SarosSwap

Smart Contract Audit Report
Prepared for Saros Finance

Date Issued:	Dec 3, 2021
Project ID:	AUDIT2021043
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021043
Version	v1.0
Client	Saros Finance
Project	SarosSwap
Auditor(s)	Weerawat Pawanawiwat Patipon Suwanbol
Author	Weerawat Pawanawiwat
Reviewer	Suvicha Buakhom
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Dec 3, 2021	Full report	Weerawat Pawanawiwat

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by Saros Finance, Inspex team conducted an audit to verify the security posture of the SarosSwap program between Nov 18, 2021 and Nov 22, 2021. During the audit, Inspex team examined the program and the overall operation within the scope to understand the overview of the SarosSwap program. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 1 high, 2 medium, and 1 very low-severity issues. With the project team's prompt response, 1 high and 2 medium-severity issues were resolved or mitigated in the reassessment, while 1 very low-severity issue was acknowledged by the team. Therefore, Inspex trusts that the SarosSwap program has sufficient protections to be safe for public use. However, in the long run, Inspex suggests resolving all issues found in this report.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Saros Finance is a Defi super network built on Solana. With three fundamental building blocks - SarosSwap (AMM), SarosFarm (Pool), SarosPad (LaunchPad), Saros aims to attract more builders and users to the Solana Ecosystem.

SarosSwap is based on the Solana Program Library's Token Swap Program. The users can swap from one token to another on the platform's liquidity pools with a small fee, and the liquidity providers can deposit tokens as the platform's liquidity to gain their share of the swapping fee.

Scope Information:

Project Name	SarosSwap
Website	https://saros.finance/swap
Smart Contract Type	Solana Program
Chain	Solana
Programming Language	Rust

Audit Information:

Audit Method	Whitebox
Audit Date	Nov 18, 2021 - Nov 22, 2021
Reassessment Date	Dec 2, 2021

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following program was audited and reassessed by Inspex in detail:

Initial Audit: (Commit: 75d6da14c3ed93a3c5c7fbb87f6e5dcac7b30400)

Program	Location (URL)
saros-swap	https://github.com/coin98/saros-swap/tree/75d6da14c3/program/src

Reassessment: (Commit: 75f7007505cd09efaaa01cfe45868d071ef36a02)

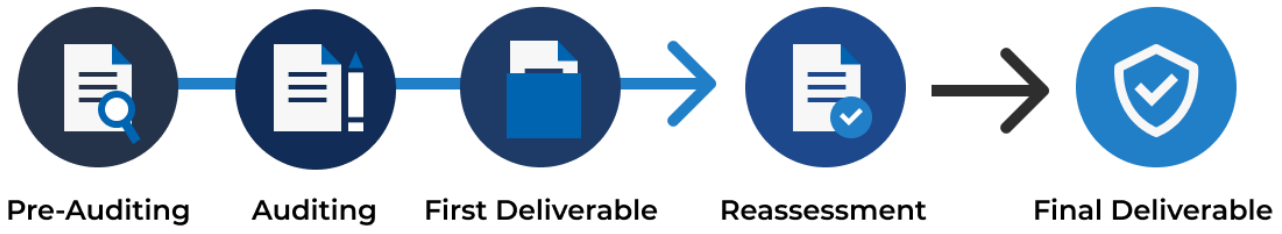
Program	Location (URL)
saros-swap	https://github.com/coin98/saros-swap/tree/75f7007505/program/src

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Integer Overflows and Underflows
Bad Randomness
Use of Known Vulnerable Component
Use of Deprecated Component
Solana Account Confusions
Missing Rent Exemption Checking
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Implicit Type Inference
Function Declaration Inconsistency
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact**: a measure of the damage caused by a successful attack

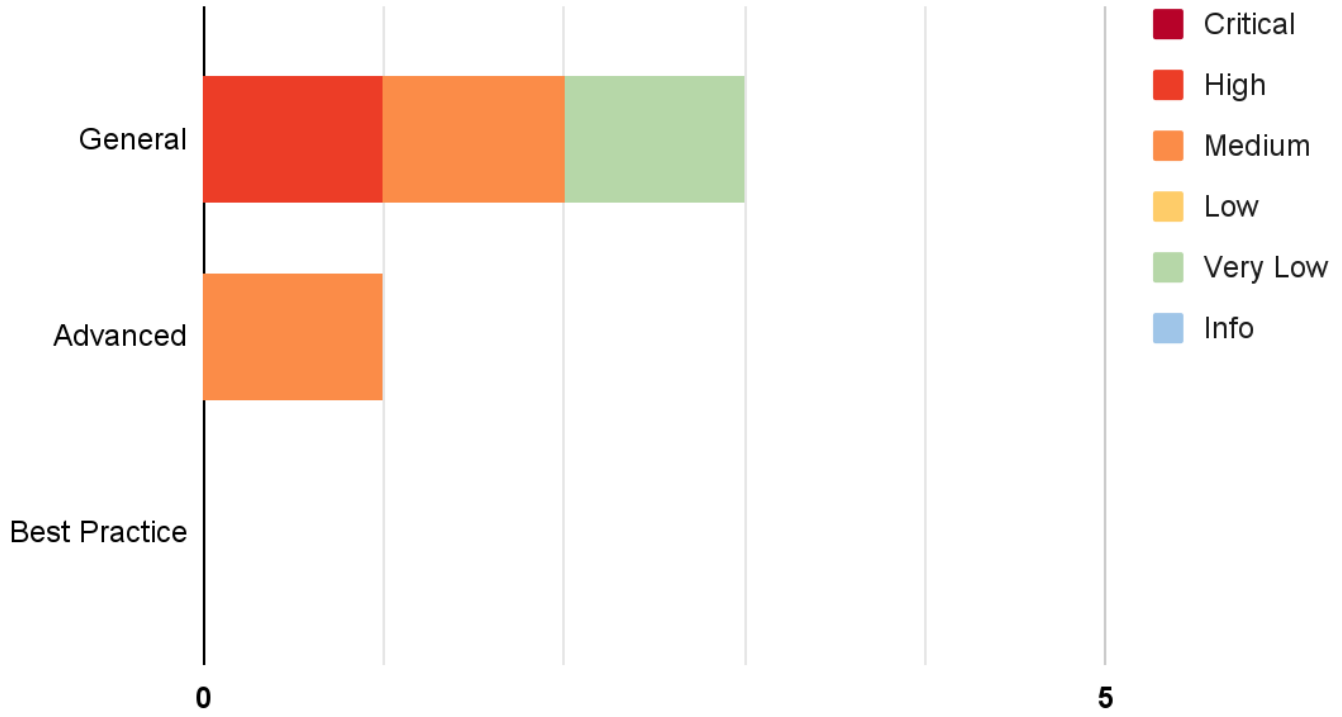
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Impact \ Likelihood	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

4. Summary of Findings

From the assessments, Inspex has found 4 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Upgradability of Solana Program	General	**High**	Resolved *
IDX-002	Unchecked Account Rent Exemption	General	**Medium**	Resolved *
IDX-003	Improper Deposit Allowance Checking	Advanced	**Medium**	Resolved
IDX-004	Use of Outdated Dependency	General	**Very Low**	Acknowledged

* The mitigations or clarifications by Saros Finance can be found in Chapter 5.

5. Detailed Findings Information

5.1. Upgradability of Solana Program

ID	IDX-001
Target	saros-swap
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** High **Impact:** High The logic of the program can be arbitrarily changed. This allows the upgrade authority to change the logic of the program in favor of the platform, e.g., transferring the users' funds to the platform owner's account. **Likelihood:** Medium Only the program upgrade authority can redeploy the program to the same program address; however, there is no restriction to prevent the authority from inserting malicious logic.
Status	Resolved * Saros Finance team has confirmed that they will mitigate this issue by deploying the program using a multisig account owned by multiple trusted parties. The platform users should verify that the multisig account is the upgrade authority of the program, and the parties with the permission to use the multisig account are trusted before using the platform.

5.1.1. Description

Programs on Solana can be deployed through the upgradable BPF loader to make them upgradable, allowing the program's upgrade authority to redeploy the program with the new logic, bug fixes, or upgrades to the same program address.

However, there is no restriction on how and when the program will be upgraded. This opens up an attack surface on the program, allowing the upgrade authority to redeploy the program with malicious logic and gain unfair benefits from the users, for example, transferring funds out from the users' accounts.

5.1.2. Remediation

Inspex suggests deploying the program as an immutable program to prevent the program logic from being modified.

However, if the upgradability is needed, Inspex suggests mitigating this issue by the following options:

- Using a multisig account controlled by multiple trusted parties as the upgrade authority
- Implementing a community-run governance to control the redeployment of the program

5.2. Unchecked Account Rent Exemption

ID	IDX-002
Target	saros-swap
Category	General Smart Contract Vulnerability
CWE	CWE-826: Premature Release of Resource During Expected Lifetime
Risk	**Severity:** Medium **Impact:** High Accounts can be deleted due to insufficient lamports for rent, locking the tokens of the liquidity providers and causing the users to be unable to swap within the pool. This results in monetary damage for the liquidity providers and loss of reputation for the platform. **Likelihood:** Low Without the checking, it is possible that the swap account will be initialized with insufficient lamport in the account to reach the rent exemption criterion. However, lamports can be manually transferred to the account to make it rent-exempted.
Status	Resolved * Saros Finance team has clarified that the team has a practice to call `getMinimumBalanceForRentExemption` to transfer enough lamports before creating a new account, so this case will not happen. The platform users should verify that the swap accounts contain sufficient lamports for rent exemption before providing liquidity into the platform.

5.2.1. Description

Swap accounts are not checked for rent exemption, causing rent to be collected from the account every epoch. When the account is out of lamport, the account will be deleted, and the data stored for that pair will be lost. Without the account, the users will not be able to swap the tokens within that pair, and the pool token held by the liquidity providers will lose its value.

This causes significant monetary damage to the liquidity providers and disrupts the availability of the platform.

5.2.2. Remediation

Inspex suggests checking the amount of lamports inside the swap account on the initialization to make sure that it is rent-exempt.

5.3. Improper Deposit Allowance Checking

ID	IDX-003
Target	saros-swap
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium Deposits can still be made on the curves that are not designed for more deposits to be added. This may allow an attacker to deposit and withdraw to gain a part of the pool's tokens, causing unintended impact to the token price and liquidity. **Likelihood:** Medium This attack can only be done mainly by the swap pair initializer of the pair with offset curve, but it is profitable for the attacker, so the motivation for the attack is high.
Status	Resolved Saros Finance team has resolved this issue as suggested in commit `75f7007505cd09efaaa01cfe45868d071ef36a02` by adding a validation for the `DepositSingleTokenTypeExactAmountIn` instruction.

5.3.1. Description

In the `saros-swap` program, the `allows_deposits()` function is implemented as a flag to prevent deposits from being done in some curves.

src/curve/calculator.rs

```
168 /// Some curves function best and prevent attacks if we prevent deposits
169 /// after initialization.  For example, the offset curve in offset.rs,
170 /// which fakes supply on one side of the swap, allows the swap creator
171 /// to steal value from all other depositors.
172 fn allows_deposits(&self) -> bool {
173     true
174 }
```

This flag is checked in the `DepositAllTokenTypes` instruction at line 517.

src/processor.rs

```
495 /// Processes an [DepositAllTokenTypes](enum.Instruction.html).
496 pub fn process_deposit_all_token_types(
497     program_id: &Pubkey,
498     pool_token_amount: u64,
```

```
499     maximum_token_a_amount: u64,
500     maximum_token_b_amount: u64,
501     accounts: &[AccountInfo],
502 ) -> ProgramResult {
503     let account_info_iter = &mut accounts.iter();
504     let swap_info = next_account_info(account_info_iter)?;
505     let authority_info = next_account_info(account_info_iter)?;
506     let user_transfer_authority_info = next_account_info(account_info_iter)?;
507     let source_a_info = next_account_info(account_info_iter)?;
508     let source_b_info = next_account_info(account_info_iter)?;
509     let token_a_info = next_account_info(account_info_iter)?;
510     let token_b_info = next_account_info(account_info_iter)?;
511     let pool_mint_info = next_account_info(account_info_iter)?;
512     let dest_info = next_account_info(account_info_iter)?;
513     let token_program_info = next_account_info(account_info_iter)?;
514
515     let token_swap = SwapVersion::unpack(&swap_info.data.borrow())?;
516     let calculator = &token_swap.swap_curve().calculator;
517     if !calculator.allows_deposits() {
518         return Err(SwapError::UnsupportedCurveOperation.into());
519     }
```

However, it is not checked in the `DepositSingleTokenTypeExactAmountIn` instruction, allowing the users to deposit tokens into the swap pair.

Unintended consequences can happen when deposits are done to the swap pairs that do not accept deposits.

For example, in the offset curve, if the users deposit the token on the offset side into the pair, the swap pair initializer can directly withdraw their liquidity and directly gain the users' deposits.

5.3.2. Remediation

Inspex suggests including the deposit checking via the `allows_deposits()` function in the `DepositSingleTokenTypeExactAmountIn` instruction. This can be done the same way as the fix applied in the SPL `token-swap` program. For example, the condition is added to the following source code in line 747-750, and since the reference is saved in the `calculator` variable, the reference can also be used at line 800:

src/processor.rs

```
728 /// Processes DepositSingleTokenTypeExactAmountIn
729 pub fn process_deposit_single_token_type_exact_amount_in(
730     program_id: &Pubkey,
731     source_token_amount: u64,
732     minimum_pool_token_amount: u64,
733     accounts: &[AccountInfo],
```

```
734 ) -> ProgramResult {
735     let account_info_iter = &mut accounts.iter();
736     let swap_info = next_account_info(account_info_iter)?;
737     let authority_info = next_account_info(account_info_iter)?;
738     let user_transfer_authority_info = next_account_info(account_info_iter)?;
739     let source_info = next_account_info(account_info_iter)?;
740     let swap_token_a_info = next_account_info(account_info_iter)?;
741     let swap_token_b_info = next_account_info(account_info_iter)?;
742     let pool_mint_info = next_account_info(account_info_iter)?;
743     let destination_info = next_account_info(account_info_iter)?;
744     let token_program_info = next_account_info(account_info_iter)?;
745
746     let token_swap = SwapVersion::unpack(&swap_info.data.borrow())?;
747     let calculator = &token_swap.swap_curve().calculator;
748     if !calculator.allows_deposits() {
749         return Err(SwapError::UnsupportedCurveOperation.into());
750     }
751     let source_account =
752         Self::unpack_token_account(source_info,
    token_swap.token_program_id())?;
753     let swap_token_a =
754         Self::unpack_token_account(swap_token_a_info,
    token_swap.token_program_id())?;
755     let swap_token_b =
756         Self::unpack_token_account(swap_token_b_info,
    token_swap.token_program_id())?;
757
758     let trade_direction = if source_account.mint == swap_token_a.mint {
759         TradeDirection::AtoB
760     } else if source_account.mint == swap_token_b.mint {
761         TradeDirection::BtoA
762     } else {
763         return Err(SwapError::IncorrectSwapAccount.into());
764     };
765
766     let (source_a_info, source_b_info) = match trade_direction {
767         TradeDirection::AtoB => (Some(source_info), None),
768         TradeDirection::BtoA => (None, Some(source_info)),
769     };
770
771     Self::check_accounts(
772         token_swap.as_ref(),
773         program_id,
774         swap_info,
775         authority_info,
776         swap_token_a_info,
777         swap_token_b_info,
```

```
778             pool_mint_info,
779             token_program_info,
780             source_a_info,
781             source_b_info,
782             None,
783         )?;
784
785         let pool_mint = Self::unpack_mint(pool_mint_info,
    token_swap.token_program_id())?;
786         let pool_mint_supply = to_u128(pool_mint.supply)?;
787         let pool_token_amount = if pool_mint_supply > 0 {
788             token_swap
789                 .swap_curve()
790                 .deposit_single_token_type(
791                     to_u128(source_token_amount)?,
792                     to_u128(swap_token_a.amount)?,
793                     to_u128(swap_token_b.amount)?,
794                     pool_mint_supply,
795                     trade_direction,
796                     token_swap.fees(),
797                 )
798                 .ok_or(SwapError::ZeroTradingTokens)?
799         } else {
800             calculator.new_pool_supply()
801         };
```

Reference: https://github.com/solana-labs/solana-program-library/pull/2590

5.4. Use of Outdated Dependency

ID	IDX-004
Target	saros-swap
Category	General Smart Contract Vulnerability
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Very Low **Impact:** Low Outdated dependencies have publicly known issues and bugs. It is possible that attackers can use those flaws to attack the program and cause monetary loss or business impact to the platform and its users. **Likelihood:** Low With the current dependency version, it is very unlikely that the publicly known bugs and issues will affect the target program.
Status	**Acknowledged** Saros Finance team has acknowledged this issue and decided not to fix this issue in the current release.

5.4.1. Description

A dependency used by the `saros-swap` is outdated. The version may have publicly known inherent bugs that may potentially be used to cause damage to the program or the users of the program.

Cargo.toml

```
15 [dependencies]
16 arrayref = "0.3.6"
17 enum_dispatch = "0.3.7"
18 num-derive = "0.3"
19 num-traits = "0.2"
20 solana-program = "1.8.1"
21 spl-math = { version = "0.1", features = [ "no-entrypoint" ] }
22 spl-token = { version = "3.2", features = [ "no-entrypoint" ] }
23 thiserror = "1.0"
24 arbitrary = { version = "0.4", features = ["derive"], optional = true }
25 roots = { version = "0.0.7", optional = true }
```

The `arbitrary` crate used is outdated. The version specified in the program is "0.4"; however, the latest stable version at the time of the audit is "1.0.3".

5.4.2. Remediation

Inspex suggests upgrading the dependency to the latest stable version. However, the compatibility of the components must be checked to make sure that the program is functional as intended.

6. Appendix

6.1. About Inspex

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

[1] “OWASP Risk Rating Methodology.” [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE