Cyberscope

Audit Report

GroWealth

November 2024

Source	Github Repository
Commit	8e5bbbf542465b951ed3bbc7556fedbcebed316f
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	8e5bbbf542465b951ed3bbc7556fedbcebed316f
Network	SOL

Audit Updates

Initial Audit	05 Nov 2024

Source Files

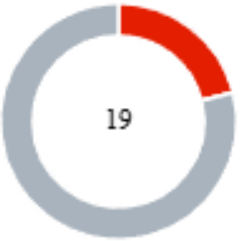
Filename	SHA256
constant.rs	e2dc286788fd90eaf234325c17379fe9f43865621ac30d64541c1db22b8deb1b
error.rs	dbe67a3b757ec5ed17bd85b9f3d6e3d5b9290bbbd82b418f369fd13b56b3a300
events.rs	1b20c6ec014c43568e00f82104d91611d170216d8d30ccb652a7ca50d2ea8c7f
lib.rs	fb9152ef885253cbad34a859ff6da9fa48f646ee6e5c650e6370dcecc7874f34
state.rs	48dd32d2e2c883105b2e657f23262afd7e1722daa5dd72506dce29b4be1220f7
processor/create_presale.rs	c1a17db8aa86c9abc2f9c9f95597360d8dabad4bc64f0f7678bf357c8fa6e06b
processor/grant_access.rs	dd59a89d79671f37e8e3d2c0503a8f2cfead3064410e62fd2c1c652c188615bf
processor/initialize.rs	3c07283f8fe65da32941040eb0e83d494c3020ccafac556f7d9f3998b0f014d1

processor/mod.rs	2ca9496c83223685b0016f75ecc9f992151e1dba5aa617d8dff19c4d26ee05bb
processor/purchase_token.rs	4f6053d15c58403394083bac15dbbd831e42c120e29b33737151d306fec30a25
processor/rewoke_access.rs	cba1291a9679985f084e9a0b63ec40c755e3a31e926e64d97fb29da020ac2fd7
processor/update_presale.rs	02edec283b9dff9620e96df290ae26db2294d5cc801fa85f869ec16dba489772
processor/withdraw_token.rs	34bba249f7c41252748dac90c43cc311c88a53b4e79b2796d3145c55b671b0bd

Contract Readability Comment

The audit scope is to check for security vulnerabilities, validate the business logic, and propose potential optimizations. The contract does not adhere to key principles of Solana and Rust development regarding resource efficiency, code readability, and proper data structures. Given these issues, it cannot be considered production-ready. The development team is strongly advised to re-evaluate the business logic and follow Solana and Rust best practices. It is recommended to optimize resource usage, simplify function definitions, and use descriptive variable names to enhance auditability, efficiency, and security.

Findings Breakdown



Critical	4
Medium	0
Minor / Informative	15

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	4	0	0	0
Medium	0	0	0	0
Minor / Informative	15	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	FI	Frontunnable Initialization	Unresolved
Critical	MPM	Missing Payment Mechanism	Unresolved
Critical	PSL	Potential System Lockout	Unresolved
Critical	RPM	Restrictive Presale Mechanism	Unresolved
Minor / Informative	IFN	Inconsistent Function Naming	Unresolved
Minor / Informative	RSR	Redundant Signer Requirement	Unresolved
Minor / Informative	IEE	Inconsistent Event Emission	Unresolved
Minor / Informative	IKCU	Inconsistent Key Comparison Use	Unresolved
Minor / Informative	ISNC	Inconsistent Seed Naming Convention	Unresolved
Minor / Informative	ISM	Ineffective State Management	Unresolved
Minor / Informative	ISA	Inefficient Space Allocation	Unresolved
Minor / Informative	LCM	Lamports Constant Misuse	Unresolved
Minor / Informative	PPSI	Potential Presale Status Inconsistency	Unresolved
Minor / Informative	PCR	Program Centralization Risk	Unresolved

●	RAAC	Redundant Account Authority Constraints	Unresolved
●	RCP	Redundant Code Present	Unresolved
●	UAD	Unused Account Declaration	Unresolved
●	UEC	Unused Error Codes	Unresolved
●	USF	Unused Struct Field	Unresolved

FI - Frontunnable Initialization

Criticality	Critical
Location	processor/initialize.rs#L12
Status	Unresolved

Description

The `initialize_handler` function lacks any access control mechanism, allowing any account to call it. This function sets the caller as the authority of the `presale_program_data` account, granting them full administrative control over the presale configuration. Without restrictions, any user can initialize the presale, taking ownership and potentially compromising the intended control of the contract.

```
pub fn initialize_handler(ctx: Context<Initialize>, args:
InitializeArgs) -> Result<()> {
    let presale_program_data: &mut Account<PresaleProgramData>
= &mut ctx.accounts.presale_program_data;
    presale_program_data.token_mint = args.token_mint;
    presale_program_data.authority =
ctx.accounts.authority.key();
    presale_program_data.bump = ctx.bumps.presale_program_data;

    emit!(InitializeEvent{
        initializer: ctx.accounts.authority.key(),
        token_mint: args.token_mint,
    });
    Ok(())
}
```

Recommendation

Add access control to the `initialize_handler` function to ensure that only an authorized entity can call it. Consider using the program's upgrade authority as the designated initializer or specify a hardcoded address of a trusted account. This change prevents unauthorized users from taking control and secures the initialization process.

MPM - Missing Payment Mechanism

Criticality	Critical
Location	process/purchase_token.rs#L8
Status	Unresolved

Description

The `purchase_token_handler` function allows the caller to receive tokens from the presale account without requiring any form of payment in return. Typically, in presale scenarios, buyers are expected to pay in native currency or another specified asset, based on a predefined exchange rate or business logic set by the contract owner. The absence of a payment mechanism in this function creates an inconsistency with standard presale practices, which could lead to the unintended consequence of tokens being distributed for free.

```
pub fn purchase_token_handler(ctx: Context<PurchaseToken>,
token_amount: u64) -> Result<()> {
    let clock = Clock::get()?;
    let current_unix_timestamp = clock.unix_timestamp as u64;

    require!(
        current_unix_timestamp >=
ctx.accounts.presale_account.start_time &&
            current_unix_timestamp <=
ctx.accounts.presale_account.end_time,
        PresaleErrorCodes::InvalidTime
    );

    require!(
        ctx.accounts.buyer.key() ==
ctx.accounts.presale_account.authority.key(),
        PresaleErrorCodes::Unauthorized
    );

    // check token_amount is within the buyable range

    require!(
        token_amount >=
ctx.accounts.presale_account.minimum_buyable_amount &&
            token_amount <=
ctx.accounts.presale_account.maximum_buyable_amount,
        PresaleErrorCodes::InvalidPurchaseAmount
    );

    // now transfer the tokens from presale to buyer account
    let bump = ctx.accounts.presale_account.bump;
    msg!("Hello wolrd!!!");

    anchor_spl::token_interface::transfer_checked(

ctx.accounts.transfer_presale_token_to_buyer().with_signer(&[&[
PREFIX, &[bump]]]),
        token_amount,
        ctx.accounts.token_mint.decimals
    )?;
    // ctx.accounts.presale_account.total_tokens -=
token_amount;

    emit!(PurchaseTokenEvent {
        buyer: ctx.accounts.buyer.key(),
        bought_token_amount: token_amount,
        rest_token_amount:
ctx.accounts.presale_account.total_tokens,
    });
    Ok(())
```

```
}
```

Recommendation

Implement a payment mechanism that ensures buyers pay an appropriate amount when purchasing tokens. This could involve transferring native currency or another asset from the buyer to the presale account, following a fixed exchange rate or a business logic determined by the contract owner. This adjustment aligns the presale process with common practices and ensures that token distribution is appropriately compensated.

PSL - Potential System Lockout

Criticality	Critical
Location	processor/rewoke_access.rs#L5
Status	Unresolved

Description

The `rewoke_access_handler` function resets the `creator_account.reciever` to the default public key. This action effectively breaks the existing constraints that check the relationship between the `buyer`, `authority`, and `creator_account.reciever` across multiple functions. Since these constraints require a valid matching authority, resetting the `reciever` results in failed authorization checks, rendering key parts of the program unusable. Additionally, the `grant_access_handler` function cannot be called again due to the `init` constraint, which only allows the `creator_account` to be created once. Consequently, there is no mechanism to restore access, permanently disabling essential functionality.

```
pub fn rewoke_access_handler(ctx: Context<RewokeAccess>) ->
Result<()>{

    if ctx.accounts.presale_program_data.authority ==
Pubkey::default(){
        return  err!(PresaleErrorCodes::PresaleDoesnotExist);
    }

    // msg!("Number: {}", num);

    let creator_account = &mut ctx.accounts.creator_account;
    let key = creator_account.reciever.key();
    creator_account.reciever = Pubkey::default();

    emit!(RewokeAccessEvent {
        old_authority: key,
    });
    Ok(())

}
```

Recommendation

Review the design of the `rewoke_access_handler` function to ensure that resetting the `creator_account.reciever` does not lock the system or break critical authorization logic. Consider implementing a mechanism to securely reassign authority or avoid resetting the `reciever` to an unusable state. This change ensures the program remains operational and does not risk becoming irreversibly locked.

RPM - Restrictive Presale Mechanism

Criticality	Critical
Location	processor/initialize.rs#L12 processor/purchase_token.rs#L18
Status	Unresolved

Description

The current implementation of the presale mechanism limits token purchasing exclusively to the authority account that initializes the presale. The code enforces constraints that require the buyer to be the same as the authority or a designated creator account, effectively preventing other users from participating. This design defeats the purpose of a presale, which is to allow multiple users to acquire tokens during the presale period. As a result, the presale functions more like a self-transfer of tokens for the authority, rather than an opportunity for broader user participation.

```
pub fn initialize_handler(ctx: Context<Initialize>, args:
InitializeArgs) -> Result<()> {
    let presale_program_data: &mut Account<PresaleProgramData>
= &mut ctx.accounts.presale_program_data;
    presale_program_data.token_mint = args.token_mint;
    presale_program_data.authority =
ctx.accounts.authority.key();
    presale_program_data.bump = ctx.bumps.presale_program_data;

    emit!(InitializeEvent{
        initializer: ctx.accounts.authority.key(),
        token_mint: args.token_mint,
    });
    Ok(())
}

require!(
        ctx.accounts.buyer.key() ==
ctx.accounts.presale_account.authority.key(),
        PresaleErrorCodes::Unauthorized
    );
```

IFN - Inconsistent Function Naming

Criticality	Minor / Informative
Location	lib.rs#L39
Status	Unresolved

Description

The function `rewoke_access` in the contract contains a naming inconsistency. The term "rewoke" does not align with standard English or common terminology used in programming for revoking access. The correct term, "revoke," more accurately describes the functionality of withdrawing permissions or access rights. The use of an incorrect term could create confusion for developers, auditors, or other stakeholders who review or interact with the code, potentially leading to misunderstandings about the purpose of the function.

```
pub fn rewoke_access(ctx:Context<RewokeAccess>) -> Result<()>{
        rewoke_access::rewoke_access_handler(ctx)
    }
```

Recommendation

It is recommended to use consistent and meaningful naming throughout the codebase to improve readability and clarity. The function name should be updated to reflect its intended behavior accurately. Additionally, associated references or documentation should be checked and aligned to avoid further confusion. This practice helps ensure that the contract remains understandable and maintainable.

RSR - Redundant Signer Requirement

Criticality	Minor / Informative
Location	processor/grant_access.rs#L5
Status	Unresolved

Description

The `grant_access` function unnecessarily includes `creator` as a signer in the account struct, despite the fact that it is not used for any authorization or signing within the function. The body of the function only uses the `creator` public key passed as an argument to update the `creator_account.reciever` field. There is no instance where `ctx.accounts.creator` is accessed or required for signature verification. This redundant signer requirement complicates the transaction by imposing an unnecessary constraint on the `creator` account, without enhancing security or serving any functional purpose.

```
pub fn grant_access_handler(ctx: Context<GrantAccess>, creator: Pubkey)
-> Result<()> {
    let presale_program_data = &mut ctx.accounts.presale_program_data;

    let creator_account = &mut ctx.accounts.creator_account;
    creator_account.reciever = creator;
    creator_account.bump = ctx.bumps.creator_account;

    emit!(GrantAccessEvent {
        new_authority: creator,
        provider: presale_program_data.authority.key(),
    });
    Ok(())
}
```

Recommendation

Remove the `creator` account as a signer from the `grant_access` function, as it is not used for authorization or any signing operation within the function. Simplifying the account structure in this way reduces transaction complexity, avoids confusion, and ensures the function remains efficient and straightforward.

IEE - Inconsistent Event Emission

Criticality	Minor / Informative
Location	processor/update_presale.rs#L47
Status	Unresolved

Description

The `update_presale_handler` function updates several fields in the `presale_account`, but the `start_time` field is commented out and remains unchanged. Despite this, the `start_time` from the arguments is still emitted in the UpdatePresaleEvent, creating a misleading inconsistency between the actual data and the event emitted. Users or off-chain listeners relying on the event data may be misinformed about the current state of the presale, leading to potential confusion or misinterpretation.

```
//update the presale data
    // presale_account.start_time = args.start_time;
    presale_account.end_time = args.end_time;
    presale_account.maximum_buyable_amount =
args.maximum_buyable_amount;
    presale_account.minimum_buyable_amount =
args.minimum_buyable_amount;
    presale_account.is_presale_ended = args.is_presale_ended;

    emit!(UpdatePresaleEvent {
        start_time: args.start_time,
        end_time: args.end_time,
    });
```

Recommendation

It is advised to ensure that only the fields that are actually updated in the `presale_account` are included in the emitted event. This adjustment will make the event data accurate and reflective of the true state of the presale, improving transparency and reliability.

IKCU - Inconsistent Key Comparison Use

Criticality	Minor / Informative
Location	processor/create_presale.rs#L78
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() ==
creator_account.reciever.as_ref()
        @PresaleErrorCodes::Unauthorized,
    )]
    pub authority: Signer<'info>,

    #[account(
        mut,
        seeds = [MINTER_SEED],
        bump,
        constraint = creator_account.reciever ==
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.

ISNC - Inconsistent Seed Naming Convention

Criticality	Minor / Informative
Location	constant.rs#L1
Status	Unresolved

Description

The `PREFIX` constant in the contract uses `b"Presale"` with an uppercase "P," while other seeds, such as `program_data` and `creator_seed`, are defined in lowercase. This inconsistency in naming conventions can introduce potential confusion for developers and auditors, as it breaks uniformity in how seeds are structured. Standard practices typically favor lowercase for seed constants to maintain readability and to align with common conventions across smart contract ecosystems.

```
pub const PREFIX: &[u8] = b"Presale";
```

Recommendation

It is recommended to adopt a consistent naming convention across all seed constants, preferably using lowercase for uniformity. Ensuring that seed names follow the same format throughout the codebase improves readability and helps prevent errors or misunderstandings in account derivation or when referencing seeds. This adjustment supports better maintainability and coherence in the contract’s design.

ISM - Ineffective State Management

Criticality	Minor / Informative
Location	state.rs#L12 processor/purchase_token.rs#L40 processor/withdraw_token.rs#L13,31
Status	Unresolved

Description

The `total_tokens` field in the `PresaleAccount` struct is intended to track the number of tokens available in the presale. However, all instances in the codebase where this field would be updated or modified are commented out. As a result, the value of `total_tokens` remains unchanged and always returns 0, rendering the field ineffective for managing the presale's token state. This issue affects the accuracy of event emissions and log messages, misleading participants and developers about the actual state of the presale.

```
#[account]
#[derive(Default, InitSpace, Debug)]
pub struct PresaleAccount {
    pub authority: Pubkey,
    pub start_time: u64,
    pub end_time: u64,
    pub minimum_buyable_amount: u64,
    pub maximum_buyable_amount: u64,
    pub token_price_in_lamports: u64,
    pub total_tokens: u64,
    pub is_presale_ended: bool,
    pub bump: u8,
}

// ctx.accounts.presale_account.total_tokens -= token_amount;

    emit!(PurchaseTokenEvent {
        buyer: ctx.accounts.buyer.key(),
        bought_token_amount: token_amount,
        rest_token_amount:
ctx.accounts.presale_account.total_tokens,
    });

msg!(
        "Presale token amount: {}, asked amount: {}",
        ctx.accounts.presale_account.total_tokens /
LAMPORTS_PER_SOL,
        token_amount / LAMPORTS_PER_SOL
    );

// ctx.accounts.presale_account.total_tokens -= token_amount;

    msg!(
        "After withdraw , presale token amount {}",
        ctx.accounts.presale_account.total_tokens /
LAMPORTS_PER_SOL
    );
```

Recommendation

It is recommended to review and implement the logic necessary for updating the `total_tokens` field to ensure it accurately reflects the remaining token amount throughout the presale lifecycle. Properly managing this state ensures that event emissions and logs provide reliable and meaningful information, supporting the correct operation of the presale contract.

ISA - Inefficient Space Allocation

Criticality	Minor / Informative
Location	processor/create_presale.rs#L95
Status	Unresolved

Description

The `presale_account` allocation includes an additional 1000 bytes on top of the space already required for the account's discriminator and data. This excessive allocation is inefficient and increases the cost of account creation on Solana. Allocating more space than necessary can lead to higher rent fees and resource wastage, especially in an environment where cost-efficiency is crucial.

```
#[account(
        init_if_needed,
        payer = authority,
        seeds = [PREFIX],
        bump,
        space = 8 + PresaleAccount::INIT_SPACE + 1000
    )]
    pub presale_account: Box<Account<'info, PresaleAccount>>,
```

Recommendation

It is advisable to review and optimize the space calculation for the presale_account by removing any unnecessary padding. This change reduces the cost of account creation and ensures that resources are used more efficiently, aligning with best practices for Solana smart contract development.

LCM - Lamports Constant Misuse

Criticality	Minor / Informative
Location	processor/withdraw_token.rs#L13,33
Status	Unresolved

Description

The contract uses the `LAMPORTS_PER_SOL` constant from the Solana native token library to divide and format token amounts, which is not appropriate for non-native tokens. `LAMPORTS_PER_SOL` represents the conversion factor between lamports and SOL, the native currency of the Solana blockchain. Using this constant to divide token amounts may lead to incorrect calculations or misleading token balance displays, as token decimals for SPL tokens are typically defined by the mint and may vary. If the intention is to handle token decimals, the correct approach is to use the decimal places specified by the token mint.

```
msg!(
        "Presale token amount: {}, asked amount: {}",
        ctx.accounts.presale_account.total_tokens /
LAMPORTS_PER_SOL,
        token_amount / LAMPORTS_PER_SOL
    );

msg!(
        "After withdraw , presale token amount {}",
        ctx.accounts.presale_account.total_tokens /
LAMPORTS_PER_SOL
    );
```

Recommendation

Review and adjust the usage of `LAMPORTS_PER_SOL` to ensure accurate token calculations. If the goal is to account for token decimals, use the token's decimal value from the mint instead of the SOL-specific constant. This ensures that all token operations are consistent with the token's defined properties and prevents potential inaccuracies or confusion.

PPSI - Potential Presale Status Inconsistency

Criticality	Minor / Informative
Location	processor/update_presale.rs#L45
Status	Unresolved

Description

The `is_presale_ended` boolean in the PresaleAccount struct is manually updated by the contract owner, independent of the `end_time` timestamp. This design introduces a risk of inconsistency between the boolean flag and the actual status of the presale, as the flag may not accurately reflect whether the presale has ended based on the timeline. Lastly, the `is_presale_ended` isn't checked when purchasing tokens, in order to check if the presale has ended, as its name suggests. Such inconsistencies can lead to confusion and mismanagement of the presale's lifecycle, especially if the owner fails to update the status appropriately.

```
presale_account.end_time = args.end_time;
presale_account.maximum_buyable_amount =
args.maximum_buyable_amount;
presale_account.minimum_buyable_amount =
args.minimum_buyable_amount;
presale_account.is_presale_ended = args.is_presale_ended;
```

Recommendation

It is recommended to tie the is_presale_ended flag to the end_time timestamp or automate the status change based on the presale's timing logic. This approach ensures the status of the presale is always accurate and reduces the risk of errors or manipulation, thereby maintaining a reliable and consistent representation of the presale's state.

PCR - Program Centralization Risk

Criticality	Minor / Informative
Location	processor/create_presale.rs#L20 processor/update_presale.rs#L13
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 to specify the token that will be used in the presale, since only one token can be part of the sale and 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. Additionally, the program owner has the authority to update the parameters of the presale by calling the `update_presale` function.

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

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

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.

RAAC - Redundant Account Authority Constraints

Criticality	Minor / Informative
Location	processor/create_presale.rs#L78
Status	Unresolved

Description

The contract defines two constraints that essentially perform the same check: verifying that the `authority` key matches the `creator_account.reciever` key. Since the first constraint already ensures this match, the second constraint is redundant and does not add any security or functionality benefits. Redundancy in access control checks can lead to confusion and unnecessary complexity in the code.

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

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

Recommendation

It is suggested to remove the redundant constraint, simplifying the code while still ensuring the necessary security. This adjustment helps streamline the contract and improves maintainability without affecting the logic or security of the function.

RCP - Redundant Code Present

Criticality	Minor / Informative
Location	processor/update_presale.rs#L13 processor/purchase_token.rs#L33
Status	Unresolved

Description

The codebase contains unnecessary debug messages, such as `"Hello world!!!"`, and multiple commented-out code snippets. These elements do not contribute to the program's functionality and instead increase the complexity and clutter of the code. Unused or leftover debug statements can mislead developers or auditors, making the codebase harder to read and maintain. Similarly, commented-out code fragments suggest incomplete or outdated functionality, which can create confusion regarding the intended behavior of the program.

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

    require!(
        ctx.accounts.authority.key() ==
ctx.accounts.presale_account.authority,
        PresaleErrorCodes::Unauthorized
    );
    // let clock = Clock::get()?;
    // let current_unix_timestamp = clock.unix_timestamp as
u64;

    // to make presale editable at every time (make comment
below lines)
    // require!(
    //     current_unix_timestamp <
ctx.accounts.presale_account.start_time,
    //     PresaleErrorCodes::PresaleHasStarted
    // );
    ...
}

msg!("Hello wolrd!!!");

// ctx.accounts.presale_account.total_tokens -= token_amount;
```

Recommendation

It is recommended to remove all unnecessary debug messages and clean up any commented-out code that is not intended for future use. If any of the commented-out logic is necessary, it should be properly implemented and documented. Cleaning up the code improves clarity, reduces potential misunderstandings, and ensures that the contract is ready for deployment and auditing without extraneous elements.

UAD - Unused Account Declaration

Criticality	Minor / Informative
Location	processor/update_presale.rs#L79
Status	Unresolved

Description

The `update_presale_handler` function includes the `presale_program_data` account, which is not used anywhere in the logic of the function. Defining and allocating space for an account that is not utilized increases the complexity of the code and may lead to unnecessary costs associated with the function's execution. Moreover, maintaining unused accounts can cause confusion for developers and auditors, making the code harder to understand and review.

```
#[account(
        init_if_needed,
        seeds = [PREFIX, PROGRAM_DATA],
        bump,
        payer = authority,
        space = 8 + PresaleProgramData::INIT_SPACE
    )]
    pub presale_program_data: Box<Account<'info,
PresaleProgramData>>,
```

Recommendation

It is recommended to remove the `presale_program_data` account from the function if it is not required. Simplifying the account setup ensures the code is efficient and easier to maintain while also reducing potential resource overhead.

UEC - Unused Error Codes

Criticality	Minor / Informative
Location	error.rs#L4
Status	Unresolved

Description

There are error codes that are not used anywhere in the programs. Maintaining unused error codes can introduce confusion and may give the false impression that specific access control mechanisms are implemented when they are not. It also increases the complexity of understanding the contract and poses a risk of errors in future updates if developers mistakenly believe this error code is actively used.

```
#[error_code]
pub enum PresaleErrorCodes {
    ...
    #[msg("Insufficient funds")]
    InsufficientFunds,
    #[msg("Invalid amount")]
    InvalidAmount,
    ...
    #[msg("presale has been ended")]
    PresaleEnded,
    #[msg("Invalid Token Price")]
    InvalidTokenPrice,
    ...
    #[msg("Error during calculating the price ")]
    CalculationError,
    ...
    #[msg("overflow while calculating payment")]
    Overflow,
    ...
    #[msg("Presale has been started")]
    PresaleHasStarted
}
```

Recommendation

Review the contract to determine if there is an intended use for those error codes. If they are required, ensure that they are correctly implemented where needed. Otherwise, if there is no need for those error codes, consider removing them to keep the codebase clean and maintainable. Clear and concise error handling contributes to a more efficient and secure smart contract design.

USF - Unused Struct Field

Criticality	Minor / Informative
Location	state.rs#L5
Status	Unresolved

Description

The `PresaleAccount` struct contains a field named `token_price_in_lamports` that is not utilized anywhere in the codebase. Including unused fields in account structures increases the size of the account, leading to higher storage costs and inefficient use of resources. This practice can also lead to confusion for developers or auditors, as the presence of an unused field suggests that there might be missing or incomplete logic related to the presale functionality.

```
#[account]
#[derive(Default, InitSpace, Debug)]
pub struct PresaleAccount {
    pub authority: Pubkey,
    pub start_time: u64,
    pub end_time: u64,
    pub minimum_buyable_amount: u64,
    pub maximum_buyable_amount: u64,
    pub token_price_in_lamports: u64,
    pub total_tokens: u64,
    pub is_presale_ended: bool,
    pub bump: u8,
}
```

Recommendation

It is recommended to remove the `token_price_in_lamports` field if it is not needed. This change optimizes the account size, reduces storage costs, and clarifies the purpose and design of the account structure, making the codebase cleaner and easier to maintain.

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