BLOCKSEC

Security Audit
Report for one-intro-swap

Date: May 14, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**
1.1 About Target Contracts 1
1.2 Disclaimer 1
1.3 Procedure of Auditing 2
1.3.1 Software Security 2
1.3.2 DeFi Security 2
1.3.3 NFT Security 2
1.3.4 Additional Recommendation 3
1.4 Security Model 3

Chapter 2 Findings **4**
2.1 Note 4
2.1.1 Inconsistent initial liquidity calculation 4
2.1.2 Potential centralization risk 5

Report Manifest

Item	Description
Client	1Intro
Target	one-intro-swap

Version History

Version	Date	Description
1.0	May 14, 2024	First release

Signature

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Rust
Approach	Semi-automatic and manual verification

The focus of this audit is the `programs/one-intro-swap` within the one-intro-swap [1]. Please note that other external dependencies in the repository, including the solana development framework `Anchor` [2], are considered reliable in terms of both functionality and security, these files are not included in the scope of the audit.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
one-intro-swap	`Version 1`	`317225207312e958a227172186a169e923890302`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

[1] https://github.com/1intro/1intro-programs

[2] https://www.anchor-lang.com/

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver
* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [3] and Common Weakness Enumeration [4]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[3] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[4] https://cwe.mitre.org/

Chapter 2 Findings

In total, we did not find potential security issues. Besides, we have **two** notes.

- Note: 2

ID	Severity	Description	Category	Status
1	-	Inconsistent initial lp liquidity calculation	Note	-
2	-	Potential centralization risk	Note	-

The details are provided in the following sections.

2.1 Note

2.1.1 Inconsistent initial liquidity calculation

Description The instruction `create_pool_state` collects underlying assets from the creator and mints LP tokens, where the `initial_lp_supply` is calculated using the function `get_initial_lp_supply`.

```
5   impl<'info> CreatePoolState<'info> {
6       pub fn process(&mut self, pool_auth_pda_bump: u8, params: CreatePoolStateParams) -> Result
            <()> {
7           ...
37          // mint lp token to creator
38          let initial_lp_supply = PoolState::get_initial_lp_supply(&params.token_balances);
39          if initial_lp_supply < MIN_INITIAL_POOL_SUPPLY || initial_lp_supply >
                MAX_INITIAL_POOL_SUPPLY {
40              return Err(ErrorCode::ValidationInvalidInitialPoolSupply.into());
41          }
42          self.pool_state.mint_lp_token(
43              pool_state_pubkey,
44              &mut self.pool_lp_mint,
45              &self.creator_lp_token_account.to_account_info(),
46              &self.pool_auth_pda,
47              &self.token_program,
48              initial_lp_supply,
49          )?;
```

Listing 2.1: src/instructions/create_pool_state.rs

Meanwhile, when a pool becomes empty, users can provide an initial liquidity again via the instruction `join_pool`. However, in such a scenario, the initial liquidity is assigned by the `params.pool_out_amount`, instead of using the same calculation method in instruction `create_pool_state`.

```
7   impl<'info> JoinPool<'info> {
8       pub fn process(&mut self, params: JoinPoolParams) -> Result<()> {
9           ...
61          // mint lp token to user
62          let pool_state_pubkey = self.pool_state.key();
63          self.pool_state.mint_lp_token(
```

```
64              pool_state_pubkey,
65              &mut self.pool_lp_mint,
66              &self.user_lp_token_account.to_account_info(),
67              &self.pool_auth_pda,
68              &self.token_program,
69              params.pool_out_amount,
70          )?;
```

Listing 2.2: src/instructions/join_pool.rs

2.1.2 Potential centralization risk

Description The `one-intro-swap` program is a DEX that enables users to create pools. The admin of the program has certain privileges, such as adjusting the fee settings, which can influence the liquidity providers' profit.

Feedback from the Project we understand this concern and may consider to remind/highlight the fee settings in frontend, thanks for pointing out.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS