Protocol Audit Report

Version 1.0

Tim Sigl

September 5, 2024

Protocol Audit Report

Tim Sigl

2024/08

Prepared by: Lead Security Researcher Tim Sigl

Table of Contents

Protocol Summary

Rumpel enables points earned via any incentive or loyalty scheme – from Ethena, Zircuit, Symbiotic, etc – to be tokenized, traded, and used in DeFi without lockups or derivatives.

Contest Summary

Sponsor: Rumpel Point Tokenization Protocol

Dates: Aug 26th, 2024 - Aug 29th, 2024

See more contest details here

Disclaimer

The Tim-team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in this document correspond with the following commit hash:

```
a5f71ca5d5ad67a09cf241ad1b78b5d9a0319635
```

Scope

498 nSLOC

```
.
|-- contracts
|   |-- PToken.sol
|   +-- PointTokenVault.sol
+-- sense-finance/rumpel-wallet
    +-- src
        |-- InitializationScript.sol
        |-- RumpelGuard.sol
        |-- RumpelModule.sol
        +-- RumpelWalletFactory.sol
```

Issues found

Severity	Number of Findings
High	0
Medium	1
Low	0
Info	0
Total	1

Findings

Medium

[M-1] PointTokenVault does not handle non-18-decimal reward tokens resulting in too high or too low rewards

Summary The lack of decimal adjustment in `PointTokenVault::redeemRewards()` will cause incorrect reward distribution for users when the operator will set a non-18 decimal reward token, leading to miscalculations in token conversions.

Disclaimer regarding invalidation This issue must not fall under the “Admin Input/call validation” invalidation criteria because there is no assumption made anywhere about the reward token. Other

than the PToken which is always deployed via deployPToken which enforces 18 decimals and the heads up regarding weird point-earning tokens which should be wrapped to avoid weird interactions, there are no assumptions made about the reward token. During the contest steve_fifrom the protocol confirmed that reward tokens can be pretty much any token.

Root Cause In PointTokenVault.sol:191 the protocol is assuming that amountToClaim and rewardsPerPToken have 18 decimals -> Otherwise pTokensToBurn would be calculated incorrectly since pTokens will always have 18 decimals.

The same with the fee calculation in PointTokenVault.sol:211-213 which is deducted from the rewards that are paid out (e.g. if reward tokens have 6 decimals and you deduct a fee with 18 decimals you'll get an underflow (revert))

Internal pre-conditions The operator calls PointTokenVault::setRedemption with a non-18-decimal ERC20 reward token.

External pre-conditions n/a

Attack Path The operator calls PointTokenVault::setRedemption with a non-18-decimal ERC20 reward token Given the attacker already has some pTokens they call PointTokenVault::redeemRewards Depending on whether the decimals of the reward are higher (2.1) or lower (2.2) than 18 we have different scenarios: 2.1 The user won't be able to withdraw any meaningful amount of rewards since pTokensToBurn would be higher than expected 2.2 The user can claim way more reward tokens than they should be able to since pTokensToBurn would be lower than expected (this case is shown in the PoC) Impact As explained in the attack path depending on whether the decimals of the reward token are higher or lower than 18 it results in two different impacts. The worse one, that is also in the PoC, is where the decimals of the reward token are lower than 18.

The user can redeem their pToken for way more reward tokens than anticipated. In the PoC below (mocking USDC as a reward token) the user can get 1,000,000x the amount of reward tokens than they should get.

PoC Paste the following PoC into PointTokenVault.t.sol:

```
1 function test_redemptionWrongForNon18Decimals() public {
2     uint8 decimals = 6;
3     MockERC20 usdcReward = new MockERC20("USDC Reward", "USDC",
          decimals);
4
```

```
5       // Give vault reward tokens to distribute
6       usdcReward.mint(address(pointTokenVault), 1_000_000_000e6);
7
8       // Operator sets reward token to a non-18-decimal reward token
9       vm.startPrank(operator);
10      // For 1 pToken : 2 reward tokens  (e18 - e6) -> 2e12
            rewardsPerToken
11      pointTokenVault.setRedemption(eigenPointsId, usdcReward, 2e12,
            false);
12      vm.stopPrank();
13
14      // This is equal to vitalik buying pTokens somewhere
15      vm.startPrank(address(pointTokenVault));
16      pointTokenVault.pTokens(eigenPointsId).mint(vitalik, 1e18);
17      vm.stopPrank();
18
19
20      bytes32[] memory empty = new bytes32[](0);
21
22      // Vitalik redeems 1 pToken for 2 reward tokens
23      vm.startPrank(vitalik);
24      // pointTokenVault.redeemRewards(PointTokenVault.Claim(
            eigenPointsId, 2e6, 2e6, empty), vitalik);
25      pointTokenVault.redeemRewards(PointTokenVault.Claim(eigenPointsId,
            1e6*2e6, 1e6*2e6, empty), vitalik);
26      vm.stopPrank();
27
28      assertEq(usdcReward.balanceOf(vitalik), 1e6*2e6);
29      assertEq(pointTokenVault.pTokens(eigenPointsId).balanceOf(vitalik),
            0);
30  }
```

Mitigation Two possible mitigations: 1. (worse) Only allow reward tokens with 18 decimals 2. Modify redeemRewards to account for different decimals e.g.

```
1  uint256 rewardTokenDecimals = params.rewardToken.decimals();
2  uint256 decimalAdjustment = 10**(18 - rewardTokenDecimals);
```