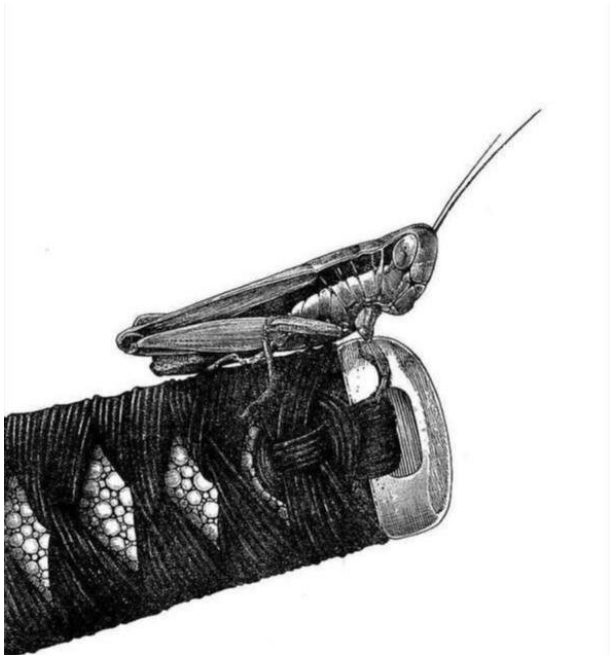

TimeRift Audit Report

Version 1.0

Mahdi Rostami

https://github.com/0xmahdirostami

https://twitter.com/0xmahdirostami

December 27, 2023

TimeRift Audit Report

Mahdi Rostami

December 16, 2023

Prepared by: mahdi rostami

Table of Contents

TimeRift Security Review

A security review of the Possum lab, TimeRift smart contracts protocol was done by mahdirostami. This audit report includes all the vulnerabilities, issues and code improvements found during the security review.

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 find as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**."

- Secureum

Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High**
  - Funds are directly or nearly directly at risk.
  - There's a severe disruption of protocol functionality or availability.
- **Medium**
  - Funds are indirectly at risk.
  - There's some level of disruption to the protocol's functionality or availability.
- **Low**
  - Funds are not at risk.
  - However, a function might be incorrect, state might not be handled appropriately, etc.
  - can lead to any kind of unexpected behaviour with some of the protocol's functionalities that's not so critical.

Likelihood

- **High**
  - attack path is possible with reasonable assumptions that mimic on-chain conditions and the cost of the attack is relatively low to the amount of funds that can be stolen or lost.
- **Medium**
  - only conditionally incentivized attack vector, but still relatively likely.
- **Low**
  - has too many or too unlikely assumptions or requires a huge stake by the attacker with little or no incentive.

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.

Executive summary

Overview

Project Name	TimeRift
Repository	Link
Commit Hash	c18c975
Docs	Link
Methods	Manual Review

Scope

File	nSLOC
Contracts (1)	268
/TimeRift.sol	268

Compatibilities

- Solc Version: 0.8.19
- Chain(s) to deploy contract to: Arbitrum

Known Issues

None

Issues found

Severity	Count
Critical	0
High	0
Medium	1
Low	0
Informational	4

Findings

Medium Severtiy

[M-1] user doesn't get compound rewards.

Description: The TimeRift contract aims to reward users who frequently distribute Energy Bolts. However, the current implementation doesn't consider compounding rewards for users who frequently distribute PSM tokens. The issue lies in the usage of `userStake.stakedTokens` in the `TimeRift::_collectEnergyBolts`.

```
1           uint256 energyBoltsCollected = ((time - userStake.
               lastCollectTime) *
2 @>              userStake.stakedTokens *
3               ENERGY_BOLTS_ACCRUAL_RATE) / (100 * SECONDS_PER_YEAR);
```

This calculation doesn't account for compounding rewards, leading to users not receiving the expected compound rewards.

Impact: Users don't receive compound rewards for frequently distributing PSM tokens. **Likelihood, Feasibility:** Likelihood is high as the issue occurs consistently.

Tools Used: Manual Review

Recommended Mitigation: use `user.exchangeBalance` instead of `userStake.stakedTokens` in both `TimeRift::_collectEnergyBolts` and `TimeRift::getUserEnergyBolts`.

```
1  @@ -214,7 +214,7 @@ contract TimeRift is ReentrancyGuard, Ownable {
2
3           uint256 time = block.timestamp;
4           uint256 energyBoltsCollected = ((time - userStake.
               lastCollectTime) *
5  -            userStake.stakedTokens *
6  +            userStake.exchangeBalance *
7               ENERGY_BOLTS_ACCRUAL_RATE) / (100 * SECONDS_PER_YEAR);
8
9
10 @@ -407,7 +407,7 @@ contract TimeRift is ReentrancyGuard, Ownable {
11
12          uint256 time = block.timestamp;
13          uint256 energyBoltsCollected = ((time - userStake.
               lastCollectTime) *
14 -            userStake.stakedTokens *
15 +            userStake.exchangeBalance *
16              ENERGY_BOLTS_ACCRUAL_RATE) / (100 * SECONDS_PER_YEAR);
```

Proof of Concept: (Proof of Code) add the following test:

```
1  function test_compounding() external {
2          // Whitelist PSM Treasury
3          testAddToWhitelist_Success();
4
5          uint256 stakeAmount = 1e22;
6          uint256 timePassed = 31536000;
7          vm.startPrank(alice);
8          FLASH_Token.approve(address(timeRift), stakeAmount);
9          timeRift.stake(stakeAmount);
10         vm.stopPrank();
11         vm.startPrank(bob);
12         FLASH_Token.approve(address(timeRift), stakeAmount);
13         timeRift.stake(stakeAmount);
```

```
14          vm.stopPrank();
15
16          // alice distribute two times, bob one times
17          vm.warp(timePassed/2);
18          vm.startPrank(alice);
19          (uint256 user_energyBolts) = timeRift.getUserEnergyBolts(alice)
                ;
20          timeRift.distributeEnergyBolts(PSM_Treasury, user_energyBolts);
21          vm.stopPrank();
22          ( , , , ,uint256 user_exchangeBalance) = timeRift.stakes(alice)
                ;
23
24          console2.log("alice exchangeBalance after first distribution",
                user_exchangeBalance);
25
26          vm.warp(timePassed);
27
28          vm.startPrank(alice);
29          (uint256 user_energyBolts_2) = timeRift.getUserEnergyBolts(
                alice);
30          timeRift.distributeEnergyBolts(PSM_Treasury, user_energyBolts_2
                );
31          vm.stopPrank();
32          ( , , , ,uint256 user_exchangeBalance_2) = timeRift.stakes(
                alice);
33
34          console2.log("alice exchangeBalance after second distribution",
                 user_exchangeBalance_2);
35
36          vm.startPrank(bob);
37          (uint256 user_energyBolts_bob) = timeRift.getUserEnergyBolts(
                bob);
38          timeRift.distributeEnergyBolts(PSM_Treasury,
                user_energyBolts_bob);
39          vm.stopPrank();
40          ( , , , ,uint256 user_exchangeBalance_bob) = timeRift.stakes(
                bob);
41
42          console2.log("bob exchangeBalance after first distribution",
                user_exchangeBalance_bob);
43      }
```

Logs before modifying code:

```
1  Running 1 test for test/TimeRiftTest.t.sol:TimeRiftTest
2  [PASS] test_compounding() (gas: 463066)
3  Logs:
4    alice exchangeBalance after first distribution
         174999995243531200243531
5    alice exchangeBalance after second distribution
         249999995243531200243531
```

```
6    bob exchangeBalance after first distribution 24999999524353120243531
```

Logs after modifying code:

```
[PASS] test_compounding() (gas: 463066)
Logs:
  alice exchangeBalance after first distribution
     17499999524353120243531
  alice exchangeBalance after second distribution
     30624999167617960426179
  bob exchangeBalance after first distribution 24999999524353120243531
```

Remark: Fixed.

Informational Severity

[I-1] Lack of Check for Whitelisted Addresses.

Description: The functions `addToWhitelist` and `removeFromWhitelist` currently do not check whether the address exists in the whitelist or not. **Recommended Mitigation:** It is advisable to add checks to these functions. The modified code with the suggested checks is provided below:

```
@@ -350,6 +350,7 @@ contract TimeRift is ReentrancyGuard, Ownable {
     /// @dev The function updates the whitelist mapping.
     /// @param _destination The address to add to the whitelist.
     function addToWhitelist(address _destination) external onlyOwner {
+        if (whitelist[_destination]) revert();
         whitelist[_destination] = true;

         emit WhitelistAdded(_destination);
@@ -359,6 +360,7 @@ contract TimeRift is ReentrancyGuard, Ownable {
     /// @dev The function updates the whitelist mapping.
     /// @param _destination The address to remove from the whitelist.
     function removeFromWhitelist(address _destination) external
        onlyOwner {
+        if (!whitelist[_destination]) revert();
         whitelist[_destination] = false;

         emit WhitelistRemoved(_destination);
```

[I-2] Lack of Zero Address Check for Whitelisted Addresses.

Description: `addToWhitelist` doesn't have zero address check. **Recommended Mitigation:** add some checks for this functions:

```
@@ -350,6 +350,7 @@ contract TimeRift is ReentrancyGuard, Ownable {
     /// @dev The function updates the whitelist mapping.
     /// @param _destination The address to add to the whitelist.
     function addToWhitelist(address _destination) external onlyOwner {
+        require(_destination != address(0));
         whitelist[_destination] = true;
```

[I-3] Improve Readability and Optimization in DistributeEnergyBolts

Description: In the `TimeRift::DistributeEnergyBolts` function, the condition if (available*PSM* <= *_amount* 2) can be enhanced for better readability and optimization. Changing it to if (available*PSM* < *_amount* 2) would be more appropriate, as when available_PSM is equal to _amount * 2, there is no need for additional operations and checks.

Recommended Mitigation:

```
@@ -269,7 +269,7 @@ contract TimeRift is ReentrancyGuard, Ownable {

         /// @dev Calculate the correct amount of exchange balance increase and distributed tokens.
         /// @dev The increase of the user's exchange balance has priority over distributing tokens to the destination.
-        if (available_PSM <= _amount * 2) {
+        if (available_PSM < _amount * 2) {
```

[I-4] Add a White List Address in Constructor

Description: Currently, after deploying the contract, the owner must set a whitelist address by calling `addToWhitelist` to allow users to distribute and gain PSM tokens. Forgetting to make this important transaction can lead to potential issues.

Recommended Mitigation:

- To streamline the deployment process and prevent oversight, add the ability to set a whitelist address directly in the constructor.

```
@@ -38,7 +38,8 @@ contract TimeRift is ReentrancyGuard, Ownable {
         address _PSM_TREASURY,
         uint256 _MINIMUM_STAKE_DURATION,
         uint256 _ENERGY_BOLTS_ACCRUAL_RATE,
-        uint256 _WITHDRAW_PENALTY_PERCENT
+        uint256 _WITHDRAW_PENALTY_PERCENT,
+        address _WHITELIST_DESTINATION
     ) Ownable() {
         if (_FLASH_ADDRESS == address(0)) {
```

```
9                 revert InvalidInput();
10 @@ -52,6 +53,9 @@ contract TimeRift is ReentrancyGuard, Ownable {
11            if (_PSM_TREASURY == address(0)) {
12                revert InvalidInput();
13            }
14 +          if (_WHITELIST_DESTINATION == address(0)) {
15 +              revert InvalidInput();
16 +          }
17            if (
18                _MINIMUM_STAKE_DURATION < 2592000 ||
19                _MINIMUM_STAKE_DURATION > 31536000
20 @@ -74,6 +78,10 @@ contract TimeRift is ReentrancyGuard, Ownable {
21            MINIMUM_STAKE_DURATION = _MINIMUM_STAKE_DURATION;
22            ENERGY_BOLTS_ACCRUAL_RATE = _ENERGY_BOLTS_ACCRUAL_RATE;
23            WITHDRAW_PENALTY_PERCENT = _WITHDRAW_PENALTY_PERCENT;
24 +          whitelist[_WHITELIST_DESTINATION] = true;
25 +
26 +          emit WhitelistAdded(_WHITELIST_DESTINATION);
27 +
28        }
```

With this modification, the whitelist address is set during contract deployment, reducing the chance of oversight and ensuring a smoother deployment process.

- Certainly, calling the addToWhitelist function in the deploy script is another valid approach. This way, the deployment script can deploy the contract and then immediately call the addToWhitelist function with the desired whitelist address.