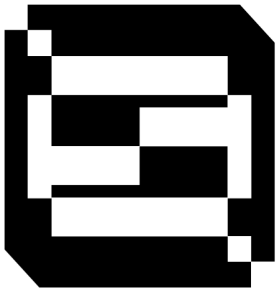

OFFSIDE Labs

Jupiter DAO

Smart Contract Security Assessment

March 2024

Prepared for:

Raccoons

Prepared by:

Offside Labs

Zhuo Zhang

Siji Feng

Contents

1 About Offside Labs

Offside Labs is a leading security research team, composed of top talented hackers from both academia and industry.

We possess a wide range of expertise in modern software systems, including, but not limited to, *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. We are also at the forefront of innovative areas like *cryptocurrencies* and *blockchain technologies*. Among our notable accomplishments are remote jailbreaks of devices such as the **iPhone** and **PlayStation 4**, and addressing critical vulnerabilities in the **Tron Network**.

Our team actively engages with and contributes to the security community. Having won and also co-organized *DEFCON CTF*, the most famous CTF competition in the Web2 era, we also triumphed in the **Paradigm CTF 2023** within the Web3 space. In addition, our efforts in responsibly disclosing numerous vulnerabilities to leading tech companies, such as *Apple*, *Google*, and *Microsoft*, have protected digital assets valued at over **$300 million**.

In the transition towards Web3, Offside Labs has achieved remarkable success. We have earned over **$9 million** in bug bounties, and **three** of our innovative techniques were recognized among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs

2 Executive Summary

Introduction

Offside Labs completed a security audit of *Jupiter*'s *DAO* smart contracts, starting on March 7th, 2024, and concluding on March 15th, 2024.

DAO Project Overview

The DAO utilizes three programs: locked-voter, govern, and smart-wallet. Staking JUP in locked-voter grants voting power. User create a draft proposal on govern program. Council activates a draft proposal from smart-wallet program. After users vote, successful proposals are queued in smart-wallet for execution. Voters may receive incentives from the council for their participation in the governance process.

Audit Scope

The assessment scope contains mainly the smart contracts of the *DAO* program for the *Raccoons Team*.

The audit is based on the following specific branches and commit hashes of the codebase repositories:

- DAO
  - Branch: main
  - Commit Hash: 344cc209165eadc52a4c8dd9a0e681b6a659a890
  - Codebase Link

We listed the files we have audited below:

- DAO
  - programs/locked-voter
  - programs/govern
  - programs/smart-wallet

Findings

The security audit revealed:

- 1 medium issues
- 2 low issues
- 2 informational issues

Further details, including the nature of these issues and recommendations for their remediation, are detailed in the subsequent sections of this report.

3 Summary of Findings

ID	Title	Severity	Status
01	The Constraint of min_stake_duration can be Bypassed	Medium	Fixed
02	Deadlock in Smart Wallet when Threshold Equals to owners.len()	Low	Fixed
03	Voting Power does not Linearly Decrease after Unstaking	Low	Acknowledged
04	escrow_tokens Account Not Closed Upon Withdrawal	Informational	Fixed
05	Unable to Close Invalid Transactions	Informational	Acknowledged

4 Key Findings and Recommendations

4.1 The Constraint of min_stake_duration can be Bypassed

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic

Description

When extending the lock duration, the code mandates a minimum staking period. However, when increasing the locked amount, there is no check on the locking period. As such, a malicious user could potentially circumvent the minimum staking period requirement by initially staking just `1` token, waiting for the locking period to decrease to a desired duration, and then using `increase_locked_amount` to deposit the originally intended amount.

This has particular negative impact given the current method of calculating voting power. Specifically, the voting power does not decrease linearly with the current time, but rather in relation to the proposal's end time. Consider `T1`, `T2`, and `T3` as the current time, the end time of proposal voting, and the end time of the subject escrow (i.e., the end time of locking), respectively. The code enforces `T1 < T2 < T3`. If we use `V` to denote the voting power when `max_lockas` is `true`, the current code implementation in fact enforces the voting power as `V * min(T3 - T2, max_stake_duration) / max_stake_duration`.

Therefore, a malicious user doesn't need to lock a large number of tokens at the very beginning. Instead, they can opt to lock just 1 token at an opportune moment, allowing them to control the duration between `T3` and `T2`. They then lock as many tokens as desired at or near the time of `T2`. Unlike ordinary users, this approach enables them to lock their tokens for any duration they prefer, thereby undermining the intended purpose of locking funds.

Recommendation

When increasing the locked amount, add checks ensuring that the remaining lock duration exceeds `increase_locked_amount` (if the lock is already in place).

Mitigation Review Log

Raccoons: Explained in RP#47.

4.2 Deadlock in Smart Wallet when Threshold Equals to owners.len()

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic

Description

In a smart wallet, at least `k` signers must approve transactions, where `k` represents the `threshold`. Additionally, governance stipulates that the `governor` must be one of the smart wallet's signers (in `Govern::create_governor`).

If the threshold equals the number of owners (i.e., `owners.len()`), the governor's approval is necessary for smart wallet to initiate transactions. Conversely, the `governor` requires the smart wallet's signature to send approval, leading to a deadlock.

This issue can arise in the following scenarios, with the latter being significantly more likely:

1. When defining the threshold, the code only checks `threshold <= owners.len()`, allowing a configuration that can lead to deadlock.
2. When updating the smart wallet's owners, if the number of new owners is fewer than the existing threshold, as illustrated in the code snippet.

```
pub fn set_owners(&mut self, owners: Vec<Pubkey>) -> Result<()> {
    let smart_wallet = &mut self.smart_wallet;
    if (owners.len() as u64) < smart_wallet.threshold {
        smart_wallet.threshold = owners.len() as u64;
    }

    ...
}
```

Recommendation

1. Validate the threshold is less than the number of owners when creating governor.
2. Add sanity checks in `set_owners`.

Mitigation Review Log

Raccoons: Fixed in PR#48.

4.3 Voting Power does not Linearly Decrease after Unstaking

Severity: Low	Status: Acknowledged
Target: Smart Contract	Category: Logic

Description

Per the description posted online, after unstaking (i.e., setting `max_lock` as `true`), voting power will linearly decrease.

However, according to the `CastVote` instruction of `locked-voter`, the voting power is calculated as follows.

```
pub fn cast_vote(&mut self, side: u8) -> Result<()> {
    let voting_power = self.future_voting_power()?;
    ...
}

/// The voting power of the escrow at the time the proposal's voting
    ↪ ends.
fn future_voting_power(&self) -> Result<u64> {
    Ok(unwrap_int!(self.escrow.voting_power_at_time(
        &self.locker,
        self.proposal.voting_ends_at
    )))
}
```

As a result, the voting power is actually decided by the time of the proposal's voting ends, instead of the current time.

That is, consider `T1`, `T2`, and `T3` as the current time, the end time of proposal voting, and the end time of the subject escrow, respectively. The code enforces `T1 < T2 < T3`. We also use `V` to denote the voting power when `max_lockas` is `true`.

As such, the voting power described in the document seems to be `V * min(T3 - T1, max_stake_duration) / max_stake_duration`. As such, the voting power is linearly decreased.

But the code implementation in fact enforce the voting power as `V * min(T3 - T2, max_stake_duration) / max_stake_duration`. This means that the voting power is not dependent on the current time. In other words, regardless of when you vote after unstaking, as long as your vote is cast before the proposal deadline, your voting power remains unchanged (which is hence not linear decrement).

Recommendation

The issue could come from ambiguity in the documentation or the implementation. It is therefore recommended to:

1. Update the documentation to clarify the description, or
2. Reimplement `self.future_voting_power()` in a more proper way.

Mitigation Review Log

Raccoons: Explained in PR#49.

4.4 Informational and Undetermined Issues

escrow_tokens Account not Closed upon Withdrawal

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Logic

When withdrawing from the `locked-voter`, the `escrow` account is closed, but the associated token account `escrow_tokens` is not closed, resulting in permanent inaccessibility.

Raccoons: Fixed in PR#50.

Unable to Close Invalid Transactions

Severity: Informational	Status: Acknowledged
Target: Smart Contract	Category: Logic

A transaction can only be closed when all signers unapprove it. However, this condition is too strict in two scenarios:

1. If the transaction is executed, it is burnt by setting `executed_at = unix_timestamp`, making it no longer usable.
2. If the owners of the wallet are updated, the final unapprove is rejected due to the incorrect `owner_set_seqno`.

In both cases, we are unable to close the transaction account.

5 Disclaimer

This audit report is provided for informational purposes only and is not intended to be used as investment advice. While we strive to thoroughly review and analyze the smart contracts in question, we must clarify that our services do not encompass an exhaustive security examination. Our audit aims to identify potential security vulnerabilities to the best of our ability, but it does not serve as a guarantee that the smart contracts are completely free from security risks.

We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster the security of their smart contracts.

It is important to note that the scope of our audit is limited to the areas outlined within our engagement and does not include every possible risk or vulnerability. Continuous security practices, including regular audits and monitoring, are essential for maintaining the security of smart contracts over time.

Please note: we are not liable for any security issues stemming from developer errors or misconfigurations at the time of contract deployment; we do not assume responsibility for any centralized governance risks within the project; we are not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By using this report, the client acknowledges the inherent limitations of the audit process and agrees that our firm shall not be held liable for any incidents that may occur subsequent to our engagement.

This report is considered null and void if the report (or any portion thereof) is altered in any manner.