Frank castle

Bio Audit

April 2025

Bio_Final_report

Content

About Frank Castle 🦀

Frank Castle is a professional smart contract security researcher with a focused expertise in auditing Rust-based contracts and decentralized infrastructure across leading blockchain ecosystems, including Solana , Polkadot , and Cosmos (CosmWasm).🦀

Frank Castle has audited Lido, GMX ,Pump.fun, LayerZero, Synthetix , Hydration, something.cool ,DUB Social and several multi-million protocols.

with more than ~35 Rust Audit , ~25 Solana Audits , and +200 criticals/highs found , All the reports can be found here

For private audit or consulting requests please reach out to me via Telegram @castle_chain , **Twitter** (@0xfrank_auditor) or **Discord** (castle_chain).

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Summary of Findings

ID	Title	Severity	Status
[M-01]	Unrestricted token selling may cause sale to fail	Medium	Resolved
[L-01]	Lack of `init_if_needed` support for Token2022 in Anchor	Low	Resolved
[L-02]	Missing decimals check in deposit token handler	Low	Resolved

Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Audit Scope

All the programs in `boop-solana/programs` :

- boop
- fee_spiltter
- reward_pool
- sboop

Commit Hash : https://github.com/Boop-fun/boop-solana/commit/b7f7533ad783c1a489e4977e013f18ece4bca277

Findings

1. Medium Findings

1.1 Unrestricted token selling may cause sale to fail

Severity

Impact: Medium

Likelihood: Medium

Description

Any buyer can sell their tokens just before the sale ends, causing the total revenue to fall below the minimum threshold required for a successful token sale. This is possible because the availability of the sell function is only restricted by the time range, as shown in the code below:

```
let curr_time = Clock::get()?.unix_timestamp;
```

```
require!(
    curr_time >= token_stats.start_time,
    DesciLaunchpadError::InvalidStartTime
);

require!(
    curr_time <= token_stats.end_time,
    DesciLaunchpadError::InvalidEndTime
);
```

There are no additional checks to prevent selling if it breaches the minimum required threshold, making the sale susceptible to last-minute withdrawals that can sabotage the sale.

Recommendations

Introduce a check to block selling transactions that would cause the total raised amount to fall below the minimum threshold. Only allow sell trades if the resulting amount still satisfies the minimum threshold condition after the transaction.

2. Low Findings

2.1 Lack of `init_if_needed` support for Token2022 in Anchor

Anchor currently does not support the `init_if_needed` attribute for Token2022 accounts in the same way it does for normal tokens. This oversight requires developers to explicitly specify the `token_program` as `token2022_program` in their account attributes, even though the mint account's `toAccountInfo` should inherently provide this information.

Reference: https://github.com/solana-foundation/anchor/issues/3498.

Recommendation:

Consider the during the integration.

2.2 Missing decimals check in deposit token handler

The `deposit_token_handler` function in the program does not verify if the `args.decimals` value is within an acceptable range before proceeding. This oversight can lead to the creation of tokens with an invalid or unintended number of decimals, potentially causing issues with token calculations and user interactions. Ensuring that the decimals are within a predefined maximum limit is crucial for maintaining consistency and preventing errors in token operations.

```
    let min_threshold = (((args.sale_supply as f64) / (10u64.pow(decimals as u32) as f64))
```

```
        * (args.price_per_token as f64)) as u64;

    if args.max_threshold.is_some() {
        require!(
            min_threshold <= args.max_threshold.unwrap(),
            DesciLaunchpadError::InvalidThreshold
        );
    }

    ctx.accounts.stats.tokens_created += 1;

    let token_stats = &mut ctx.accounts.token_stats;

    token_stats.token_id = ctx.accounts.stats.tokens_created;
    token_stats.token_mint = ctx.accounts.token_mint.key();
    token_stats.decimals = decimals;
    token_stats.payment_token = args.payment_token;
    token_stats.sale_supply = args.sale_supply;
    token_stats.limit_per_wallet = args.limit_per_wallet;
    token_stats.price_per_token = args.price_per_token;
    token_stats.start_time = args.start_time;
    token_stats.end_time = args.end_time;
    token_stats.cooldown_duration = args.cooldown_duration;
    token_stats.min_threshold = min_threshold;
    token_stats.max_threshold = args.max_threshold;
    token_stats.sell_enabled = args.sell_enabled;
    token_stats.bump = ctx.bumps.token_stats;

    spl_token_2022::onchain::invoke_transfer_checked(
        ctx.accounts.token_program.key,
        ctx.accounts.authority_token.to_account_info(),
        ctx.accounts.token_mint.to_account_info(),
        ctx.accounts.stats_token.to_account_info(),
        ctx.accounts.authority.to_account_info(),
        ctx.remaining_accounts,
        token_stats.sale_supply,
        decimals,
        &[],
    )?;

    Ok(())
```

Recommendations: Implement a check for the `args.decimals` value within the `deposit_token_handler` function. (`MAX_DECIMALS`)