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

shieldify

Goose Run

SECURITY REVIEW

Date: 20 Sep 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

Goose Run 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. The Goose Run 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 8 days with a total of 256 hours dedicated to the audit by four researchers from the Shieldify team.

The code is exceptionally well-implemented, showcasing the developers' high level of expertise. The second part of the audit focused on the borrow and repay functionality, as well as the protocol's solvency calculations. The audit report helped identify a missing refund mechanism, token buy/sell deadlines, and some overlooked validation checks.

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**
Repository	fairlaunch-contracts
Type of Project	DeFi, Lending
Audit Timeline	8 days
Review Commit Hash	ab72c4feb39bf1a25804dd573785ec43c6b7897c
Fixes Review Commit Hash	1677d0eb2aed4b9d02280f1584f796c8deda1280

5.2 Scope

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

File	nSLOC
src/LaunchFactory.sol	117
src/LaunchToken.sol	22
src/Swapper.sol	109
src/TokenManager.sol	162
src/TokenManagerDeployer.sol	8
src/TokenManagerLens.sol	250
src/libraries/Distribution.sol	65
src/libraries/ArrayOperations.sol	33
src/libraries/LaunchConstants.sol	7
Total	**773**

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]	The Excess ETH in `buyToken()` Is Not Returned to the User	Medium	Fixed
[L-01]	Missing Deadline Check in `buyToken()` and `sellToken()`	Low	Acknowledged
[L-02]	Missing Validation for `ethLaunchFee` in `setParameters()`	Low	Acknowledged

7. Findings

[M-01] The Excess ETH in `buyToken()` Is Not Returned to the User

Severity

Medium Risk

Description

The `buyToken()` function in the `Swapper` contract allows users to buy tokens using ETH:

```
function buyToken(
    address recipient,
    IERC20 token,
    uint256 amountEthIn,
    uint256 amountOutMinimum,
    address referer
) public payable returns (uint256 amountOut, IMaverickV2Pool pool) {
// eth in
    pool = _getPool(token);
    bool tokenAIn = pool.tokenA() != token;

// extract fee; raw eth at this point
    uint256 netEthForUser = _extractEthFee(amountEthIn, referer);

// pay for token; function takes care of wrapping input eth value
    amountOut = _exactInputSingle(recipient, pool, tokenAIn,
        netEthForUser);

// check slippage
    if (amountOut < amountOutMinimum) revert TooLittleReceived(
        amountOutMinimum, amountOut);
}
```

However, the function does not handle the scenario where the user sends more ETH (`msg.value`) than required for the transaction. If the user sends excess ETH, it is not returned, leading to overpayment. This could result in financial losses for users if they mistakenly send more ETH than needed for the swap.

While the function deducts fees and performs the swap based on the provided `amountEthIn` , any excess ETH sent by the user remains in the contract.

Impact

Users who send more ETH than required will not receive a refund for the excess amount.

Location of Affected Code

File: src/Swapper.sol#buyToken()

Recommendation

After performing the swap and deducting the required ETH (`amountEthIn`), return any excess ETH back to the user.

Team Response

Fixed

[L-01] Missing Deadline Check in `buyToken()` and `sellToken()`

Severity

Low Risk

Description

The `buyToken()` and `sellToken()` functions in the `Swapper` contract to swap ETH for a specific ERC-20 token and vice versa.

```
function buyToken(
    address recipient,
    IERC20 token,
    uint256 amountEthIn,
    uint256 amountOutMinimum,
    address referer
) public payable returns (uint256 amountOut, IMaverickV2Pool pool) {
// eth in
    pool = _getPool(token);
    bool tokenAIn = pool.tokenA() != token;

// extract fee; raw eth at this point
    uint256 netEthForUser = _extractEthFee(amountEthIn, referer);

// pay for token; function takes care of wrapping input eth value
    amountOut = _exactInputSingle(recipient, pool, tokenAIn,
       netEthForUser);

// check slippage
    if (amountOut < amountOutMinimum) revert TooLittleReceived(
       amountOutMinimum, amountOut);
}
```

```
function sellToken(
    address recipient,
    IERC20 token,
    uint256 amountTokenIn,
    uint256 amountOutMinimum,
    address referer
) public returns (uint256 amountOut, IMaverickV2Pool pool) {
// eth out
    pool = _getPool(token);
    bool tokenAIn = pool.tokenA() == token;

// swap token for weth
    uint256 amountOutBeforeFee = _exactInputSingle(address(this), pool,
       tokenAIn, amountTokenIn);

// unwrap weth to eth
    weth.withdraw(amountOutBeforeFee);

// take fee
    amountOut = _extractEthFee(amountOutBeforeFee, referer);

// check slippage
    if (amountOut < amountOutMinimum) revert TooLittleReceived(
       amountOutMinimum, amountOut);

    Address.sendValue(payable(recipient), amountOut);
}
```

However, these functions do not include a **deadline check** to ensure that the swap is executed within a specific time frame.

This omission could expose the contract to certain risks, such as stale transactions that are executed later than intended.

Although the slippage check helps mitigate the impact of price changes, swaps executed without a time limit could still be exposed to high market volatility over a longer period. If market conditions change drastically, the user's transaction could be negatively impacted.

Impact

Users have no way to enforce a time limit on their transactions, meaning their swap could be executed at an unexpected time, which may no longer be beneficial.

Location of Affected Code

File: src/Swapper.sol#buyToken()

File: src/Swapper.sol#sellToken()

Recommendation

We recommend adding a **deadline parameter** to both the `buyToken()` and `sellToken()` functions to ensure that transactions are executed within a specified time frame. This will give users additional protection against market volatility and stale transactions.

Team Response

Acknowledged, the user is still able to set slippage limits and/or manually cancel their transaction anytime they like.

[L-02] Missing Validation for `ethLaunchFee` in `setParameters()` Function

Severity

Low Risk

Description

The `setParameters()` function in the `LaunchFactory` contract allows the owner to update important parameters like the protocol fee proportion (`_protocolFeeProportionD18`) and the ETH launch fee (`ethLaunchFee`).

```
function setParameters(
    address _protocolFeeCollector,
    uint128 _ethLaunchFee,
    uint128 _protocolFeeProportionD18
) public onlyOwner {
    if (_protocolFeeProportionD18 > 1e18) revert InvalidFeeParameter();
    protocolFeeCollector = _protocolFeeCollector;
    ethLaunchFee = _ethLaunchFee;
    protocolFeeProportionD18 = _protocolFeeProportionD18;
}
```

While there is a check that ensures the `_protocolFeeProportionD18` remains within acceptable limits (i.e., not greater than `1e18` or 100%), there is no such check for `ethLaunchFee`, which could potentially be set to an unintended or unreasonable value.

Impact

Changing the launch fee to an inappropriate value could affect the protocol's revenue model or user experience, especially if the fee is set too high or too low by mistake.

Location of Affected Code

File: src/LaunchFactory.sol#setParameters()

Recommendation

To prevent `ethLaunchFee` from being set to an inappropriate value, we recommend adding a validation check to ensure that it remains within reasonable bounds.

Team Response

Acknowledged.

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

shieldify

Thank you!