code4rena

Jupiter

Smart Contract Security Assessment

Version 2.0

Audit dates: Oct 07 — Oct 09, 2024

Audited by: J4X
0xtheC0der

Contents

1. Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2. Executive Summary

2.1 About Jupiter

Jupiter is one of the largest decentralized trading platform and one of the most active governance community in crypto.

2.2 Scope

Repository	github.com/jup-ag/smart-wallet-99/tree/main
Commit Hash	5f2b486d5afef3933c65b17baa37db12729fd7e9
Mitigation Hash	https://github.com/jup-ag/smart-wallet-99/pull/9/commits/57ac08cc755c87b7da22dc5dfa4b8873e4c517d9

2.3 Audit Timeline

DATE	EVENT
Oct 07, 2024	Kick-off call
Oct 07, 2024	Audit start
Oct 09, 2024	Audit end
Oct 21, 2024	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	1
Low Risk	4
Informational	0
Total Issues	6

3. Findings Summary

ID	DESCRIPTION	STATUS

H-1	Keeper can be drained by withdrawing dust	Resolved
M-1	Potential smart wallet DoS due to accidental account closure	Acknowledged
L-1	Missing prefixes in PDA `seeds`	Acknowledged
L-2	Tx could get stuck in jito due to low tip	Acknowledged
L-3	Missing support for token2022	Acknowledged
L-4	Tokens can not be withdrawn to user PDAs	Acknowledged

4. Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] Keeper can be drained by withdrawing dust

Severity: High	Status: Resolved

Context:

- src/instructions/withdraw.rs#L19

Description:

The smart wallet allows users to interact in two ways: withdrawing funds via the `withdraw()` entry point and swapping funds via the `flash_swap_approve()` entry point. However, these entry points will not be called by the user but by the keeper. The keeper gets supplied a signature by the user, which he can use to act on his behalf. As a result, the keeper will incur the gas fees for executing the TX. When looking at the `flash_swap_approve()` function, one can see that these fees are directly charged to the user.

```
let total_fees_lamports = math::checked_add(priority_fee_lamports,
SIGNATURES_FEE)?;
require_gte!(
    ctx.accounts.vault.lamports(),
    total_fees_lamports,
    Errors::InsufficientSOLForFees
);

// Transfer SOL base fees
msg!("Transfer SOL base fees.");
system_program::transfer(
    CpiContext::new_with_signer(
        ctx.accounts.system_program.to_account_info(),
        system_program::Transfer {
            from: ctx.accounts.vault.to_account_info(),
            to: ctx.accounts.keeper.to_account_info(),
        },
        signer_seeds,
    ),
    total_fees_lamports,
```

```
)?;
```

However, if one looks at the same process in `withdraw()`, one can see that no fees are charged to the user. As a result, a malicious user can consistently withdraw dust amounts, griefing the keeper and burning all his funds.

Recommendation:

We recommend charging a fee similar to how the swap function is implemented and sending it to the keeper.

Jupiter:

Issue resolved by adding fees https://github.com/jup-ag/smart-wallet-99/pull/9

Zenith:

Verified.

4.2 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] Potential smart wallet DoS due to accidental account closure

Severity: Medium	Status: Acknowledged

Context:

- src/instructions/withdraw.rs#L72-L82
- src/instructions/flash_swap.rs#L210-L220
- src/instructions/flash_swap.rs#L228-L238

Description:

In the instances mentioned above, native SOL is transferred from the user's `vault` account. However, it is not checked if the transfer could cause the `vault`'s lamports balance to fall below the rent exemption threshold. In case this happens, Solana will automatically close the account.

Since the `vault` is one of the smart wallet's core accounts, its withdraw & swap functionality will be subject to DoS once the `vault` account is closed. Therefore, token accounts owned by the `vault` will also be inaccessible.

Recommendation:

We recommend introducing a transfer helper function with a check which assures that accounts stay rent exempt after native SOL transfers.

Jupiter:

Users are allowed to have the Sol funds to be low, but they will have issues paying for fees, this will be resolved on FE asking the users to transfer in more Sol. We also allow for vault accounts to be closed if users transfer out all their Sol funds. Since the Vault accounts belong to the System Program, they will be re-opened once the user transfers in > 0.00089088 SOL (they are unable to transfer less due to the minimum rent exemption required).

TAs owned by the vault will technically be not accessible if the vault is closed (because the vault cannot pay for fees), but the user can always reopen the vault, thus this won't be an issue.

Zenith:

Acknowledged.

4.3 Low Risk

A total of 4 low risk findings were identified.

[L-1] Missing prefixes in PDA `seeds`

Severity: Low	Status: Acknowledged

Context:

- src/instructions/initialize.rs#L30
- src/instructions/initialize.rs#L39

Description:

Whenever PDA (Program Derived Address) accounts are used in a Solana programs, it is common to prefix their `seeds` with a string describing the unique purpose of the account to avoid collisions with other PDAs. This can especially become a problem when `seeds` are fully user-specified, see first instance, and when the program is expected to be upgraded with new PDA accounts in the future.

Recommendation:

We recommend specifying a custom `seeds` prefix for all PDAs. *Example*:

```
#[account(
  seeds=[b"VAULT", user_signer[0 ..32].as_ref(),
user_signer[32..64].as_ref()],    // <---
  bump
)]
vault: SystemAccount<'info>,

#[account(
  init,
  payer=payer,
  space=VaultInfo::LEN,
  seeds=[b"VAULT_INFO", vault.key().as_ref()],    // <---
  bump
)]
vault_info: Account<'info, VaultInfo>,
```

Jupiter:

Acknowledged, will not fix currently as this requires a migration for all users

[L-2] Tx could get stuck in jito due to low tip

Severity: Low	Status: Acknowledged

Context:

- src/lib.rs#L34

Description:

The smart wallet uses jito to process transactions. Jito charges a fixed fee, and a tip is added. This is effectively the same as the solana priority fee, so the higher the tip, the faster the transaction will be executed. As the transaction works in an auction style, a transaction may get consistently ignored because others provide higher values on each auction. To mitigate this, the recommendation is to increase the tip if a transaction is stuck for too long.

However, in the current system, the user can only set the priority fee when he first sends the TX, and he can not increase the tip later if the TX gets stuck.

Recommendation:

We recommend adding functionality that allows a user to transfer an additional fee to the keeper, which will then be used to increase the tip of the corresponding bundle.

Jupiter:

Acknowledged. We expect swaps to expire within 30s (nonce expiry) if they get stuck due to low Jito fees. Thus there will not be a fix planned for this case.

[L-3] Missing support for token2022

Severity: Low	Status: Acknowledged

Context:

- src/instructions/flash_swap.rs#L15
- src/instructions/withdraw.rs#L12

Description:

The current implementation does not support tokens using the token2022 program. As a result, deposited tokens using that program will get stuck.

Recommendation:

We recommend adding support for the token2022 program.

Jupiter:

Acknowledged

[L-4] Tokens can not be withdrawn to user PDAs

Severity: Low	Status: Acknowledged

Context:

- src/instructions/withdraw.rs#L130

Description:

The `withdraw()` function can be called to withdraw a user's funds from the vault. The user must provide the `user` account in the context accounts to define the address to which he intends to withdraw.

```
#[account(mut)]
user: SystemAccount<'info>,
```

As we can see, this will only allow the passing of system-owned PDAs here. So users that want to withdraws to PDAs owned by them will not be able to do that.

Recommendation:

We recommend adapting this to allow withdrawals to any kind of account.

Jupiter:

Acknowledged.