Sec3™

Security Assessment Report

Fraction

September 15, 2025

Summary

The Sec3 team was engaged to conduct a thorough security analysis of the Fraction.

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.

Task	Type	Commit
Fraction	Solana	131180f95a5acca3e4c656c9fe34d80cb9d42ff1

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

Table of Contents

Result Overview

Issue	Impact	Status
FRACTION		
[M-01] Redundant and unusable logic for WSOL distribution	Medium	Resolved
[L-01] Participant validation should occur at configuration time	Low	Resolved
[L-02] Missing validation of participant token accounts in `claim_and_distribute`	Low	Resolved
[I-01] Ineffective length validation for `name` as PDA seed	Info	Resolved

Findings in Detail

FRACTION
[M-01] Redundant and unusable logic for WSOL distribution

The `claim_and_distribute` instruction is the core function for distributing tokens to participants. It contains a special conditional branch to handle the distribution of Wrapped SOL (WSOL), which is distinct from the logic for all other SPL tokens.

```
/* programs/fraction/src/instructions/claim_and_distribute.rs */
if self.treasury_mint.key() == WSOL_MINT {
    return self.handle_wsol_distribution(signer);
}
```

However, this special-case logic for WSOL is both redundant and inconsistent with the provided client-side SDK. The on-chain program's WSOL path requires a temporary, signer-controlled WSOL account (`temp_wsol_account`) to be provided. Conversely, the corresponding SDK function for creating this instruction unconditionally sets this account to `null`, meaning it is never passed to the program.

```
/* sdk/instructions/claim.ts */
const ix = await program.methods.claimAndDistribute().accountsStrict({
    authority: fraction.authority,
    botWallet: fraction.botWallet,
    fractionConfig: config,
    treasury: treasuryAssociatedTokenAccount,
    treasuryMint: mint,
    tempWsolAccount: null,
    botTokenAccount: botAssociatedTokenAccount,
    participantTokenAccount0: participantsAssociatedTokenAccount[0],
    participantTokenAccount1: participantsAssociatedTokenAccount[1],
    participantTokenAccount2: participantsAssociatedTokenAccount[2],
    participantTokenAccount3: participantsAssociatedTokenAccount[3],
    participantTokenAccount4: participantsAssociatedTokenAccount[4],
    tokenProgram: TOKEN_PROGRAM_ID,
    systemProgram: SystemProgram.programId,
}).instruction()
```

This discrepancy means any attempt to distribute WSOL using the provided SDK will fail due to missing accounts. To resolve this, the special handling for WSOL should be removed. WSOL should be treated like any other SPL token and distributed directly from the treasury's ATA. If the intention was to distribute native SOL, a wrap-and-unwrap flow would need to be implemented, which is not present.

Resolution

This issue has been fixed by `65cd68c`.

FRACTION

[L-01] Participant validation should occur at configuration time

The `claim_and_distribute` instruction correctly includes a safety check to prevent funds from being distributed to the System Program address (`anchor_lang::system_program::ID`), which would result in a permanent loss of those funds. This check is performed for each participant during the critical distribution phase.

```
/* programs/fraction/src/instructions/claim_and_distribute.rs */
if participant_wallet == anchor_lang::system_program::ID && share_bps > 0 {
    return Err(FractionError::SystemProgramParticipant.into());
}
```

However, from a secure design perspective, this validation is performed too late in the process. Relying solely on a check at the moment of distribution means that a `FractionConfig` can be created and exist on-chain in a misconfigured and dangerous state. If this misconfiguration is not caught before the `claim_and_distribute` function is called, the transaction will revert, potentially lock user's fund.

To create a more robust system, it is recommended to move this validation check "up a level." The same check should be added to the `initialize_fraction` and `update_fraction` instructions. By validating the participant list at the time of configuration, the program can prevent a malformed `FractionConfig` from ever being created, ensuring that all on-chain configurations are guaranteed to be valid and distributable.

Resolution

This issue has been fixed by `b6ff785`.

FRACTION

[L-02] Missing validation of participant token accounts in `claim_and_distribute`

The `claim_and_distribute` instruction is a core function, callable by a bot, designed to distribute funds from the treasury to the various participants as defined in the `FractionConfig`.

However, the instruction contains a validation flaw: it completely trusts that the token accounts provided by the bot are the correct ones. The program fails to verify that each `participant_token_account` is an ATA and the owner of each `participant_token_account` matches the corresponding participant's wallet address stored in the on-chain `fraction_config`.

A malicious or misconfigured bot can provide an arbitrary token account in place of a legitimate participant's account. When the instruction executes, the funds intended for that participant will be transferred to the incorrect or attacker-controlled account, resulting in a direct and permanent loss of funds.

As an additional layer of defense-in-depth, it is also recommended to store the `treasury_mint` address in the `FractionConfig` during initialization. The `claim_and_distribute` instruction should then also verify that the `treasury_mint` matches this stored address.

Resolution

This issue has been fixed by `b6ff785`.

FRACTION

[I-01] Ineffective length validation for `name` as PDA seed

The `initialize_fraction` instruction creates the `FractionConfig` PDA using the user-provided `name` string as one of its seeds. Solana imposes a strict 32-byte limit on the length of any individual PDA seed.

```
/* programs/fraction/src/instructions/initialize_fraction.rs */
#[account(init, payer = authority, space = 1 + FractionConfig::INIT_SPACE, seeds = [b"fraction_config", authority.key().as_ref(), name.as_ref()], bump)]
pub fraction_config: Box<Account<'info, FractionConfig>>,
```

However, the program's logic and state are configured with a conflicting and incorrect maximum length for this `name`. The `FractionConfig` struct allocates space for a name up to 50 bytes long, and the `initialize_fraction` instruction explicitly checks that the provided `name` is no longer than 50 bytes.

```
/* programs/fraction/src/states/fraction_config.rs */
pub struct FractionConfig {
    pub authority: Pubkey,
    #[max_len(50)]
    pub name: String,
...
}

impl FractionConfig {
    pub const MAX_NAME_LENGTH: usize = 50;
}

/* programs/fraction/src/instructions/initialize_fraction.rs */
require!(
    name.len() <= FractionConfig::MAX_NAME_LENGTH,
    FractionError::NameTooLong
);
```

The on-chain length check is therefore misleading and ineffective. This also results in wasted on-chain space, as rent is paid for 50 bytes of string data when only 32 can ever be used.

Resolution

This issue has been fixed by `b6ff785` and `65cd68c`.

Appendix: Methodology and Scope of Work

Assisted by the Sec3 Scanner developed in-house, the manual audit 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 SendAI Inc. (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

The Sec3 audit team comprises a group of computer science professors, researchers, and industry veterans with extensive experience in smart contract security, program analysis, testing, and formal verification. We are also building automated 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™