# Security Assessment for Tsunami Finance

Report by Ulam Labs

# Margin trading and spot exchange

Findings and Recommendations Report Presented to:

Tsunami Finance Team

August 4, 2023 Version: 0.1

Presented by:

Ulam Labs

Grabiszynska 163/502,

53-332 Wroclaw, POLAND

# Executive Summary

## Overview

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Ulam Labs Security Teams took to identify and validate each issue, and any applicable recommendations for remediation.

## Scope

The audit has been conducted on the commit **43ade07d1f465b197a0dfc01100c3e6cd84a075f** of Tsunami Finance private GitHub Repository.

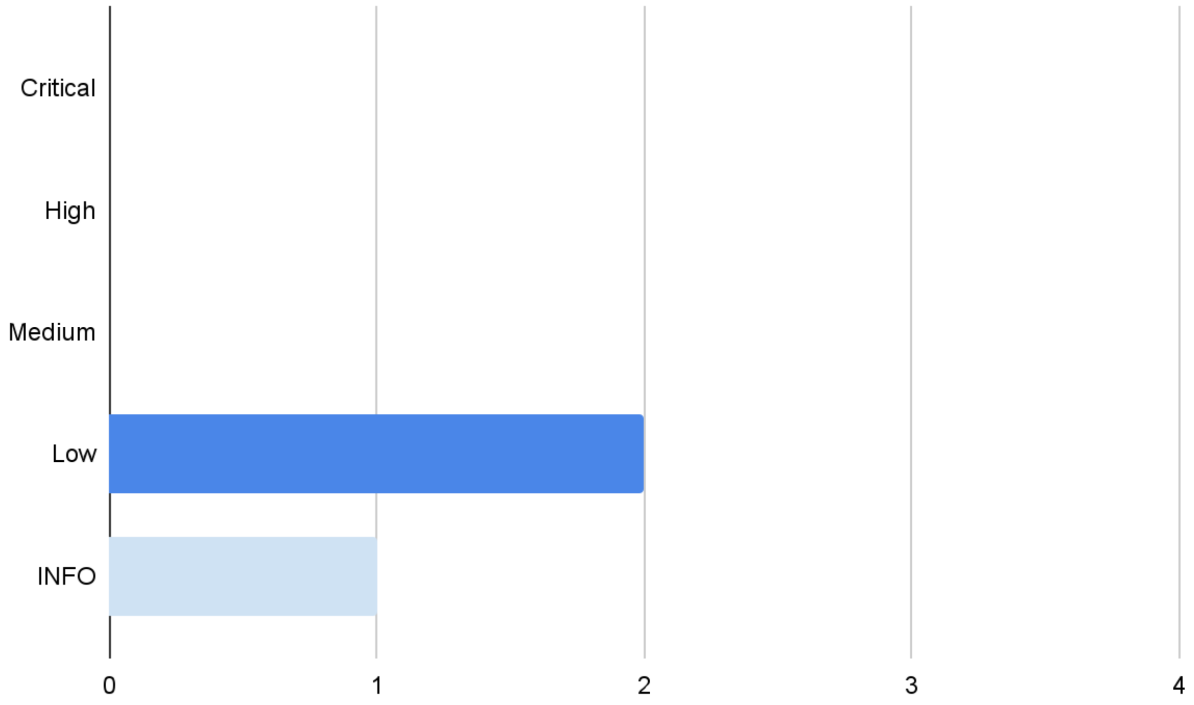

Chart 1: Findings by severity.

## Key findings

During the Security Assessment, following findings have been discovered:

- 0 findings with a CRITICAL severity rating,
- 0 findings with a HIGH severity rating,
- 0 findings with a MEDIUM severity rating,
- 2 findings with a LOW severity rating.
- 1 findings with an INFO severity rating.

## Disclaimer

This report does not constitute legal or investment advice. The preparers of this report present it as an informational exercise documenting the due diligence involved in the secure development of the target contract only, and make no material claims or guarantees concerning the contract's operation post-deployment. The preparers of this report assume no liability for any and all potential consequences of the deployment or use of the contract.

Smart contracts are still a nascent software arena, and their deployment and public offering carries substantial risk. This report makes no claims that its analysis is fully comprehensive, and recommends always seeking multiple opinions and audits.

This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system.

The possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims which impact a large quantity of funds.

# Technical analysis & findings

## Function pow can unexpectedly overflow

Finding ID: **TF-1**
Module: math.move
Severity: Low
Status: Closed

### Description

Function **pow** takes as its arguments 64-bit **base** and 8-bit **exp**. It returns a 128-bit result. Caller may expect that such function should support up to 128-bit results, but actually even valid 64-bit results may cause integer overflow.

```
public fun pow(base: u64, exp: u8): u128 {
    let result_value: u128 = 1;
    while (exp > 0) {
        if (exp & 1 == 1) {
            result_value = result_value * (base as u128)
        };
        base = base * base;
        exp = exp >> 1
    };
    result_value
}
```

The problem lies in **base** multiplication. As this variable is 64-bit, it is impossible to create **result** greater than 64-bit. In addition, as **base** is calculated after **result_value**, some “almost” 128-bit results like **pow(2, 63)** also will cause integer overflow.

### Impact

Function **pow** is used by only one function: **pow_10**. Function **pow_10** is used to convert decimals or precision to number. If any coin or price have decimals of **19**, the contract will always crash.

### Solution

There are two steps to make the **pow** function ready to return a 128-bit result.

First, **base** should be assigned to 128-bit variable **base128** and all the calculations should be performed on **base128** variable. Casting inside the loop is not required anymore.

Second, **exp** should be recalculated before **base128** and if **exp** is zero, the loop should be terminated. Loop condition is not needed anymore.

### Status

Problem acknowledged by the team, but as the problem is very unlikely, correction was not delivered.

## Staking rewards does not depend on time

Finding ID: **TF-2**
Contract: stake_rewards.move
Severity: Info
Status: Closed

### Description

Tsunami finance uses following formula to distribute staking rewards:

$$cumulative\_reward\_numerator = rewards\_in\_reward\_coin \ * \ 10^{FEE\_UNITS\_PRECISION}$$

$$staking\_reward\_units = \frac{cumulative\_reward\_numerator}{total\_staked\_coin\_supply}$$

$$cumulative\_staking\_reward\_units \mathrel{+}= staking\_reward\_units$$

Each user has its own $cumulative\_staking\_reward\_units$ to track how many rewards have been already claimed.

Pending rewards are calculated using formulas:

$$delta\_bw\_user\_global\_staking\_fee\_units = g.cumulative\_staking\_reward\_units$$

$delta\_bw\_user\_global\_staking\_fee\_units \ -= \ u.user\_cumulative\_staking\_reward\_units$

$$new\_staking\_fees\_to\_claim = \frac{delta\_bw\_user\_global\_staking\_fee\_units * user\_staked\_coins\_amount}{10^{FEE\_UNITS\_PRECISION}}$$

$pending\_staking\_rewards\_to\_claim \ += \ new\_staking\_fees\_to\_claim$

$u.user\_cumulative\_staking\_reward\_units = g.user\_cumulative\_staking\_reward\_units$

As we can see, user rewards depends only on $user\_staked\_coins\_amount$ and $user\_cumulative\_staking\_reward\_units$. It means that the user has incentive to stake as much as possible, as early as possible. However $user\_cumulative\_staking\_reward\_units$ does not increase linearly and staking just one transaction before reward distribution is the same as staking after last distribution.

It creates the environment, when honest users keep locking their funds for a long time get the same amount of rewards as malicious users running their own node and stake just before distribution and un-stake just after.

### Impact

Onchain data shows that staking is deployed, but not enabled, so there are no funds at risk right now.

### Solution

Problem has been presented to the team and nothing is planned to be delivered, because staking functionality will only be interacted with from another set of whitelisted modules that claim all staking rewards.

## Gas usage increases with open position count

Finding ID: **TF-3**
Contract: position.move
Severity: Low
Status: Closed

### Description

Function **get_position_id_if_exists** returns position id for given user, basket, position and collateral coin. Positions are checked using a while loop and if a matching position is found, its id is returned.

The problem with such implementation is that if a malicious user creates lots of positions, everyone who creates a position afterwards will have to bear higher gas costs.

### Impact

Number of open positions is limited to 20 000, and even if someone wants to create so many positions, gas costs will increase only up to cents.

### Solution

To solve this problem, position location should be found off chain and confirmed on chain.

Mentioned proposal has been rejected, because it would complicate the design and keeper scripts too much without giving too much in return.

## General observations

### General safety

The Tsunami finance team put in a lot of effort to make their smart contracts safe.
First thing increasing safety is good design. All the entry points have a minimal set of parameters, which makes the interface easy to use and hard to do something wrong.
There are also some limitations in contract usage. Only order/position creator, whitelisted keeper and admin can interact with the system. Of course it does not eliminate the risk of malicious behavior, but drastically decreases attacker possibilities to do something harmful and remain anonymous.
Last, probably the most important thing done to increase safety is a huge amount of tests. Almost each case, when something could go wrong, has been covered.

## Other

### Severity classification

We have adopted a severity classification inspired by the Immunefi Vulnerability Severity Classification System - v2. It can be found here.