HyperLaunch

Security Audit Report

February 22, 2025

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

BugBlow utilized a widely adopted approach to performing a security audit. Below is a breakout of how our team was able to identify and exploit vulnerabilities.

1.2.1 Code and architecture review:

- Review the source code.
- Read the project's documentation.
- Review the project’s architecture.

Stage goals

- Build a good understanding of how the application works.

1.2.2 Check code against vulnerability checklist:

- Understand the project’s critical assets, resources, and security requirements.
- Manual check for vulnerabilities based on the auditor's checklist.
- Run static analyzers.

Stage goals

- Identify and eliminate typical vulnerabilities (gas limit, replay, flash-loans attack, etc.)

1.2.3 Threat & Attack Simulation:

- Analyze the project's critical assets, resources and security requirements.
- Exploit the found weaknesses in a safe local environment.
- Document the performed work.

Stage goals

- Identify vulnerabilities that are not listed in static analyzers that would likely be exploited by hackers.
- Develop Proof of Concept (PoC).

1.2.4 Report found vulnerabilities:

- Discuss the found issues with developers
- Show remediations.
- Write an initial audit report.

Stage goals

- Verify that all found issues are relevant.

1.2.5 Fix bugs & re-audit code:

- Help developers fix bugs.
- Re-audit the updated code again.

Stage goals

- Double-check that the found vulnerabilities or bugs were fixed and were fixed correctly.

1.2.4. Deliver final report:

- The Client deploys the re-audited version of the code
- The final report is issued for the Client.

Stage goals

- Provide the Client with the final report that serves as security proof.

Severity classification

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Vulnerabilities leading to assets theft, fund access locking, or any other loss of funds.
High	Vulnerabilities that can trigger contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Vulnerabilities that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Vulnerabilities that do not have a significant immediate impact and could be easily fixed.

1.3 Project Overview

The Fair Launch project implements a two-phase token distribution mechanism, starting with an internal liquidity pool that sells and buys tokens directly from the contract. Once a specific market cap threshold is reached, the contract automatically locks liquidity on a DEX via an NFT position, preventing immediate rug pulls. Ownership and deployment details are stored on a dedicated Storage contract, while the Event Tracker logs all buys, sells, and listing events.

1.4 Project Dashboard

Project Summary

Title	Description
Client name	HyperLaunch
Project name	Fair Launch
Timeline	14.02.2025 - 22.02.2025
Number of Auditors	3

Project Last Log

Date	Data Hash	Note
10.02.2025	ef8a90bff71e6c524a06de4228e3db3dcfbe1e884dcf2669fcde5b2ce7a1433e	Sha256 hash of contracts_fair_launch.zip

Project Scope

The audit covered the changes introduced in Fair Launch contracts.

File name
contracts/Storage.sol
contracts/SimpleERC20.sol
contracts/FairLaunchMemesAiAgentsPool.sol
contracts/FairLaunchMemesAiAgentsDeployer.sol
contracts/EventTracker.sol
contracts/Errors.sol
contracts/Clones.sol
hardhat.config.js
contracts/LiquidityLockerV3.sol
contracts/ConnectorLock.sol
contracts/interfaces/ILiquidityLockerV3.sol
contracts/interfaces/IMorphexV3SwapCallback.sol
contracts/interfaces/INonfungiblePositionManager.sol
contracts/interfaces/ISwapRouter.sol
Contracts/mock/FairLaunchMemesAiAgentsPool.sol

The Client is aware of the known “dangerous tokens” issue. The Client is planning to filter all tokens before adding them to the protocol.

1.5 Summary of findings

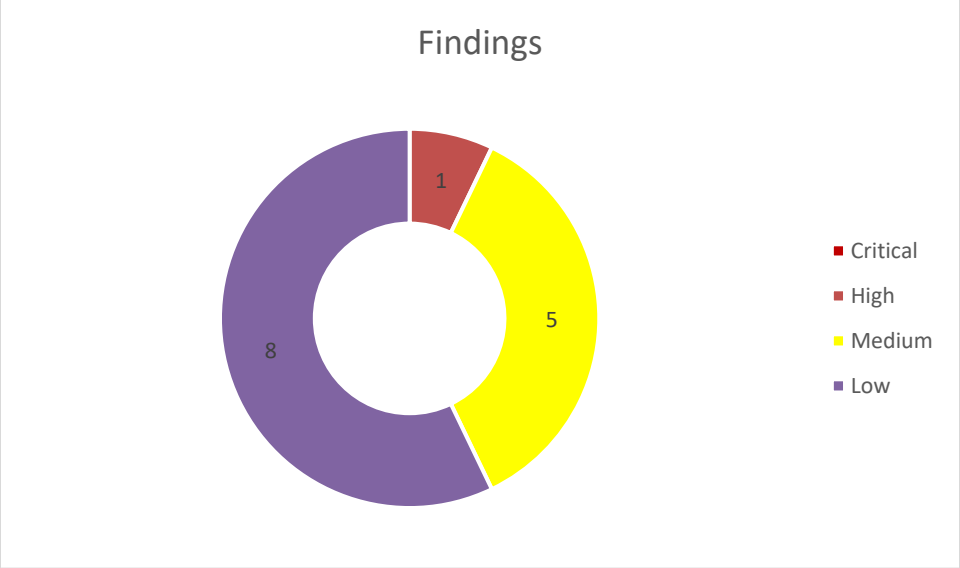

Figure 1. Findings chart

Severity	# of Findings
CRITICAL	0
HIGH	1
MEDIUM	5
LOW	8

2. FINDINGS REPORT

2.1 Critical

Not found

2.2 High

2.2.1 A token can be listed on a DEX with almost 0 liquidity

Status: Fixed

Description

Upon reaching the market threshold, a token is supposed to be listed on a DEX. However, the token's "initiate dex" flag is set before the liquidity is transferred to the DEX. It is possible to set the flag, while skipping the liquidity transfer (line 5).

A certain condition must be met - `nativePer` must be more than 0. If the owner of the token sets token.pool.nativePer to 0 (using changeNativePer) just before reaching the market cap, the liquidity will never be transfered to DEX, but all trades will stop and the `initiateDex` flag will be set (line 2). This will break the protocol.

```
1. if (currentMarketCap >= listThresholdCap) {
2.     token.pool.tradeActive = false;
3.     IToken(fToken).initiateDex();
4.     token.pool.reserveETH -= token.pool.initialReserveEth;
5.     if (token.pool.nativePer > 0) {
6.         _addLiquidityETH(
7.             fToken,
8.             (IERC20(fToken).balanceOf(address(this)) *
9.             token.pool.nativePer) / HUNDRED,
10.            (token.pool.reserveETH * token.pool.nativePer) / HUNDRED
11.        );
```

Remediation

Either revert the transaction if token.pool.nativePer was less or equal to zero, or transfer the liquidity first, and only then set the flag and lock the trades. The latter one is also known as Checks and Effects pattern.

2.3 Medium

2.3.1 DoS: exceeded gas limit

Status: Fixed

Description

The isUserFunToken function uses an array of an undetermined length to run over. Too many tokens per user will increase the gas consumption and will eventually make the function fail.

```
    function _isUserFunToken(address fToken) internal view returns (bool) {
        for (uint i = 0; i < userTokens[msg.sender].length; ) {
            if (fToken == userTokens[msg.sender][i]) {
                return true;
            }
            unchecked {
                i++;
            }
        }
        return false;
    }
```

Remediation

To check whether the token belongs to user, there a cheap way to do it: store the user tokens in a map as well. And the function would like this:

```
function _isUserFunToken(address fToken) internal view returns (bool) {
    bool isUserFunToken = isUserTokens[msg.sender][fToken];
    return isUserFunToken;
}
```

2.3.2 Affiliate address can be the sender themselves

Status: Acknowledged

Description

There are no checks on whether the affiliate address is any different from the sender. The sender can keep the affiliate fee.

Remediation

The solution is not trivial. If there is no opportunity to validate the affiliate address (whether it doesn't belong to the sender), then what is the point of allowing to receive extra fee?

2.3.3 No validation whether the transfer was successful

Status: Acknowledged

Description

In the sellTokens function, there is no check whether the tokens transfer was successful, although the transferFrom function returns a Boolean result. At the moment, the function reverts is something went wrong, but if the behavior changes in the future, this may break the protocol as it will lead to inconsistency in storage.

Remediation

The solution is not trivial. If there is no opportunity to validate the affiliate address (whether it doesn't actually belong to the sender), then what is the point of allowing to receive extra fee?

2.3.4 Any pool owner can rewrite the fun contract data at any point in time

Status: Fixed

Description

The updateFunData (calls updateData) can be called by the pool owner at any point in time, rewriting the contract's details. The data may be inconsistent with the actual data in the pool (for example totalSupply). This is a scam risk for the pool users, if the Storage data is used anywhere.

Remediation

We do not see the need to manually rewrite the pool's data in Storage once the pool has been deployed.

2.3.5 EVM Compatibility

Status: Fixed

Description

Your hardhat configuration does not specify the EVM's version. By default, hardhat sets the Paris EVM version. Your Lumia chain supports a different version (Shanghai), the OP codes and precompiles have different behavior in different EVM versions.

Remediation

Specify the EVM version in the hardhat configuration. However, keep in mind, that shanghai supports Solidity 0.8.20, so you make your contracts' versions "^0.8.19" to that the older versions could compile your contracts.

```
module.exports = {
  solidity: {
    version: "0.8.20",
    settings: {
      viaIR: true,
      evmVersion: "shanghai",
      optimizer: {
        enabled: true,
        runs: 200,
      },
    },
  },
};
```

An example of differences in different EVM versions can be found here [2].

2.4 Low

2.4.1 Bad naming

Status: Fixed

Description

Everywhere the pool storage is used, its variable is named `token`. This is very confusing as it is not what it is for. The actual token is *fToken*.

```
FunTokenPool storage token = tokenPools[fToken];
```

Remediation

Rename all similar lines to

```
FunTokenPool storage pool = tokenPools[fToken];
```

2.4.2 Wrong error

Status: Fixed

Description

The validateTransfer function allows trades if the token is already listed on a DEX, or if the transaction came from the agent pool contract (before the market cap is reached). However, the error message `not dex listed` indicates that the trades are only allowed after listing on DEX, which is not correct.

```
require(_validateTransfer(msg.sender), "not dex listed");
```

Also, we advise you to double-check this is the logic, you intended to have. This is also related to 2.2.1 (High). Once, the `dexInitiated` variable is set to true, the users can trade the tokens directly even though the DEX has not received any liquidity.

```
function _validateTransfer(address _from) internal view returns (bool) {
    if (dexInitiated) {
        return true;
    }
    if (_from == initialFrom || _from == deployerFrom) {
        return true;
    }
    return false;
}
```

Remediation

The error message should be "unauthorized sender or not listed on DEX yet".

2.4.3 Compilation error

Status: Fixed

Description

There is a missing interface method in *INonfungiblePositionManager* to make the contracts compile successfully.

Remediation

Add this method to the interface.

```
function safeTransferFrom(
    address from,
    address to,
    uint256 tokenID
) external;
```

2.4.4 Duplicate function

Status: Fixed

Description

The function getBaseAddr is a duplicate of getBaseToken. The do absolutely the same.

Remediation

Consider removing one of them.

2.4.4 No allowance check

Status: Acknowledged

Description

The transferFrom functions relies on the underflow revert behavior (since Solidity 0.8.0) if the allowance is less than the function is sending. But this

```
  function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public override returns (bool) {
        require(_validateTransfer(msg.sender), "not dex listed");
        _transfer(sender, recipient, amount);
        _approve(sender, msg.sender, _allowances[sender][msg.sender] - amount);
        return true;
    }
```

Remediation

Return false if the allowance is less than the sent amount.

2.4.5 Voter is not initialized

Status: Fixed

Description

The voter address is not initialized in the constructor of the Ownable contract, although it is later expected, the sender must be equal to voter. The modifier is not used anywhere, so this is not critical. However, if you are planning to use the modifier somewhere, this will lead to vulnerabilities.

```
constructor() {
    owner = msg.sender;
}
```

```
modifier onlyVoter() {
    require(msg.sender == voter);
    _;
}
```

Remediation

Initialize the voter or remove the variable and the modifier.

2.4.6 Unused fields

Status: Acknowledged

Description

Some of the fields (marked red) in INonfungiblePositionManager are unused. Removing them will save some gas used to allocate memory.

```
    struct Position {
        uint96 nonce; // unused
        address operator; // unused
        address token0;
        address token1;
        uint24 fee;
        int24 tickLower;
        int24 tickUpper;
        uint128 liquidity;
        uint256 feeGrowthInside0LastX128; // unused
        uint256 feeGrowthInside1LastX128; // unused
        uint128 tokensOwed0; // unused
        uint128 tokensOwed1; // unused
    }
```

Remediation

Remove the unused fields.

2.4.7 Unlimited fee name

Status: Fixed

Description

There's no hardcoded maximum size for a string in Solidity. The actual limit is determined by the block gas limit. Should your logic change (i.e. someone else besides the owner can add fee names), this will cause problems.

```
function addOrEditFee(string memory _name, uint256 _lpFee, uint256 _collectFee) public onlyOwner {
    bytes32 nameHash = keccak256(abi.encodePacked(_name));

    FeeStruct memory newFee = FeeStruct(_name, _lpFee, _collectFee);
    _fees[nameHash] = newFee;
```

```
    if (!_feeLookUp.contains(nameHash)) {
        _feeLookUp.add(nameHash);
        emit OnAddFee(nameHash, newFee.name, newFee.lpFee, newFee.collectFee);
    } else {
        emit OnEditFee(nameHash, newFee.name, newFee.lpFee, newFee.collectFee);
    }
}
```

Remediation

Limit the fee name to length 256. Also, consider moving "DEFAULT" name to a constant (i.e. defaultFeeName = 'DEFAULT'.

2.4.8 Bad naming

Status: fixed

Description

In the collect function, the balance0 and balance1 variables represent balances before and after (each pair) collecting process.

```
uint256 balance0 = IERC20(_token0).balanceOf(address(this));
uint256 balance1 = IERC20(_token1).balanceOf(address(this));

userLock.nftPositionManager.collect(INonfungiblePositionManager.CollectParams(userLock.nftId,
 address(this), _amount0Max, _amount1Max));

balance0 = IERC20(_token0).balanceOf(address(this)) - balance0;
balance1 = IERC20(_token1).balanceOf(address(this)) - balance1;
```

The current names are confusing (and wrong), so they may lead to bugs in the future.

Remediation

Consider renaming them to (balance0before, balance0afterm balance1before, balance1after).

CONSCLUSION

This security audit found no critical vulnerabilities in the Fair Launch set of contracts, but did identify multiple areas that require attention. One high-severity issue (Sections 2.2.1) can lead to a token being listed on a DEX with zero liquidity. Several medium-severity issues (Sections 2.3.1

through 2.3.6) were also discovered, encompassing potential DoS from large arrays, and inconsistent storage or fee logic. Lastly, multiple low-severity findings (Sections 2.4.1 through 2.4.5) relate to code style, missing interface definitions, uninitialized variables, and minor error-message inaccuracies.

Addressing how liquidity is transferred when initiateDex() is called (Section 2.2.1) is crucial to ensure the token cannot become tradable with insufficient liquidity.

Addressing these issues will significantly enhance the protocol's security and reliability.

REFERENCES

[1] https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

[2] https://docs.linea.build/get-started/build/ethereum-differences