Cyberscope

Audit Report
GroWealth

January 2025

Source	Github Repository
Commit	2034512dce77efee14e9c1d8dc6638703e63cb94
Audited by	© cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Commit	2034512dce77efee14e9c1d8dc6638703e63cb94
Network	SOL

Audit Updates

Initial Audit	05 Nov 2024
Corrected Phase 2	19 Dec 2024
Corrected Phase 3	02 Jan 2025

Source Files

Filename	SHA256
constant.rs	920aec5f5a3eb8e251eec09e382219ac629c0d5a7c0d9140bef514987169c468
error.rs	2a136edabc037f3935286f51fc4618a1310ed8c9f3efb57e24d5c7a8f56f5519
events.rs	9b5b535eb052f1cfd2401db6cdeea0b73f1885ea184217c2e1ccd7e714feb907
lib.rs	e6c25d8d428cfa03ca2a7ee89484ab199916e15de18fac7cb3ddb7c37ee82755
state.rs	68055662744f67c0f047197a44ec63be8b905e2ab1be07f041757a3050b368ba
processor/create_presale.rs	b8d699100afdbe8f3c21d854b8d67fe55491e7a601e41b8605be738c353e9814
processor/grant_access.rs	bc6771b11426fe457b77ae780006f6bd235baff210af5d723c007e3d594a4230

processor/initialize.rs	f79546a56e65a327724102393a8afe227fab9156f8a1fc251827886330f094eb
processor/mod.rs	abedce22b9ba191e7aa5fd28c7fd74ab137190a2240590ba8a08ed2bdf27620c
processor/purchase_token.rs	4d0b2976050b2b0c8774132828cf789edea59a3871d01ffb36b6e947f957af12
processor/revoke.rs	79ea05bd8f4a9d1b3a7930d7c498afe81823cb60518e0fc8a7cac3a001b46e53
processor/update_presale.rs	6bb0c429c6d0f6c5ddfd4098219900ff914c87550d6e320bc0547cbb3c5250ab
processor/withdraw_token.rs	a1cf5d08a8f40c99cda8ef4c6561878bb868d4011e33293a5dbf41da4d03826f

Overview

The contract manages the lifecycle of a token presale. It creates and configures the presale, defines who is allowed to administer it, and handles the transfer of tokens both to participants and back to the presale authority.

The `initialize` function sets up core parameters in the presale program data. It ties the presale to a specific token mint and establishes an authority with special permissions, including the ability to freeze token accounts if necessary. It stores these details in the newly created presale program data account, ensuring that future actions know which token and authority to reference.

The `create_presale` function creates or reinitializes the presale by configuring its start time, end time, minimum and maximum buyable token amounts, total tokens for sale, and token price. It transfers the specified number of tokens from an administrative account to a presale-owned token account. By using a single seed for the presale account, the contract associates each presale with a fixed set of seeds, effectively allowing only one instance of the presale data to manage a particular token mint. Because of its design, this function can be called again after a previous presale's end time to start a new or adjusted presale period under the same account.

The `grant_access` function allows the super authority to authorize a new creator to manage the presale. It stores the newly granted creator's public key in a dedicated creator account. This establishes the permissions needed for the new creator to interact with other aspects of the presale, such as creating or updating a presale session.

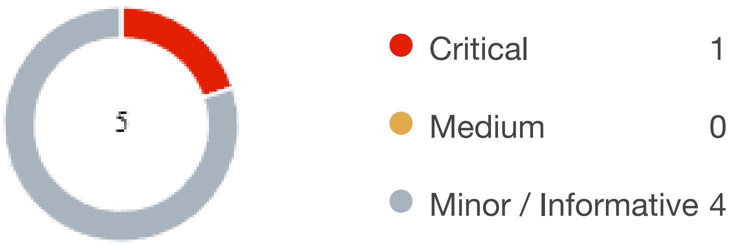
The `revoke_access` function reverts the creator authority back to the super authority by adjusting the stored creator key. This lets the super authority remove delegated rights from the previous creator, ensuring that only the designated party can manage or modify the presale afterward.

The `update_presale` function updates the existing presale's configuration by adjusting its end time, as long as the current time is still within the presale window. It checks that the caller is the presale's authority and makes sure the updated end time does not violate logical constraints, such as setting it earlier than the start time.

The `purchase_token` function allows participants to buy tokens from the presale. It verifies the payment amount based on the token price and ensures that the purchase falls within the presale's defined rules. It then transfers the user's payment from their token account to the presale's beneficiary account before moving the presale tokens to the buyer's account. Because the contract freezes participants' token accounts after the transfer, the designated freeze authority is required to thaw them whenever the buyer wishes to move or spend the purchased tokens.

The `withdraw_token` function permits the presale authority to withdraw any remaining tokens that were set aside for the presale. It transfers these tokens back from the presale's token account to an administrative account, reducing the total tokens allocated to the presale and freeing them up for other uses.

Findings Breakdown



Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	4	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
🔴	IHUA	Incorrect Hardcoded USDC Address	Unresolved
●	IKCU	Inconsistent Key Comparison Use	Unresolved
●	PPR	Potential Presale Reinitialization	Unresolved
●	PCR	Program Centralization Risk	Unresolved
●	SSI	Single Seed Initialization	Unresolved

IHUA - Incorrect Hardcoded USDC Address

Criticality	Critical
Location	constant.rs#L7 processor/purchase_token.rs#L109,129
Status	Unresolved

Description

The contract uses a hardcoded token address for USDC that is intended for mainnet, but this address is incorrect. In addition, the contract hardcodes a separate address for test environments. This makes the contract rely on a wrong reference when it is deployed on mainnet and causes potential issues with token transfers and payments. It also prevents flexible updates or expansions if the contract needs to run on other environments or networks, since there is no mechanism to handle different token addresses without altering the contract code.

```
pub static USDC_TEST_MINT_PUBKEY_STR: &str =
"4zMMC9srt5Ri5X14GAgXhaHii3GnPAEERYPJgZJDncDU";

pub static USDC_MINT_PUBKEY_STR: &str =
"EPjFWdd5AufqSSqeM2q6wwFACHXyYQyRPoMtybdD3iUC";

#[account(mut,
        constraint = buyer_payment_ata.mint.key().to_string()
== USDC_TEST_MINT_PUBKEY_STR
        @ PresaleErrorCodes::InvalidPaymentMint
    )]
    pub buyer_payment_ata: Account<'info, TokenAccountLegacy>,

#[account(
        mut,
        constraint = payment_mint.key().to_string() ==
USDC_TEST_MINT_PUBKEY_STR
        @ PresaleErrorCodes::InvalidPaymentMint
    )]
    pub payment_mint: Account<'info, MintLegacy>,
    #[account(
```

Recommendation

It is recommended to correct the address of USDC, to correctly point to the correct token. Furthermore, it is recommended to remove the direct reliance on hardcoded addresses for mainnet and test environments. One way to achieve this is to introduce environment-based checks or feature flags that select the correct address dynamically for each network, so that the contract references the valid token address for the environment in which it is deployed. This ensures accurate deployment on all networks, including mainnet.

IKCU - Inconsistent Key Comparison Use

Criticality	Minor / Informative
Location	processor/revoke.rs#L14 processor/withdraw_token.rs#L31
Status	Unresolved

Description

The smart contract uses two different approaches for comparing public keys: one with the `as_ref()` method and one without. While both methods are technically correct and achieve the same result, the inconsistency reduces code readability and may cause confusion for developers maintaining the contract. Using a uniform approach throughout the codebase is essential for clarity and simplicity.

```
#[account(
        mut,
        constraint = authority.key().as_ref() ==
presale_program_data.super_authority.key().as_ref()
        @PresaleErrorCodes::Unauthorized,
    )]
    pub authority: Signer<'info>,

#[account(
        mut,
        seeds = [CREATOR_SEED, authority.key().as_ref()],
        bump,
        constraint = creator_account.creator.key() ==
authority.key()
        @PresaleErrorCodes::InvalidCreator
    )]
    pub creator_account: Box<Account<'info, CreatorAccount>>,
```

Recommendation

It is recommended to adopt a single, consistent method for key comparisons. This change improves readability and ensures that the code follows a clear and unified style, making it easier for future developers to understand and maintain.

PPR - Potential Presale Reinitialization

Criticality	Minor / Informative
Location	processor/create_presale.rs#L16
Status	Unresolved

Description

The current implementation of the presale creation function permits multiple invocations that can effectively reinitialize or overwrite the presale’s configuration. The use of `init_if_needed` on key presale-related accounts and the conditions allowing a new presale start time to be set after a previous one allows the authority to restart the presale. This creates a situation where the presale may begin again, potentially contradicting expectations that once it is started and concluded, it should not be altered in such a fundamental way. Additionally,the presale can end abruptly, if it is currently active, by setting a new start time in the future. Furthermore, the presence of a separate update function, which allows only the extension of the end time while a presale is still ongoing, suggests that the intended design likely aims to restrict the ability to modify critical presale parameters after initiation. The combination of these behaviors creates a lack of clarity around how the presale is meant to operate over its lifecycle, and it may enable scenarios where a presale can be abruptly redefined or relaunched, potentially confusing participants and undermining trust.

```
pub fn create_presale_handler(ctx: Context<CreatePresale>,
args: CreatePresaleArgs) -> Result<()> {
    let presale_account = &mut ctx.accounts.presale_account;
    let clock = Clock::get()?;
    let current_unix_timestamp = clock.unix_timestamp as u64;
    if presale_account.end_time != 0 {
        require!(
            current_unix_timestamp > presale_account.end_time,
            PresaleErrorCodes::PresaleAlreadyActive
        );
    }
    require!(
        presale_account.start_time == 0 ||
presale_account.end_time <= args.start_time,
        PresaleErrorCodes::PresaleAlreadyActive
    );
    require!(
        current_unix_timestamp < args.start_time &&
args.end_time > args.start_time,
        PresaleErrorCodes::InvalidTime
    );
    require!(
        args.maximum_buyable_amount >
args.minimum_buyable_amount,
        PresaleErrorCodes::InvalidPurchaseAmount
    );
    presale_account.authority = args.authority.key();
    presale_account.start_time = args.start_time;
    presale_account.end_time = args.end_time;
    presale_account.minimum_buyable_amount =
args.minimum_buyable_amount;
    presale_account.maximum_buyable_amount =
args.maximum_buyable_amount;
    presale_account.total_tokens += args.presale_token_amount;
    presale_account.token_price_in_usdc =
args.token_price_in_usdc;
    presale_account.bump = ctx.bumps.presale_account;
    anchor_spl::token_interface::transfer_checked(
        ctx.accounts.transfer_token_to_presale_ata(),
        args.presale_token_amount,
        ctx.accounts.token_mint.decimals
    )?;
    emit!(CreatePresaleEvent {
        authority: ctx.accounts.authority.key(),
        token_amount: args.presale_token_amount,
        start_time: args.start_time,
        end_time: args.end_time,
        minimum_buyable_amount: args.minimum_buyable_amount,
        maximum_buyable_amount: args.maximum_buyable_amount,
        token_price_in_usdc: args.token_price_in_usdc,
    });
    Ok(())
}
```

Recommendation

It is recommended to re-evaluate the presale logic to establish a clear and consistent lifecycle. The core values and parameters of the presale should not be modifiable through the creation function after it has already started, unless the explicit business logic allows for a safe and transparent process to do so. By clarifying the intended behavior and ensuring that the creation function cannot overwrite a running or completed presale, the contract’s integrity and the participants’ expectations will be better preserved.

PCR - Program Centralization Risk

Criticality	Minor / Informative
Location	processor/create_presale.rs#L16 processor/update_presale.rs#L7 processor/purchase_token.rs#L68
Status	Unresolved

Description

The program'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 program owner has the authority to restart the presale multiple times, even after it has concluded. This means that the owner can modify key parameters, such as the start and end times, and relaunch the presale at will. Additionanlly, the program owner has the authority to update the parameters of the presale by calling the `update_presale` function. Lastly, when users purchase token, their token accounts get frozen indefinetely. In order for users to be able to transfer their tokens, the freeze authority of the token that is used in the presale, have to unfreeze/thaw their accounts.

```
pub fn create_presale(ctx: Context<CreatePresale>, args:
CreatePresaleArgs) -> Result<()> {
        create_presale::create_presale_handler(ctx, args)
    }

pub fn update_presale(ctx: Context<UpdatedData>, args:
UpdatePresaleArgs) -> Result<()> {
        update_presale::update_presale_handler(ctx, args)
    }

let cpi_accounts = anchor_spl::token_interface::FreezeAccount {
        account:
ctx.accounts.buyer_token_ata.to_account_info(),
        mint: ctx.accounts.token_mint.to_account_info(),
        authority:
ctx.accounts.freeze_authority.to_account_info(),
    };
    let cpi_program =
ctx.accounts.token_program.to_account_info(); // Token-2022
program
    let cpi_ctx = CpiContext::new(cpi_program, cpi_accounts);
    anchor_spl::token_interface::freeze_account(cpi_ctx)?;
```

Recommendation

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

SSI - Single Seed Initialization

Criticality	Minor / Informative
Location	processor/create_presale.rs#L64
Status	Unresolved

Description

The `presale_account` uses a constant seed in its initialization, which restricts it to managing only one token mint. If multiple token mints are intended to be supported, the current design causes all presales to overwrite the same account due to the `init_if_needed` directive. This creates potential issues with overwriting data and limits the contract's functionality for handling multiple token mints.

```
#[derive(Accounts)]
pub struct CreatePresale<'info> {
    #[account(
        mut,
    )]
    pub authority: Signer<'info>,
    #[account(
        mut,
        seeds = [CREATOR_SEED, authority.key().as_ref()],
        bump,
        constraint = creator_account.creator.key().as_ref() ==
authority.key().as_ref()
        @PresaleErrorCodes::InvalidCreator
    )]
    pub creator_account: Box<Account<'info, CreatorAccount>>,
    #[account(
        init_if_needed,
        payer = authority,
        seeds = [PRESALE_SEED],
        bump,
        space = 8 + PresaleAccount::INIT_SPACE
    )]
    pub presale_account: Box<Account<'info, PresaleAccount>>,
    #[account(
        init_if_needed,
        payer = authority,
        associated_token::mint = token_mint,
        associated_token::authority = presale_account
    )]
    pub presale_token_ata: Box<InterfaceAccount<'info,
TokenAccount>>,
    #[account(
        mut,
    )]
    pub admin_token_ata: Box<InterfaceAccount<'info,
TokenAccount>>,
    #[account(seeds = [PRESALE_SEED, PROGRAM_DATA_SEED], bump =
presale_program_data.bump)]
    pub presale_program_data: Box<Account<'info,
PresaleProgramData>>,
    #[account(
        mut,
        constraint=token_mint.key().as_ref() ==
presale_program_data.token_mint.as_ref()
        @PresaleErrorCodes::InvalidMintedToken,
    )]
    pub token_mint: Box<InterfaceAccount<'info, Mint>>,
    pub token_program: Program<'info, Token2022>,
    pub associated_token_program: Program<'info,
AssociatedToken>,
    pub system_program: Program<'info, System>,
    pub rent: Sysvar<'info, Rent>,
}
```

Recommendation

It is recommended to clarify the intended functionality of the presale. If only one token mint is supported, remove `init_if_needed` and ensure proper error handling for attempts to initialize multiple presales. If multiple token mints are intended, include the token mint as part of the seed and modify the design of `presale_program_data` to support multiple token mints effectively. This ensures the contract functions as intended and avoids unintended overwrites or limitations.

Summary

The GroWealth contract implements a presale mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

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

cyberscope.io