Sec3™

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

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 */
pub struct ClaimCtx<'info> {
    #[account(mut, has_one = recipient, has_one = escrow_token)]
    pub escrow: AccountLoader<'info, Escrow>,

    #[account(mut)]
    pub escrow_token: Box<Account<'info, TokenAccount>>,

    #[account(mut)]
    pub recipient: Signer<'info>,

    #[account(mut)]
    pub recipient_token: Box<Account<'info, TokenAccount>>,
}
```

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 */
anchor-lang = { version = "0.28.0", features = ["event-cpi"] }

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

/* jup-lock/programs/locker/src/instructions/claim.rs */
#[event_cpi]
#[derive(Accounts)]
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 */
pub fn handle_create_vesting_plan(
) -> Result<()> {
    emit!(EventCreateVestingPlan {
    });
}

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

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 */
pub fn handle_update_recipient(
    ctx: Context<UpdateRecipientCtx>,
    new_recipient: Pubkey,
) -> Result<()> {
    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 */
pub fn handle_create_escrow_metadata(
    ctx: Context<CreateEscrowMetadataCtx>,
    params: &CreateEscrowMetadataParameters,
) -> Result<()> {
    let escrow_metadata = &mut ctx.accounts.escrow_metadata;
    escrow_metadata.escrow = ctx.accounts.escrow.key();
    escrow_metadata.recipient_email = params.recipient_email.clone();
    Ok(())
}
```

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 */
pub struct CreateVestingPlanCtx<'info> {
    #[account(mut)]
    pub base: Signer<'info>,

    #[account(
        init,
        seeds = [
            b"escrow".as_ref(),
            base.key().as_ref(),
        ],
    )]
    pub escrow: AccountLoader<'info, Escrow>,

    #[account(mut)]
    pub escrow_token: Box<Account<'info, TokenAccount>>,

pub fn handle_create_vesting_plan(
    ctx: Context<CreateVestingPlanCtx>,
    params: &CreateVestingPlanParameters,
) -> Result<()> {
    let escrow_token = anchor_spl::associated_token::get_associated_token_address(
        &ctx.accounts.escrow.key(),
        &ctx.accounts.sender_token.mint,
    );
    require!(
        escrow_token == ctx.accounts.escrow_token.key(),
        LockerError::InvalidEscrowTokenAddress
    );
```

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

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.

Sec3™