0xCommit

Security Audit Report

SOEX - Solana Programs

Treasury , OG and Premium CVT Programs

Version: Final

Date: 2nd November 2024

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL LICENSE.

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

Introduction

Purpose of this report

0xCommit has been engaged by **SOEX - Solana Programs** to perform a security audit of several Solana Programs components.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine solana program bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebases Submitted for the Audit

The audit has been performed on the following GitHub repositories:

Version	List of programs	Source
1	Treasury , OG and Premium CVT	https://gitlab.dcircle.io/dcchain/soex_solana_x/-/tree/master/programs

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Overview

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Summary of Findings

Sr. No.	Program	Description	Severity	Status
1	Treasury	Handle_Receive_Sol function does not check for sol_mint with global config	High	Resolv...
2	Treasury	No Mechanism to change config	Medium	Resolv...
3	Treasury	Segregate call into two functions	Medium	Resolv...
4	Treasury	Nonce is not used from config, rather a separate instance of it is created.	Low	Resolv...
5	Treasury	Open is not used	Low	Resolv...
6	OG	No nonce is present which leads the ED25519 signature to be reusable	High	Resolv...
7	OG	Missing check if NFT ID is already minted or not. (Tentative)	High	Resolv...
8	OG	No Mechanism to change collectionAmount	Medium	Resolv...
9	OG	Missing checks for max copies in set_max_supply function	Low	Resolv...
10	OG	Incomplete Function	Low	Resolv...

Sr. No.	Program	Description	Severity	Status
11	Premium_cvt	No Mechanism to change collectionAmount struct	Low	Resolv...
12	Premium_cvt	Utility phase_supply_over can be changed to boolean	Low	Ackno...
13	Premium_cvt	No nonce is present which leads the ED25519 signature to be reusable	Medium	Resolv...
14	Premium_cvt	Unused Variable in handle_mint	Low	Resolv...

Detailed Findings

1. Receive sol does not check for sol_mint with global config

Severity: High

Program - Treasury

Description

For Receive_sol struct in used for handle_receive_sol function there is missing cross check with global_config like it is done in the case of soex_mint.

```
45      #[account(mut)] // Security issue here
46      pub sol_mint: Box<Account<'info, Mint>>,
47
48      #[account(mut,address = global_config.load().unwrap().soex_mint)]
49      pub soex_mint: Box<Account<'info, Mint>>,
```

Also in global_config initialized during initialization does not factor in sol_mint.

```
#[account(zero_copy(unsafe))]
#[repr(C)]
pub struct Config {
    pub manager: Pubkey,     // Admin of the program
    pub soex_mint: Pubkey,   // Account for recver staking fee
    pub fee_account: Pubkey, // Account for recver staking fee
    pub oracle: Pubkey,
    pub nonce: u64, // not used here
    pub open: bool,
} // Missing sol_mint
```

Due to this issue sol_mint parameters can be any arbitrary key.

Remediation

Add sol_mint as a parameter during the initialization process and add sol_mint as derived from global_config in receive_sol struct.

Status

Resolved

2. No mechanism to change config

Severity: Medium

Program - Treasury

Description

The config parameters once set during initialization cant be edited. This functionality is needed for stability of the treasury programs. As parameters can change over the life cycle of the program.

Following parameters in config may require alteration over time.

```
#[account(zero_copy(unsafe))]
#[repr(C)]
pub struct Config {
    pub manager: Pubkey,     // May be required to change over time
    pub soex_mint: Pubkey,   // May be required to change over time
    pub fee_account: Pubkey, // May be required to change over time
    pub oracle: Pubkey,      // May be required to change over time
    pub nonce: u64,
    pub open: bool,
}
```

Remediation

Add functions which allow modification of config. (Note - These changes are subject to admin control.)

Status

Resolved

3. Segregate calls into two separate functions.

Severity: Medium

Program - Treasury

Description

Receive_sol function calls handle_receive_sol which has two flows one for handing sol and other for SOEX both function call transfer_token_from_pool which is identical in nature it would be better if both functions are segregated leading to better administrative control over the system and removes the risk of accidental calls.

Remediation

Segregate receive_sol function into two distinct functions.

Status

Resolved

4. Nonce is not used from config, rather a separate instance of it is created.

Severity: Low

Program - Treasury

Description

The Config struct set during the initialization phase has nonce but that nonce is not used in receive_sol function but rather a separate struct is created in receive_sol which handles nonce. There are two implementation of nonce while only one is needed to handle nonce across the program.

```
37  #[account(zero_copy(unsafe))]
38  #[repr(C)]
39  pub struct Config {
40      pub manager: Pubkey,
41      pub soex_mint: Pubkey,
42      pub fee_account: Pubkey,
43      pub oracle: Pubkey,
44      pub nonce: u64, // not used here
45      pub open: bool,
46  }
```

Nonce implementation in state/init.rs

```
#[account]
pub struct AccountManager {
    pub nonce: u64,
}
```

Nonce implementation in state/receive_sol.rs

Remediation

The nonce implementation in state/receive_sol.rs is redundant, rather nonce from config should only be used for operation.

Status

Resolved

5. Open is not used

Severity: Low **Program - Treasury**

Description

During initialization of the program there is a boolean value called open setup, The intended use of the this function seems to be to enable pausing and unpausing functionality but it is not used any where in the program.

Remediation

Either remove the boolean open from config or add pausing functionality. If pausing functionality is added then there will be a need to add admin controlled, enable pause and disable pause functionality.

Status

Resolved

6. No nonce is present which leads the ED25519 signature to be reusable.

Severity: High

Program - OG

Description

In the function handle_mint the hash used for ED25519 signature does not use any nonce while generating the hash. This can lead to repudiation attack by virtue of reuse of signature.

```
let mut msg : Vec<u8> = vec![];
msg.extend(ctx.accounts.payer.key().to_bytes());
msg.extend(nft_id.to_le_bytes());
msg.extend(copies.to_le_bytes());
let hash : [u8; 32] = keccak::hash(&msg).to_bytes();
msg!("hash {}", Pubkey::new_from_array(hash));
let ix: Instruction = load_instruction_at_checked( index: 2, &ctx.accounts.ix_sysvar)?;
utils::verify_ed25519_ix(&ix, &collection.creater.to_bytes(), &hash, &signature)?;
```

Remediation

Implement a nonce for each ED25519 signature used in the program. Note nonce should be auto incrementing in nature and nonce should be set to 0 during the initialization phase.

Status

Resolved

7. Missing check if NFT ID is already minted or not. (Tentative)

Severity: High **Program - OG**

Description

In the function handle_mint there is no check to see if the nft id is already minted or not.

Remediation

Maintain a list of NFT ID minted and if the NFT ID is minted then program should throw error.

Status

Resolved

(Note - This is tentative issue need to dig deeper into this for full closure)

8. No mechanism to change Collection amount

Severity: Medium

Program - OG

Description

The Collection Amount parameters once set during initialization cant be edited. This functionality is needed for stability of the OG programs. As parameters can change over the life cycle of the program.

Following parameters in config may require alteration over time.

```
22  #[account(zero_copy(unsafe))]
23  #[repr(C)]
24  pub struct CollectionAccount {
25      pub admin: Pubkey,          // May be required to change over time
26      pub creater: Pubkey,        // May be required to change over time
27      pub oracle: Pubkey,         // May be required to change over time
28      pub current_supply: u32,
29      pub max_supply: u32,
30      pub max_copies: u32,
31      pub current_copies: u32,
32  }
```

Remediation

Add functions which allow modification of config. (Note - These changes are subject to admin control.)

Status

Resolved

8. Missing checks for max copies in set_max_supply function

Severity: Medium

Program - OG

Description

In the function set_max_supply there are checks for max_supplies but there seems to be need for max_copies also in place.

```
50      pub fn set_max_supply(
51          ctx: Context<SetMaxSupply>,
52          new_max_supply: u32,
53          new_max_copies: u32,
54      ) -> Result<()> {
55          let collection : &mut RefMut<CollectionAccount> = &mut ctx.accounts.collection.load_mut()?;
56          require!(
57              collection.admin == ctx.accounts.payer.key(),
58              ErrorCode::InvalidAdmin
59          );
60          // Security - There seems to be missing function to check for max_copies in here
61          assert!(new_max_supply >= collection.current_supply);
62          collection.max_supply = new_max_supply;
63          collection.max_copies = new_max_copies;
64
65          Ok(())
66      }
```

Remediation

Add a assert to check perform max_copies if needed.

Status

Resolved

9. Changed max copies and max supply should be greater than existing values in set_max_supply function

Severity: Low **Program - OG**

Description

In the function set_max_supply the changes to the new_max_supplies and new_max_copies must be greater than already set values, currently that check is mising. As this can cause economics issues while interfacing with other contracts.

```
50      pub fn set_max_supply(
51          ctx: Context<SetMaxSupply>,
52          new_max_supply: u32,
53          new_max_copies: u32,
54      ) -> Result<()> {
55          let collection : &mut RefMut<CollectionAccount> = &mut ctx.accounts.collection.load_mut()?;
56          require!(
57              collection.admin == ctx.accounts.payer.key(),
58              ErrorCode::InvalidAdmin
59          );
60          assert!(new_max_supply >= collection.current_supply);
61          collection.max_supply = new_max_supply; // Add checks to ensure new_max_supply is greater than exisiting values.
62          collection.max_copies = new_max_copies; // Add checks to ensure new_max_supply is greater than exisiting values.`
63
64          Ok(())
65      }
```

Remediation

Add checks for following

- new_max_supply must be greater than existing set values for max_supply.
- new_max_copies must be greater than existing set values for max_copies.

Status

Resolved

10. Incomplete function

Severity: Low **Program - OG**

Description

In the function init_copies seems to be incomplete as it does not change any state of the contract.

```
36      pub fn init_copies(_ctx: Context<InitCopies>, nft_id: u64) -> Result<()> {
37          msg!("Initializing copies for NFT ID: {}", nft_id);
38          Ok(())
39      }
```

Remediation

Remove the function if it does not serve any purpose.

Status

Resolved

11. No mechanism to change certain variables in Collection amount

Severity: Low

Program - Premium_CVT

The Collection Amount parameters once set during initialization cant be edited. This functionality is needed for stability of the Premium_cvt program. As parameters can change over the life cycle of the program.

Following parameters in config may require alteration over time.

```
pub struct CollectionAccount {
    pub admin: Pubkey,
    pub creater: Pubkey,        // Will Require changes over time
    pub oracle: Pubkey,         // Will Require changes over time
    pub ids: [u8; 8192],
    pub max_supply: u32,        // Will Require changes over time
    pub current_supply: u32,
    pub price: u64,             // Will Require changes over time
    pub phase_supply_start_time: [u64;3],
    pub phase_supply_quit_deadline: [u64;3],
    pub phase_supply_max_supply: [u32;3],
    pub phase_supply_current_supply:[u32;3],
    pub phase_supply_over:[u32;3],
    pub current_phase: u32,
    pub authority:u32,
    pub lock_max_nft_id: u32,
    pub lock_mint_amount:u32,
    pub max_supply_added:u32,
}
```

Remediation

Add functions which allow modification of config. (Note - These changes must be subject to admin control)

Status

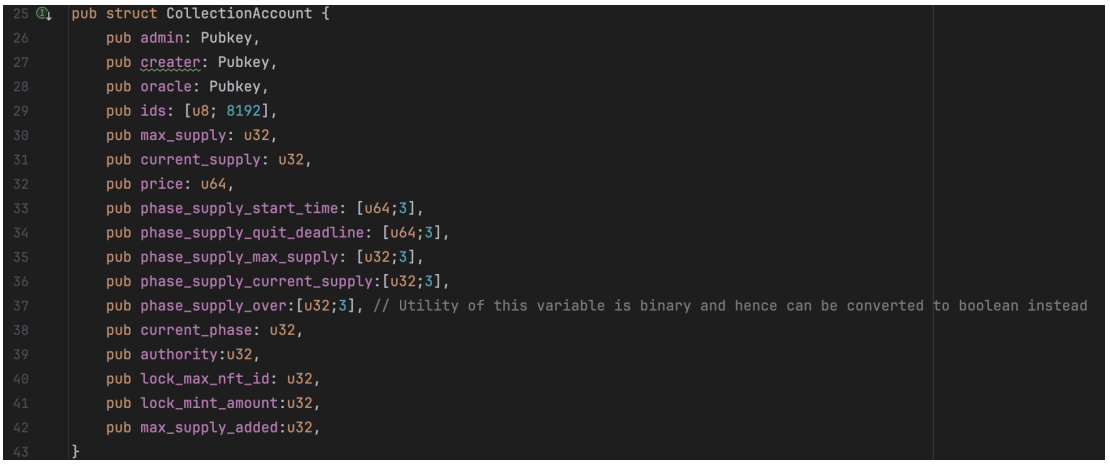
Resolved

12. Utility phase_supply_over can be changed to boolean

Severity: Low

Program - Premium_CVT

In CollectionAmount struct, variable phase_supply_over is binary and hence it can be converted to boolean without any problem currently it is set as u32 which takes far more greater space than needed.



```
pub struct CollectionAccount {
    pub admin: Pubkey,
    pub creater: Pubkey,
    pub oracle: Pubkey,
    pub ids: [u8; 8192],
    pub max_supply: u32,
    pub current_supply: u32,
    pub price: u64,
    pub phase_supply_start_time: [u64;3],
    pub phase_supply_quit_deadline: [u64;3],
    pub phase_supply_max_supply: [u32;3],
    pub phase_supply_current_supply:[u32;3],
    pub phase_supply_over:[u32;3], // Utility of this variable is binary and hence can be converted to boolean instead
    pub current_phase: u32,
    pub authority:u32,
    pub lock_max_nft_id: u32,
    pub lock_mint_amount:u32,
    pub max_supply_added:u32,
}
```

Remediation

Change the variable datatype to boolean from u32.

Status

Acknowledged

13. No nonce is present which leads the ED25519 signature to be reusable.

Severity: Medium

Program - Premium_CVT

Description

In the function handle_mint the hash used for ED25519 signature does not use any nonce while generating the hash. This can lead to repudiation attack by virtue of reuse of signature.

```
34      let mut msg : Vec<u8> = vec![];
35      msg.extend(ctx.accounts.payer.key().to_bytes());
36      msg.extend(nft_id.to_le_bytes());
37      msg.extend(collection.current_phase.to_le_bytes());
38      msg.extend(user_storage_index.to_le_bytes());
39
40      let hash : [u8; 32] = keccak::hash(&msg).to_bytes();
41      msg!("hash {}", Pubkey::new_from_array(hash));
42
43      let ix: Instruction = load_instruction_at_checked( index: 2, &ctx.accounts.ix_sysvar)?;
44      utils::verify_ed25519_ix(&ix, &collection.creater.to_bytes(), &hash, &signature)?;
```

This can be a issue as same collection if created twice can be closed using older signature.

Remediation

Implement a nonce for each ED25519 signature used in the program. Note nonce should be auto incrementing in nature and nonce should be set to 0 during the initialization phase.

Status

Resolved

14. Unused Variable in handle_Mint

Severity: Low **Program - Premium_CVT**

Description

In the function handle_mint the variable “user_storage_index” is not used functionally in the program anywhere.

Remediation

Remove the variable if it does not serve any purpose.

Status

Resolved