HoneyFun Tokenomics Security Review

February 19, 2025 - February 20, 2025

Date: February 23, 2025

Conducted by: **KeySecurity**
gkrastenov, Lead Security Researcher

Table of Contents

1 About KeySecurity

KeySecurity is a new, innovative Web3 security company that hires top-talented security researchers for your project. We have conducted over 35+ security reviews for various projects, collectively holding over $300,000,000 in TVL. For security audit inquiries, you can reach out to us on Twitter/X or Telegram `@gkrastenov` or check our previous work `here`.

2 About HoneyFun

`Honeyfun AI` is pioneering the co-ownership framework for AI agents specifically tailored for the Berachain ecosystem, focusing on defi, gaming and entertainment. We envision AI agents as pivotal revenue-generating entities in the future, as we believe the era of Utility AI Agents is just beginning, and in the coming years, their untapped potential across every field will be revealed.

3 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

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** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

4.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

4.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

5 Executive summary

Overview

Project Name	HoneyFun Tokenomics
Repository	https://github.com/honey-fun/honey-fun-tokenomics
Commit hash	997ae9252631286557d564b438095800dfcad695
Review Commit hash	f9f8940e7400ae7a44fafdb2d0a60ff1f0999e30
Documentation	https://docs.honey.fun/
Methods	Manual review

Scope

AiBera.sol
Airdrop.sol
HoneyFunAccessControl.sol
Launchpool.sol
Roles.sol
StakingWithTiers.sol
ToInitialize.sol
TokenVesting.sol
stToken.sol

Timeline

February 19, 2025	Audit kick-off
February 20, 2025	Preliminary report
February 21, 2025	Mitigation review

Issues Found

Severity	Count
High	2
Medium	1
Low	0
Information	3
Total	**6**

6 Findings

6.1 High

6.1.1 A malicious user can manipulate the duration for every user

Severity: *High*

Context: StakingWithTiers.sol#L34

Description: For reward calculation, three variables are used: the staked amount, APR, and duration, where duration represents the elapsed time. The elapsed time is calculated as `block.timestamp - _lastInteractionTimestamp[user]`.

The `lastInteractionTimestamp` is updated every time the `updateReward` function is called through (`_stake`, `restake`, `unstake`, `unstakeInstant`, and `getReward`). The first time a user interacts with the system is when they make a stake, setting `_lastInteractionTimestamp` to `block.timestamp`. This way, every time a user stakes, their last interaction timestamp is updated.

This introduces a vulnerability: a malicious user can make a small stake using the `stakeAs` function, where `wallet != msg.sender`. By depositing a small amount, they can update the last interaction timestamp of any user, effectively resetting their reward calculation to zero.

```
    function stakeAs(address wallet, uint256 amount) public {
         // wallet != msg.sender
        _stake(wallet, amount);
    }
```

Attack Vector (The Unstaking Time is 30 Days):

1. Alice stakes her tokens, and `_lastInteractionTimestamp[Alice]` is set to `block.timestamp`.
2. 30 days are almost over, so Alice is preparing to claim her rewards.
3. A malicious user makes a small stake to Alice's wallet (`wallet == address(Alice)`), causing `_lastInteractionTimestamp[Alice]` to be updated to `block.timestamp` again.

```
    modifier updateReward(address wallet) {
        uint256 rewards = calculateRewards(msg.sender);
        _rewardBalance[msg.sender] += rewards;
        // wallet == address(Alice)
        _lastInteractionTimestamp[wallet] = block.timestamp;
        _;
    }
```

4. Alice claims her reward, but the duration is now calculated as:`block.timestamp - _lastInteractionTimestamp[Alice] => block.timestamp - block.timestamp = 0`.
5. Alice receives 0 rewards because the malicious user made a small stake before she could claim her rewards, resetting the last interaction timestamp.

```
    function calculateRewards(address user) public view returns (uint256)
        uint256 amountStaked = _stakedAmount[user];
        uint256 apr = getAPR(user);
```

```
        uint256 duration = block.timestamp - _lastInteractionTimestamp[user];

        return (amountStaked * apr * duration) / (365 days * APR_DENOMINATOR);
    }
```

The user will lose his reward due to the malicious user.

Recommendation: In the `updateReward` modifier, use the wallet address for calculating the reward and updating the `_rewardBalance` mapping.

Resolution and Client comment: Resolved. Commit: `#f9f8940`

6.1.2 updateReward modifier missing for stake function

Severity: *High*

Context: StakingWithTiers.sol#L229

Description: The `updateReward` modifier is missing from the `stakeAs` function in the `Launchpool` contract. The `updateReward` modifier ensures that rewards are calculated and saved *before* adding more tokens.

If a user calls the `stake` function twice, the current earned reward will not be calculated. As a result, when they later call the `getReward` function, they will lose the newly staked amount, leading to a partial loss of their rewards.

Recommendation: Add `updateReward` modifier to `stake` function.

Resolution and Client comment: Resolved. Commit: `#f9f8940`

6.2 Medium

6.2.1 Manager can not set claimable amounts during the active claiming period

Severity: *Medium*

Context: Airdrop.sol#L85

Description: When the manager calls the `setClaimableAmounts` function in the Airdrop contract, the sum of all amounts is calculated, and it is checked whether `totalLocked + sum > token.balanceOf(address(this))`. This means that the contract should have at least `totalLocked` tokens, i.e., `token.balanceOf(address(this))>= totalLocked`.

When a user claims their tokens, half of the amount is sent to them, and the other half is sent to the vesting contract. This means that tokens are transferred from the contract, decreasing its token balance.

The `totalLocked` variable stays unchanged, which will be a problem if the manager plans to call the `setClaimableAmounts` function while the claiming period is active.

Example:

- At the beginning, `totalLocked = 100K` and `token.balanceOf(address(this))= 100K`.
- The user claims a 50K amount. `totalLocked` remains 100K, but `token.balanceOf(address(this))` is now 50K.
- The manager tries to add 100K more, transferring 100K to the contract, so `token.balanceOf(address(this))` will be 150K. However, `totalLocked + sum` will be 200K, which will cause the transaction to revert.

```
        if (totalLocked + sum > token.balanceOf(address(this))) {
            revert BalanceTooSmall();
        }
```

Recommendation: Decrement `totalLocked` when users claim their tokens.

Resolution and Client comment: Resolved. Commit: `#f9f8940`

6.3 Information

6.3.1 Redundant import

Severity: *Information*

Context: Launchpool.sol#L11

Description: The following import is redundant in the Launchpool contract.

```
import {Test, console} from "forge-std/Test.sol";
```

Recommendation: Remove the import.

Resolution and Client comment: Resolved. Commit: `#f9f8940`

6.3.2 totalStaked is never decremented

Severity: *Information*

Context: StakingWithTiers.sol#L207

Description: The variable `totalStaked` in `StakingWithTiers` is only incremented by the staked amount or when the user restakes their reward. After the user unstakes, the variable is not decremented.

The `totalStaked` will only increase and will not store the actual current staked amount in the contract**.**

Recommendation: Decrement `totalStaked` when the user unstakes.

Resolution and Client comment: Resolved. Commit: `#f9f8940`

6.3.3 Tiers can not be changed

Severity: *Information*

Context: StakingWithTiers.sol#L196

Description: The Manager can not change the `Tiers` in the `StakingWithTiers` contract. Once they are set, they can not be changed. If this is the intended behavior to maintain transparency for the user, then you can remove delete `_tiers`, as the array will be empty when the contract is initialized.

```
    function _setTiers(Tier[] memory tiers_) internal {
        delete _tiers;

        for (uint256 i = 0; i < tiers_.length; i++) {
            _tiers.push(tiers_[i]);
        }
        emit TiersSet(tiers_);
    }
```

Resolution and Client comment: Acknowledged.