HoneyFun Tokenomics v2 Security Review

February 24, 2025 - February 25, 2025

Date: March 10, 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	4f03bca774456e1d9cfa19fc0d5c35171162cd27
Review Commit hash	bdc5a3a587a78b235ac12fe2093fee83bfb062c4
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
TokenVestingFreezable.sol
stToken.sol

Timeline

February 24, 2025	Audit kick-off
February 25, 2025	Preliminary report
February 27, 2025	Mitigation review

Issues Found

Severity	Count
Critical	1
High	0
Medium	1
Low	0
Information	1
Total	**3**

6 Findings

6.1 Critical

6.1.1 A malicious user can drain all funds from the TokenVesting contract

Severity: *Critical*

Context: TokenVesting.sol#L131

Description: The `vestingSchedule.claimedAmount` is incorrectly updating, which can lead to the draining of all funds from the contract if a user makes multiple claim calls.

Attack Vector:

1. The user has a vesting schedule with `100,000 tokens`, where `50% of the duration has passed`.
2. He calls the **`claim`** function and claim `25,000 tokens` (initial release) plus '`50% of the remaining 75,000 tokens`, total of `62,500 tokens`.

```
return tgeTokens + vestedTokens - claimedAmount;
// return 25,000 + 32,500 - 0 = 62,500 tokens
// unclaimedAmount = 62,500 tokens
```

3. The vesting schedule's `claimedAmount` is set to `62,500`, and the contract transfers this amount to the user.
4. The user calls the **`claim`** function again. Since the check `vestingSchedule.totalAmount == vestingSchedule.claimedAmount` is equal to false, the function proceeds. However, their available claimable tokens become 0, as:

```
return tgeTokens + vestedTokens - claimedAmount;
// return 25,000 + 32,500 - 62,500 = 0 tokens
// unclaimedAmount = 0 tokens
```

5. The vesting schedule is updated incorrectly, setting `vestingSchedule.claimedAmount = 0`, and a `zero transfer` from the contract to the user occurs.
6. The user calls the **`claim`** function again. Since `claimedAmount` was reset to 0, the unclaimed amount becomes `62,500 tokens again`, as:

```
return tgeTokens + vestedTokens - claimedAmount;
// return 25,000 + 32,500 - 0 = 62,500 tokens
// unclaimedAmount = 62,500 tokens
```

7. The vesting schedule's `claimedAmount` is updated to `62,500`, and the contract transfers this amount again.
8. The user repeats this process multiple times until they drain all funds from the contract.

This issue arises due to the incorrect assignment of:

```
vestingSchedule.claimedAmount = unclaimedAmount;
```

This allows the user to repeatedly reset and reclaim the same vested tokens, leading to an infinite drain of funds.

Recommendation: The correct way to update claimedAmount is:

```
vestingSchedule.claimedAmount += unclaimedAmount;
```

Resolution and Client comment: Resolved. Commit: `#bdc5a3a5`

6.2 Medium

6.2.1 Frozen wallet can call claim funciton

Severity: *Medium*

Context: TokenVestingFreezable.sol#L32

Description: The frozen wallet can call the `claim` function to receive its vested tokens. Also, when the airdrop contract calls the vesting contract through the `addVesting` function, it never checks if the beneficiary is a frozen address.

```
    function claim(uint256[] calldata scheduleIds) external virtual override {
        return _claim(scheduleIds);
    }

    function addVesting(address beneficiary, uint256 amount)
        external
        onlyVestingManagerAccess
        validateAddress(beneficiary)
        validateAmount(amount)
    {}
```

Recommendation: Do not allow a frozen wallet to claim its tokens and block the adding of vesting tokens to a frozen wallet.

Resolution and Client comment: Resolved. Commit: `#bdc5a3a5`

6.3 Information

6.3.1 The variable _frozenWallets is never used

Severity: *Information*

Context: TokenVestingFreezable.sol#L13

Description: The mapping`_frozenWallets` is never used in the `TokenVestingFreezable` contract. When a wallet is frozen, the inherited mapping `_freezedAddresses` is used instead.

```
mapping(address => uint256) internal _frozenWallets; // TokenVestingFreezable
mapping(address => bool) internal _freezedAddresses; // TokenVesting
```

Recommendation: Use the `_frozenWallets` mapping from the child contract, not the inherited contract.

Resolution and Client comment: Resolved. Commit: `#bdc5a3a5`