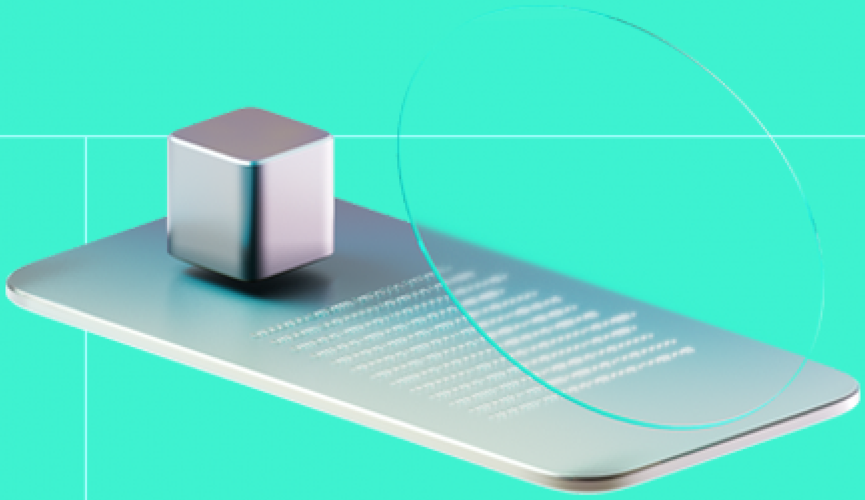
Smart Contract Code Review And Security Analysis Report



Customer: Unizen

Date: 12/02/2025

We express our gratitude to the Unizen team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Unizen is a next-generation DEX aggregator, offering developers, traders, and businesses the ability to unlock unparalleled token swap capabilities.

Document

Name	Smart Contract Code Review and Security Analysis Report for Unizen
Audited By	Stepan Chekhovskoi
Approved By	Ataberk Yavuzer
Website	https://unizen.io
Changelog	27/01/2025 - Initial Report
	12/02/2025 - Second Report
Platform	Solana
Language	Rust
Tags	DeFi, DeX Aggregator
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/unizen-io/unizen-solana-swap
Initial Commit	653803a62fbe5c9ebb96c52d7cbeb4afec3eba10
Second Commit	a4592e0418b6a5afe26138822ae358e4c2c2b299

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

1	1	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	1
Low	0

Vulnerability	Severity
F-2025-8517 - Inaccessible Functionality due to Read-Only Account	Medium

Documentation quality

- Functional requirements are missed.
- Technical description is not provided.

Code quality

- The code is well-structured.
- The development environment is configured.

Test coverage

Code coverage of the project is **0%**.

- The protocol does not have any test cases.

Table of Contents

System Overview

Unizen Solana Contract wraps Jupiter AMM swap functionality.

The contract implements optional fee mechanism which provides ability to integrate an off-chain service to cover Gas fees in arbitrary token.

The contract provides validation for swap minimum amount out.

Privileged roles

- The contract **owner** is able to upgrade the functionality.

Potential Risks

- **System Reliance on External Contracts**: The functioning of the system significantly relies on Jupiter AMM external contracts. Any flaws or vulnerabilities in these contracts adversely affect the audited project, potentially leading to security breaches or loss of funds.
- **Flexibility and Risk in Contract Upgrades**: The project's contracts are upgradable, allowing the administrator to update the contract logic at any time. While this provides flexibility in addressing issues and evolving the project, it also introduces risks if upgrade processes are not properly managed or secured, potentially allowing for unauthorized changes that could compromise the project's integrity and security.
- **Integrator Specifies Zero Share Percent**: The project introduces the integrator role for accounts executing transactions on behalf of the protocol users. The integrators may charge a fee for the service. The smart contract provides the ability for integrators to share the fee with the protocol development team. However, integrators may specify the fee-sharing as zero and not reward the protocol.
- **Misconfiguration of Jupiter Transactions**: The integrators are allowed to provide arbitrary swap data to Jupiter AMM. Users should carefully validate the source and receive token accounts, check the swap and minimum output amounts.

Findings

Vulnerability Details

F-2025-8517 - Inaccessible Functionality due to Read-Only Account - Medium

Description: The `SwapSolForTokens` instruction requires `integrator_wsol_ata` to be provided as read-only while it is a token account dedicated for fees collection.

```
#[derive(Accounts)]
pub struct SwapSolForTokens<'info> {
    pub integrator_wsol_ata: Account<'info, TokenAccount>,
    ...
}
```

This may lead to the instruction fails due to trying write to read-only account.

Assets:

- instructions/swap_sol_for_tokens.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]

Status: Fixed

Classification

Impact:	2/5
Likelihood:	4/5
Exploitability:	Independent
Complexity:	Simple
Severity:	Medium

Recommendations

Remediation: Consider declaring the account as mutable.

```
#[derive(Accounts)]
pub struct SwapSolForTokens<'info> {
    #[account(mut)]
    pub integrator_wsol_ata: Account<'info, TokenAccount>,
    ...
}
```

Resolution: The Finding is fixed before or in the commit `a4592e0418b6a5afe26138822ae358e4c2c2b299`.

The account is declared as mutable.

Observation Details

F-2025-8511 - Redundant Lifetime Declarations - Info

Description:

Redundant Lifetime Declarations refer to unnecessary or duplicate specifications of lifetimes within a program, particularly in the context of the Solana framework, which relies on the Rust programming language. Rust's ownership system enforces strict rules on how references and data lifetimes are handled, and excessive or redundant lifetime annotations can cause code to become unnecessarily verbose, harder to maintain, and potentially lead to misunderstandings about the intended ownership structure.

In the Solana framework, this issue may occur in programs where developers explicitly declare lifetimes for accounts, data structures, or function parameters when Rust’s compiler can infer them automatically. While redundant lifetime annotations do not directly lead to security vulnerabilities, they increase code complexity, which can obscure critical logic and make auditing more challenging.

The `'info` lifetime declaration can be elided:

```
pub fn swap_on_jupiter<'info>(
    remaining_accounts: &[AccountInfo],
    jupiter_program: Program<'info, Jupiter>,
    data: Vec<u8>,
)

pub fn take_integrator_fee<'info>(
    accounts: AccountsForFee,
    in_amount: u64,
    fee_percent: u64,
    share_percent: u64,
)
```

The `remaining_accounts` array copying can be avoided:

```
let accounts_infos: Vec<AccountInfo> = remaining_accounts
    .iter()
    .map(|acc| AccountInfo { ..acc.clone() })
    .collect();

invoke_signed(
    &Instruction {
        program_id: *jupiter_program.key,
        accounts,
            data,
        },
        &accounts_infos,
        &[],
    )
```

The explicit `as_ref()` can be elided:

```
    let signer_seeds: &[&[&[u8]]] = &[
        &[constants::AUTHORITY_SEED, authority_bump.as_ref()],
        &[constants::WSOL_SEED, wsol_bump.as_ref()],
    ];

    let signer_seeds: &[&[&[u8]]] = &[&[constants::AUTHORITY_SEED, authority_bump
    .as_ref()]];
```

Assets:

- helpers.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]

Status: Fixed

Recommendations

Remediation: Consider removing the redundancies to keep the code clean.

```
    pub fn swap_on_jupiter(
        remaining_accounts: &[AccountInfo],
        jupiter_program: Program< Jupiter>,
        data: Vec<u8>,
    )

    pub fn take_integrator_fee(
        accounts: AccountsForFee,
        in_amount: u64,
        fee_percent: u64,
        share_percent: u64,
    )
```

```
    invoke_signed(
        &Instruction {
            program_id: *jupiter_program.key,
            accounts,
            data,
        },
        remaining_accounts,
    &[],
)
```

```
let signer_seeds: &[&[&[u8]]] = &[
    &[constants::AUTHORITY_SEED, authority_bump],
    &[constants::WSOL_SEED, wsol_bump],
];

let signer_seeds: &[&[&[u8]]] = &[&[constants::AUTHORITY_SEED, authority_bump
]];
```

Resolution: The Finding is mostly fixed before or in the commit `a4592e0418b6a5afe26138822ae358e4c2c2b299`.

The redundancies are eliminated.

F-2025-8516 - Redundant Accounts in Context - Info

Description:

The `CloseProgramWsol` and `CreateWsolTokenIdempotent` instructions accept the `user` account, however, the program never uses it.

```
#[derive(Accounts)]
pub struct CloseProgramWsol<'info> {
    pub user: Signer<'info>,
    ...
}

#[derive(Accounts)]
pub struct CreateWsolTokenIdempotent<'info> {
    pub user: Signer<'info>,
    ...
}
```

Redundancies may lower the code quality.

Assets:

- instructions/create_program_wsol_idempotent.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]
- instructions/close_program_wsol.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]

Status: Fixed

Recommendations

Remediation: Consider removing redundant `user` account from the `CloseProgramWsol` and `CreateWsolTokenIdempotent` contexts.

Resolution: The Finding is fixed before or in the commit `a4592e0418b6a5afe26138822ae358e4c2c2b299`.

The redundant signer is removed from the transaction contexts.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Definitions

Severities

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution.

Potential Risks

The "Potential Risks" section identifies issues that are not direct security vulnerabilities but could still affect the project’s performance, reliability, or user trust. These risks arise from design choices, architectural decisions, or operational practices that, while not immediately exploitable, may lead to problems under certain conditions. Additionally, potential risks can impact the quality of the audit itself, as they may involve external factors or components beyond the scope of the audit, leading to incomplete assessments or oversight of key areas. This section aims to provide a broader perspective on factors that could affect the project's long-term security, functionality, and the comprehensiveness of the audit findings.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	https://github.com/unizen-io/unizen-solana-swap
Initial Commit	653803a62fbe5c9ebb96c52d7cbeb4afec3eba10
Second Commit	a4592e0418b6a5afe26138822ae358e4c2c2b299
Whitepaper	N/A
Requirements	https://docs.unizen.io
Technical Requirements	README.md

Asset	Type
constants.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]	Smart Contract
errors.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]	Smart Contract
helpers.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]	Smart Contract
instructions/close_program_wsol.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]	Smart Contract
instructions/create_program_wsol_idempotent.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]	Smart Contract
instructions/mod.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]	Smart Contract
instructions/swap_sol_for_tokens.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]	Smart Contract
instructions/swap_tokens_for_sol.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]	Smart Contract
instructions/swap_tokens_for_tokens.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]	Smart Contract
instructions/take_integrator_fee.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]	Smart Contract
lib.rs [https://github.com/unizen-io/unizen-solana-swap/invitations]	Smart Contract

Appendix 3. Additional Valuables

Additional Recommendations

The smart contracts in the scope of this audit could benefit from the introduction of automatic emergency actions for critical activities, such as unauthorized operations like ownership changes or proxy upgrades, as well as unexpected fund manipulations, including large withdrawals or minting events. Adding such mechanisms would enable the protocol to react automatically to unusual activity, ensuring that the contract remains secure and functions as intended.

To improve functionality, these emergency actions could be designed to trigger under specific conditions, such as:

- Detecting changes to ownership or critical permissions.
- Monitoring large or unexpected transactions and minting events.
- Pausing operations when irregularities are identified.

These enhancements would provide an added layer of security, making the contract more robust and better equipped to handle unexpected situations while maintaining smooth operations.