



Security Assessment Report

Jupiter Locker

August 05, 2024

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Jupiter Locker smart contracts.

The artifact of the audit was the source code of the following programs, excluding tests, in a private repository.

The initial audit focused on the following versions and revealed 4 issues or questions.

program	type	commit
jup-lock	Solana	4560ddc52077673f80ee9af0542a3747fcf899c9

This report provides a detailed description of the findings and their respective resolutions.

Table of Contents

Result Overview 3

Findings in Detail 4

 [I-01] Unchecked "recipient_token" in "claim" 4

 [I-02] CPI events enabled without using "emit_cpi!" 5

 [I-03] Add instructions to update "escrow_metadata" 6

 [I-04] Initialize the "escrow_token" ATA account when needed 7

Appendix: Methodology and Scope of Work 8

Result Overview

Issue	Impact	Status
JUP-LOCK		
[I-01] Unchecked "recipient_token" in "claim"	Info	Resolved
[I-02] CPI events enabled without using "emit_cpi!"	Info	Resolved
[I-03] Add instructions to update "escrow_metadata"	Info	Resolved
[I-04] Initialize the "escrow_token" ATA account when needed	Info	Acknowledged

Findings in Detail

JUP-LOCK

[I-01] Unchecked "recipient_token" in "claim"

Since the "recipient" signs the "claim" instruction, it's legitimate for the "recipient_token" to be any token account.

```
/* jup-lock/programs/locker/src/instructions/claim.rs */
006 | pub struct ClaimCtx<'info> {
007 |     #[account(mut, has_one = recipient, has_one = escrow_token)]
008 |     pub escrow: AccountLoader<'info, Escrow>,
009 |
010 |     #[account(mut)]
011 |     pub escrow_token: Box<Account<'info, TokenAccount>>,
012 |
013 |     #[account(mut)]
014 |     pub recipient: Signer<'info>,
015 |
016 |     #[account(mut)]
017 |     pub recipient_token: Box<Account<'info, TokenAccount>>,
021 | }
```

However, if the "recipient" is mistakenly passed the "escrow_token" as the "recipient_token", even though the claimed amount is not transferred out from the "escrow_token", the "recipient" can no longer claim it.

Consider validating the "recipient_token" so that it cannot be the same as the "escrow_token".

Resolution

This issue has been resolved by commit "e1f9e33".

JUP-LOCK

[I-02] CPI events enabled without using "emit_cpi!"

In "Cargo.toml", the "event-cpi" feature of "anchor-lang" is enabled. In "create_vesting_plan" and "claim", the "#[event_cpi]" is added to both account structs.

```
/* jup-lock/programs/locker/Cargo.toml */
020 | anchor-lang = { version = "0.28.0", features = ["event-cpi"] }

/* jup-lock/programs/locker/src/instructions/create_vesting_plan.rs */
023 | #[event_cpi]
024 | #[derive(Accounts)]
025 | pub struct CreateVestingPlanCtx<'info> {

/* jup-lock/programs/locker/src/instructions/claim.rs */
004 | #[event_cpi]
005 | #[derive(Accounts)]
006 | pub struct ClaimCtx<'info> {
```

The "#[event_cpi]" annotation appends 2 extra accounts to the account structs.

```
#[account(seeds = [b"__event_authority"], bump)]
pub event_authority: AccountInfo<'info>,
pub program: AccountInfo<'info>,
```

However, both instructions invoke the "emit!" instead of "emit_cpi!".

```
/* jup-lock/programs/locker/src/instructions/create_vesting_plan.rs */
060 | pub fn handle_create_vesting_plan(
063 | ) -> Result<()> {
117 |     emit!(EventCreateVestingPlan {
126 |     });
128 | }

/* jup-lock/programs/locker/src/instructions/claim.rs */
043 | pub fn handle_claim(ctx: Context<ClaimCtx>, max_amount: u64) -> Result<()> {
064 |     emit!(EventClaim {
068 |     });
070 | }
```

Resolution

This issue has been resolved by commit "e1f9e33".

JUP-LOCK

[I -03] Add instructions to update "escrow_metadata"

The "escrow.recipient" can be updated after the "escrow_metadata" is created.

```
/* jup-lock/programs/locker/src/instructions/update_recipient.rs */
012 | pub fn handle_update_recipient(
013 |     ctx: Context<UpdateRecipientCtx>,
014 |     new_recipient: Pubkey,
015 | ) -> Result<()> {
016 |     escrow.update_recipient(new_recipient);
```

Currently, there is no instruction to update the "escrow_metadata", especially its "recipient_email" field.

```
/* jup-lock/programs/locker/src/instructions/create_escrow_metadata.rs */
039 | pub fn handle_create_escrow_metadata(
040 |     ctx: Context<CreateEscrowMetadataCtx>,
041 |     params: &CreateEscrowMetadataParameters,
042 | ) -> Result<()> {
043 |     let escrow_metadata = &mut ctx.accounts.escrow_metadata;
044 |     escrow_metadata.escrow = ctx.accounts.escrow.key();
048 |     escrow_metadata.recipient_email = params.recipient_email.clone();
049 |     Ok(())
050 | }
```

Resolution

This issue has been resolved by commit "e1f9e33".

JUP-LOCK

[I-04] Initialize the "escrow_token" ATA account when needed

In "create_vesting_plan", the "escrow" is a PDA derived from the signer's "base" account.

Before calling this instruction, the caller must create the ATA "escrow_token" for the not yet created "escrow" account.

```

/* jup-lock/programs/locker/src/instructions/create_vesting_plan.rs */
025 | pub struct CreateVestingPlanCtx<'info> {
026 |     #[account(mut)]
027 |     pub base: Signer<'info>,
028 |
029 |     #[account(
030 |         init,
031 |         seeds = [
032 |             b"escrow".as_ref(),
033 |             base.key().as_ref(),
034 |         ],
038 |     )]
039 |     pub escrow: AccountLoader<'info, Escrow>,
040 |
041 |     #[account(mut)]
042 |     pub escrow_token: Box<Account<'info, TokenAccount>>,

060 | pub fn handle_create_vesting_plan(
061 |     ctx: Context<CreateVestingPlanCtx>,
062 |     params: &CreateVestingPlanParameters,
063 | ) -> Result<()> {
080 |     let escrow_token = anchor_spl::associated_token::get_associated_token_address(
081 |         &ctx.accounts.escrow.key(),
082 |         &ctx.accounts.sender_token.mint,
083 |     );
085 |     require!(
086 |         escrow_token == ctx.accounts.escrow_token.key(),
087 |         LockerError::InvalidEscrowTokenAddress
088 |     );

```

Consider adding an "init_if_needed" option for "escrow_token" to allow for its initialization if necessary.

Resolution

The team clarified that the caller should create the ATA outside.

Appendix: Methodology and Scope of Work

The Sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in smart contract security, program analysis, testing and formal verification, performed a comprehensive manual code review, software static analysis and penetration testing.

Assisted by the Sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Sec3 (the "Company") and Raccoon Labs Pte. Ltd. (the "Client"). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

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

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, Sec3 (formerly Soteria) is a leading blockchain security company. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At Sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our [website](#) and follow us on [twitter](#).

