our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Builda

SECURITY REVIEW

Date: 22 April 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 working with the best talents in the space.

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 Builda

Builda is a restaking ETH aggregator that allows users to deposit ETH and LRT tokens to receive bldETH in return. Users can deposit ETH or LRT tokens to receive bldETH in return, Builda protocol supports kelp(rsETH), renzo(ezETH) and etherFI(eETH) tokens, and user can decide which LRT token to deposit to the Builda protocol and the **Aggregator.sol** contract will divide the deposited amount on the three supported LRT protocols(tokens) based on their weight which is set by the admin of the **Aggregator.sol**, this makes it easier for users to participate in different LRT protocols by depositing using Builda protocol.

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 lasted 10 days and a total of 350 hours have been spent by the four smart contract security researchers from the Shieldify team.

Overall, the code is well-written and implements foundational good practices. The audit report contributed by identifying several issues of varying severity, impacting improper initialization setup and potential manipulation of the **bldETH** amount, among other less severe findings.

The developers at Builda have done an excellent job with their testing suite - 100% unit and integration tests coverage, together with fuzzing and formal verification for some of the functionality.

The lack of extensive documentation required a lot of explanation from the Builda team but with swift and effective communication all questions on our end were clarified in detail.

We want to extend our gratitude to the Builda team for their splendid communication and professionalism.

5.1 Protocol Summary

Project Name	**Builda**
Repository	builda-protocol
Type of Project	Aggregator to the Restaking Ecosystem
Audit Timeline	14 days
Review Commit Hash	23066dc1c56a59cf84e971b678aa7b5da98be0ab
Fixes Review Commit Hash	c7e5b975867d1979a3ec29814865f98ee79ad766

5.2 Scope

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

File	nSLOC
src/interfaces/IAggregator.sol	18
src/interfaces/IBLDETH.sol	5
src/interfaces/ILRTProtocolsManager.sol	3
src/Aggregator.sol	368
src/BLDETH.sol	191
src/LRTProtocolsManager.sol	82
Total	**667**

6. Findings Summary

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

- **Critical** and **High** issues: **0**
- **Medium** issues: **2**
- **Low** issues: **5**

ID	Title	Severity	Status
[M-01]	Attacker Can Manipulate the Amount of **bldETH** that Should Be Minted to Users	Medium	Fixed
[M-02]	Initializing **Aggregator** Contract Fails If **setTargetWeights()** Function Is Not Called Manually Beforehand	Medium	Acknowledged
[L-01]	Discrepancy Between Intended and Actual Behavior	Low	Fixed
[L-02]	Users Choosing Which LRT Token to Withdraw May Harm Other Users	Low	Fixed
[L-03]	Withdraw/Deposit Function Lacks to Call **updateRSETHPrice()**	Low	Fixed
[L-04]	Centralization Risk	Low	Fixed
[L-05]	Unnecessary Checks	Low	Fixed

7. Findings

[M-01] Attacker Can Manipulate the Amount of bldETH that Should Be Minted to Users

Severity

Medium Risk

Description

The function **depositLRT()** allows the user to deposit LRT eth tokens to the protocol and mint bldETH tokens to the user, however, the calculation that happened in the mint function opens the possibility for a malicious user to manipulate it, this is possible because the mint function depends on the calculation below: **amount * getTotalShares / getTotalAssets**

The **totalAsset** will return the **balanceOf()** to the **Aggregator.sol** contract this allows a malicious user to send LRT eth directly to the contract and manipulate the minted amount for each user, probably this scenario happens by someone who wants to mint more **bldETH** with less LRT depositing, the PoC below shows how transfer LRT directly can affect the mint calculation.

Proof of Concept

```
function testDepositAttackSameBlock() public virtual {
// Note: Remove Bob transfer to show the fair amount that Alice should
   get (bldETH amount)
    uint256 rounding = 4;
    vm.startPrank(bob);
    uint256 amount = 1 ether;
    vm.deal(bob, amount);
    super.depositDirectOnLRT(s_eETH, amount, bob);
    s_eETH.transfer(address(s_aggregator), amount);
    vm.stopPrank();

    uint256 amount1 = 1 ether;
    vm.deal(alice, amount1);
    vm.startPrank(alice);
    super.depositDirectOnLRT(s_eETH, amount1, alice);
    super.approveAndDepositLRTOnBuilda(s_eETH, alice);
    vm.stopPrank();

    uint256 amount2 = 1 ether;
    vm.deal(alice, amount2);
    vm.startPrank(alice);
    super.depositDirectOnLRT(s_eETH, amount2, alice);
    super.approveAndDepositLRTOnBuilda(s_eETH, alice);

    uint256 aliceBLDBalance = s_aggregator.bldETH().balanceOf(alice);
    uint256 aliceActualDepositedAmount = amount1 + amount2;

    console.log("Alice BLD Balance: ", aliceBLDBalance);

    assertEq(aliceBLDBalance + rounding, aliceActualDepositedAmount);
    vm.stopPrank();
}
```

Location of Affected Code

File: src/Aggregator.sol#L538

```
function totalAssets() public view returns (uint256, uint256, uint256) {
    return (
        etherFiEETH.asset.balanceOf(address(this)),
        renzoEzETH.asset.balanceOf(address(this)),
        kelpRsETH.asset.balanceOf(address(this))
    );
}
```

Recommendation

Consider creating a mapping which handles any deposit and withdrawal rather than depending on the **totalAssets()** function which makes use of **balanceOf(address(this))**.

Team Response

Fixed.

[M-02] Initializing Aggregator Contract Fails If setTargetWeights() Function Is Not Called Manually Beforehand

Severity

Medium Risk

Description

The **initialize()** function inside the **Aggregator** is called following a specific mechanism to prevent the first deposit issue, this mechanism will deposit the amount of eth to the LRT protocol and divide it among the supported LRT protocols depending on their weight, this happens by calling the `_depositOnLRTProtocols()` function.

The `_depositOnLRTProtocols()` function calls the deposit function of each LRT protocol and the etherFi deposit function and kelp Deposit function will revert when **msg.value == 0**, and if we take a look at the **initialize** function we can see that the **targetWeight** for each protocol is not set only the asset address is set.

This will prevent deploying the aggregator contract because the calculation in the `_depositOnLRTProtocols()` function will return zero disregarding what is the value of **msg.value** if **setTargetWeights()** is not called beforehand.

Location of Affected Code

File: src/Aggregator.sol

```
function initialize(
    IConfigAggregator memory _configAggregator,
    address _lrtManager,
    IAssets memory _assets
) external payable initializer {
    uint256 initialAmount = msg.value;
    if (initialAmount == 0) {
        revert Aggregator__AmountMustBeGreaterThanZero();
    }
    __Ownable_init(msg.sender);
    __ReentrancyGuard_init();
    __UUPSUpgradeable_init();
    bldTreasury = _configAggregator.treasury;
    bldETH = BLDETH(_configAggregator.bldETH);
    lrtManager = LRTProtocolsManager(_lrtManager);
    etherFiEETH = _assets.etherFieEth;
    renzoEzETH = _assets.renzoEzEth;
    kelpRsETH = _assets.kelpRsETH;
    allowedTokens[address(_assets.etherFieEth.asset)] = true;
    allowedTokens[address(_assets.renzoEzEth.asset)] = true;
    allowedTokens[address(_assets.kelpRsETH.asset)] = true;
    admins[_configAggregator.admin] = true;
    exitBaseFee = 0;
    exitFeeLow = 100;
    feeFactor = 0;

    _depositOnLRTProtocols(initialAmount);
    _mintInitialShares();
}
```

```
// code

function _depositOnLRTProtocols(uint256 _amount) private {
// check if etherFi deposit is not paused
    lrtManager.etherFiDeposit().deposit{value: _amount * etherFiEETH.
       targetWeight / TOTAL_WEIGHT_BASIS}();
    lrtManager.renzoDeposit().depositETH{value: _amount * renzoEzETH.
       targetWeight / TOTAL_WEIGHT_BASIS}();
    IKelpLRTDepositPool lrtKelpDeposit = lrtManager.kelpDeposit();
    uint256 ethAmountForKelp = _amount * kelpRsETH.targetWeight /
       TOTAL_WEIGHT_BASIS;
    uint256 minReceiveFromDeposit = lrtKelpDeposit.getRsETHAmountToMint(
       address(ETH_TOKEN), ethAmountForKelp);
    lrtKelpDeposit.depositETH{value: ethAmountForKelp}(
       minReceiveFromDeposit, "0");
    emit Deposit(msg.sender, _amount, address(0));
}
```

Recommendation

Consider executing the **setTargetWeights()** to set the target's weight before calling the `_depositOnLRTProtocols()` function.

Team Response

Acknowledged and fixed as proposed.

[L-01] Discrepancy Between Intended and Actual Behavior

Severity

Low Risk

Description

The documentation does not correspond to the actual implementation in several places and this can lead to serious issues:

- In the **Aggregator** contract the validation of the parameter `_feeFactor` in **setFeeStructure()** should ensure the value is greater than zero according to the NatSpec documentation. This validation is missing which could lead to removing the withdrawal fee. However this function can be called only by the admin, therefore it is a centralization risk.

Location of Affected Code

File: src/Aggregator.sol#L354

```
...
//_feeFactor must be higher than zero.

function setFeeStructure(uint256 _exitBaseFee, uint256 _exitFeeLow,
    uint256 _feeFactor) external onlyAdmin {
```

Recommendation

Consider implementing the described validation check in the **setFeeStructure()** function in the **Aggregator.sol** contract.

Team Response

Acknowledged and fixed as proposed.

[L-02] Users Choosing Which LRT Token to Withdraw May Harm Other Users

Severity

Low Risk

Description

The function **withdrawLRT()** allows users to burn the **bldETH** amount and get in return an equal value of LRT tokens, users can decide which LRT token they want to get by setting the **assetToWithdraw** variable in the **withdrawLRT()** function, this opens the possibility to the below scenario to happen.

- Alice deposits 10 rsETH tokens by calling the **depositLRT()** function and gets 10 bldETH in return. Afterwards, Bob deposits 10 eETH tokens and gets 10 bldETH in return.
- Time passes and the kelpDAO(rsETH) protocol got exploited or faced issues and the rsETH price is now worth less than 1 eth for each rsETH.
- Alice notices this and decides to call the **withdrawLRT()** function to burn 10 bldETH and get **eETH** in return instead of the **rsETH** that she deposited, eETH is worth 1 eth for each token, and Alice gets 10 eETH tokens back.
- Bob decides to call the **withdrawLRT()** function to burn 10 bldETH and in return, he is forced to get rsETH instead of eETH because Alice got all eETH in the protocol, Bob will gain a huge loss because of Alice's action and he'll gain less amount worth of eth because of the rsETH price exploit/issue.

This allows any user to get any LRT token which could lead to other users losing trust and even their funds.

Location of Affected Code

File: src/Aggregator.sol#L272

```
function withdrawLRT(address assetToWithdraw, uint256 bldShares)
    external
    payable
    nonReentrant
    isAllowed(assetToWithdraw)
    notPaused
{
```

Recommendation

Consider adding a mapping that stores the balance of users according to the LRT they deposit to prevent this issue.

Team Response

Acknowledged and fixed as proposed.

[L-03] Withdraw/Deposit Function Lacks to Call updateRSETHPrice()

Severity

Low Risk

Description

Both **depositLRT/withdrawLRT** functions depend on the **rsETHPrice** directly when the user deposits or withdraws the **rsETH** token, this can lead to price manipulation because the **rsETH** price can get updated by anyone by calling the **kelpOracle.updateRSETHPrice()** function, an attacker or whale address that holds a lot of **rsETH** token can manipulate the amount of **bldETH** that can be minted to users by sending/burning a huge amount of **rsETH** or ETH to the kelp protocol and calls the **updateRSETHPrice()** function to change the **rsETH** price, this can be profitable for the attacker in case of deposit 1 rsETH when the **rsETH** price is 1.01 and then deposit a huge amount of ETH to increase the **rsETH** price to 1.1 rsETH, this will mint 1 bldETH to the attacker with depositing 1.01 rsETH and getting in return 1.1 rsETH back (the deposited eth will gain profit too because it is sent to the kelp protocol). In case, the attacker is not a whale address who holds eth, the **rsETH** price will increase or decrease when calling the **updateRSETHPrice()** function to gain profit for himself or grief other users.

This is possible because both **depositLRT/withdrawLRT** functions depend on the **ethToKelpEth()** function.

Location of Affected Code

File: src/Aggregator.sol

```
function depositLRT(address asset, uint256 amount, uint256
   minBldETHAmount)
    external
    nonReentrant
    isAllowed(asset)
    notPaused
{

function withdrawLRT(address assetToWithdraw, uint256 bldShares)
    external
    payable
    nonReentrant
    isAllowed(assetToWithdraw)
    notPaused
{
```

File: src/LRTProtocolsManager.sol#L162

```
function ethToKelpEth(
   uint256 _ethAmount
) external view onlyAggregator returns (uint256) {
   uint256 kelEthPrice = kelpOracle.rsETHPrice();
   return (_ethAmount * SCALE_FACTOR) / kelEthPrice;
}
```

Recommendation

Consider calling the ' **updateRSETHPrice()** function inside the **ethToKelpEth()** function.

Team Response

Acknowledged and fixed as proposed.

[L-04] Centralization Risk

Severity

Low Risk

Description

In the current implementation setup, the owner is meant to be a single Externally Owned Account (EOA) which could potentially introduce security risks regarding single points of failure or mismanagement.

Specifically, the **withdrawLRT()** function is decorated with the **notPaused** modifier. This would allow the owner to potentially "pause" the protocol and prevent users from withdrawing their funds. This could become an actual problem if the owner is a single EOA.

Location of Affected Code

File: src/Aggregator.sol#L272

```
    function withdrawLRT(address assetToWithdraw, uint256 bldShares)
        external
        payable
        nonReentrant
        isAllowed(assetToWithdraw)
        notPaused
    {
```

Recommendation

Consider removing the pausing functionality from the **withdrawLRT()** function and using a multi-sig or governance as the protocol owner, which would require more than a single entity to perform such an action. ## Team Response

Acknowledged, will be considered.

[L-05] Unnecessary Checks

Severity

Low Risk

Description

1. In the **depositETH()** function there is a redundant zero value check, as the amount is ensured to be greater than **minDepositETH()** which returns **1e16**.
2. In the **convertBldEthToLRTShares()** function **bldShares == 0** check is unnecessary since there is already such a zero-value check in the **convertEthToLRTshares()** function.

Location of Affected Code

File: src/Aggregator.sol

```
function depositETH() external payable nonReentrant notPaused {
     if (amount == 0) {
          revert Aggregator__AmountMustBeGreaterThanZero();
     }

// code
]

function convertBldEthToLRTShares(address asset, uint256 bldShares)
   public view isAllowed(asset) returns (uint256) {
     if (bldShares == 0) {
          return 0;
     }

// code
}
```

Recommendation

Consider removing the unnecessary checks.

Team Response

Acknowledged and fixed as proposed.

, our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Thank you!