verichains

SECURITY AUDIT OF

UNIZEN SOLANA AGGREGATOR PROGRAM

Public Report

Feb 11, 2025

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Solana	A decentralized blockchain built to enable scalable, user-friendly apps for the world.
SOL	A cryptocurrency whose blockchain is generated by the Solana platform.
Lamport	A fractional native token with the value of 0.000000001 SOL.
Program	An app interacts with a Solana cluster by sending it transactions with one or more instructions. The Solana runtime passes those instructions to program.
Instruction	The smallest contiguous unit of execution logic in a program.
Cross-program invocation (CPI)	A call from one smart contract program to another.
Anchor	A framework for Solana's Sealevel runtime providing several convenient developer tools for writing smart contracts.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Feb 11, 2025. We would like to thank the Unizen for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Unizen Solana Aggregator Program. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team found no vulnerabilities in the given version of Unizen Solana Aggregator Program, only some notes and recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Unizen Solana Aggregator Program

The Unizen Ecosystem is housing and aggregating trades across trusted first and third-party exchange modules. Unizen is currently powered by the below liquidity providers but is continuously adding on more sources of liquidity.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the Unizen Solana Aggregator Program. It was conducted on commit `653803a62fbe5c9ebb96c52d7cbeb4afec3eba10` from git repository link *https://github.com/unizen-io/unizen-solana-swap*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
5ba1ae04fc4b70e6bffef194fc94ffdac56ae61012c997e0a348afbb9632255e	./instructions/take_integrator_fee.rs
a83f8092499aa7dee5a0ef19b6c067bd6fcf91e178d7b7dd6e19b5a92f141a28	./instructions/swap_tokens_for_tokens.rs
3f2b04ca502723885e63c31c311e2cce57fdd55ffb142646820bf92cc5856ff8	./instructions/swap_sol_for_tokens.rs
f2ad6feff831689bbc4b2a0a04810a5558895909bf39edfb8d778f0159897fd4	./instructions/swap_tokens_for_sol.rs
4273e5d721585cfcce04003a9c4a7cd8f5ceb280237341ff51f6b03f66df6484	./instructions/mod.rs
2c4eb69d0e3ba16d31a137f72eb2cdcad3f4a992a589cedf0d8361fb1f3658da	./instructions/create_program_wsol_idempotent.rs
df600f18f9fc858034d2bf99d0a33a3d201679b6bc0c1c8a44ecd2264a45fd37	./instructions/close_program_wsol.rs
4bc036f7535ad3735587ec06018ce079e4b734732a74f24bd01346fdd1174fc5	./constants.rs
a88ffe7431f6b56099329c09ecc9a7dea776a9d1ea20b5a23011af63c4ab813b	./lib.rs
0f81c3e4777e4e6cb7a6ffc20816c6a585b1585a112578ff379e3f6bc9d273f1	./helpers.rs

24cc06d863b3b9bdbcf8f68f293e6272eab8f31c8bd09b74f73043de5c6f37eb	./errors.rs

1.3. Audit Methodology

Our security audit process for Solana smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the Solana smart contract:

- Arithmetic Overflow and Underflow
- Signer checks
- Ownership checks
- Rent exemption checks
- Account confusions
- Bump seed canonicalization
- Closing account
- Signed invocation of unverified programs
- Numerical precision errors
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.

SEVERITY LEVEL	DESCRIPTION
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Unizen acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Unizen understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Unizen agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Unizen will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Unizen, the final report will be considered fully accepted by the Unizen without the signature.

2. AUDIT RESULT

2.1. Overview

The Unizen Solana Aggregator Program was written in `Rust` programming language and `Anchor` framework.

This program is a Solana-based token swap aggregator that integrates with **Jupiter** to facilitate various types of token swaps while adding fee management capabilities.

- **Supported Swap Types:**
  - `SOL` to Token swaps and vice versa.
  - Token to Token swaps.
- **Fee Management:**
  - Supports fee collection for integrators.
  - Implements a revenue sharing model between **Unizen** and **integrators** through `fee_percent` and `share_percent` parameters.
- **Native SOL Handling:**
  - Automatically wraps and unwraps `SOL` to `WSOL` (Wrapped SOL) as needed.
  - Manages program-owned `WSOL` accounts for token to `SOL` swaps.
  - Includes idempotent `WSOL` account creation and cleanup.
- **Integration with Jupiter:**
  - Leverages Jupiter's swap infrastructure for executing the actual token swaps.
  - Passes through Jupiter's route data for optimal swap execution.

The program acts as a middleware layer that adds fee management and SOL handling capabilities on top of Jupiter's swap functionality, making it suitable for integrators who want to offer token swaps while earning fees from the transactions.

2.2. Findings

During the audit process, the audit team found no vulnerabilities in the given version of Unizen Solana Aggregator Program, only some notes and recommendations.

#	Issue	Severity	Status
1	Unsafe public functions	INFORMATIVE	Acknowledged
2	Skip the fee deduction process if the fee percentage is zero	INFORMATIVE	Fixed
3	Redundant constant NATIVE_MINT	INFORMATIVE	Acknowledged

2.2.1. Unsafe public functions INFORMATIVE

Affected files:

- lib.rs

These functions below are public functions which support DApps to swap through combine instructions to execute instead of calling to **Unizen** Aggregator program. Using these functions will not enforce the slippage check, which is an important part of the swap process.

Furthermore, if DApps forget to close program WSOL after swapping, user will not receive the swapped SOL and anyone can call the `close_program_wsol` function to steal the swapped WSOL in the program.

```
pub fn take_integrator_fee(
    ctx: Context<TakeIntegratorFee>,
    amount_in: u64,
    fee_percent: u64,
    share_percent: u64,
) -> Result<()> {
    instructions::take_integrator_fee(ctx, amount_in, fee_percent, share_percent)
}

pub fn create_program_wsol_idempotent(ctx: Context<CreateWsolTokenIdempotent>) ->
Result<()> {
    instructions::create_program_wsol_idempotent(ctx)
}

pub fn close_program_wsol(ctx: Context<CloseProgramWsol>) -> Result<()> {
    instructions::close_program_wsol(ctx)
}
```

RECOMMENDATION

- We do not recommend to public these functions. Instead, DApps should call to Unizen Aggregator program to swap tokens which will ensure that the slippage check is enforced and user will receive their SOL.

UPDATES

- **Feb 11, 2025**: This issue has been acknowledged by Unizen team.

2.2.2. Skip the fee deduction process if the fee percentage is zero INFORMATIVE

Affected files:

- helpers.rs

The `take_integrator_fee` function calculates the total fee and transfers the fee to the Unizen and integrator accounts. However, the function does not check if the fee percentage is zero before transferring the fee to integrator. This can lead to unnecessary fee transfers when the fee percentage is zero.

```
pub fn take_integrator_fee<'info>(
    accounts: AccountsForFee,
    in_amount: u64,
    fee_percent: u64,
    share_percent: u64,
) -> Result<()> {
    let total_fee = in_amount * fee_percent / constants::FEE_DENOM;
    let mut unizen_fee: u64 = 0;

    if share_percent > 0 {
        unizen_fee = total_fee * share_percent / constants::FEE_DENOM;
        msg!("Transfer fee to Unizen");
        token::transfer(
            CpiContext::new(
                accounts.token_program.to_account_info(),
                token::Transfer {
                    from: accounts.user_token_account.to_account_info(),
                    to: accounts.unizen_token_account.to_account_info(),
                    authority: accounts.user.to_account_info(),
                },
            ),
            unizen_fee,
        )?;
    }

    msg!("Transfer fee to integrator");
    token::transfer(
        CpiContext::new(
            accounts.token_program.to_account_info(),
            token::Transfer {
                from: accounts.user_token_account.to_account_info(),
                to: accounts.integrator_token_account.to_account_info(),
                authority: accounts.user.to_account_info(),
            },
        ),
        total_fee - unizen_fee,
    )?;

    Ok(())
}
```

RECOMMENDATION

- Adding a check to skip the fee deduction process if the fee percentage is zero. This will prevent unnecessary fee transfers when the fee percentage is zero.

UPDATES

- **Feb 11, 2025**: This issue has been acknowledged and fixed by Unizen team.

2.2.3. Redundant constant NATIVE_MINT INFORMATIVE

Affected files:

- create_program_wsol_idempotent.rs
- swap_sol_for_tokens.rs
- swap_tokens_for_sol.rs
- constants.rs

The constant `NATIVE_MINT` is redundant and can be replaced with `spl_token::native_mint::id()` to ensure the consistency of the codebase in Solana ecosystem.

```
#[account(address = NATIVE_MINT)]
pub sol_mint: Account<'info, Mint>,

// constants.rs
#[constant]
pub const NATIVE_MINT: Pubkey = pubkey!("So11111111111111111111111111111111111111112");
```

UPDATES

- **Feb 11, 2025**: This issue has been acknowledged by Unizen team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jan 21, 2025*	Public Report	Verichains Lab
1.1	*Feb 11, 2025*	Public Report	Verichains Lab

Table 2. Report versions history