

Pineapple: Staking Contract

Security Review

Cantina Managed review by:
Cccz, Security Researcher

June 23, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 High Risk . . . 4
  - 3.1.1 Users can enjoy high APR without continue lock . . . 4
- 3.2 Medium Risk . . . 5
  - 3.2.1 Updates to `stakingMultiplier` and `BASE_SCORE_VALUE` may prevent users from withdrawal . . . 5
- 3.3 Low Risk . . . 6
  - 3.3.1 `restakeStake()` is missing system status check . . . 6
  - 3.3.2 Reward race issue . . . 6
- 3.4 Informational . . . 7
  - 3.4.1 Redundant Check . . . 7
  - 3.4.2 Centralization risk . . . 7

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Pineapple have simplified the complexities of crypto trading into a seamless and intuitive experience.

From Jun 21st the Cantina team conducted a review of PineappleDEX-staking-contract on commit hash 72f8cada. The team identified a total of **6** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	1	1	0
Low Risk	2	1	1
Gas Optimizations	0	0	0
Informational	2	1	1
Total	**6**	**4**	**2**

3 Findings

3.1 High Risk

3.1.1 Users can enjoy high APR without continue lock

Severity: High Risk

Context: PappleStaking.sol#L376-L390

Description: Users can choose the lock period when entering the system, and the reward will be a higher APR during the lock period and a base APR outside of the lock period.

```
if (rewardDuration > lockPeriod && stake.stakingType != 0) {
    uint256 lockedReward = (stake.tokenAmount *
        lockPeriod *
        stakingAPR[uint8(stake.stakingType)]) / (365 days * BASE);
    uint256 flexReward = (stake.tokenAmount *
        (rewardDuration - lockPeriod) *
        stakingAPR[0]) / (365 days * BASE);
    return lockedReward + flexReward;
} else {
    // default case
    return
        (stake.tokenAmount *
            rewardDuration *
            stakingAPR[uint8(stake.stakingType)]) / (365 days * BASE);
}
```

However, if `rewardDuration <= lockPeriod`, the reward will always be the higher APR. e.g., if a user locks for 1 year and claims the reward after 1.5 years, the reward will be 15% APR for 1 year and 3% APR for 0.5 years. Nonetheless, if the user claims the reward every 0.5 years, the reward will always be 15% APR. This actually makes the user enjoy the higher APR after the lock period without having to continue lock.

Recommendation: Change to:

```
--- a/contracts/PappleStaking.sol
+++ b/contracts/PappleStaking.sol
@@ -369,24 +369,25 @@ contract Staking is
         Stake storage stake = stakes[_id];
         if (stake.finished) return 0;

-        uint256 rewardDuration = block.timestamp - stake.lastClaimTime;
-        uint256 lockPeriod = stakingPeriod[uint8(stake.stakingType)];
-
-        // if a locked stake has finished it continues to accrue base interest until withdrawn (flexReward)
-        if (rewardDuration > lockPeriod && stake.stakingType != 0) {
-            uint256 lockedReward = (stake.tokenAmount *
-                lockPeriod *
-                stakingAPR[uint8(stake.stakingType)]) / (365 days * BASE);
-            uint256 flexReward = (stake.tokenAmount *
-                (rewardDuration - lockPeriod) *
-                stakingAPR[0]) / (365 days * BASE);
-            return lockedReward + flexReward;
-        } else {
-            // default case
-            return
-                (stake.tokenAmount *
-                    rewardDuration *
+        uint256 highAPRTime = stake.startTime + stakingPeriod[uint8(stake.stakingType)];
+        if(block.timestamp > highAPRTime) {
+            if(highAPRTime > stake.lastClaimTime) {
+                uint256 lockedReward = (stake.tokenAmount *
+                    (highAPRTime - stake.lastClaimTime) *
                     stakingAPR[uint8(stake.stakingType)]) / (365 days * BASE);
+                uint256 flexReward = (stake.tokenAmount *
+                    (block.timestamp - highAPRTime) *
+                    stakingAPR[0]) / (365 days * BASE);
+                return lockedReward + flexReward;
+            } else {
+                return (stake.tokenAmount *
+                    (block.timestamp - stake.lastClaimTime) *
+                    stakingAPR[0]) / (365 days * BASE);
+            }
```

```
+        } else {
+            return (stake.tokenAmount *
+                (block.timestamp - stake.lastClaimTime) *
+                stakingAPR[uint8(stake.stakingType)]) / (365 days * BASE);
         }
     }
```

Pineapple: Fixed in commit 3e537348.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 Updates to `stakingMultiplier` and `BASE_SCORE_VALUE` may prevent users from withdrawal

Severity: Medium Risk

Context: PappleStaking.sol#L247-L250

Description: When the user stakes and withdraws, `userTotalScore` is updated based on the result of `_calculateStakeScore()`.

```
userTotalScore[msg.sender] += _calculateStakeScore(
    _amount,
    _stakingType
);
// ...
userTotalScore[stake.user] -= _calculateStakeScore(
    stake.tokenAmount,
    stake.stakingType
);
```

The result of `_calculateStakeScore()` is affected by `stakingMultiplier` and `BASE_SCORE_VALUE`, and they may be changed by the owner.

```
return
        (_amount *
            stakingMultiplier[uint8(_stakingType)] *
            BASE_SCORE_VALUE) / denominator;
}
// ...
function setStakingMultiplier(
    uint8 tier,
    uint256 mult_
) external onlyOwner nonReentrant whenNotPaused {
    require(tier < maxStakingTypes, "Invalid tier");
    uint256 oldMultiplier = stakingMultiplier[tier];
    stakingMultiplier[tier] = mult_;
    emit StakingMultiplierSet(tier, oldMultiplier, mult_);
}
...
function setBaseScoreValue(
    uint256 _newValue
) external onlyOwner nonReentrant whenNotPaused {
    uint256 oldValue = BASE_SCORE_VALUE;
    BASE_SCORE_VALUE = _newValue;
    emit BaseScoreValueSet(oldValue, _newValue);
}
```

If `stakingMultiplier` and `BASE_SCORE_VALUE` increase, due to the increase of `_calculateStakeScore()` during the stake period, this may cause the update of `userTotalScore` to underflow when the user withdraws.

```
userTotalScore[stake.user] -= _calculateStakeScore(
    stake.tokenAmount,
    stake.stakingType
);
```

For example, when a user stakes, `userTotalScore` increases by 100. During the stake period, the `stakingMultiplier` increases. When the user withdraws, `userTotalScore` decreases by 110, resulting in an underflow.

Recommendation: There are two options:

1. When reducing `userTotalScore`, if the reduced value is greater than `userTotalScore`, directly set `userTotalScore` to 0. This does not completely get rid of the issue, because if `stakingMultiplier` and `BASE_SCORE_VALUE` are reduced, the user will have some extra points.
2. Add a `score` field to the `Stake` structure, when the user stakes, save the `score` and increase `userTotalScore`, when the user withdraws, subtract the value of the `score` field directly from `userTotalScore`.

Pineapple: Fixed in commit 2aa5a274.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 `restakeStake()` is missing system status check

Severity: Low Risk

Context: PappleStaking.sol#L294-L300

Description: When `isOpen` is false or `depositsPaused` is true, the protocol prohibits users from calling `stakeTokens()` to enter the system, however `restakeStake()` lacks these checks, allowing users to bypass these status checks and enter the system.

```
function stakeTokens(
    uint256 _amount,
    uint256 _stakingType
) external nonReentrant whenNotPaused {
    require(isOpen, "Staking is not available");
    require(!depositsPaused, "Deposits are paused");
```

Recommendation: Change to:

```
  function restakeStake(uint256 _id) external nonReentrant whenNotPaused {
      Stake storage oldStake = stakes[_id];
+     require(isOpen, "Staking is not available");
+     require(!depositsPaused, "Deposits are paused");
      require(!oldStake.finished, "Stake already withdrawn");
      require(oldStake.user == msg.sender, "Not stake owner");
      uint256 stakingType = oldStake.stakingType;
      uint256 stakingDuration = block.timestamp - oldStake.startTime;
```

Pineapple: Fixed in commit 2aa5a274.

Cantina Managed: Fix verified.

3.3.2 Reward race issue

Severity: Low Risk

Context: PappleStaking.sol#L220-L232, PappleStaking.sol#L310-L312

Description: In `withdrawStake()` and `restakeStake()`, if there are not enough rewards in the system, the system will send the remaining rewards to the user and finish the user's stake.

```
uint256 balance = stakingToken.balanceOf(address(this));

// Contract must at least hold all staked principal
require(
    balance >= totalAmountOfStakes,
    "Insufficient principal balance"
);

// Only the surplus over *all* live principal is usable as reward
uint256 availableReward = balance - totalAmountOfStakes;
uint256 paidReward = rewardAmount <= availableReward
    ? rewardAmount
    : availableReward;
```

One problem is that if multiple users withdraw at the same time, the rewards only satisfy some users, but all users think the rewards are enough. Then when the users' transactions are executed, some users will get little or no rewards due to the reward race issue.

Recommendation: It is recommended to provide a `minRewardOut` parameter in `withdrawStake()` and `restakeStake()` to allow users to control the slippage of the rewards they receive.

Pineapple: Acknowledged. We will ensure that the contract is always sufficiently funded. I will talk this over with leadership, but adding a minReward parameter will in my opinion not benefit the user whatsoever, it would actually do the opposite. If there are no rewards left for some reason then there are no rewards to pay out, regardless of this parameter. Adding this parameter will only ever prevent the user from withdrawing their principal and not benefit them in any way, something we do not want. If the contract is sufficiently funded then this race condition is irrelevant anyway.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Redundant Check

Severity: Informational

Context: PappleStaking.sol#L235

Description: The following check in `withdrawStake()` is incorrect.

```
- require (balance + paidReward >= totalAmountOfStakes, "Insufficient reward liquidity");
+ require (balance >= totalAmountOfStakes + paidReward, "Insufficient reward liquidity");
```

And it is redundant because the previous check and calculation ensures that it is always true:

```
require(
    balance >= totalAmountOfStakes,
    "Insufficient principal balance"
);

// Only the surplus over *all* live principal is usable as reward
uint256 availableReward = balance - totalAmountOfStakes;
uint256 paidReward = rewardAmount <= availableReward
    ? rewardAmount
    : availableReward;

// Contract must at least hold all staked principal + paid reward
require (balance + paidReward >= totalAmountOfStakes, "Insufficient reward liquidity");
```

Recommendation: It is recommended to remove this check.

Pineapple: Fixed in commit 2aa5a274.

Cantina Managed: Fix verified.

3.4.2 Centralization risk

Severity: Informational

Context: PappleStaking.sol#L499-L523

Description: The `setStakingPeriod()` and `setStakingAPR()` functions allow the owner to call to change the stake period and APR, and their changes may cause the user's funds to be locked for longer periods of time or to get a lower APR:

```
function setStakingPeriod(
    uint8 tier,
    uint256 seconds_
) external onlyOwner nonReentrant whenNotPaused {
    require(tier < maxStakingTypes, "Invalid tier");
    uint256 oldPeriod = stakingPeriod[tier];
    stakingPeriod[tier] = seconds_;
    emit StakingPeriodSet(tier, oldPeriod, seconds_);
}
function setStakingAPR(
    uint8 tier,
    uint256 apr_
) external onlyOwner nonReentrant whenNotPaused {
    require(tier < maxStakingTypes, "Invalid tier");
    uint256 oldAPR = stakingAPR[tier];
    stakingAPR[tier] = apr_;
    emit StakingAPRSet(tier, oldAPR, apr_);
}
```

Recommendation: It is recommended to document this issue to inform the user.

Pineapple: Acknowledged. We will add a small disclaimer to the site. However this will only ever be changed to the user's benefit.

Cantina Managed: Acknowledged.