fuzzland

WELabs Audit Report

Signature: 97445997838aa0ebe0d34e786673039b2c3d61ca4f180ada323e5f461aa19f50

WELabs Audit Report

Scope

Project Name	WELabs
Repo	Launchpad
Commit	`5ad8da58ceb77f04a2c12a7f77c54dc218124785`
Fix Commit	`2c41efb00fb4c578af88d9f852bd32e8c60c2ea0`
Language	Solidity
Scope	contracts/Launchpad.sol,contracts/BondingCurve.sol

Disclaimer

The audit does not ensure that it has identified every security issue in the smart contracts, and it should not be seen as a confirmation that there are no more vulnerabilities. The audit is not exhaustive, and we recommend further independent audits and setting up a public bug bounty program for enhanced security verification of the smart contracts. Additionally, this report should not be interpreted as personal financial advice or recommendations.

Auditing Process

- Static Analysis: We perform static analysis using our internal tools and Slither to identify potential vulnerabilities and coding issues.
- Fuzz Testing: We execute fuzz testing with our internal fuzzers to uncover potential bugs and logic flaws.
- Invariant Development: We convert the project into Foundry project and develop Foundry invariant tests for the project based on the code semantics and documentations.
- Invariant Testing: We run multiple fuzz testing tools, including Foundry and ItyFuzz, to identify violations of invariants we developed.
- Formal Verification: We develop individual tests for critical functions and leverage Halmos to prove the functions in question are not vulnerable.
- Manual Code Review: Our engineers manually review code to identify potential vulnerabilities not captured by previous methods.

Vulnerability Severity

We divide severity into three distinct levels: high, medium, low. This classification helps prioritize the issues identified during the audit based on their potential impact and urgency.

- **High Severity Issues** represent critical vulnerabilities or flaws that pose a significant risk to the system's security, functionality, or performance. These issues can lead to severe consequences such as fund loss, or major service disruptions if not addressed immediately. High severity issues typically require urgent attention and prompt remediation to mitigate potential damage and ensure the system's integrity and reliability.
- **Medium Severity Issues** are significant but not critical vulnerabilities or flaws that can impact the system's security, functionality, or performance. These issues might not pose an immediate threat but have the potential to cause considerable harm if left unaddressed over time. Addressing medium severity issues is important to maintain the overall health and efficiency of the system, though they do not require the same level of urgency as high severity issues.
- **Low Severity Issues** are minor vulnerabilities or flaws that have a limited impact on the system's security, functionality, or performance. These issues generally do not pose a significant risk and can be addressed in the regular maintenance cycle. While low severity issues are not critical, resolving them can help improve the system's overall quality and user experience by preventing the accumulation of minor problems over time.

Below is a summary of the vulnerabilities with their current status, highlighting the number of issues identified in each severity category and their resolution progress.

	Number	**Resolved**
High Severity Issues	0	0
Medium Severity Issues	1	1
Low Severity Issues	0	0
Info Severity Issues	2	2

Findings

[Med] Excess ETH not refunded in `launchpad()`

The `launchpad()` function requires users to send at least `createFees` amount of ETH, but if users send more than the required `amount`, the excess ETH is not refunded and remains locked in the contract.

```
// contracts/Launchpad.sol
function launchpad(
    string calldata name,
    string calldata symbol,
    string calldata url
) external payable returns (address token, address curve) {
    require(msg.value >= createFees, "Must send 0.001 ether");
    (token, curve) = _launchpad(name, symbol, url);
    TransferHelper.safeTransferETH(feeTo, createFees);
}
```

Recommendation:

Calculate and refund the excess ETH to the sender:

```
function launchpad(
    string calldata name,
    string calldata symbol,
    string calldata url
) external payable returns (address token, address curve) {
    require(msg.value >= createFees, "Must send 0.001 ether");
    (token, curve) = _launchpad(name, symbol, url);
    TransferHelper.safeTransferETH(feeTo, createFees);
    // Refund excess ETH
    uint256 excess = msg.value - createFees;
    if (excess > 0) {
        TransferHelper.safeTransferETH(msg.sender, excess);
    }
}
```

Status: fixed

[Info] Typo in error message

In the `initialize()` function of the `BondingCurve` contract, there is a typo in the error message of the require statement. "`adderss`" is misspelled and should be "`address`".

```
// contracts/BondingCurve.sol
require(_token != address(0), 'zero adderss');
```

Recommendation:

Correct the spelling in the error message:

```
require(_token != address(0), 'zero address');
```

Status: fixed

[Info] Duplicate check

The `Launchpad` contract's check `_beneficiaryAddresses.length == _totalAmounts.length` is duplicated, as it already exists in the `LinearVesting` contract's constructor that it calls.

```
// contracts/Launchpad.sol
function launchpadInitialBuy(
      string calldata name,
      string calldata symbol,
      string calldata url,
      address[] calldata _beneficiaryAddresses,
      uint256[] calldata _totalAmounts,
      uint256 _duration,
      uint256 amount
  ) external payable returns (address token, address curve, address vesting)
{
      //......
      require(
          _beneficiaryAddresses.length == _totalAmounts.length,
          "BondingCurve: Mismatched lengths"
      );
      //......
  }
```

```
// contracts/LinearVesting.sol
constructor(
      address _erc20,
      address[] memory _beneficiaryAddresses,
      uint256[] memory _totalAmounts,
      uint256 _duration
  ) {
      require(
          _beneficiaryAddresses.length == _totalAmounts.length,
          "Mismatched lengths"
      );
      //......
```

Recommendation:

```
diff --git a/contracts/Launchpad.sol b/contracts/Launchpad.sol
index 1cb3676..f6193d1 100644
--- a/contracts/Launchpad.sol
+++ b/contracts/Launchpad.sol
@@ -97,10 +97,6 @@ contract Launchpad is Ownable2Step {
         uint256 buyValue = msg.value - createFees;
         require(buyValue != 0, "Must send some ether");
         require(_beneficiaryAddresses.length <= MAX_INITIAL_BUY, "reached th
e max");
-        require(
-            _beneficiaryAddresses.length == _totalAmounts.length,
-            "BondingCurve: Mismatched lengths"
-        );
 
         uint256 totalAmount = 0;
         for (uint256 i = 0; i < _beneficiaryAddresses.length; i++) {
```

Status: fixed