Switchboard

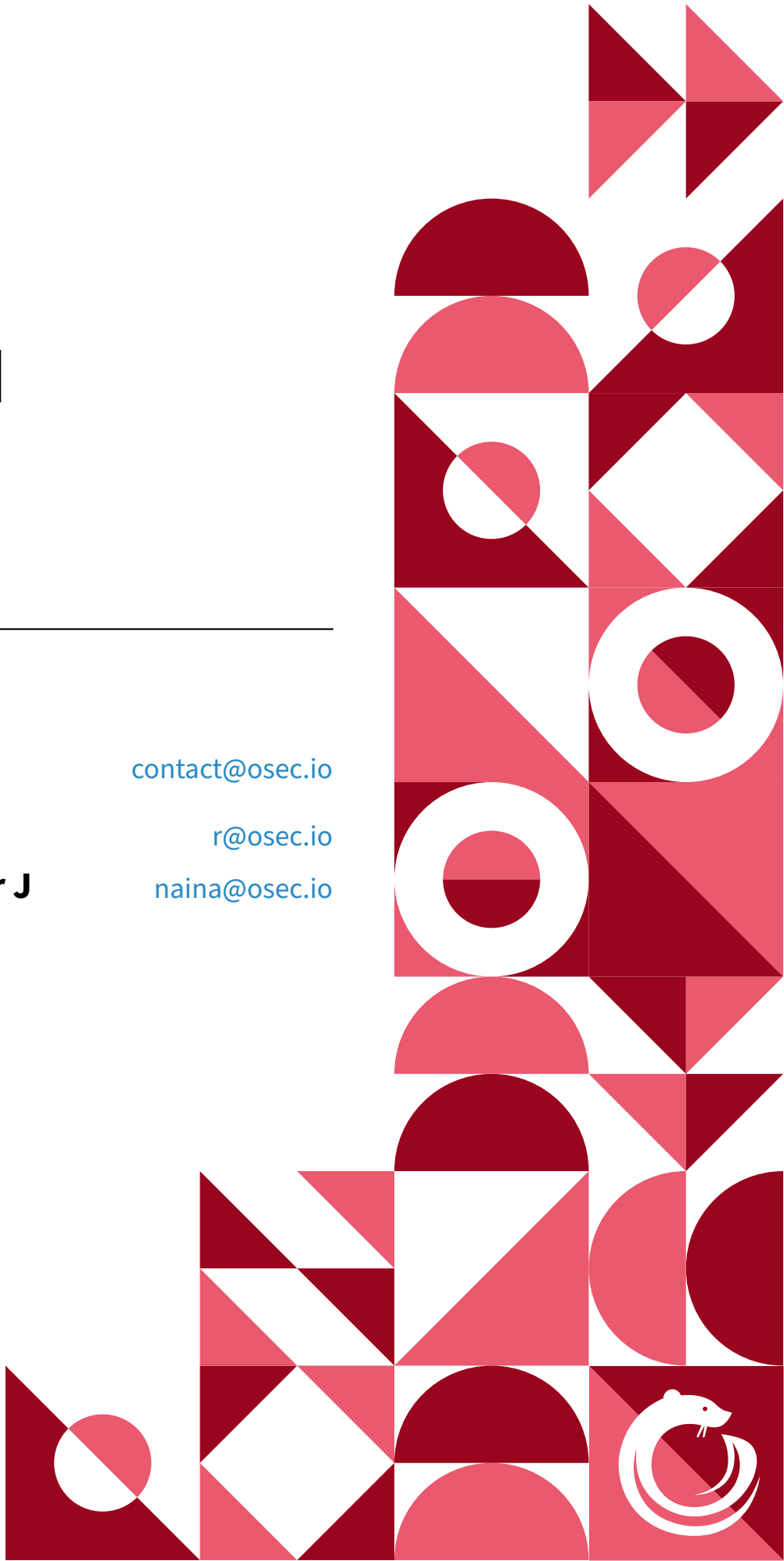
Audit

Presented by:

OtterSec contact@osec.io

Robert Chen r@osec.io

Naveen Kumar J naina@osec.io



Contents

Appendices

01 | Executive Summary

Overview

Switchboard engaged OtterSec to perform an assessment of the `switchboard-aptos` program. This assessment was conducted between October 26th and November 11th, 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 **[not yet delivered]**.

Key Findings

Over the course of this audit engagement, we produced 6 findings total.

In particular, we found a denial of service related to job removals (OS-SWB-ADV-00) and an issue in the math library with inconsistent results on unmatched decimals (OS-SWB-ADV-01).

We also made a number of recommendations around round data read limitations (OS-SWB-SUG-00, OS-SWB-SUG-01), contract access control (OS-SWB-SUG-02), and general code suggestions (OS-SWB-SUG-03).

Overall, we commend the Switchboard team for being responsive and knowledgeable throughout the audit.

02 | Scope

The source code was delivered to us in a git repository at github.com/switchboard-xyz/switchboard-aptos. This audit was performed against commit `f929dac`.

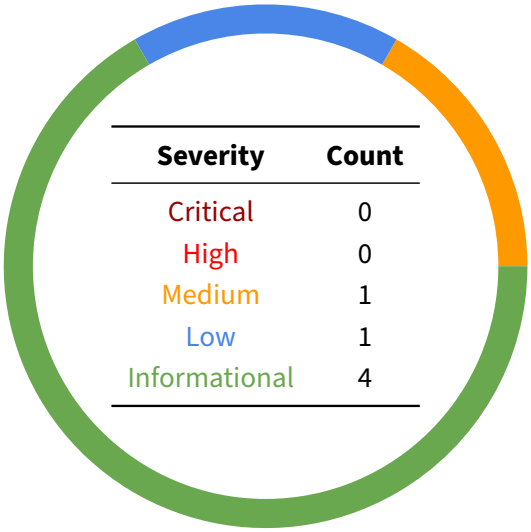
A brief description of the programs is as follows.

Name	Description
Switchboard	Permissionless data feeds on-chain, built on Aptos

03 | Findings

Overall, we report 6 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 do not have an immediate impact but will help mitigate future vulnerabilities.



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

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-SWB-ADV-00	Medium	TODO	Denial of service when removing a job
OS-SWB-ADV-01	Low	TODO	Inconsistent Results On Unmatched Decimals

OS-SWB-ADV-00 [med] | Remove Job DOS

Description

The function `aggregator::remove_job` skips the execution silently when the supplied job address doesn't exist in `aggregator_job_data`. This should abort because when `aggregator_remove_job_action::actuate` calls this function and skips execution, the following call to decrement the `job_ref_count` will be reduced.The `job_ref_count` is used to keep track of number of references that a job has. So If a job gets added count increases and count decreases when removed.

sources/actions/aggregator/aggregator_remove_job_action.move RUST

```
fun actuate(_account: &signer, params: &AggregatorRemoveJobParams) {
    aggregator::remove_job(params.aggregator_addr, params.job_addr);
    job::sub_ref_count(params.job_addr);
}

public(friend) fun remove_job(addr: address, job: address) acquires
    ↪ AggregatorJobData {
    let aggregator_job_data = borrow_global_mut<AggregatorJobData>(addr);
    let (is_in, idx) = vector::index_of(&aggregator_job_data.job_keys,
    ↪ &job);
    if (!is_in) {
        return
    };
    [..]
}
```

Remediation

Abort if the supplied job doesn't exist.

sources/schemas/aggregator.move RUST

```
public(friend) fun remove_job(addr: address, job: address) acquires
    ↪ AggregatorJobData {
    let aggregator_job_data = borrow_global_mut<AggregatorJobData>(addr);
    let (is_in, idx) = vector::index_of(&aggregator_job_data.job_keys,
    ↪ &job);
    assert!(is_in, E_JOB_NOT_FOUND);
    [..]
}
```

OS-SWB-ADV-01 [low] | Inconsistent Results On Unmatched Decimals

Description

`SwitchboardDecimal` can store a decimal value along with its decimals and sign. When a new decimal is initialized, it will be scaled to 9 decimals by default(`MAX_DECIMALS`), which is the maximum limit. All of the operations inside the math library assume that all the passed `SwitchboardDecimal` are scaled to `MAX_DECIMALS`. With the function, `math::normalize` scaling can be reversed; Thus, when an unscaled value is sent with a scaled value, the outcome is erroneous.

Proof of Concept

test.move RUST

```
#[test]
#[expected_failure(abort_code = 7331)]
fun test_decimal() {

    let x = new(1, 8, false);   // Scaled   Value - 10    [9 dec]
    let y = new(4, 7, false);   // Scaled   Value - 400   [9 dec]
    let result = add(&x,&y);    // Scaled   Value - 410   [9 dec]

    assert!(result.value == 410, 1337);

    normalize(&mut x);          // Unscaled Value - 1     [8 dec]
    normalize(&mut y);          // Unscaled Value - 4     [7 dec]
    result = add(&x,&y);        // Scaled   Value - 5     [9 dec]   //
    ↪  Actual
                                // Scaled   Value - 410   [9 dec]   //
    ↪  Expected

    assert!(result.value == 410, 7331);
}
```

Note: This has no impact when used in the context of the switchboard project because the input values were internally scaled and passed to other functions.

Remediation

Check if the passed in `SwitchboardDecimal`'s are scaled before performing any operation.

Patch

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-SWB-SUG-00	Suggestions Related to Limitation on Round Data Reading
OS-SWB-SUG-01	Block Removal Of Job When The Aggregator Is Locked
OS-SWB-SUG-02	Permission Contract Access Control Suggestions
OS-SWB-SUG-03	General Code Suggestions

OS-SWB-SUG-00 | Round Data Read Limitation Suggestions

Description

The public getter functions which return round data should abort if the value `aggregator::limit_reads_to_whitelist` is set to true. In the function `aggregator::latest_round`, there was no check to limit the users even if the value is set to true.

sources/schemas/aggregator.move RUST

```
public fun latest_round(addr: address): ([..]) acquires AggregatorRound,
    ↪  AggregatorReadConfig {
    let aggregator = borrow_global_mut<AggregatorReadConfig>(addr);
    assert!(aggregator.read_charge == 0, errors::PermissionDenied());
    [..]
}
```

This would allow anyone to read the data of the last round when the `read_charge` is 0, regardless of whether it is allowed to read or not.

Remediation

Add a check to ensure the value `limit_reads_to_whitelist` is set to false.

RUST

```
assert!(
    aggregator_read_config.read_charge == 0
                                &&
    !aggregator_read_config.limit_reads_to_whitelist,
errors::PermissionDenied());
```

Patch

OS-SWB-SUG-01 | Block Removal Of Job When The Aggregator Is Locked

Description

When the value `aggregator.is_locked` is set to true, the aggregator should not allow the job addition and removal. The `aggregator_remove_job_action::validate` should check if the aggregator is locked before removing the job.

```rust
public fun validate(account: &signer, params: &AggregatorRemoveJobParams)
{
   assert!(aggregator::exist(params.aggregator_addr),
   errors::AggregatorNotFound());
   assert!(job::exist(params.job_addr), errors::JobNotFound());
   assert!(aggregator::has_authority(params.aggregator_addr, account),
   errors::InvalidAuthority());
}
```

This would allow jobs to be removed even if the aggregator is locked.

Remediation

Add a check to see if the aggregator is locked or not.

```rust
assert!(!aggregator::is_locked(params.aggregator_addr),
    ↪  errors::AggregatorLocked());
```

OS-SWB-SUG-02 | Permission Contract Access Control

Description

Generally, it's a good practice to use the `friend` concept in Move for better access control. In the contract `Permission`, the functions `set` and `unset` were declared as public. It is preferred to let them only be invoked by their friends.

RUST

```
public fun set(permission: &mut Permission, code: u64) {
    [..]
}
public fun unset(permission: &mut Permission, code: u64) {
    [..]
}
```

Though it is not possible to get `&mut Permission` of others, it is better to restrict the access of functions that modifies the state of resources.

Remediation

Use `public(friend)` instead of `public` for the setter to have better access control.

RUST

```
public(friend) fun set(permission: &mut Permission, code: u64);
public(friend) fun unset(permission: &mut Permission, code: u64);
```

OS-SWB-SUG-03 | General Code Suggestions

Description

1. Dead Code in the function `lease_extend_action::actuate`. This code snippet seems to do nothing. It is better to remove dead code.

sources/actions/lease/lease_extend_action.move RUST

```
fun actuate<CoinType>(account: &signer, params: &LeaseExtendParams)
    ↪ {
    if (escrow::balance<CoinType>(params.aggregator_addr, queue_addr)
    ↪ < max_round_cost &&
        escrow::balance<CoinType>(params.aggregator_addr, queue_addr)
    ↪ + params.load_amount >= max_round_cost) {
    };
}
```

2. As a whole, the project does not contain end-to-end tests. It would be better to have end-to-end tests to test edge cases.

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