SALUS SECURITY

MAR 2025

CODE SECURITY ASSESSMENT

SWAPX

Overview

Project Summary

- Name: Swapx - buy sell
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/hi-swapx/buy_sell
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	SwapX - buy sell
Version	v3
Type	Solidity
Dates	Apr 10 2025
Logs	Mar 31 2025; Apr 15 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	4
Total informational issues	3
Total	8

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction **4**

1.1 About SALUS 4

1.2 Audit Breakdown 4

1.3 Disclaimer 4

Findings **5**

2.1 Summary of Findings 5

2.2 Notable Findings 6

1. The contract lacks a function to remove SupportToken 6

2. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom() 7

3. Unable to withdraw native token 8

4. Centralization risk 9

5. Missing events for functions that change critical state 10

2.3 Informational Findings 11

6. Missing two-step transfer ownership pattern 11

7. Use of floating pragma 12

8. Redundant code 13

Appendix **14**

Appendix 1 - Files in Scope 14

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 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 with 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.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	The contract lacks a function to remove SupportToken	Medium	Business Logic	Resolved
2	Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	Low	Business Logic	Resolved
3	Unable to withdraw native token	Low	Business Logic	Resolved
4	Centralization risk	Low	Centralization	Acknowledge
5	Missing events for functions that change critical state	Low	Logging	Resolved
6	Missing two-step transfer ownership pattern	Informational	Business Logic	Acknowledge
7	Use of floating pragma	Informational	Configuration	Resolved
8	Redundant code	Informational	Redundancy	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. The contract lacks a function to remove SupportToken	
Severity: Medium	Category: Business Logic
Target: - src/BuySell.sol	

Description

The contract only includes the `addSupportToken` function for adding tokens, but lacks a corresponding function to revoke support for a token.

If a malicious or vulnerable token is added, it could harm the project, with no way to remove it afterward.

Recommendation

It is recommended to add a function in the contract to revoke support for a specific token.

Status

The team has resolved this issue in commit 51933fb.

2. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()

Severity: Low	Category: Business Logic
Target: - src/BuySell.sol	

Description

Tokens not compliant with the `ERC20` specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Incorrectly checking for return values can lead to exceptions in calls to tokens that are not ERC20-compliant, such as the `transfer()` and `transferFrom()` functions of USDT where no return value exists. Checking the return value is a requirement, as written in the EIP-20 specification:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT assume that
false is never returned!
```

Recommendation

Consider using the `SafeERC20` library implementation from OpenZeppelin and call `safeTransfer` or `safeTransferFrom` when transferring `ERC20` tokens.

Status

The team has resolved this issue in commit 70b2fe9.

3. Unable to withdraw native token

Severity: Low	Category: Business Logic
Target: - src/BuySell.sol	

Description

The `deposit()` function is marked as payable, allowing users to send native tokens when depositing. However, the contract lacks any function to withdraw these native tokens. If users accidentally send native tokens to the contract, the funds will be locked indefinitely, with no way to recover them.

src/BuySell.sol:L33

```
function deposit(DepositParams calldata params) public payable {
    ...
}
```

Recommendation

Consider adding native token handling to the `withdraw` function or removing the `payable` modifier.

Status

The team has resolved this issue in commit 70b2fe9.

4. Centralization risk

Severity: Low	Category: Centralization
Target: - src/BuySell.sol	

Description

The `BuySell` contract has a privileged account. The `owner` can add `SupportToken` and `withdraw` all funds from the contract. If the private key is compromised, an attacker could add malicious tokens or directly drain the contract's assets, causing financial loss to the project.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

5. Missing events for functions that change critical state

Severity: Low	Category: Logging
Target: - src/BuySell.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the `BuySell` contract, events are lacking in the `addSupportToken` function.

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been resolved by the team with commit 70b2fe9.

2.3 Informational Findings

6. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - src/BuySell.sol	

Description

The `BuySell` contract inherits from the `Ownable` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Status

This issue has been acknowledged by the team.

7. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - src/BuySell.sol	

Description

```
pragma solidity ^0.8.28;
```

The `BuySell` contract uses a floating compiler version `^0.8.28`

Using a floating pragma `^0.8.28`statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

The team has resolved this issue in commit 70b2fe9.

8. Redundant code

Severity: Informational	Category: Redundancy
Target: - src/BuySell.sol	

Description

Unused code should be removed before deploying the contract to mainnet. We have identified the following code are not being utilized:

src/BuySell.sol:L17

```
error NotSupportToken(address tokenAddress);
```

Recommendation

Consider removing the redundant code.

Status

The team has resolved this issue in commit 70b2fe9.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 98cbf05:

File	SHA-1 hash
contracts/BuySell.sol	01e466278ae84a6096b2e4f8b564aed1169c67f9