

Secure Code Review

Findings and Recommendations Report Presented to:

Solcial

October 06, 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
Key Findings.....	4
Scope and Rules Of Engagement	5
TECHNICAL ANALYSIS & FINDINGS	6
Findings.....	7
KS-01 Subscribers count can be easily manipulated	8
KS-02 Pending rewards is not zeroed after an owner claims rewards	11
KS-03 Lack validation of token-program id	12
METHODOLOGY	13
Tools	13
Vulnerability Scoring Systems.....	14
KUDELSKI SECURITY CONTACTS	Error! Bookmark not defined.

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-subscription program.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place between September 20th 2023 and October 6th 2023, and it was focused on the following objectives:

- 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 **2 high findings** and **1 informational** finding 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 issue, 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
- Connections and CPI calls to other programs

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.

- The social-subscription codebase review resulted in 2 high severity findings related to assets management. These issues should be addressed as soon as possible since they can result in loss of user funds.
- The mentioned findings have an easy-to-implement fix and shouldn't require too much effort to be developed.
- The informational finding is related to accounts validation, to have a program with good practices implemented.

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.

Commit Hash
c3430a37df7f38aff3fabf0817dd70376fd289e0
In-Scope Contracts
solcial-subscription

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Secure Code Review, we identified **2 high findings** and **1 informational** finding according to our Vulnerability Scoring System.

The following chart displays the findings by severity.

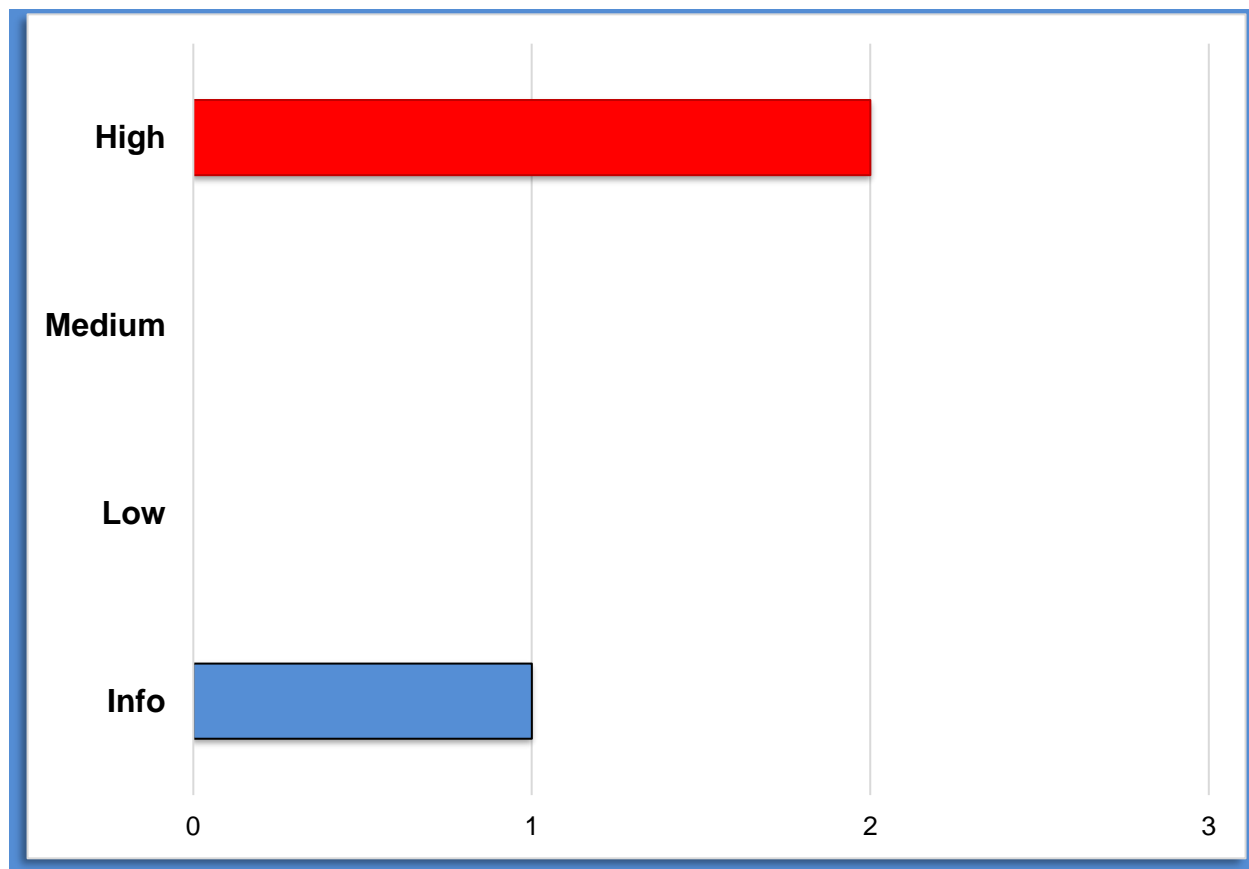


Figure 1: Findings by Severity

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
1	High	Subscribers count can be easily manipulated
2	High	Pending rewards is not zeroed after an owner claims rewards
3	Informational	Lack of validation of token-program ids

Table 2: Findings Overview

KS-01 Subscribers count can be easily manipulated

Severity

High

Impact	Likelihood	Difficulty
High	High	Low

Description


The amount of subscribers for a Subscription can be easily manipulated without spending any tokens.

If a user calls the `create_user_account` instruction with a `duration = 0`, then the `subscription.amount` is not incremented, but the `subscription.subscribers` is incremented by 1.

Impact

As the amount claimed by a subscription's owner is directly proportional to the amount of subscribers, a malicious subscription's owner can call the `create_user_account` function to increment the `subscription.subscribers` number and subsequently increment the amount of tokens claimed.

Evidence



```
user_create_account.rs

40 #[derive(Debug, BorshDeserialize)]
41 pub struct Config {
42     duration: u64,
43 }
44
45 pub fn create_user_account(accounts: Accounts, config: Config) -> ProgramResult {
46     msg!("Call create_user_account");
47     // TODO: check susbscription account state
48     // TODO: check owner is signer
49
49     Snipped
```

Duration can be set to 0.


```
user_create_account.rs
80 let amount = config.duration.saturating_mul(subscription.price);
81
82 if accounts.subscription_vault.lamports() == 0 {
83     msg!("Create subscription_vault");
84     invoke(
85         &spl_associated_token_account::instruction::create_associated_token_account(
86             accounts.payer.key,
87             accounts.subscription_vault.key,
88             accounts.mint.key,
89         ),
90         &[
91             accounts.payer.clone(),
92             accounts.subscription_vault.clone(),
93             accounts.subscription.clone(),
94             accounts.mint.clone(),
95             accounts.system_program.clone(),
96             accounts.token_program.clone(),
97             accounts.associated_token_program,
98         ],
99     );
100 }
101 // do token transfer
102 token_deposit(
103     accounts.token_program,
104     accounts.mint,
105     accounts.payer_token_account,
106     accounts.subscription_vault,
107     accounts.payer.clone(),
108     amount,
109     decimals,
110 );
```

Snipped

Amount of tokens transferred is 0 since `config.duration` is 0. Line 102 will be executed without problem.

```
user_create_account.rs
142 subscription.subscriber_count = subscription.subscriber_count.saturating_add(1);
143 subscription.vault_amount = subscription.vault_amount.saturating_add(amount);
```

Snipped

`subscriber_count` is incremented by 1.

```
owner_claim.rs
52 let number_of_follower = subscription.subscriber_count;
53 let price = subscription.price;
54
55 let amount = compute_amount(&now, &last_claimed, number_of_follower, price)
56 + subscription.pending_reward;
57
58 let mint = Mint::unpack(&accounts.mint.data.borrow());
59 let bump = &[bump];
60 let bump_seed = subscription.compute_seed(bump);
61
62 token_withdraw(
63     accounts.token_program,
64     accounts.subscription_vault,
65     accounts.mint,
66     accounts.recipient_token_account,
67     accounts.subscription,
68     &bump_seed,
69     amount,
70     mint.decimals,
71 );
```

Snipped

```
owner_claim.rs
142 pub fn compute_amount(
143     now: &OffsetDateTime,
144     last_claimed: &OffsetDateTime,
145     number_of_follower: u64,
146     price: u64,
147 ) -> u64 {
148     let (now_year, now_mounth, _now_day) = now.to_calendar_date();
149     let (last_year, last_mounth, _last_day) = last_claimed.to_calendar_date();
150     let delta_year = now_year - last_year;
151     let delta_mounth = if delta_year == 0 {
152         now_mounth as i32 - last_mounth as i32
153     } else {
154         (12 - last_mounth as i32) + now_mounth as i32
155     };
156     let duration_mounth = ((delta_year - 1).max(0) * 12 + delta_mounth) as u64;
157     duration_mounth * price * number_of_follower
158 }
159
```

Snipped

The image on the right shows that the amount to be transferred to the owner is directly proportional to `number_of_follower`.

The `number_of_follower` is obtained from the subscription account, as shown on the image to the left.

Recommendation

In `user_create_account` validate that the duration is bigger than 0.

Affected Resource

- `src/instructions/user_create_account.rs`

KS-02 Pending rewards is not zeroed after an owner claims rewards

Severity

High

Impact	Likelihood	Difficulty
High	High	Low

Description

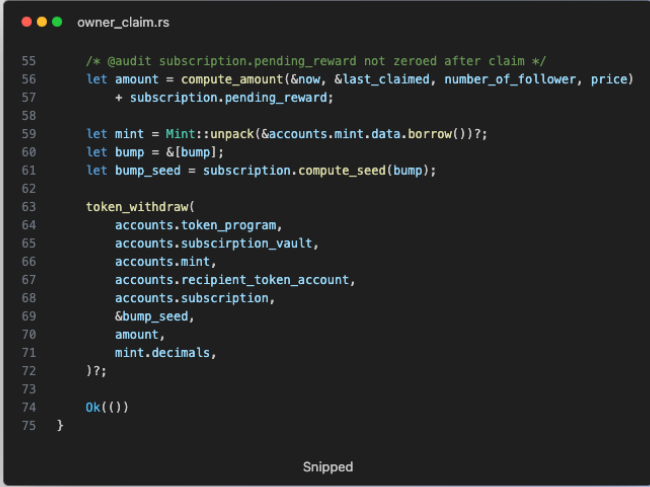
The `owner_claim` function transfers tokens proportional to the amount of subscribers plus a `pending_reward` amount.

After claiming, the `pending_reward` is expected to be 0 again, which is not the case for the current implementation.

Impact

A subscription owner can call the mentioned function multiple times, even in the same instruction, to drain the subscription vault without having to wait.

Evidence



```
owner_claim.rs
55  /* @audit subscription.pending_reward not zeroed after claim */
56  let amount = compute_amount(&now, &last_claimed, number_of_follower, price)
57    + subscription.pending_reward;
58
59  let mint = Mint::unpack(&accounts.mint.data.borrow());
60  let bump = &[bump];
61  let bump_seed = subscription.compute_seed(bump);
62
63  token_withdraw(
64    accounts.token_program,
65    accounts.subscription_vault,
66    accounts.mint,
67    accounts.recipient_token_account,
68    accounts.subscription,
69    &bump_seed,
70    amount,
71    mint.decimals,
72  );
73
74  Ok(())
75 }
```

The image above shows that the `pending_reward` value is used to transfer tokens in the `owner_claim` function but is not zeroed after.

Recommendation

Set `subscription.pending_reward` to 0 after transferring tokens in the `owner_claim` function.

Affected Resource

- `src/instructions/owner_claim.rs`

KS-03 Lack of validation of token-program id

Severity

Informational

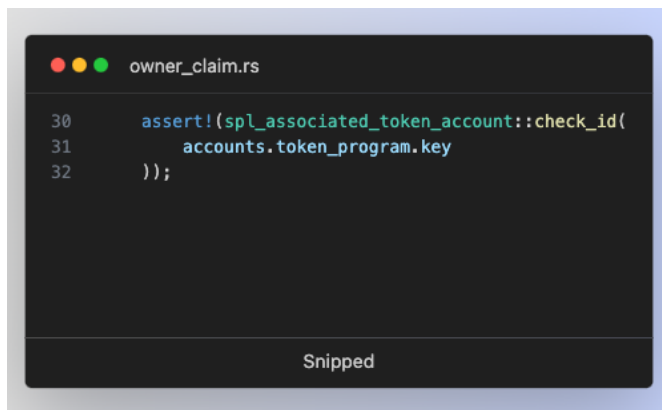
Description

The `token-program` id provided as a parameter to some instructions is not checked to be the expected id.

It is considered a good practice to validate on chain the `token-program` id and any other program id provided to prevent unexpected results.

Recommendation

Add the corresponding validation in the mentioned affected resources. The image below shows an implementation example.



```
owner_claim.rs  
  
30     assert!(spl_associated_token_account::check_id(  
31         accounts.token_program.key  
32     ));  
  
Snipped
```

Affected Resource

- `/src/instructions/owner_claim.rs`
- `/src/instructions/user_create_account.rs`

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. A link for more information about the tool is provided as well.

- 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