

Audit Report **GroWealth**

January 2025

Source Github Repository

Commit 26665d1fa8d51f0706c3f9e8527c530065124240

Audited by © cyberscope



Table of Contents

| Table of Contents | 1 |
|--|----|
| Risk Classification | 2 |
| Review | 3 |
| Audit Updates | 3 |
| Source Files | 3 |
| Overview | 5 |
| Findings Breakdown | 7 |
| Diagnostics | 8 |
| PPR - Potential Presale Reinitialization | 9 |
| Description | 9 |
| Recommendation | 11 |
| PCR - Program Centralization Risk | 12 |
| Description | 12 |
| Recommendation | 13 |
| SSI - Single Seed Initialization | 14 |
| Description | 14 |
| Recommendation | 16 |
| Summary | 17 |
| Disclaimer | 18 |
| About Cyberscope | 19 |



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:

- 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.
- 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.
- 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 | 26665d1fa8d51f0706c3f9e8527c530065124240 |
|---------|--|
| Network | SOL |

Audit Updates

| Initial Audit | 05 Nov 2024 |
|-------------------|-------------|
| Corrected Phase 2 | 19 Dec 2024 |
| Corrected Phase 3 | 02 Jan 2025 |
| Corrected Phase 4 | 08 Jan 2025 |

Source Files

| Filename | SHA256 |
|-----------------------------|--|
| constant.rs | a1d9ab6c7ac3767d91cc562bb948607bacaa5baced948d652b83f3e466 ef5a71 |
| error.rs | 2a136edabc037f3935286f51fc4618a1310ed8c9f3efb57e24d5c7a8f56f5 519 |
| events.rs | 9b5b535eb052f1cfd2401db6cdeea0b73f1885ea184217c2e1ccd7e714f eb907 |
| lib.rs | 296d42ccceb8566981b89b2fa9254ee4130a3711d75839a0094696957f b88514 |
| state.rs | 68055662744f67c0f047197a44ec63be8b905e2ab1be07f041757a3050b 368ba |
| processor/create_presale.rs | b8d699100afdbe8f3c21d854b8d67fe55491e7a601e41b8605be738c35 3e9814 |



| processor/grant_access.rs | bc6771b11426fe457b77ae780006f6bd235baff210af5d723c007e3d594 a4230 |
|-----------------------------|--|
| processor/initialize.rs | f79546a56e65a327724102393a8afe227fab9156f8a1fc251827886330f0 94eb |
| processor/mod.rs | abedce22b9ba191e7aa5fd28c7fd74ab137190a2240590ba8a08ed2bdf 27620c |
| processor/purchase_token.rs | ce8ad2e107fa6f9136b28e4b0eadcc2b1070bf18578cdbe5ffe182a00f45 5b77 |
| processor/revoke.rs | 79ea05bd8f4a9d1b3a7930d7c498afe81823cb60518e0fc8a7cac3a001b 46e53 |
| processor/update_presale.rs | 6bb0c429c6d0f6c5ddfd4098219900ff914c87550d6e320bc0547cbb3c5 250ab |
| processor/withdraw_token.rs | 2e9264f9a7c525c71c4045447d00d318860a13ada2731c1259c0821cba 99f92c |



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 | 0 | 0 | 0 | 0 |
| Medium | 0 | 0 | 0 | 0 |
| Minor / Informative | 3 | 0 | 0 | 0 |

Diagnostics

Critical
 Medium
 Minor / Informative

| Severity | Code | Description | Status |
|----------|------|------------------------------------|------------|
| • | PCR | Program Centralization Risk | Unresolved |
| • | PPR | Potential Presale Reinitialization | Unresolved |
| • | SSI | Single Seed Initialization | Unresolved |

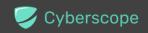


PPR - Potential Presale Reinitialization

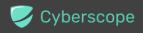
| Criticality | Minor / Informative |
|-------------|---------------------------------|
| Location | processor/create_presale.rs#L16 |
| Status | Unresolved |

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,</pre>
        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. Additionanly, 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

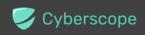
| Criticality | Minor / Informative |
|-------------|---------------------------------|
| Location | processor/create_presale.rs#L64 |
| Status | Unresolved |

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()],
        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,</pre>
TokenAccount>>,
    #[account(
        mut,
    pub admin token ata: Box<InterfaceAccount<'info,</pre>
TokenAccount>>,
    #[account (seeds = [PRESALE SEED, PROGRAM DATA SEED], bump =
presale program data.bump) ]
    pub presale program data: Box<Account<'info,</pre>
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,</pre>
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 <code>init_if_needed</code> 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 <code>presale_program_data</code> 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