SECURITY AUDIT REPORT

for

Avix Presale & Vesting

Prepared By: Xiaomi Huang

PeckShield

October 21, 2025

Document Properties

Client	Avix
Title	Security Audit Report
Target	Avix Presale & Vesting
Version	1.0
Author	Xuxian Jiang
Auditors	Matthew Jiang, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	October 21, 2025	Xuxian Jiang	Final Release
1.0-rc1	October 15, 2025	Xuxian Jiang	Release Candidate #1

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the `Avix`'s presale/vesting contracts, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Avix

`Avix` is an ecosystem where anyone can pay, trade, stake, and launch safely—all on the high-performance `Solana` blockchain. And the protocol token `AVIX` is designed around payments and rewards: spend with the card, stake for yield, and trade. This audit covers the presale and vesting contracts on `Solana`. The basic information of audited contracts is as follows:

Table 1.1: Basic Information of Avix Presale & Vesting

Item	Description
Name	Avix
Type	Solidity & Solana
Language	EVM & Rust
Audit Method	Whitebox
Latest Audit Report	October 21, 2025

In the following, we show the Git repositories of reviewed files and the commit hash values used in this audit.

- https://github.com/cowchainworkspace/avix-presale.git (7b78430)
- https://github.com/cowchainworkspace/avix-vesting.git (1121523)

And here are the commit IDs after all fixes for the issues found in the audit have been checked in:

- https://github.com/cowchainworkspace/avix-presale.git (de1b65d)
- https://github.com/cowchainworkspace/avix-vesting.git (a4547a1)

1.2 About PeckShield

PeckShield Inc. [7] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [6]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact, and can be accordingly classified into four categories, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [5], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings. Moreover, in case there is an issue that may affect an active protocol that has been deployed, the public version of this report may omit such issue, but will be amended with full details right after the affected protocol is upgraded with respective fixes.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 Findings

2.1 Summary

Here is a summary of our findings after analyzing the implementation of the `Avix`'s presale/vesting contracts. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	1	■
Medium	1	■
Low	1	■
Informational	0	
Total	3	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 high-severity vulnerability, 1 medium-severity vulnerability, and 1 low-severity vulnerability.

Table 2.1: Key Audit Findings

ID	Severity	Title	Category	Status
PVE-001	High	Revisited SOL Amount Calculation in buy_with_sol.rs	Business Logic	Resolved
PVE-002	Medium	Possible Inconsistent Use of round_stats Account	Business Logic	Resolved
PVE-003	Low	Redundant Code Removal	Coding Practices	Resolved

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 | Detailed Results

3.1 Revisited SOL Amount Calculation in buy_with_sol.rs

- ID: PVE-001
- Severity: High
- Likelihood: High
- Impact: High
- Target: `buy_with_sol`
- Category: Business Logic [4]
- CWE subcategory: CWE-837 [2]

Description

The `Avix`'s presale contract on `Solana` allows users to participate in the token presale with three types of payment tokens: `SOL`, `USDC`, and `USDT`. In the process of calculating the required `SOL`s for the intended token presale, we notice current implementation calculates an incorrect `SOL` amount.

In the following, we show the code snippet of the related function, i.e., `buy_with_sol::handler()`. The purpose of this code snippet is to compute the required `SOL`s for the intended amount of token presale. Our analysis shows it does not properly take into account the `SOL-USD` price feed's built-in decimals. With that, the required `SOL` amount or `total_sol_lamports` (line 146) should be computed as follows, `total_cost_usd * 10^SOL_DECIMALS / (sol_price_usd * 10^(12+price_data.exponent))`, not current `total_cost_usd * 10^SOL_DECIMALS / (sol_price_usd)` (lines 146-150).

```
138     let sol_price_usd = price_data.price as u128;
139     let token_price_usd = current_round.token_price_usd as u128;
140
141     // Calculate total cost: (token_amount * token_price_usd) / sol_price_usd
142     let total_cost_usd = (token_amount as u128)
143         .checked_mul(token_price_usd)
144         .ok_or(ErrorCode::ArithmeticOverflow)?; // USD cost * 10^12
145
146     let total_sol_lamports = total_cost_usd
147         .checked_mul(10u128.pow(SOL_DECIMALS as u32))
148         .ok_or(ErrorCode::ArithmeticOverflow)?
149         .checked_div(sol_price_usd)
150         .ok_or(ErrorCode::ArithmeticOverflow)? as u64;
```

```

    require_gt!(total_sol_lamports, 0);
```

Listing 3.1: `buy_with_sol::handler()`

Recommendation Revisit the above-mentioned routine to properly compute the required `SOL` amount for the intended token amount purchase.

Status The issue has been fixed in the following commit: de1b65d.

3.2 Possible Inconsistent Use of round_stats Account

- ID: PVE-002
- Severity: Medium
- Likelihood: Low
- Impact: Medium
- Target: `buy_with_sol`, `buy_with_usd`
- Category: Business Logic [4]
- CWE subcategory: CWE-837 [2]

Description

The token presale in `Avix` may be organized in multiple rounds and each round will keep its round-specific states in `round_stats`. In the analysis of its accounting, we notice an issue that may lead to an inconsistent update among different rounds.

In the following, we show the definition of the related `BuyWithSol` accounts. We notice the `round_stats` account is specific to current presale round that is consistent with the presale configuration in `presale_config.current_round` (line 52). However, the respective handler may advance the current round if the round becomes inactive and exceeds the round's `soft_cap`. As a result, if current round is instead advanced to next round, the associated `round_stats` should also be updated to the one related to next round. However, current implementation still updates the old `round_stats`, not the new one related to the new round. Note this issue also affects another instruction in `BuyWithUsd`.

```
12 #[derive(Accounts)]
13 pub struct BuyWithSol<'info> {
14     /// CHECK: Program-derived admin PDA for presale management
15     #[account(
16         seeds = [ADMIN_SEED.as_bytes()],
17         bump
18     )]
19     pub admin_pda: UncheckedAccount<'info>,
20 
21     /// CHECK: Treasury wallet that receives funds
22     #[account(mut)]
23     pub treasury: UncheckedAccount<'info>,
24 
```

```
    #[account(mut)]
    pub user: Signer<'info>,

    #[account(
        has_one = treasury @ ErrorCode::Unauthorized,
        has_one = admin_pda @ ErrorCode::Unauthorized,
        seeds = [PRESALE_SEED.as_bytes()],
        bump = presale_config.bump
    )]
    pub presale_config: Account<'info, PresaleConfig>,

    #[account(address = SOL_USD_PRICE_FEED_ACCOUNT)]
    pub price_update: Account<'info, PriceUpdateV2>,

    #[account(
        init_if_needed,
        payer = user,
        space = UserContribution::DISCRIMINATOR.len() + UserContribution::INIT_SPACE,
        seeds = [USER_CONTRIBUTION_SEED.as_bytes(), user.key().as_ref()],
        bump
    )]
    pub user_contribution: Account<'info, UserContribution>,

    #[account(
        init_if_needed,
        payer = user,
        space = RoundStats::DISCRIMINATOR.len() + RoundStats::INIT_SPACE,
        seeds = [ROUND_STATS_SEED.as_bytes(), presale_config.current_round.to_le_bytes()
            .as_ref()],
        bump
    )]
    pub round_stats: Account<'info, RoundStats>,

    pub system_program: Program<'info, System>,
}
```

Listing 3.2: The `BuyWithSol` Accounts

Recommendation Revise the above-mentioned routines to properly update the intended `round_stats` account, which is always consistent with current presale round.

Status The issue has been fixed in the following commit: de1b65d.

3.3 Redundant Code Removal

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: Multiple Contracts
- Category: Coding Practices [3]
- CWE subcategory: CWE-1126 [1]

Description

The Avix presale/vesting contracts are well-engineered and make extensive use of the Anchor framework to facilitate the program organization. In the meantime, we also notice the definition or inclusion of redundant state/code, which can be safely removed.

In the following, we show the list of defined error code in the Avix presale program. We notice it includes a number of unused ones, including PresaleNotStarted, BelowMinContribution, StageSupplyExhausted, NoContributionsToRefund, RefundsNotAvailable, PresalePaused, and PresaleNotPaused .

```
#[error_code]
pub enum ErrorCode {
    #[msg("Presale has not started yet")]
    PresaleNotStarted,
    #[msg("Presale has already ended")]
    PresaleEnded,
    #[msg("Invalid round configuration")]
    InvalidRoundConfig,
    #[msg("Contribution amount is below minimum")]
    BelowMinContribution,
    #[msg("Contribution amount exceeds maximum per wallet")]
    ExceedsMaxContribution,
    #[msg("Stage token supply exhausted")]
    StageSupplyExhausted,
    #[msg("Global hard cap reached")]
    HardCapReached,
    #[msg("Soft cap not reached, refunds available")]
    SoftCapNotReached,
    #[msg("User has no contributions to refund")]
    NoContributionsToRefund,
    #[msg("Refunds not available yet")]
    RefundsNotAvailable,
    #[msg("Invalid payment token")]
    InvalidPaymentToken,
    #[msg("Arithmetic overflow")]
    ArithmeticOverflow,
    #[msg("Unauthorized access")]
    Unauthorized,
    #[msg("Presale is currently paused")]
```

```
    PresalePaused,
    #[msg("Presale is not paused")]
    PresaleNotPaused,
}
```

Listing 3.3: The List of Defined `error` Code

Moreover, it has defined a constant `ROUND_SEED`, which is not used either. The vesting program has a built-in `VestingInfo` account, which includes a field named `authority`, which is not currently used.

Recommendation Revise the above-mentioned routines to ensure the redundant code is removed.

Status The issue has been fixed in the following commits: de1b65d and a4547a1.

4 Conclusion

In this audit, we have analyzed the design and implementation of the `Avix`'s presale/vesting contracts with the goal of creating an ecosystem where anyone can pay, trade, stake, and launch safely—all on the high-performance `Solana` blockchain. It has a protocol token named `AVIX` designed around payments and rewards: spend with the card, stake for yield, and trade. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

References

[1] MITRE. CWE-1126: Declaration of Variable with Unnecessarily Wide Scope. https://cwe.mitre.org/data/definitions/1126.html.

[2] MITRE. CWE-837: Improper Enforcement of a Single, Unique Action. https://cwe.mitre.org/data/definitions/837.html.

[3] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[4] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[5] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[6] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[7] PeckShield. PeckShield Inc. https://www.peckshield.com.