This limits the possibility for certain contracts and for certain businesses, which may prefer separating deposit and withdrawal addresses. If, for any reason, a depositor cannot or should not receive funds due to any issues in a contract, in accounting, or in the management of an externally owned account or multisig, it would be preferable if the funds could be sent to a specified address.

Recommendation: Add a parameter, `_receiver`, and have the funds be transferred to that address.

Byzantine Finance: Fixed in commit 9d03b2aa.

Spearbit: Fix verified.

5.2.3 No functionality for removing support for a token deposit

Severity: Low Risk

Context: ByzantineDeposit.sol#L245-L253

Description: If a token is supported but ends up being deemed buggy, or a token is erroneously supported, there is no way to drop support. For instance, one can imagine whitelisting the wrong token, in which case the contract may need to be redeployed, or funds get stuck. It would be beneficial to be able to remove the support before any of those tokens are deposited, as soon as the error is discovered.

Recommendation: Add a function for delisting a single token, that takes a boolean toggle. It's unlikely that you will need a function accepting an array for this scenario.

Byzantine Finance: Fixed in commit 65352e26.

Spearbit: Fix verified.

5.2.4 No functionality for removing support for a vault

Severity: Low Risk

Context: ByzantineDeposit.sol#L274-L283

Description: If a vault is supported but ends up being deemed buggy, or a vault is erroneously supported, there is no way to drop support. For instance, one can imagine whitelisting the wrong vault, in which case there is a risk of a loss of funds. It would be beneficial to be able to remove the support before any of those vault are deposited to, as soon as the error is discovered.

Recommendation: Add a function for delisting a single vault, that takes a boolean toggle. It's unlikely that you will need a function accepting an array for this scenario.

Byzantine Finance: Fixed in commit 30932059.

Spearbit: Fix verified.

5.3 Gas Optimization

5.3.1 Remove unnecessary variable initializations

Severity: Gas Optimization

Context: ByzantineDeposit.sol#L88, ByzantineDeposit.sol#L152, ByzantineDeposit.sol#L176, ByzantineDeposit.sol#L212

Description: The `ByzantineDeposit.sol` contract contains multiple instances where variables are explicitly initialized to their default values or where new variables are declared when existing ones could be reused. This wastes gas as Solidity initializes variables to defaults, and new declarations increase stack usage.

Recommendation: Consider removing the unused variable initialization to optimize the code.

Found instances:

1. In `moveToVault()`, a new amount variable is created when the existing `_amount` parameter could be reused:

```
   function moveToVault(
   IERC20 _token,
   address _vault,
   uint256 _amount,
   address _receiver
   ) external onlyWhenNotPaused(PAUSED_VAULTS_MOVES) nonReentrant {
   // ...
-      uint256 amount = _amount;
       // ...
       else if (_token == stETHToken) {
-          amount = wstETH.unwrap(_amount);
+          _amount = wstETH.unwrap(_amount);
       }
       // ...
   }
```

2. In `depositERC20()` and `withdraw()`, same unnecessary amount variable initialization:

```
  function depositERC20(
  IERC20 _token,
  uint256 _amount
  ) external onlyWhenNotPaused(PAUSED_DEPOSITS) onlyIfCanDeposit(msg.sender) {
    // ...
-   uint256 amount = _amount;
    if (_token == stETHToken) {
        stETHToken.approve(address(wstETH), _amount);
-       amount = wstETH.wrap(_amount);
+       _amount = wstETH.wrap(_amount);
        // ...
  }
```

3. In `recordByzantineVaults()`, the loop counter is initialized to 0, which is redundant:

```
function recordByzantineVaults(
    address[] calldata _vaults
) external onlyOwner {
-     for (uint256 i = 0; i < _vaults.length;)
+     for (uint256 i; i < _vaults.length;)
    // ...
```

4. The state variable `isPermissionlessDeposit` is initialized to false, which is redundant:

```
- bool public isPermissionlessDeposit = false;
+ bool public isPermissionlessDeposit;
```

Byzantine Finance: Fixed in commit c8bffaec.

Spearbit: Fix verified.

5.3.2 Use custom errors instead of `require`

Severity: Gas Optimization

Context: ByzantineDeposit.sol#L96, ByzantineDeposit.sol#L130, ByzantineDeposit.sol#L147, ByzantineDeposit.sol#L168, ByzantineDeposit.sol#L179, ByzantineDeposit.sol#L203, ByzantineDeposit.sol#L204, ByzantineDeposit.sol#L234, ByzantineDeposit.sol#L248

Description: `revert` with a custom error is more gas efficient than `require`.

Recommendation: Change

```
require(isByzantineVault[_vault], "ByzantineDeposit.moveToVault: vault is not recorded");
```

to

```
if (!isByzantineVault[_vault]) revert VaultNotRecorded();
```

with a top level declared error:

```
error VaultNotRecorded();
```

Byzantine Finance: Fixed in commit a01d5fdf.

Spearbit: Fix verified.

5.3.3 Allow setting whitelisted addresses with an array

Severity: Gas Optimization

Context: ByzantineDeposit.sol#L233-L237

Description: There are likely many addresses that need whitelisting. One transaction per address can be tedious to create and cost more gas than whitelisting a batch of addresses. Also, there is no need to check for the 0 address. The 0 address cannot deposit anyway.

Recommendation: Change the signature to take an array of structs, which take an address and a boolean. Or simply an array of addresses to be whitelisted. Remove the 0 address check.

Byzantine Finance: Fixed in commit fc535c98.

Spearbit: Fix verified.

5.4 Informational

5.4.1 Use `Ownable2Step` from OpenZeppelin instead of `Ownable`

Severity: Informational

Context: ByzantineDeposit.sol#L5

Description: The `ByzantineDeposit.sol` contract inherits from OpenZeppelin's Ownable contract and uses its single-step ownership transfer pattern. This is risky as it can lead to the contract becoming permanently ownerless if the ownership is accidentally transferred to an incorrect or inaccessible address.

Recommendation: Consider using OpenZeppelin's `Ownable2Step` instead of Ownable. This implements a two-step ownership transfer pattern:

```
- import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
+ import {Ownable2Step} from "@openzeppelin/contracts/access/Ownable2Step.sol";

- contract ByzantineDeposit is Ownable, Pausable, ReentrancyGuard {
+ contract ByzantineDeposit is Ownable2Step, Pausable, ReentrancyGuard {
```

Byzantine Finance: Fixed in commit d2fbafab.

Spearbit: Fix verified.

5.4.2 Increase test coverage

Severity: Informational

Context: ByzantineDeposit.sol#L41

Description: There is less than complete test coverage of key contracts under review. Adequate test coverage and regular reporting is an essential process in ensuring the codebase works as intended. Insufficient code coverage may lead to unexpected issues and regressions arising due to changes in the underlying smart contract implementation.

Recommendation: Add to test coverage ensuring all execution paths are covered.

```
function test_depositETH_ZeroValue() public {
        vm.startPrank(alice);
        vm.expectRevert();
        deposit.depositETH();
    }

    function test_setCanDeposit_ZeroAddress() public {
        vm.startPrank(byzantineAdmin);
        vm.expectRevert();
        deposit.setCanDeposit(address(0), true);
    }

    function test_withdraw_failTransferEth() public {
        vm.startPrank(byzantineAdmin);
        deposit.setCanDeposit(address(scUser), true);

        _unpauseWithdrawals();

        deal(address(scUser), 1 ether);
        vm.startPrank(address(scUser));
        deposit.depositETH{value: 1 ether}();

        IERC20 token = deposit.beaconChainETHToken();
        vm.expectRevert();
        deposit.withdraw(token, 1 ether);
    }
```

Byzantine Finance: Fixed in commit 5b198fb7.

Spearbit: Fix verified.

5.4.3 Sanity check deposit asset matches vault

Severity: Informational

Context: ByzantineDeposit.sol#L213-L221

Description: Consider checking that the asset to be pulled matches with the vault asset. While such a check isn't necessary because:

- Calling an `ERC4626` vault with ETH will revert because `deposit()` isn't payable.
- Calling a `ERC7535` vault with an ERC20 will revert with `ERC7535AssetsShouldBeEqualToMsgValue()`, at least with the given mock implementation. This may not apply generally, but the spec does say that `deposit()` MUST rely on `msg.value`.

  MUST use `msg.value` as the primary input parameter for calculating the shares output.
- Calling a `ERC4626` vault with an incorrect ERC20 should revert due to insufficient approval.

It would be beneficial for a clearer revert reason.

Proof of Concept:

```
function test_Move_MismatchedVaultReverts() public {
  vm.prank(byzantineAdmin);
  deposit.addDepositToken(fUSDC);

  // Alice deposits various assets
  _depositETH(alice, 5 ether);
  _depositStETH(alice, 5 ether);
  _depositfUSDC(alice, 5 ether);

  // Unpause withdrawals
  _unpauseVaultMoves();

  // Record the vault
  _recordVaults();

  // Case 1: Should revert when trying to move ETH to a ERC4626 vault because non-payable
  vm.startPrank(alice);
  vm.expectRevert();
  deposit.moveToVault(beaconChainETHToken, address(vault4626fUSDC), 5 ether, alice);

  // Case 2: Should revert when trying to move ERC20 tokens to a ERC7535 vault
  uint256 depositAmount = deposit.depositedAmount(alice, stETH);
  vm.expectRevert(abi.encodeWithSelector(ERC7535.ERC7535AssetsShouldBeEqualToMsgValue.selector));
  deposit.moveToVault(stETH, address(vault7535ETH), depositAmount, alice);

  // Case 3: Should revert when trying to move mismatched ERC20 tokens to a ERC4626 vault
  // testing with mainnet, revert reason from stETH.transferFrom() is ALLOWANCE_EXCEEDED
  vm.expectRevert();
  deposit.moveToVault(fUSDC, address(vault4626stETH), 5 ether, alice);
}
```

Recommendation:

```
+ require(address(_token) == IERC4626(_vault).asset(), "mismatching assets");
  if (_token == beaconChainETHToken) {
      // ...
  }
```

Byzantine Finance: Fixed in commit 64ed6b9d.

Spearbit: Fix verified.