shieldify

Goose Run V1

SECURITY REVIEW

Date: 29 October 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 Goose Run V1

Goose Run V1 is a series of smart contracts that facilitate fair tokens launched through the Maverick V2 AMM. Users can select a launch liquidity distribution that suits their token launch price schedule. Goose Run V1 pools also support lending launch tokens.

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 7 days with a total of 224 hours dedicated to the audit by four researchers from the Shieldify team. This is the second review of the Goose Run V1 codebase conducted by the same research team.

The audit report helped identify a short lookback period in `createTokenManager()` function and several other lower-impact issues. The raffle-vault system has been rigorously tested and verified to ensure that prize chances are truly random, probabilities are accurately calculated, and fairness is maintained at all times.

The protocol's team has done a great job with their test suite and provided support and responses to all of the questions that the Shieldify researchers had.

5.1 Protocol Summary

Project Name	**Goose Run V1**
Repository	fairlaunch-contracts
Type of Project	DeFi, Lending
Audit Timeline	7 days
Review Commit Hash	cb3b10370a029e39694dde45c51027bfb5b1cf75
Fixes Review Commit Hash	0992b679413caf369a7c3d51b004267f1ba4d25a

5.2 Scope

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

File	nSLOC
src/LaunchFactory.sol	142
src/LaunchToken.sol	28
src/Swapper.sol	322
src/TokenManager.sol	178
src/TokenManagerDeployer.sol	8
src/TokenManagerLens.sol	223
src/DistributionUsd.sol	37
src/DistributionEth.sol	90
src/FeeVault.sol	32
src/VotingDistributor.sol	247
src/RaffleVault.sol	165
src/RaffleVaultDeployer.sol	9

src/HistoricalBalanceNonTransferableERC20.sol	99
src/libraries/PoolInspection.sol	86
src/libraries/Distribution.sol	86
src/libraries/LaunchConstants.sol	9
src/libraries/ArrayOperations.sol	33
Total	**1794**

6. Findings Summary

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

- **Medium** issues: **1**
- **Low** issues: **2**

ID	Title	Severity	Status
[M-01]	Short Lookback Period in `createTokenManager()` Function	Medium	Fixed
[L-01]	Discrepancy in `MIN_LOOKBACK` Constant	Low	Fixed
[L-02]	A Malicious User Can Steal Another User's Excess ETH	Low	Fixed

7. Findings

[M-01] Short Lookback Period in `createTokenManager()` Function

Severity

Medium Risk

Description

The `createTokenManager` function specifies a **30-seconds lookback period** for calculating the Time-Weighted Average Price (TWAP) when creating a new pool through the `createPermissioned` function.

```
IMaverickV2Pool pool = factory.createPermissioned(
    0,
    0,
    TICK_SPACING,
    uint32(30 seconds), seconds
    tokenIsA ? _tempLaunchData.token : quoteToken,
    tokenIsA ? quoteToken : _tempLaunchData.token,
    tokenIsA ? lens.lastTick() : -lens.lastTick(),
    1, // 1 => only mode static in pool
```

```
        address(swapper),
        false,
        true
    );
```

The lookback period is critical in determining TWAP, which smooths price volatility and helps ensure price accuracy over time. A 30-second lookback period is excessively short and can lead to unstable TWAP values.

Impact

A short TWAP lookback window heightens exposure to rapid price changes, enabling potential price manipulation.

Location of Affected Code

File: src/LaunchFactory.sol#L143

Recommendation

Consider increasing the lookback period.

Team Response

Fixed, the lookback period was set to 2 minutes.

[L-01] Discrepancy in `MIN_LOOKBACK` Constant

Severity

Low Risk

Description

The `MIN_LOOKBACK` constant is defined as **1 second** in the contract code:

```
uint64 constant MIN_LOOKBACK = 1 seconds;
```

While the official Maverick documentation specifies a minimum lookback of **30 minutes**: `Link`

Although this constant is currently unused, using such a short lookback period could introduce critical vulnerabilities in future implementations if it were applied to TWAP calculations.

Impact

If `MIN_LOOKBACK` is applied in the future, a 1-second period could cause unreliable TWAP calculations due to high sensitivity to brief price fluctuations.

Location of Affected Code

File: src/v-2-common/libraries/Constants.sol#L11

Recommendation

Update `MIN_LOOKBACK` to match the value specified in Maverick documentation to avoid future issues.

Team Response

Fixed.

[L-02] A Malicious User Can Steal Another User's Excess ETH

Severity

Low Risk

Description

The `refundEth()` function allows anyone to claim the entire ETH balance of the contract, which could include excess ETH paid by users during interactions with functions like `openLootBoxes()`.

```
function refundEth() public payable {
    if (address(this).balance > 0) Address.sendValue(payable(msg.sender),
    address(this).balance);
}
```

The function lacks access control, allowing any caller to execute it. This creates a front-running risk: if a legitimate user attempts to retrieve their excess ETH, a malicious actor could monitor the mempool and submit a transaction with higher gas to claim the ETH balance first.

Impact

An unauthorized user can drain the ETH balance of the contract, leading to the loss of excess ETH that users may have paid.

Location of Affected Code

File: src/RaffleVault.sol#L163

Recommendation

Ensure any excess ETH is refunded directly within the `openLootBoxes()` function. This approach helps maintain control over refund transactions and avoids ETH balance thefts.

Team Response

Fixed.

shieldify

Thank you!