Cyberscope

A TAC Security Company

Audit Report

Fair

November 2025

Sha256

a628e12427066634e24db6e8f4de4054138efd4f740f2c386218c88a21c156b9

bd41716e4da4817a1713f72269beb59c2efcc50a18e6319d6dc937a23f94e02f

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

Audit Updates

Initial Audit	05 Sep 2025 https://github.com/cyberscope-io/audits/blob/main/fair/v1/audit.pdf
Corrected Phase 2	23 Oct 2025 https://github.com/cyberscope-io/audits/blob/main/fair/v2/audit.pdf
Corrected Phase 3	03 Nov 2025

Source Files

Filename	SHA256
lib.rs	a628e12427066634e24db6e8f4de4054138efd4f740f2c386218c88a21c156b9
ARCHITECTURE.md	bd41716e4da4817a1713f72269beb59c2efcc50a18e6319d6dc937a23f94e02f

Overview

The **FairTokenVault** contract manages a controlled token sale and redemption system on Solana, ensuring transparent and secure handling of both **tokens** and **SOL**. It allows users to buy tokens using SOL and redeem them back on a 1:1 basis, with strict rules enforcing correct account relationships, authority management, and safe settlement. The program leverages **Program Derived Addresses (PDAs)** to hold SOL and token assets securely, using deterministic seeds for both mint authority and vault management. This design guarantees that all operations remain verifiable, consistent, and protected from unauthorized access.

Initialization Functionality

An **administrator** initializes the system by providing a pre-created SPL token mint that meets strict conditions (e.g., zero supply, correct decimals, no freeze authority). During initialization, the program transfers the mint's **MintTokens** authority from the admin to a dedicated PDA, establishes a **SOL vault PDA** and **token vault PDA**, and persists all configuration parameters in a `Config` account. This process creates a predictable and tamper-resistant foundation for token distribution, ensuring that only the program (via its PDAs) can mint or transfer tokens during the sale.

Token Purchase Functionality

The `buy_fair_token` instruction enables users to purchase tokens by sending **SOL** to the vault. Before finalization, tokens are minted directly to the user's token account under PDA authority. After finalization, purchases instead draw from the pre-minted tokens stored in the token vault. All transfers are performed through verified PDAs, guaranteeing that only the correct vaults and mints are used. The function also triggers automatic finalization once the sale period ends, locking the token supply and transitioning the system into post-sale mode.

Token Redemption Functionality

The `redeem_fair_token` instruction allows users to return tokens in exchange for **SOL**. Before finalization, tokens are **burned**, permanently reducing supply; after finalization, they are transferred back into the token vault. In both cases, SOL is securely transferred from the vault PDA to the redeemer using a **System Program CPI with PDA signer seeds**, ensuring safe and authorized payments. The contract also handles the edge case where the SOL vault balance reaches zero, allowing it to be automatically re-created when new SOL is later received.

Finalization Functionality

The **finalization mechanism** locks the token's mint authority and ensures that the total supply meets a defined minimum threshold. Once triggered—either automatically after the sale period or implicitly during a buy/redeem after expiry—the contract mints any shortfall of tokens into the vault to meet the minimum supply, then revokes the mint authority entirely. After finalization, token minting becomes impossible, SOL redemptions and token transfers are routed through the vault, and the token economy enters a fully immutable state.

Security Properties & Program Invariants

- No developer-only profit paths — No privileged minting/transfer/withdraw flows that benefit the deployer beyond what is available to any user.
- Perpetual 1:1 redemption — Program redeems at 1 lamport per base unit (DECIMALS=9) both pre- and post-finalization; no privileged actor can disable redemption.
- Pricing invariant is hard-coded — 1 lamport == 1 base unit; no runtime price parameter; no rounding.
- Sale window constraints — initialize(sale_end) must place sale_end within 45–90 days of current cluster time, enforced on-chain.
- Finalization semantics — On finalization, program may mint a shortfall to a token-vault SPL account to reach MIN_SUPPLY_TOKENS * 10^DECIMALS, then permanently revokes mint authority (MintTokens=None); after that, buys are served

strictly from token-vault inventory (no minting possible). Finalization is lazy/auto-triggered by the next buy or redeem after `sale_end`.

- Vault controls and solvency model — SOL vault is a system-owned PDA; SOL moves only via PDA signer seeds; no direct privileged withdraw path. Token vault is a program-owned PDA controlling the token-vault SPL account for post-finalization flows. During finalization the contract adds tokens without adding SOL, solvency for those tokens relies on future post-finalization buys bringing SOL into the vault.
- Burn/transfer authority behavior — Pre-finalization redeem: user burns from own ATA (authority=user), program pays SOL. Post-finalization redeem: user transfers to token-vault account (authority=user), program pays SOL. No hidden authority to burn from user accounts.
- Admin gating scope — Only initialize is address-gated; no admin-only withdraws or price changes.

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	5

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	MT	Mints Tokens	Acknowledged
	PAO	Potential Arbitrage Opportunities	Acknowledged
	ZVD	Zero-Balance Vault Deallocation	Acknowledged
	IMIC	Inconsistent Metadata Immutability Check	Unresolved
	CAC	Contract Architecture Clarifications	Unresolved

MT - Mints Tokens

Criticality	Minor / Informative
Location	lib.rs#L180
Status	Acknowledged

Description

Before finalization, users are able to mint tokens by calling the `buy_fair_token` function. As a result, the contract tokens will be highly inflated.

```rust
token::mint_to(cpi_ctx, lamports_sent)?;
```

Recommendation

The team could introduce a `MAX_SUPPLY` limit to ensure that tokens minted will not be more than a reasonable amount.

Team Update

The team has acknowledged that this is not a security issue and states:

During the initial sale window, minting is unlimited at 1 SOL per token to eliminate timing games, front-running, and whale monopolization. Setting a maximum supply as suggested would defeat these objectives.

After finalization, mint authority is revoked; supply becomes fixed (the vault may sell any remainder at 1 SOL) Your comment "the contract tokens will be highly inflated" is misleading - inflation implies devaluation, and all circulating tokens are fully backed 1:1 by SOL. A large amount of tokens sold would be a highly successful and desirable outcome.

PAO - Potential Arbitrage Opportunities

Criticality	Minor / Informative
Location	lib.rs#L154,245
Status	Acknowledged

Description

The contract allows users to exchange SOL for the Fair Token and vice versa through the `buy_fair_token` and `redeem_fair_token` functions at a fixed 1:1 ratio. While this ensures a stable exchange rate within the contract, it does not account for potential fluctuations of the token price on external markets or DEXs. As a result, users could exploit discrepancies between the fixed rate and the market price, creating arbitrage opportunities that could be profitable at the expense of the contract's reserves.

```rust
pub fn buy_fair_token(ctx: Context<BuyFairToken>, lamports_sent: u64) -> Result<()>

pub fn redeem_fair_token(ctx: Context<RedeemFairToken>, amount_to_redeem: u64) -> Result<()>
```

Recommendation

The team could consider implementing a dynamic pricing mechanism or integrate price oracles to adjust the on-chain rate, mitigating the risk of arbitrage and protecting the contract's reserves from being drained.

Team Update

The team has acknowledged that this is not a security issue and states:

If market < 1 SOL → buy on DEX, redeem at 1 SOL → price returns to the floor.

If market > 1 SOL (and vault has tokens) → buy from program at 1 SOL, sell higher → pressure back toward 1 SOL.

Once all program tokens have been sold, price may rise on secondary markets, but can never fall below 1 SOL more than transiently. This is intended behavior.

The auditors' suggestion to implement a dynamic pricing mechanism would be incompatible with the 1:1 redemption guarantee and integrating a price oracle would add immense and unnecessary complexity.

ZVD - Zero-Balance Vault Deallocation

Criticality	Minor / Informative
Location	lib.rs#L245
Status	Acknowledged

Description

The SOL vault is a PDA System account with zero data. When redemptions drain it to exactly 0 lamports, the runtime will reclaim the account at the end of the transaction. That doesn't lose the address—the PDA still derives to the same pubkey—but the account object no longer exists until someone sends lamports to it again. However if any later instruction attempts to read or mutate the vault before it's re-funded, the transaction will fail because the account no longer exists on-chain.

```rust
pub fn redeem_fair_token(ctx:
Context<RedeemFairToken>, amount_to_redeem: u64)
-> Result<()>
```

Recommendation

The team should ensure to always keep a small amount of lamport to the vault to ensure that cases such as described above are avoided.

Team Update

The team has acknowledged that this is not a security issue and states:

Tokens only exist when SOL is deposited; redemptions remove the same SOL. Redemptions can never fail as the sol vault always contains reserves sufficient to redeem every circulating token.

The rent reserve, held in the sol vault, is never part of the buy/redeem accounting, so it cannot be spent. Its presence ensures that the sol vault can never be entirely depleted.

IMIC - Inconsistent Metadata Immutability Check

Criticality	Minor / Informative
Location	lib.rs#L118
Status	Unresolved

Description

The initialization function introduces a verification step to ensure the mint's Metaplex metadata is immutable before the sale begins. It does this by checking only whether the `update_authority` field of the metadata account equals the System Program ID.

However, in the Metaplex Token Metadata standard, immutability can be represented in two ways:

1. Setting the `update_authority` to the System Program ID.
2. Setting the `is_mutable` flag to `false`.

The current implementation only checks for the first condition and ignores the `is_mutable` flag. As a result, this check will incorrectly reject valid immutable metadata if that metadata's immutability is enforced by having `is_mutable` set to `false`, while the `update_authority` field still points to the original creator.

```rust
require!(
    meta.update_authority == system_program::ID,
    ErrorCode::MetadataStillMutable
);
```

Recommendation

The team is advised to update the current immutability check to correctly account for both Metaplex conventions. The check should be modified to verify that either `is_mutable` is `false` *or* `update_authority` equals the System Program ID. This ensures the program properly enforces metadata immutability as intended and maintains compatibility with all standard Metaplex metadata formats.

CAC - Contract Architecture Clarifications

Criticality	Minor / Informative
Location	ARCHITECTURE.md#L13,17,24,43,48,52 lib.rs#L118,186,227,331
Status	Unresolved

Description

The `ARCHITECTURE.md` document serves to explain the contract's design, functionality, and key invariants. The following recommendations aim to enhance the document's accuracy, technical validity, and clarity, ensuring that readers can easily understand the contract's behavior, constraints, and security properties.

Section 1:

```shell
- Users **buy** during the initial sale by sending SOL. Pre-finalization the program **mints** to the buyer.
```

However, buys are also allowed post-finalization, but they transfer tokens from the token vault to the buyer (still 1:1 in base units).

```rust
if !finalized {
    ...
} else {
    ...
    token::transfer(cpi_ctx, lamports_sent)?;
```

```
    ...
}
```

```
Shell
- **Finalization:** when the sale ends, the
program may **mint a shortfall** to the token
vault account to bring the circulating supply up
to `MIN_SUPPLY_TOKENS * 10^9` if needed, then
**revokes mint authority** permanently and marks
the sale as finalized.
```

In the code, finalization is lazy triggered: it occurs inside the next `buy_fair_token` or `redeem_fair_token` call after `sale_end` at the end of the instruction. The document only states when the sale ends.

Additionally the circulating supply mentioned includes the tokens in the vault which are not necessarily circulating.

```
Rust
if !finalized && now > sale_end {
    finalize_sale(**args**)?;
}
```

Section 2:

```
Shell
- **Config PDA** — seeds: `["config"]`

Tracks mint, sale window, finalization flag,
counters, and bumps.
```

The contract does not store bumps. They are runtime-derived by Anchor.

Section 3:

```
Shell
- **`initialize(ctx, sale_end: i64)`**
    - Gated to the **ADMIN** address.
    - Validates that `sale_end` lies within **[45,
90] days** from the current slot time.

    - Sets up PDAs and records config.
```

The document does not state the intended behavior of the contract regarding the Metadata Update Authority during initialization.

```
Rust
require!(
    meta.update_authority == system_program::ID,
    ErrorCode::MetadataStillMutable
```

```
);
```

```
Shell
- **`buy_fair_token(ctx, lamports_sent: u64)`**
...
- **`redeem_fair_token(ctx, amount_to_redeem:
u64)`**

...
```

The document does not mention the lazy triggered finalization.

```
Rust
if !finalized && now > sale_end {
    finalize_sale(**args**)?;

}
```

Recommendation

By incorporating these refinements, the `ARCHITECTURE.md` document will provide a more precise, transparent, and reliable representation of the contract's behavior—facilitating easier auditing, integration, and long-term maintainability.

Summary

Fair Token contract implements an exchange 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 TAC 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.

A **TAC Security** Company

The Cyberscope team

cyberscope.io