Jupiter Lock

Security Assessment

August 15th, 2024 — Prepared by OtterSec

Nicola Vella nick0ve@osec.io

Robert Chen r@osec.io

Table of Contents

01 — Executive Summary

Overview

Jupiter engaged OtterSec to assess the `jup-lock` program. This assessment was conducted between August 2nd and August 9th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 2 findings throughout this audit engagement.

We made recommendations for the removal of redundant code for better maintainability and clarity (OS-JPL-SUG-01), and suggested modifying the codebase for improved efficiency and security (OS-JPL-SUG-00).

Scope

The source code was delivered to us in a Git repository at https://github.com/jup-ag/jup-lock. This audit was performed against commit 4560ddc.

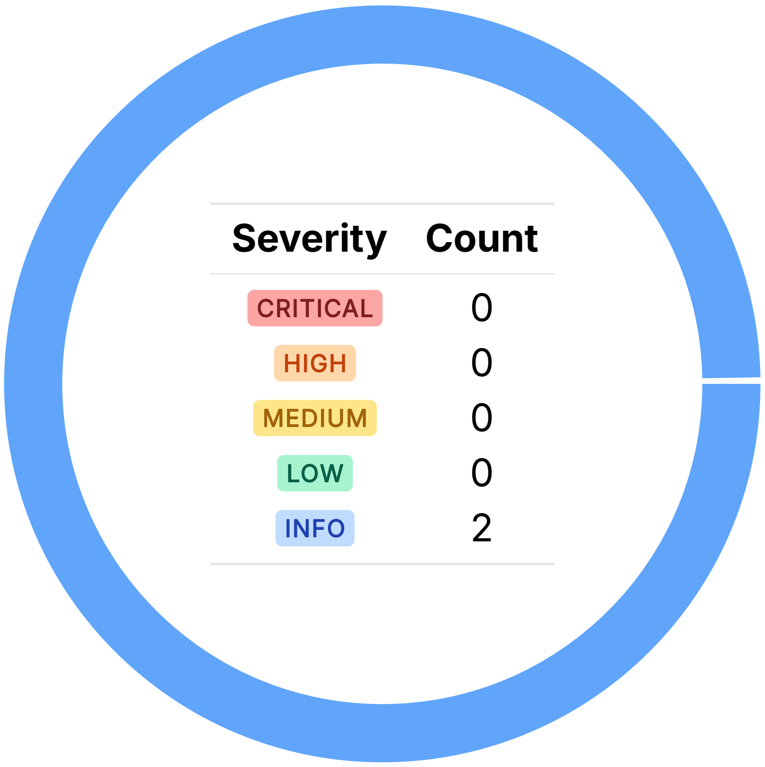
A brief description of the programs is as follows:

Name	Description
jup-lock	Open-source program to allow users to lock tokens based on a vesting plan.

02 — Findings

Overall, we reported 2 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 aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	0
HIGH	0
MEDIUM	0
LOW	0
INFO	2

03 — 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 anti-patterns and may result in security issues in the future.

ID	Description
OS-JPL-SUG-00	Recommendations for modifying the codebase for improved efficiency and adherence to coding best practices.
OS-JPL-SUG-01	Removal of redundant code for better maintainability and clarity.

Code Refactoring OS-JPL-SUG-00

Description

1. Utilize `emit_cpi` instead of `emit` in the `handle_update_vesting_escrow_recipient` for emitting the `EventUpdateVestingEscrowRecipient` event. This ensures that events are correctly logged in the context of Cross-Program Invocations, adhering to the expected format and behavior of the events.

   >_ *src/utils/system_utils.rs* RUST

   ```rust
   pub fn handle_update_vesting_escrow_recipient(
       ctx: Context<UpdateVestingEscrowRecipientCtx>,
       new_recipient: Pubkey,
       new_recipient_email: Option<String>,
   ) -> Result<()> {
       [...]
       emit_cpi!(EventUpdateVestingEscrowRecipient {
           escrow: ctx.accounts.escrow.key(),
           signer,
           old_recipient,
           new_recipient,
       });
   }
   ```

2. In `handle_create_vesting_escrow`, move the Associated Token Account (ATA) checks to `CreateVestingEscrowCtx` with Anchor macros, streamlining and simplifying the code by leveraging Anchor's built-in macros for common validation tasks.

Remediation

Implement the above-mentioned modifications.

Patch

1. Resolved in af9f413.
2. Resolved in af9f413.

Code Redundancy OS-JPL-SUG-01

Description

In the `handle_update_vesting_escrow_recipient` instruction, setting `zero_init` to true during `realloc` is not strictly necessary because the uninitialized bytes are already zeroed out by `realloc` itself.

Remediation

Remove the redundant zero initialization in the `handle_update_vesting_escrow_recipient` instruction.

Patch

Resolved in af9f413.

A — Vulnerability Rating Scale

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

CRITICAL

Vulnerabilities that immediately result in a 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 may result in a 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 may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.