code4rena

Legion

Smart Contract Security Assessment

Version 2.0

Audit dates: Sep 17 — Oct 01, 2024

Audited by: Jakub Heba

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 Legion

We’re changing the way the world communicates online The goal of Legion is to create a network where anyone can freely chat and socialize without compromising their privacy, using the hashgraph consensus.

2.2 Scope

Repository	Legion-Team/solana-contracts
Commit Hash	4685ceb41c671835f9a90a2882dad60f00a060e7
Mitigation Hash	beac737d0852c2b20b1dc8f08d5ee3c94e955273

2.3 Audit Timeline

DATE	EVENT
Sep 17, 2024	Audit start
Oct 01, 2024	Audit end
Oct 29, 2024	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	1
High Risk	1
Medium Risk	1
Low Risk	3
Informational	3
Total Issues	9

3. Findings Summary

ID	DESCRIPTION	STATUS
C-1	Lack of validation in `_claim_tokens_from_vault` allows claiming tokens from incorrect sale event	Resolved

H-1	Bypass in `deposit_funds_to_user_account` allows deposits after sale closure, leading to exploitation	Resolved
M-1	Vesting options can be updated after sale ends, leading to unfairness for participants	Resolved
L-1	Centralized control of token sale event status	Acknowledged
L-2	Risk of unfair token sale, where vault may lack tokens for user claims	Acknowledged
L-3	Modifying token mint and total supply after vault is funded and claim process starts can make user funds unclaimable	Resolved
I-1	Recommendation to use Anchor events Instead of `msg!` for logging	Resolved
I-2	Redundant and incorrect check for global configuration initialization	Resolved
I-3	Lack of support for Fee on Transfer tokens results in Incorrect accounting and refund failures	Acknowledged

4. Findings

4.1 Critical Risk

A total of 1 critical risk findings were identified.

[C-1] Lack of validation in `_claim_tokens_from_vault` allows claiming tokens from incorrect sale event

Severity: Critical	Status: Resolved

Context:

- claim_tokens_from_vault.rs

Description: In the `_claim_tokens_from_vault` function, there is no validation to ensure that the `user_account` belongs to the correct `token_sale_event`. This oversight allows users to claim tokens from a different sale event than the one they participated in, leading to potential exploitation. Malicious users could invest in a lower-value token sale (e.g., a cheap token) but claim tokens from a higher-value sale (e.g., an expensive token) during the claim phase, creating a significant unfair advantage.

```
pub fn _claim_tokens_from_vault(ctx: Context<ClaimTokensFromVault>, id:
u8) -> Result<()> {
    msg!(
        "LOG: Claim tokens from vault with user account status2: {}",
        ctx.accounts.user_account.status.to_string()
    );

    // Validate if the payer is an authorized entity (authority,
secondary authority, or user account creator)
    require!(
        ctx.accounts
            .global_config
            .is_allowed_authority(ctx.accounts.payer.key())
            || ctx.accounts.user_account.creator ==
ctx.accounts.payer.key(),
        TokenSaleError::VaultClaimCanOnlyBeDoneByAuthorityOrUser
    );
```

Exploit Scenario:

1. A user participates in a token sale for a CHEAP token and receives 5% of the total supply.
2. During the claim process, instead of providing the correct `token_sale_event` for the CHEAP token, the user supplies the `token_sale_event` for a more valuable PRICY token.
3. The absence of validation allows the user to claim PRICY tokens instead of the CHEAP tokens they initially invested in, creating an unfair outcome and potentially defrauding the token sale system.

Recommendation: Implement a validation in `_claim_tokens_from_vault` to ensure that the `user_account` belongs to the correct `token_sale_event` before allowing token claims.

Legion: Fixed with the following commit

Zenith: Verified

4.2 High Risk

A total of 1 high risk findings were identified.

[H-1] Bypass in `deposit_funds_to_user_account` allows deposits after sale closure, leading to exploitation

Severity: High	Status: Resolved

Context:

- deposit_funds_to_user_account.rs

Description: In the `deposit_funds_to_user_account` instruction, there is no check to ensure that the `token_sale_event` matches the `user_token_account`. As a result, the following check can be bypassed:

```
require!(
    ctx.accounts.user_account.status.is_open(),
    TokenSaleError::UserAccountStatusMustBeOpenForDeposit
);
```

This oversight allows malicious users to exploit the token sale process by depositing funds even after the sale status has changed, such as when the sale event is already in the claim phase. This can lead to users unfairly benefiting from the sale without locking their funds during the appropriate pre-sale or sale periods. Exploit Scenario:

1. A user interested in participating in the token sale but not willing to lock their deposit can wait until after the Token Generation Event (TGE).
2. Once the TGE occurs and the token's success is confirmed, the malicious user can bypass the check on the `token_sale_event` by directly depositing their tokens, even though the sale is in the claim phase.
3. The user can then immediately claim the tokens without having participated during the valid sale period, effectively exploiting the program logic.

It's worth mentioning that the downside of this attack is that users have to wait longer for vesting and cliff, as time starts on deposit funds, but this attack still breaks the expected user flow.

Recommendation: Add Token Sale Event Validation: Implement a check in the `deposit_funds_to_user_account` instruction to ensure that the `token_sale_event` matches the `user_token_account`. This will ensure that users can only deposit funds during the appropriate phase of the sale.

Legion: Fixed with the following commit

Zenith: Verified

4.3 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] Vesting options can be updated after sale ends, leading to unfairness for participants

Severity: Medium	Status: Resolved

Context:

- update_token_sale_event_vesting_options.rs

Description: The current implementation allows the vesting options of a token sale event to be modified and updated at any time, even after the sale has ended. This creates a significant issue for participants, as it can lead to unexpected changes in token distribution timelines, negatively affecting their expectations and investment decisions. For example, malicious actors could alter vesting conditions post-sale to delay token distribution or reduce the amount of tokens that participants receive, which is inherently unfair.

```
pub fn _update_token_sale_event_vesting_options(
    ctx: Context<UpdateTokenSaleEventVestingOptions>,
    vesting_options: TokenSaleEventVestingOptions,
) -> Result<()> {
    msg!("LOG: Setting token sale event vesting options");

    // Validate if the authority is allowed to update the vesting options
```

Recommendation: Add vesting options in the same transaction as token sale event creation, so users are aware what are the terms of the sale.

Legion: Fixed with the following commit

Zenith: Verifeid

4.4 Low Risk

A total of 3 low risk findings were identified.

[L-1] Centralized control of token sale event status

Severity: Low	Status: Acknowledged

Context:

- update_token_sale_event_status.rs

Description: The admin has the ability to change the token sale event status to "claim" at any time, or potentially never set it at all, using the `update_token_sale_event_status` instruction. This creates a centralized control point, allowing the admin to manipulate the sale process, which can lead to unfairness and a lack of trust in the system.

```
pub fn _update_token_sale_event_status(
    ctx: Context<UpdateTokenSaleEventStatusOptions>,
    new_status: TokenSaleEventStatus,
) -> Result<()> {
    msg!("LOG: Setting token sale event status");

    // Validate if the authority is allowed to update the status
    require!(
        ctx.accounts
            .global_config
            .is_allowed_authority(ctx.accounts.authority.key()),
        TokenSaleError::TokenSaleEventCanOnlyBeUpdatedByAuthority
    );

    // Validate that the status update is valid
    ctx.accounts
        .token_sale_event
        .status
        .validate_status_update(&new_status)?;

    // Apply the new status to the token sale event
    ctx.accounts.token_sale_event.status = new_status;

    Ok(())
}
```

Recommendation: Implement a decentralized or automated mechanism to handle the status transitions based on predefined conditions or timelines, reducing admin control and ensuring fairness.

Legion: Acknowledged

[L-2] Risk of unfair token sale, where vault may lack tokens for user claims

Severity: Low	Status: Acknowledged

Context:

- lib.rs

Description: There is a potential risk in the token sale process where a user invests in the sale, but when they attempt to claim their tokens, the vault may be empty. Tokens are transferred to the vault by the admin using the `_deposit_tokens_to_vault` instruction, but can also be withdrawn at any time using the `_emergency_withdraw_tokens_from_vault` instruction. This creates a situation where users might not receive their purchased tokens, leading to an unfair and unreliable token sale process.

```
pub fn emergency_withdraw_tokens_from_vault(
    ctx: Context<EmergencyWithdrawTokensFromVault>,
    id: u8,
) -> Result<()> {
    _emergency_withdraw_tokens_from_vault(ctx, id)
}
```

Recommendation: Ensure that tokens are locked in the vault during and after the sale until all user claims are fulfilled, preventing emergency withdrawals. Additionally, require that a sufficient amount of tokens is deposited into the vault before the sale begins to guarantee user claims.

Legion: Acknowledged

[L-3] Modifying token mint and total supply after vault is funded and claim process starts can make user funds unclaimable

Severity: Low	Status: Resolved

Context:

- update_token_sale_event_token_info.rs

Description: Currently, the token mint and total supply can be modified at any time using the `_update_token_sale_event_token_info` function. This creates a risk when the token sale event vault is funded and the claim process has begun. If the authority changes the token mint after funding, it can result in a mismatch between the `token_sale_event` mint and the vault mint, rendering users' funds unclaimable.

```
pub fn _update_token_sale_event_token_info(
    ctx: Context<UpdateTokenSaleEventTokenInfo>,
    total_supply: u64,
) -> Result<()> {
    msg!("LOG: Setting token sale event token info");

    // Validate if the authority is allowed to update the token
information
    require!(
        ctx.accounts
            .global_config
            .is_allowed_authority(ctx.accounts.authority.key()),
        TokenSaleError::TokenSaleEventCanOnlyBeUpdatedByAuthority
    );

    // Update the token mint address and total supply in the token sale
event
    ctx.accounts.token_sale_event.token_info.mint =
*&ctx.accounts.token_mint.key();
    ctx.accounts.token_sale_event.token_info.total_supply = total_supply;

    Ok(())
```

Issue Scenario:

1. The authority sets the token mint and total supply, and users deposit funds into the sale event vault.

2. Once the vault is funded and the claim process begins, the authority modifies the token mint.
3. This results in a mismatch between the vault mint and the `token_sale_event` mint, preventing users from claiming their tokens, as the expected mint no longer matches the vault mint.

Recommendation: Prevent any modifications to the token mint and total supply after the vault is funded and the claim process has started, ensuring consistency between the sale event and the vault and allowing users to claim their funds without issue.

Legion: Fixed with the following commit

Zenith: Verified

4.5 Informational

A total of 3 informational findings were identified.

[I-1] Recommendation to use Anchor events Instead of `msg!` for logging

Severity: Informational	Status: Resolved

Context:

- add_token_sale_event.rs

Description: Currently, the program uses the `msg!` macro to log messages during execution. While `msg!` is useful for debugging, it is not optimal for structured logging or events that can be consumed by external systems. In contrast, Anchor provides an event system that is more efficient and structured for logging, which can be easily accessed by off-chain services and explorers.

```
// - The payer is not an authorized entity (authority or secondary
authority).
pub fn _add_token_sale_event(ctx: Context<AddTokenSaleEvent>, event_id:
u64) -> Result<()> {
    msg!("LOG: Add token sale event");
```

Using Anchor events instead of `msg!` offers several advantages:

- Structured Data - Events allow logging of structured data, making it easier for external tools to process the information.
- Efficient Logging - Events are more efficient than raw messages because they are indexed and can be filtered, unlike the generic msg! output.
- Enhanced Visibility - On-chain explorers and tools like Solana's transaction history can access events more easily, improving visibility into important actions like deposits, withdrawals, and state changes.

Recommendation: Use Anchor events instead of `msg!`.

Legion: Fixed with the following commit

Zenith: Verified

[I-2] Redundant and incorrect check for global configuration initialization

Severity: Informational	Status: Resolved

Context:

- lib.rs

Description: In the `initialize_global_config` function, the following check is used to validate whether the global configuration has been initialized:

```
require!(
    account_info.owner == &crate::ID,
    TokenSaleError::GlobalConfigAlreadyExists
);
```

This check is not only redundant but also incorrect. Since the Anchor framework uses the `init` attribute, it already ensures that the account is uninitialized before proceeding. Therefore, this check adds no value and could lead to confusion. Furthermore, checking if the account is owned by the program ID (`&crate::ID`) is not a proper way to confirm initialization status, as PDAs are always owned by the program.

Problems:

- Redundancy - Anchor's init attribute automatically verifies that the account is uninitialized, so this check is unnecessary.
- Incorrect Validation - The ownership check does not reliably determine whether the global configuration has already been initialized. The correct approach would be to check if the authority field is set to a default, uninitialized state (such as `Pubkey::default()`).

Recommendation:

Remove Redundant Check: Eliminate the following line, as it is both redundant and incorrectly used:

```
require!(
    account_info.owner == &crate::ID,
    TokenSaleError::GlobalConfigAlreadyExists
);
```

Proper Validation for Initialization - If additional validation is desired (though not needed with init), the correct check would involve verifying if the authority is set to its default, uninitialized state. This would confirm whether the configuration is already set:

```
require!(
    global_config.authority == Pubkey::default(),
    TokenSaleError::GlobalConfigAlreadyExists
);
```

This check ensures that the account has not been previously initialized without relying on faulty ownership checks, providing accurate validation of the initialization status. However, if `init` is properly used, this additional check may still be unnecessary.

Legion: Fixed with the following commit

Zenith: Verified

[I-3] Lack of support for Fee on Transfer tokens results in Incorrect accounting and refund failures

Severity: Informational	Status: Acknowledged

Context:

- process_token_transfer.rs

Description: The current system does not support fee on transfer tokens, such as SPL22, but it still allows the creation of token sale events with these types of tokens. This leads to incorrect accounting and issues during deposit and refund operations. Here's a potential scenario that demonstrates the issue:

1. A token with a 5% transfer fee is used.
2. A user deposits 100 tokens into the token sale event. Due to the 5% fee on transfer, only 95 tokens are actually transferred to the user's token account in the sale event, while 5 tokens are lost to the fee.
3. The program still assumes 100 tokens are deposited in its internal accounting.
4. When the user attempts to refund or transfer tokens to the treasury, the program requires the full 100 tokens to be refunded. However, since only 95 tokens are available in the account (due to the transfer fee), the refund transaction fails, and any subsequent transfers to the treasury also fail, leading to inconsistencies and transaction failures.

```
pub fn process_token_transfer<'info>(
    _payer: &AccountInfo<'info>,
    from_token_account: &AccountInfo<'info>,
    to_token_account: &AccountInfo<'info>,
    mint: &InterfaceAccount<'info, Mint>,
    authority: &AccountInfo<'info>,
    token_program: &Interface<'info, TokenInterface>,
    amount: u64,
    bump_seeds: &[&[&[u8]]],
) -> Result<()> {
    // calculate total amount with decimals
```

This issue can cause serious problems with user experience and program functionality, particularly in refund or transfer scenarios.

Recommendation:

1. Add support for fee on transfer tokens - Update the program to recognize fee on transfer tokens and adjust internal accounting accordingly. The system should track the actual number of tokens transferred and account for any transfer fees deducted during deposits.
2. Validate token type - Before allowing the creation of a token sale event, the program should validate whether the token being used has a transfer fee and either prevent its use or handle the fee correctly.
3. Recalculate refund amounts - Ensure that the refund process is adjusted to account for any transfer fees already deducted from deposits so that users are only required to refund the net deposited amount.

Legion: Acknowledged