PancakeSwap IFO

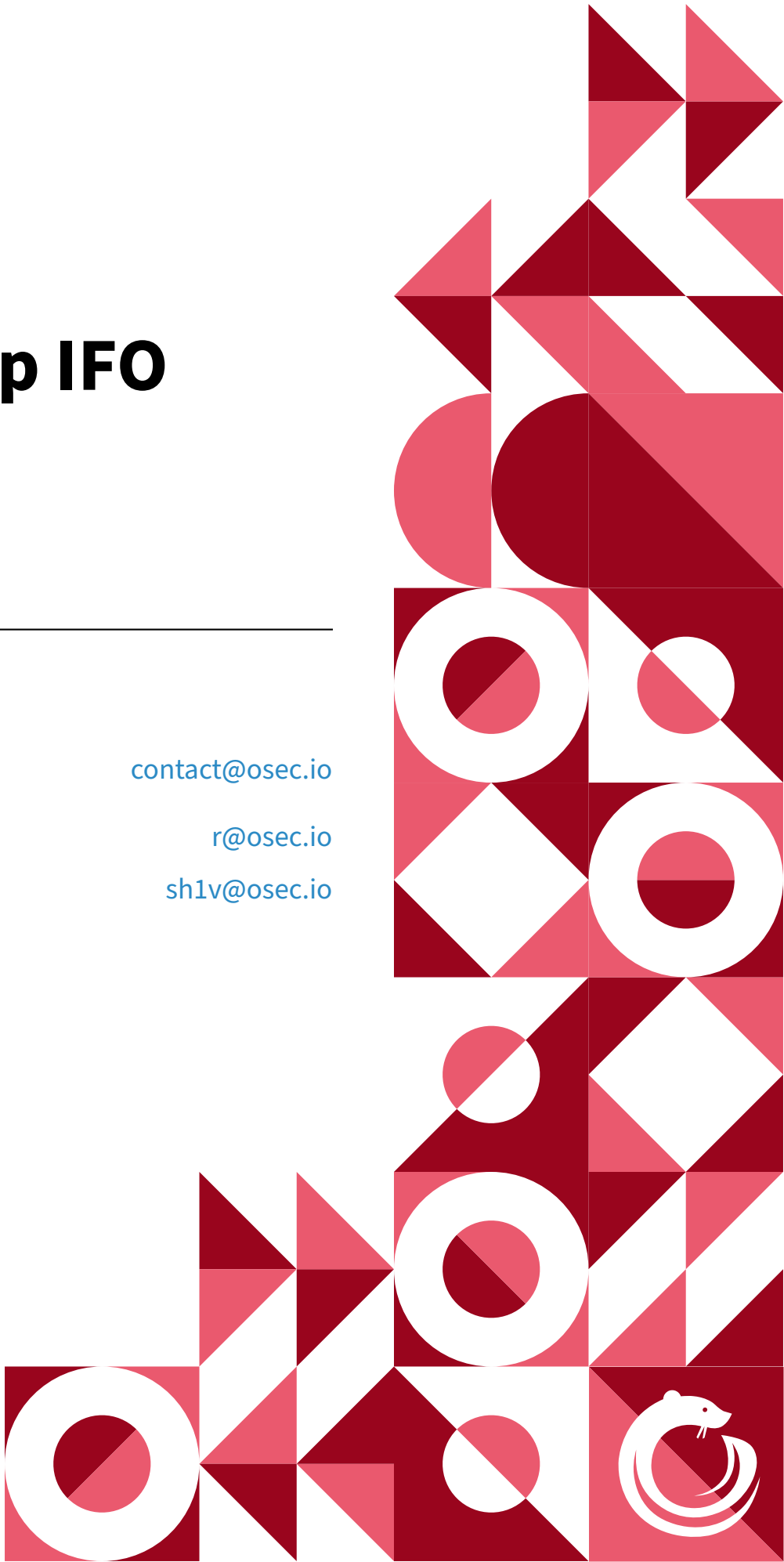
Audit

Presented by:

OtterSec contact@osec.io

Robert Chen r@osec.io

Shiva Shankar sh1v@osec.io



Contents

Appendices

01 | Executive Summary

Overview

Pancake Swap engaged OtterSec to perform an assessment of the `pancake-IFO` program. This assessment was conducted between December 5th and December 12th, 2022.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team to streamline patches and confirm remediation. We delivered final confirmation of the patches December 19th, 2022.

Key Findings

Over the course of this audit engagement, we produced 3 findings total.

In particular, we noted issues around potential overflow related denial of service vectors (OS-IFO-ADV-00). We also made suggestions around rounding and improved validation of types (OS-IFO-SUG-00, OS-IFO-SUG-01).

Overall, the Pancake Swap team was responsive to feedback and great to work with.

02 | Scope

The source code was delivered to us in a git repository at github.com/pancakeswap/aptos-contracts/. This audit was performed against commit `0b2a285`.

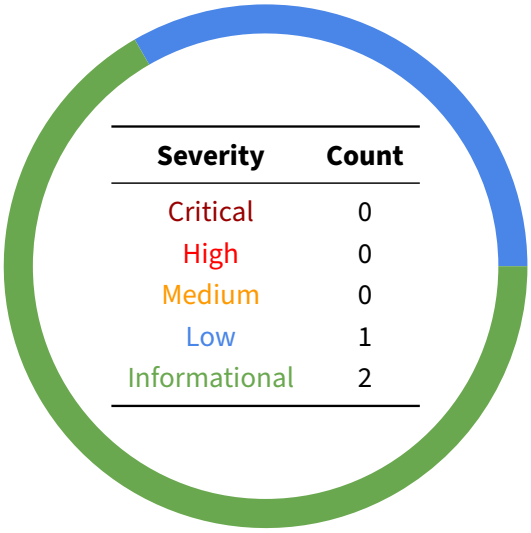
A brief description of the programs is as follows.

Name	Description
pancake-IFO	Initial farm offering contract built on Aptos

03 | Findings

Overall, we report 3 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	0
Low	1
Informational	2

04 | Vulnerabilities

Here we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-IFO-ADV-00	Low	Resolved	Possible denial of service due to overflow via multiplication

OS-IFO-ADV-00 [low] [resolved] | Multiplication Overflow DOS

Description

In `compute_release_amount`, the `vested_amount` calculation involves multiplying together the total amount to be vested by the vested seconds.

sources/IFO.move RUST

```
    let vested_seconds = vested_slice_periods * seconds_per_slice;
    let vested_amount = vesting_schedule.amount_total * vested_seconds /
    ↪  ifo_pool.vesting_duration;
```

For large quantities of tokens vested over a long period of time, this calculation might abort, causing the contract to abort. Note that this is recoverable once the tokens have fully vested.

A similar issue can be found in `harvest_pool` when calculating the initial vesting percentages. This will only be an issue if a user attempts to vest more than `u64:MAX / 100` tokens.

Remediation

Cast the values to `u128` before multiplying them together

Patch

Resolved in 37b2239.

05 | General Findings

Here we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-IFO-SUG-00	Incorrect rounding in `harvest_pool` means user can lose one token
OS-IFO-SUG-01	Possible sanity checks on `uints::get_number`.

OS-IFO-SUG-00 | Harvest Pool Rounding

Description

In `harvest_pool`, the amount of `OfferingCoin` the user gets is rounded down in both the immediate vesting and delayed vesting branches.

This means that a user could get one less token than expected.

Remediation

Store the amount that's immediately given to the user and set the vested amount as `offering_amount - initial_amount`.

OS-IFO-SUG-01 | Tighter Type Validation

Description

`uints::get_number` could enforce that the `UID` type passed in is owned by the correct module and starts with ”U”. This could mitigate type confusion issues down the line.

sources/uints.move RUST

```
    public fun get_number<UID>(): u64 {
        let struct_name = struct_name(&type_of<UID>());
        let len = length(&struct_name);
        let result: u64 = 0;
        let idx = 1; // Skip "U"
        while (idx < len) {
            result = 10 * result;
            let number = *borrow(&struct_name, idx) - 48;
            result = result + (number as u64);
            idx = idx + 1;
        };
        result
    }
```

Remediation

Check that the `UID` type is owned by the correct module and sanity check the type name format.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities that immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority or access control validation • Improperly designed economic incentives leading to loss of funds
High	Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities that could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation