Cyberscope

Audit Report

MOMOAI

April 2024

SHA256 4dbfbf32dd954ecd43995ac837c2fafc5adaae77f72b5a1ffe0496993faf750a

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/GoMomoAI/pledge_contract
Commit	a12214ba3065dbde72d99740b1d04696ee0a02c6
Network	SOL
Badge Eligibility	No

Audit Updates

Initial Audit	20 Apr 2024

Source Files

Filename	SHA256
programs/pledge_contract/src/lib.rs	4dbfbf32dd954ecd43995ac837c2fafc5adaae77f72b5a1ffe0496993faf750a

Overview

The Pledge Contract is a Solana-based program developed using the Anchor framework, which incorporates aspects of the Solana Program Library (SPL) for token management. The program is designed to facilitate the creation and management of tokenized assets, implementing functionalities for minting tokens and managing their lifecycle, including freezing and thawing token accounts.

Initialize Mint

The `initialize_mint` function is responsible for setting up a new token mint. This includes creating metadata accounts via the `create_metadata_accounts_v3` function from the SPL Metadata program. The metadata for the token includes attributes like the token name, symbol, URI, and other details related to the token's representation and management. This function also establishes the minting parameters, including the decimals, mint authority, and freeze authority, which are all set to the mint authority at initialization.

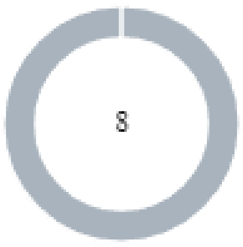
Minting Tokens

The mint_to function facilitates the actual distribution of tokens to user-specific accounts. It involves checks to ensure that tokens are only minted to eligible accounts based on a `claim_nonce`, which must match a specified `claim_id`. The function includes steps to thaw the account if it is frozen, mint the specified amount of tokens, and then re-freeze the account to secure the tokens post-minting.

Account and Authority Management

- Mint Authority: This role is central to the control of the token minting process, also serving as the freeze authority, which allows it to freeze or unfreeze token accounts as needed.
- Token Accounts: User-specific token accounts (to) are dynamically managed, being initialized as needed and controlled for security via freezing and thawing mechanisms. These accounts hold the actual tokens minted to users.

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	8

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	8	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	ITM	Incomplete Token Metadata	Unresolved
●	MT	Mints Tokens	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	NCI	Naming Convention Inconsistency	Unresolved
●	PCR	Program Centralization Risk	Unresolved
●	TFC	Token Freezing Concern	Unresolved
●	UU	UncheckedAccount Usage	Unresolved
●	UA	Update Authority	Unresolved

ITM - Incomplete Token Metadata

Criticality	Minor / Informative
Location	lib.rs#42
Status	Unresolved

Description

In the `initialize_mint` function, the URI field in the token's metadata is initialized as an empty string. This URI is crucial as it typically points to an external resource containing detailed information about the token, such as attributes and other relevant data stored in a JSON format accessible via a URL. The absence of a URI may limit the functionality and interoperability of the token, particularly on platforms that rely on this metadata to present information to users or to integrate with other services. Without a valid URI linking to comprehensive metadata, the token may not fully meet ecosystem standards that enhance user engagement and utility.

```
DataV2 {
    name: String::from("MOMO KIWI GOLD"),
    symbol: String::from("KIWIG"),
    uri: String::new(),
    seller_fee_basis_points: 0,
    creators: None,
    collection: None,
    uses: None,
},
```

Recommendation

It is recommended that the program be modified to include a valid URI during the token initialization phase. This URI should point to a reliable and secure storage location that hosts the token's detailed metadata. Ensuring that the token minted has a complete and accessible set of metadata not only aligns with best practices but also enhances the token's usability and acceptance across various platforms and applications.

MT - Mints Tokens

Criticality	Minor / Informative
Location	lib.rs#57
Status	Unresolved

Description

The program owner has the authority to mint tokens to specific users. The owner may take advantage of it by calling the `mint_to` function. As a result, the program tokens will be highly inflated.

```
pub fn mint_to(ctx: Context<MintTo>, claim_id: u32, amount: u64) ->
Result<()> {
        if ctx.accounts.user_info.claim_nonce != claim_id {
            return Err(ErrorCode::ClaimIdUsed.into());
        }

        ...

        // mint tokens
        let cpi_accounts = anchor_spl::token::MintTo {
            mint: ctx.accounts.mint.to_account_info(),
            to: ctx.accounts.to.to_account_info(),
            authority: ctx.accounts.mint_authority.to_account_info(),
        };
        let cpi_program = ctx.accounts.token_program.to_account_info();
        let cpi_ctx = CpiContext::new(cpi_program, cpi_accounts);
        anchor_spl::token::mint_to(cpi_ctx, amount)?;

        ...

        Ok(())
    }

}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the program admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	lib.rs#25,57
Status	Unresolved

Description

The program performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, to track and monitor the activity on the program. Without these events, it may be difficult for external parties to accurately determine the current state of the program.

```
pub fn initialize_mint(ctx: Context<InitalizeMint>) ->
Result<()> {
    create_metadata_accounts_v3(
    ...
}

pub fn mint_to(ctx: Context<MintTo>, claim_id: u32, amount:
u64) -> Result<()> {
    ...
}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the program. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the program will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

NCI - Naming Convention Inconsistency

Criticality	Minor / Informative
Location	lib.rs#25,105
Status	Unresolved

Description

The struct intended for initializing a new mint is named `InitalizeMint`, has a minor typographical error in the naming convention, presumably meant to be `InitializeMint`. While this does not impact the functionality of the contract, it could lead to potential confusion and slightly detracts from the overall readability of the codebase.

```
pub fn initialize_mint(ctx: Context<InitalizeMint>) ->
Result<()> {

pub struct InitalizeMint<'info> {
```

Recommendation

It is recommended that the project team adopts and adheres to consistent naming conventions for structs and other data types, particularly when they are closely associated with specific functionalities or methods. Correcting the typographical error by renaming the `InitalizeMint` struct not only aligns with general best practices but also enhances code readability and reduces potential confusion.

PCR - Program Centralization Risk

Criticality	Minor / Informative
Location	lib.rs#25,57
Status	Unresolved

Description

The programs's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion. Specifically, the admin has to properly call these functions, in order for the program to be functional.

Furthermore, these centralization risks also involve the issues described in the UA, MT, and TFC findings.

```
pub fn initialize_mint(ctx: Context<InitalizeMint>) ->
Result<()> {
    create_metadata_accounts_v3(
    ...
}

pub fn mint_to(ctx: Context<MintTo>, claim_id: u32, amount:
u64) -> Result<()> {
    ...
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the program's codebase itself. This approach would reduce external dependencies and enhance the program's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

TFC - Token Freezing Concern

Criticality	Minor / Informative
Location	lib.rs#87
Status	Unresolved

Description

The program implements a functionality where every token account is automatically frozen after tokens are minted to it via the `mint_to` function. This mechanism is controlled by the freeze authority. The freeze authority has the capability to both freeze and thaw token accounts. This setup means that after tokens are minted to a user's token account, the account becomes frozen, rendering the tokens non-transferable until the account is explicitly thawed by the freeze authority.

```
pub fn mint_to(ctx: Context<MintTo>, claim_id: u32, amount:
u64) -> Result<()> {
    ...

    // freeze account
    let cpi_context = CpiContext::new(
        ctx.accounts.token_program.to_account_info(),
        FreezeAccount {
            account: ctx.accounts.to.to_account_info(),
            mint: ctx.accounts.mint.to_account_info(),
            authority:
ctx.accounts.mint_authority.to_account_info(),
        },
    );
    token::freeze_account(cpi_context)?;

    ctx.accounts.user_info.claim_nonce += 1;
    Ok(())
}
```

Recommendation

It is recommended to reassess the business logic behind the automatic freezing of token accounts post-minting. If the immediate freezing of tokens is a desired feature, it should be clearly communicated to users through comprehensive documentation. Users should be fully informed about how this affects the usability of their tokens and under what conditions their tokens can be thawed. Additionally, it is crucial to ensure that the private keys of the freeze authority are securely managed to prevent unauthorized access and ensure that the authority can perform freezing and thawing actions as needed. Implementing robust security measures and possibly even considering multi-signature controls for these critical actions can help mitigate risks associated with the centralization of control.

UU - UncheckedAccount Usage

Criticality	Minor / Informative
Location	lib.rs#112
Status	Unresolved

Description

The `UncheckedAccount` is a type of account that bypasses the typical safety checks performed by the Anchor framework, such as data type validation and deserialization. While `UncheckedAccount` provides flexibility in handling accounts that may not fit predefined schemas, this flexibility comes at the cost of increased risk. Specifically, the program may interact with account data that is not in the expected state or format, leading to unpredictable behaviors or vulnerabilities. The absence of rigorous checks on these accounts makes the program susceptible to errors that could compromise its stability and security.

```
#[account(mut)]
pub metadata: UncheckedAccount<'info>,
```

Recommendation

It is recommended that the use of `UncheckedAccount` be carefully evaluated and, where possible, replaced with more specific account types that enable the Anchor framework to enforce data integrity and structure. If the use of `UncheckedAccount` is necessary for the program's functionality, additional safeguards should be implemented within the program code.

UA - Update Authority

Criticality	Minor / Informative
Location	lib.rs#33
Status	Unresolved

Description

The program is currently configured in a manner that allows the update authority, to retain privileges that enable the modification of crucial metadata fields. The failure to revoke the update authority leaves the token vulnerable to potential risks, as the designated address retains the capability to make changes to the metadata. This oversight could lead to unauthorized or malicious modifications that might compromise the integrity and intended functionality of the token.

```
pub fn initialize_mint(ctx: Context<InitalizeMint>) ->
Result<()> {
    create_metadata_accounts_v3(
        CpiContext::new(

ctx.accounts.token_metadata_program.to_account_info(),
                CreateMetadataAccountsV3 {
                metadata:
ctx.accounts.metadata.to_account_info(),
                mint: ctx.accounts.mint.to_account_info(),
                mint_authority:
ctx.accounts.mint_authority.to_account_info(),
                update_authority:
ctx.accounts.mint_authority.to_account_info(),
                payer:
ctx.accounts.mint_authority.to_account_info(),
                system_program:
ctx.accounts.system_program.to_account_info(),
                rent: ctx.accounts.rent.to_account_info(),
            },
        ...

    Ok(())
}
```

Recommendation

It is recommended to securely manage the private keys of the update authority. Furthermore, it is recommended to revoke the update authority privileges. This action would ensure a consistent security posture across the program's operational aspects, eliminating the discrepancy that currently allows for undue modification privileges. Implementing this recommendation would align the program's security measures, providing a more robust defense against unauthorized changes and enhancing the overall security of the program's operational environment.

Summary

The Pledge Contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io