Cyberscope

Audit Report

Anso Finance

May 2025

Repository :

https://github.com/Quantum-Bases/solana-ico-contracts/tree/Anti-whale-updated

Commit : 2231821b4c51c80d2e41ef7e9d0e23f4a48e51a9

Audited by © cyberscope

Table of Contents

Table of Contents 1
Risk Classification 3
Review 4
Audit Updates 4
Source Files 4
Overview 5
Initialization 5
Token Purchase Mechanism 5
Vesting System 5
Payment Token Management 6
Stage Progression 6
Administrative Controls 6
State Management 6
Error Handling 6
Findings Breakdown 7
Diagnostics 8
USA - Unchecked Sale Amount 10
Description 10
Recommendation 10
UTPA - Unverified Team Payment Account 11
Description 11
Recommendation 12
UUPA - Unverified User Payment Account 13
Description 13
Recommendation 14
ICE - Inconsistent Contribution Estimation 15
Description 15
Recommendation 15
MRO - Missing Refund Operation 16
Description 16
Recommendation 16
CCR - Contract Centralization Risk 17
Description 17
Recommendation 18
EPT - Early Purchase Timestamp 19
Description 19
Recommendation 19
FPM - Fixed Pricing Mechanism 20
Description 20

Recommendation 20
FTD - Fixed Token Decimals 21
Description 21
Recommendation 21
ICSP - Inconsistent Cross Stage Pricing 22
Description 22
Recommendation 22
INPT - Inconsistent Native Token Payment 23
Description 23
Recommendation 23
MCM - Misleading Comment Messages 24
Description 24
Recommendation 24
OTP - Overwritten Token Price 25
Description 25
Recommendation 25
PPPE - Purchase Past Presale End 26
Description 26
Recommendation 26
RPF - Revokable Presale Finalization 27
Description 27
Recommendation 27
TSI - Tokens Sufficiency Insurance 28
Description 28
Recommendation 28
UUI - Unsanitized User Input 29
Description 29
Recommendation 29
Summary **30**
Disclaimer **31**
About Cyberscope **32**

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

Repository	https://github.com/Quantum-Bases/solana-ico-contracts/tree/Anti-whale-updated
Commit	2231821b4c51c80d2e41ef7e9d0e23f4a48e51a9

Audit Updates

Initial Audit	04 Jun 2025

Source Files

Filename	SHA256
lib.rs	6abd6da27c583db3effcda3b8eaeef9e4fdb29d27820ba72d82ab7a4fa8600ad

Overview

The ANSO token sale contract implements a presale mechanism on Solana, designed to facilitate token distribution with a multi-stage pricing structure and vesting schedule. The protocol enables the presale of a token with predetermined pricing stages, and vesting rules, while enabling users to purchase tokens using various payment methods. The system includes features for payment token management, vesting schedules, and automated price adjustments through stage progression. This creates a controlled and structured way for the distribution of tokens while ensuring long-term alignment through vesting mechanisms.

Initialization

The protocol begins with the initialization of a `sale` account that stores global parameters and settings. This initialization sets up crucial values such as payment token addresses, funding wallet, token prices, and cap parameters. The sale account also stores the owner's address and payment token configurations, establishing the administrative structure of the protocol.

Token Purchase Mechanism

Users can purchase tokens through the `buy_tokens` function, which implements a multi-stage pricing mechanism. The system calculates token amounts based on the current stage price, verifies payment token validity, and enforces purchase limits. The purchase process includes vesting schedule initialization, with 25% of tokens unlocked immediately after the purchase is concluded and the remaining 75% vested over three months. The system automatically progresses through pricing stages as tokens are sold, with each stage having a predetermined price increase.

Vesting System

The protocol implements a comprehensive vesting system where purchased tokens are subject to a time-based release schedule. The initial 25% of tokens are unlocked once the presale is concluded, while the remaining 75% are vested over three months. This mechanism ensures long-term holder alignment and prevents immediate token release. The system tracks claimed amounts and allows users to claim their vested tokens once they become available.

Payment Token Management

The protocol supports multiple payment tokens. Each payment token is configured with its own mint address, decimals, and active status. The system allows administrators to toggle payment token status and add new payment tokens up to a maximum of three.

Stage Progression

The protocol implements an automated stage progression system where token prices increase as more tokens are sold. Each stage has a predetermined price and token allocation. The system automatically advances to the next stage when the current stage's token allocation is exhausted, with prices increasing per stage.

Administrative Controls

The platform owner has significant administrative capabilities within the protocol. They can update sale status, modify payment token configurations, adjust cap parameters, and control the presale end status. The owner can also toggle payment token status and update soft and hard caps as needed.

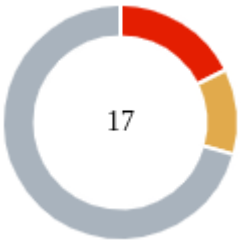
State Management

The protocol maintains several types of state accounts to track different aspects of the system. The `Sale` account stores global sale parameters and settings, while `VestingAccount` manages individual user vesting schedules. `UserPurchaseTracker` tracks user-specific purchase data, and various token accounts handle token balances.

Error Handling

The protocol implements comprehensive error handling through a custom error enum that covers various failure scenarios. These include payment token validation, arithmetic overflow protection, authorization checks, and business logic validation. The error system ensures robust operation and clear feedback for users and administrators.

Findings Breakdown



Critical	3
Medium	2
Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	2	0	0	0
Minor / Informative	12	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	USA	Unchecked Sale Amount	Unresolved
Critical	UTPA	Unverified Team Payment Account	Unresolved
Critical	UUPA	Unverified User Payment Account	Unresolved
Medium	ICE	Inconsistent Contribution Estimation	Unresolved
Medium	MRO	Missing Refund Operation	Unresolved
Minor / Informative	CCR	Contract Centralization Risk	Unresolved
Minor / Informative	EPT	Early Purchase Timestamp	Unresolved
Minor / Informative	FPM	Fixed Pricing Mechanism	Unresolved
Minor / Informative	FTD	Fixed Token Decimals	Unresolved
Minor / Informative	ICSP	Inconsistent Cross Stage Pricing	Unresolved
Minor / Informative	INPT	Inconsistent Native Payment Token	Unresolved
Minor / Informative	MCM	Misleading Comment Messages	Unresolved
Minor / Informative	OTP	Overwritten Token Price	Unresolved

●	PPPE	Purchase Past Presale End	Unresolved
●	RPF	Revokable Presale Finalization	Unresolved
●	TSI	Tokens Sufficiency Insurance	Unresolved
●	UUI	Unsanitized User Input	Unresolved

USA - Unchecked Sale Amount

Criticality	Critical
Location	lib.rs#L267
Status	Unresolved

Description

The contract implements the `buy_tokens` method to enable token purchases. When called, the function checks if the `sale_anso_account` maintains the required amount of tokens to process the sale. However, the contract fails to account for tokens that have already been sold. This inconsistency could result in a situation where tokens are sold without being available in the program's reserves, potentially leading to loss of funds.

```
require!(
ctx.accounts.sale_anso_account.amount >= amount,
CustomError::InsufficientSaleTokens
);
```

Recommendation

The contract should ensure it maintains reserves for all tokens processed through the sale mechanism. This would guarantee operational consistency and enhance user trust.

UTPA - Unverified Team Payment Account

Criticality	Critical
Location	lib.rs#L191
Status	Unresolved

Description

The contract implements the `buy_tokens` method to enable token purchases. When called, the function checks if the specified payment token is valid for the current sale. Once all checks are passed, the total_cost for the purchase is calculated in terms of the payment_token .

The contract then transfers the corresponding amount from the `user_payment_account` to the `team_payment_account` . However, it does not verify the `team_payment_account` actually reflects the teams' wallet token. As a result, users can complete the purchase by transferring tokens to themselves.

```
let team_payment_account = match
ctx.accounts.team_payment_token_account.as_ref() {
Some(account) => account,
None => return Err(CustomError::InvalidTeamPaymentAccount.into()),
};
```

```
anchor_spl::token::transfer(
CpiContext::new(
ctx.accounts.token_program.to_account_info(),
anchor_spl::token::Transfer {
from: user_payment_account.to_account_info(),
to: team_payment_account.to_account_info(),
authority: ctx.accounts.user.to_account_info(),
},
),
total_cost,
)?;
```

Recommendation

The team should implement validations to ensure that the team payment account corresponds to the account held by the team.

UUPA - Unverified User Payment Account

Criticality	Critical
Location	lib.rs#L185
Status	Unresolved

Description

The contract implements the `buy_tokens` method to enable token purchases. When called, the function checks if the specified payment token is valid for the current sale. Once all checks are passed, the `total_cost` for the purchase is calculated in terms of the `payment_token` .

The contract then transfers the corresponding amount from the `user_payment_account` to the `team_payment_account` . However, it does not verify whether the `user_payment_account` reflects the correct token. As a result, users can complete the purchase using tokens that may hold significantly less value than expected.

```
let user_payment_account = match
ctx.accounts.user_payment_token_account.as_ref() {
Some(account) => account,
None => return Err(CustomError::InvalidPaymentToken.into()),
};
```

```
anchor_spl::token::transfer(
CpiContext::new(
ctx.accounts.token_program.to_account_info(),
anchor_spl::token::Transfer {
from: user_payment_account.to_account_info(),
to: team_payment_account.to_account_info(),
authority: ctx.accounts.user.to_account_info(),
},
),
total_cost,
)?;
```

Recommendation

The team should implement validations to ensure that the user payment account corresponds to the token whitelisted by the contract for the payment.

ICE - Inconsistent Contribution Estimation

Criticality	Medium
Location	lib.rs#L160
Status	Unresolved

Description

The contract uses a fixed price for all tokens within each stage and the tokens purchased up to that moment to calculate the `total_usdt_value` contributed by a user. If a user contributes across multiple stages, the `total_usdt_value` is updated during the last call by multiplying the total tokens purchased across all stages by the price of the final stage. As a result, the contract may overestimate the user's total contribution since tokens from previous stages are multiplied by the new stage's price. This could prevent users from contributing up to their desired amount.

```
let total_usdt_value = {
let total_tokens_u128 = total_tokens_after_purchase as u128;
let price_u128 = sale.token_price as u128;
let result = (total_tokens_u128 * price_u128) / 1_000_000;
if result > u64::MAX as u128 {
return Err(CustomError::ArithmeticOverflow.into());
}
result as u64
};
require!(total_usdt_value <= 15_000_000_000,
CustomError::ExceedsMaxUsdtLimit);
```

Recommendation

The team should ensure that tokens purchased during each stage are accounted for using the respective price of that stage. This approach will maintain consistency and accurately reflect the actual funds contributed by users.

MRO - Missing Refund Operation

Criticality	Medium
Location	lib.rs#L482
Status	Unresolved

Description

The contract implements a sale mechanism with both a soft cap and a hard cap. However, it does not use these values to determine whether the presale has concluded successfully or failed to reach the hard cap. In cases where the hard cap is not met, the contract should facilitate refunds to users. Currently, tokens are non-refundable, and the presale can only be finalized by the owner regardless of the contributed amount. This may result in funds being locked indefinitely until the presale is concluded.

```
sale.is_presale_ended = is_end;
```

Recommendation

It is recommended to implement a mechanism that allows users to receive refunds if the hard cap is not reached within a reasonable timeframe. This would help build user trust in the sale mechanism.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	lib.rs#L482,534
Status	Unresolved

Description

The contract'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 owner has central control over the duration and the limits of the presale, potentially impacting participating users.

```
pub fn update_caps(
ctx: Context<UpdateSale>,
soft_cap: Option<u64>,
hard_cap: Option<u64>,
) -> Result<()> {
...
}
```

```
pub fn update_sale_status(
ctx: Context<UpdateSale>,
is_active: bool,
is_end: bool,
) -> 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 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.

EPT - Early Purchase Timestamp

Criticality	Minor / Informative
Location	lib.rs#L275
Status	Unresolved

Description

The sale mechanism employs a vesting schedule for tokens purchased through the contract. The `vesting` account records the `purchase_timestamp` when a user makes their initial purchase, but this timestamp is not updated for subsequent purchases. Consequently, tokens acquired in later transactions may be released earlier than intended, allowing users to access large quantities of tokens ahead of the intended vesting timeline.

```
let seconds_since_purchase = current_timestamp -
vesting.purchase_timestamp;
```

Recommendation

The implementation should be monitored to ensure it aligns with the intended token release schedule. Alternatively, the team could consider using a single reference point for token releases at the end of the presale.

FPM - Fixed Pricing Mechanism

Criticality	Minor / Informative
Location	lib.rs#L133
Status	Unresolved

Description

The contract facilitates the sale of tokens by supporting multiple whitelisted payment tokens. It calculates the `total_cost` to be paid based on a fixed `token_price`. For consistent operation, this implies that all units of whitelisted tokens must hold the same monetary value. If a token with lesser underlying value is used, the consistency of the contract would be disrupted.

```
let result = (amount_u128 * price_u128) / 1_000_000;
```

Recommendation

The team must ensure the consistency of operations by making sure all whitelisted tokens maintain the same underlying value.

FTD - Fixed Token Decimals

Criticality	Minor / Informative
Location	lib.rs#L44
Status	Unresolved

Description

The smart contract facilitates token-based payments for the presale using designated payment addresses. For each accepted token, a `PaymentToken` structure is defined during contract initialization. The contract assumes a fixed decimal precision of 9 for all tokens. However, this can lead to inconsistencies if tokens with a different number of decimals are used, potentially affecting payment calculations and the overall presale process.

```
let token = PaymentToken {
mint: addr,
is_active: true,
decimals: 9,
is_sol: false,
};
```

Recommendation

It is recommended to dynamically retrieve and store the decimal value of each token to ensure that recorded payment information remains consistent with the token's actual configuration. Alternatively, the contract should enforce restrictions that only allow tokens with the expected number of decimals to be used, thereby preventing discrepancies during the presale process.

ICSP - Inconsistent Cross Stage Pricing

Criticality	Minor / Informative
Location	lib.rs#L125
Status	Unresolved

Description

Users can purchase tokens through the `buy_tokens` method. When a certain limit of purchased tokens is exceeded, the contract advances to the next presale stage and increases the token price. However, if a user's purchase causes the current stage's limit to be exceeded, tokens from the next stage are still purchased at the current (lower) stage price. This behavior can lead to inconsistencies in the presale process and cause the contract's internal state to deviate.

```
let total_cost = {
let amount_u128 = amount as u128;
let price_u128 = sale.token_price as u128;
let result = (amount_u128 * price_u128) / 1_000_000;
if result > u64::MAX as u128 {
return Err(CustomError::ArithmeticOverflow.into());
}
result as u64
};
```

Recommendation

The team is advised to ensure that sold tokens are consistently purchased at the expected price ratio. This can be achieved by modifying the current implementation to track the portion of tokens purchased during each stage.

INPT - Inconsistent Native Token Payment

Criticality	Minor / Informative
Location	lib.rs#L52
Status	Unresolved

Description

The contract, during its initialization phase, supports the use of native Solana tokens for the presale. Nevertheless, during the execution of the actual `buy_tokens` method, payment tokens in the form of native currency are excluded. This inconsistency introduces redundancy in the contract, which could obscure code readability and maintainability.

```
if sol_price.is_some() {
let sol_token = PaymentToken {
mint: anchor_lang::system_program::ID,
is_active: true,
decimals: 9,
is_sol: true,
};
all_payment_tokens.push(sol_token);
}
```

```
require!(!payment_token.is_sol, CustomError::InvalidPaymentToken);
```

Recommendation

The team is advised to eliminate redundancies by ensuring greater consistency in the contract's logic. Specifically, the contract could explicitly support the use of native tokens throughout the presale process or exclude them during its initialization. Aligning this behavior will improve code readability, reduce confusion, and enhance maintainability.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	lib.rs#L131
Status	Unresolved

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code. As a result, the users will not comprehend the source code's actual implementation.

```
// Then divide by 10^9 to account for ANSO's 9 decimals
```

```
let result = (amount_u128 * price_u128) / 1_000_000;
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages.

OTP - Overwritten Token Price

Criticality	Minor / Informative
Location	lib.rs#L75
Status	Unresolved

Description

During the initialization process, the contract accepts a presale token and its corresponding sale value. However, the sale value is later overwritten with a fixed value of `3_550` for all tokens. This creates redundancy, as the user-provided value becomes unnecessary and could be omitted. Removing such redundancies would improve code size and enhance contract readability.

```
sale.token_price = 3_550;
```

Recommendation

The team is advised to eliminate redundancies by ensuring greater consistency in the contract's logic. Aligning code logic will improve code readability, reduce confusion, and enhance maintainability.

PPPE - Purchase Past Presale End

Criticality	Minor / Informative
Location	lib.rs#L305
Status	Unresolved

Description

The contract implements the `is_presale_ended` control variable, which is set by the owner to indicate the end of the presale. However, during the purchase phase, this variable can be bypassed if the hard cap has not been reached. As a result, setting `is_presale_ended` to true may not be sufficient to prevent users from purchasing tokens after the presale is intended to have finished.

```
if is_presale_ended {
let initial_unlock_amount = match amount.checked_mul(25) {
Some(val) => val / 100,
None => 0,
};
...
}
```

Recommendation

The contract should implement the necessary checks to prevent purchases after the presale has ended and during the claim phase. This can be achieved by validating the `is_presale_ended` variable at the beginning of the purchase function.

RPF - Revokable Presale Finalization

Criticality	Minor / Informative
Location	lib.rs#L482
Status	Unresolved

Description

The contract allows the owner to finalize the presale. However, the owner can also re-enable the presale after it has been finalized. This behavior, combined with the `buy_tokens` and `claim_tokens` functions, may lead to inconsistencies in the contract's internal state and in the distribution of funds.

```
pub fn update_sale_status(
ctx: Context<UpdateSale>,
is_active: bool,
is_end: bool,
) -> Result<()> {
require!(
ctx.accounts.owner.key() == ctx.accounts.sale.owner,
CustomError::UnauthorizedAdmin
);
let sale: &mut _ = &mut ctx.accounts.sale;
sale.is_active = is_active;
sale.is_presale_ended = is_end;
msg!("Sale status updated: {}", is_active);
Ok(())
}
```

Recommendation

The team is advised to ensure that once the presale is finalized, it cannot be re-enabled. This restriction would prevent inconsistencies in the contract state, ensure optimal functionality, and enhance users' trust.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	lib.rs#L11
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the tokens are to be provided to the contract from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
#[account(
init,
payer = owner,
associated_token::mint = asno_mint,
associated_token::authority = sale
)]
pub sale_anso_account: Account<'info, TokenAccount>,
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

UUI - Unsanitized User Input

Criticality	Minor / Informative
Location	lib.rs#L63
Status	Unresolved

Description

The contract accepts user information without validating that it conforms to the proper structure. Specifically, it does not ensure that variables are properly assigned and do not retain their default state.

```
sale.funding_wallet = funding_wallet;
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

Summary

The ANSO token sale contract is a token distribution protocol designed to manage the sale and release of tokens. It incorporates features such as time-based vesting schedules, a multi-stage pricing mechanism, support for multiple payment tokens, automated stage progression, purchase limits, and comprehensive error handling. 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