

# Secure Code Review

Findings and Recommendations Report Presented to:

**Solcial**

September 18, 2023  
Version: 1.0

Presented by:

Kudelski Security, Inc.  
5090 North 40th Street, Suite 450  
Phoenix, Arizona 85018

FOR PUBLIC RELEASE

## TABLE OF CONTENTS

TABLE OF CONTENTS .....	2
LIST OF FIGURES.....	2
LIST OF TABLES.....	2
EXECUTIVE SUMMARY .....	3
Overview .....	3
Major Threats Assessed .....	4
Key Findings .....	4
Scope and Rules Of Engagement.....	5
TECHNICAL ANALYSIS & FINDINGS .....	6
Findings.....	7
KS-03 Multiple outdated dependencies .....	12
KS-04 Absence of Anchor Framework.....	13
KS-05 Frequent misspellings.....	14
METHODOLOGY .....	15
Tools.....	15
Vulnerability Scoring Systems.....	16
KUDELSKI SECURITY CONTACTS .....	<b>Error! Bookmark not defined.</b>

## LIST OF FIGURES

Figure 1: Findings by Severity.....	6
-------------------------------------	---

## LIST OF TABLES

Table 1: Scope .....	5
Table : Findings Overview .....	7

## EXECUTIVE SUMMARY

### Overview

Solcial engaged Kudelski Security to perform a code review of the solcial-stacking program.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place between August 21, 2023 and September 15, 2023, and it was focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

During the Secure Code Review, we identified **1 medium**, **1 low**, and **3 informational** findings according to our Vulnerability Scoring System.

This report summarizes the engagement, tests performed, and details of the mentioned findings.

It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Teams took to identify and validate each, as well as any applicable recommendations for remediation.

The review included checks for the following:

- Unchecked math
- Proper error handling
- Validation of function inputs and outputs
- Validation of ownership
- Account creation and usage
- Permissions checks and active/inactive status checks, including permission structures & validations
- Logic flow and sequence
- Proper usage, functionality, and/or validation of instructions
- Sufficient test coverage
- Flow of instructions – from creation of the owner account, initialization of the stacking vaults, user stacking, and user unstacking – as well as state changes
- Connections and CPI calls to other programs

Some positive observations include:

- The `solcial-stacking` codebase presents a well-commented and clean architecture with attention to details as well as potential risks.

- The Solcial team responsible for this codebase were knowledgeable about the programs and provided explanations when we needed them.

## Major Threats Assessed

Threats assessed include those requested by the Solcial team as well as those we independently identified. This list includes threats deemed high-priority but is not comprehensive. They were determined not to be issues at this time. These threats include:

- There are multiple places where commented-out code and TODOs are used throughout the program. Each instance was assessed to ensure there were no missing features or checks.
  - It was determined that all TODO comments and commented out code were not missing pieces but rather additional features to be implemented in the future.
- An attacker could drain or otherwise manipulate the vault
  - The vault and associated accounts were found to be secure.
- Time misconfigurations could lead to incorrect calculations
  - All usages of time were correct and consistent.
- An attacker may be able to pass in fraudulent accounts if signers, ownership, and state are not properly checked.
  - All checks were either included or implicit through the use of PDAs
- Common security vulnerabilities usually handled by Anchor (notably: arbitrary CPI, proper checks, and account type confusion) need to be handled manually.
  - It was determined that all proper validation and checks were in place to ensure these vulnerabilities are not an issue.
- Incorrect logic flow can lead to increased attack surface
  - We meticulously mapped out the flow of logic present in the program and determined it was secure.

## Key Findings

The following are the major themes and issues identified during the testing period.

These, along with other items, within the findings section, should be prioritized for remediation to reduce the risk they pose.

- A missing validation check of the `spl_token` program ID was noted.
- There is potential for numeric overflow.
- There are multiple outdated dependencies.

## Scope and Rules Of Engagement

Kudelski performed a Secure Code Review for Solcial. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

<b>Commit Hash</b>
c3430a37df7f38aff3fabf0817dd70376fd289e0
<b>In-Scope Repositories</b>
Solcial-Solana-Client/programs/solcial-stacking

Table 1: Scope

## TECHNICAL ANALYSIS & FINDINGS

During the Secure Code Review, we identified **1 medium**, **1 low** and **3 informational** findings according to our Vulnerability Scoring System.

The following chart displays the findings by severity.

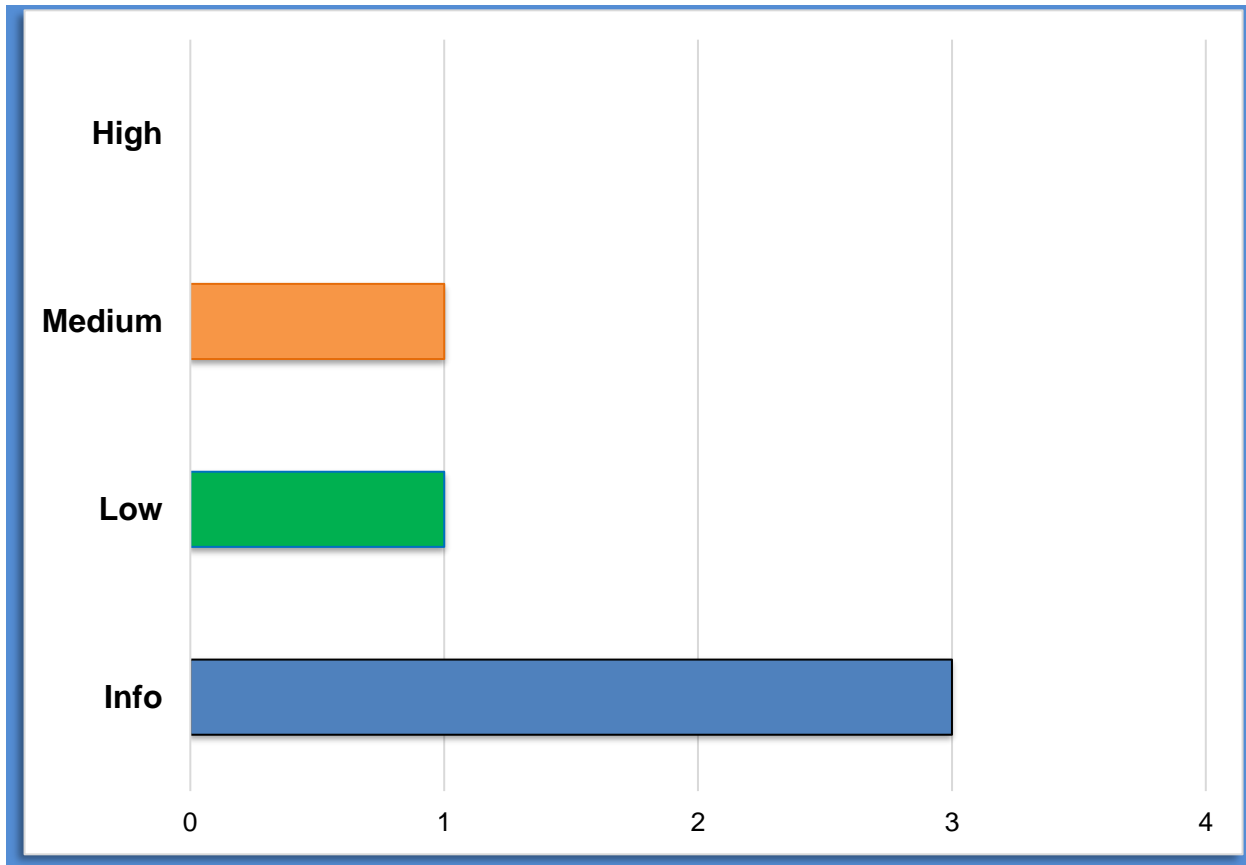


Figure 1: Findings by Severity

## Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

#	Severity	Description
KS-01	Medium	Missing validation of <code>spl_token</code> program ID
KS-02	Low	Potential numeric underflow
KS-03	Informational	Multiple outdated dependencies
KS-04	Informational	Absence of Anchor Framework
KS-05	Informational	Frequent misspellings

Table 2: Findings Overview

## KS-01 Missing validation of spl\_token program ID

Severity	Medium
----------	--------

Impact	Likelihood	Difficulty
High	Low	Medium

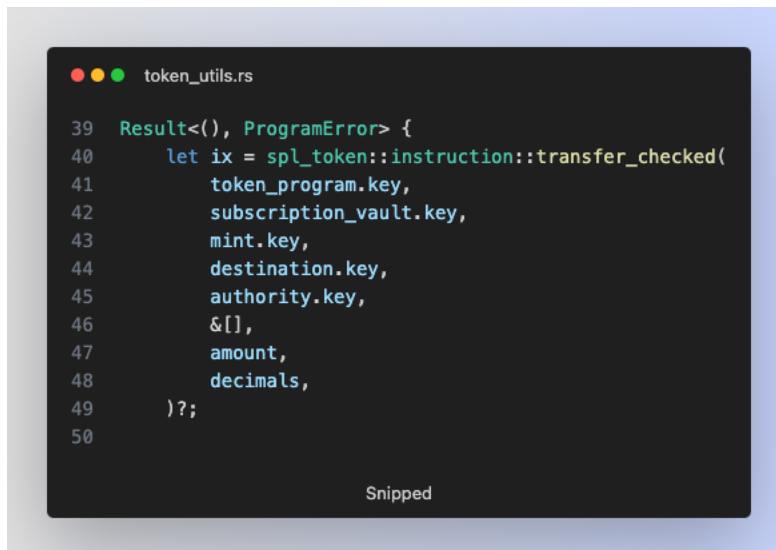
### Description

There is a missing validation check on the `spl_token` program ID. The program ID of the passed in program should match the expected program ID of the official `spl_token` program.

### Impact

If the program ID is not validated, it could allow an attacker to pass in a fake SPL token program account, leading to either loss of funds on the part of the user or incorrect calculations of rewards.

### Evidence



```
token_utils.rs

39 Result<(), ProgramError> {
40     let ix = spl_token::instruction::transfer_checked(
41         token_program.key,
42         subscription_vault.key,
43         mint.key,
44         destination.key,
45         authority.key,
46         &[],
47         amount,
48         decimals,
49     )?;
50 }
```

Snipped



```
user_unstack.rs

116 fn check_token_accounts(accounts: &Accounts) -> Result<(), ProgramError> {
117     let recipient =
118         spl_token::state::Account::unpack(&accounts.recipient_token_account.data.borrow());
119     let subscription_vault =
120         spl_token::state::Account::unpack(&accounts.stacking_vault.data.borrow());

Snipped
```

### **Affected Resource**

Every time `spl_token` is used throughout the program is a potential avenue for exploit. The entire `solcial-stacking` program is affected.

### **Recommendation**

Use the function `check_program_account` provided by the SPL token program. This checks that the program ID is the correct and expected one to ensure proper validation.

### **Reference**

[https://docs.rs/spl-token/latest/spl\\_token/fn.check\\_program\\_account.html](https://docs.rs/spl-token/latest/spl_token/fn.check_program_account.html)

<https://github.com/solana-labs/solana-program-library/tree/master/token>

## KS-02 Potential numeric underflow

Severity	Low
----------	-----

Impact	Likelihood	Difficulty
Low	Low	Medium

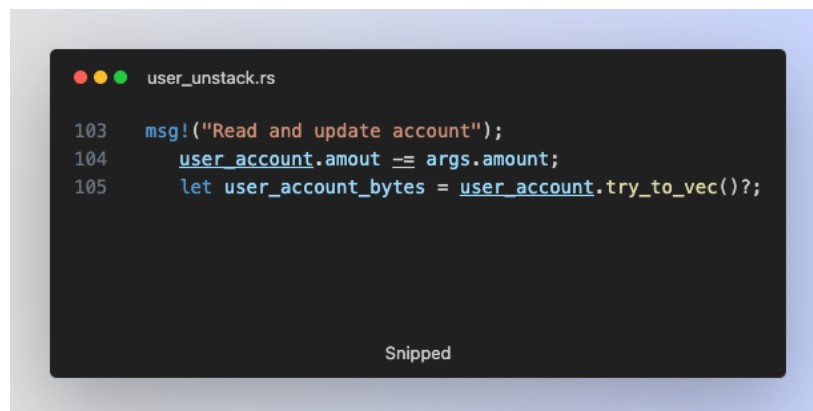
### Description

In `user_unstack`, there is an unchecked subtraction of the `args.amount` from the `user_account.amout`.

### Impact

While it is unlikely, if an underflow does occur, it would lead to an incorrect withdrawal and thus an incorrect amount in the `user_account`.

### Evidence



```
user_unstack.rs
103 msg!("Read and update account");
104 user_account.amout -= args.amount;
105 let user_account_bytes = user_account.try_to_vec()?;
```

Snipped

### Affected Resource

`src/instructions/user_unstack` line 104

### Recommendation

Use `checked_math` (`checked_sub`) or turn on `overflow-checks` in `Cargo.toml`.

### Reference

<https://medium.com/coinmonks/understanding-arithmetic-overflow-underflows-in-rust-and-solana-smart-contracts-9f3c9802dc45>

## KS-03 Multiple outdated dependencies

Severity

Informational

### **Description**

There are multiple outdated important dependencies present in `social-stacking/Cargo.toml`, including `solana-program` (1.14.12), `spl-token` (3.5.0), `borsh` (0.9.3), `spl-associated-token-account` (1.1.3), `winnow` (0.4.1), and `time` (0.3.19). Some, like `solana-program`, are outdated by more than two versions.

### **Impact**

While there are no specific vulnerabilities related to these outdated versions that we can discern, outdated dependencies can contain security vulnerabilities that have been patched in updated versions. This can introduce unnecessary vulnerabilities into the system and increase the attack surface down the line.

### **Affected Resource**

`solcial-stacking/Cargo.toml`

### **Recommendation**

Ensure that all dependencies are continually updated to the latest version. Or, if the outdated version needs to be used, ensure that version notes are continually checked to verify there are no security vulnerabilities present in the outdated version.

### **Reference**

<https://docs.rs/crate/<any crate name>>

## KS-04 Absence of Anchor Framework

Severity

Informational

### **Description**

The Anchor framework is not being used in the codebase.

### **Impact**

The Anchor framework is a well-tested and well-respected framework in the Solana ecosystem that not only increases the ease of development in Solana, but it includes multiple built-in features, including automatic serialization/deserialization, type safety checks, ownership checks, signer checks, and, most importantly, security features. Re-implementing these features from scratch opens the codebase to unnecessary security vulnerabilities and potentially a greater attack surface.

### **Affected Resource**

The entire `solcial-stacking` program is affected.

### **Recommendation**

Use the Anchor framework to avoid potential security issues or errors that would come from doing everything from scratch.

### **Reference**

<https://www.anchor-lang.com/>

## KS-05 Frequent misspellings

Severity

Informational

### Description

There are multiple instances of misspellings of common words in variable name present throughout the code, specifically “amout” in `user.rs` (also used in `user_stack.rs` and `user_unstack.rs`) and “vaultAuthoruty” in `user_unstack`.

### Impact

While the misspellings are consistent across the program and thus have no immediate ramifications, it is important to note that misspellings could have indirect security implications in the future. They increase the complexity and decrease the readability of a codebase, leading to confusion and lack of clarity for future developers as well as future development of and changes to the codebase.

### Affected Resource

```
src/instructions/user_unstack
src/instructions/user_stack
src/state/user.rs
```

### Recommendation

Correct misspellings across the codebase.

### Reference

N/A

## METHODOLOGY

During this source code review, the Kudelski Security Services team reviewed code within the project within an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase the team works through the following categories:

- Authentication
- Authorization and Access Control
- Injection and Tampering
- Configuration Issues
- Logic Flaws
- Cryptography

### **Tools**

The following tools were used during this portion of the test.

- Visual Studio Code
- Semgrep
- Cargo Audit

## Vulnerability Scoring Systems

Kudelski Security utilizes a vulnerability scoring system based on impact of the vulnerability, likelihood of an attack against the vulnerability, and the difficulty of executing an attack against the vulnerability based on a high, medium, and low rating system

### Impact

The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

#### High:

The vulnerability has a severe effect on the company and systems or has an effect within one of the primary areas of concern noted by the client.

#### Medium:

It is reasonable to assume that the vulnerability would have a measurable effect on the company and systems that may cause minor financial or reputational damage.

#### Low:

There is little to no effect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

### Likelihood

The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, and institutional knowledge

#### High:

It is extremely likely that this vulnerability will be discovered and abused.

#### Medium:

It is likely that this vulnerability will be discovered and abused by a skilled attacker

#### Low:

It is unlikely that this vulnerability will be discovered or abused when discovered.

### Difficulty

Difficulty is measured according to the ease of exploit by an attacker based on availability of readily available exploits, knowledge of the system, and complexity of attack. It should be noted that a LOW difficulty results in a HIGHER severity.



**Low:**

The vulnerability is easy to exploit or has readily available techniques for exploit.

**Medium:**

The vulnerability is partially defended against, difficult to exploit, or requires a skilled attacker to exploit.

**High:**

The vulnerability is difficult to exploit and requires advanced knowledge from a skilled attacker to write an exploit.

**Severity**

Severity is the overall score of the weakness or vulnerability as it is measured from Impact, Likelihood, and Difficulty.