Code Audit

for

Streamflow Labs

15 Rue des Halles
75001 Paris
France

https://opcodes.fr
contact@opcodes.fr

Siret :89435788800011
RCS : Paris B 894 357 888
TVA : FR35894357888

Project Information

Project	
Mission	Code audit
Client	Streamflow Labs
Start Date	12/20/2021
End Date	12/29/2021

Document Revision			
Version	Date	Details	Authors
1.0	12/28/2021	Document creation	Thibault MARBOUD Xavier BRUNI
1.0	12/29/2021	Peer review	Baptiste OUERIAGLI
2.0	01/06/2022	Public version	Thibault MARBOUD

Table of Contents

Overview

Mission Context

The purpose of the mission was to perform a code audit to discover issues and vulnerabilities in the mission scope. A comprehensive testing has been performed utilizing automated and manual testing techniques.

Mission Scope

As defined with Streamflow Labs prior to the mission, the scope of this assessment was two Solana programs. This report only concerns the Timelock-crate, another one has been edited for the Timelock implementation. The code source was supplied through the following GitHub repositories:

- https://github.com/StreamFlow-Finance/timelock-crate / b1caedd (community)
- https://github.com/StreamFlow-Finance/timelock / 08f8468 (community)

OPCODES engineers are due to strictly respect the perimeter agreed with Streamflow Labs as well as respecting ethical hacking behavior.

Note: OPCODES engineers audited the community branch.

Project Summary

The community version of Timelock-crate is token vesting library for Solana programs. It provides all the logic necessary to lock SPL tokens and distribute them to a specific recipient within a given timeframe. The community version of Timelock-crate is meant to be used as library and is NOT a standalone program. Developers implementing the library will have to be careful and understand that they are not invoking an external program.

The application logic is easy to understand, and integration tests are done using Anchor framework in the second Github repository. The code is well commented and easy to read which facilitated the audit.

Synthesis

Security Level: GOOD

The overall security level is considered as good.

OPCODES investigation did not produce major or critical severity results and showed a good understanding of needed security checks, thus preventing any harmful scenario.

In total 5 vulnerabilities have been discovered during the assessment. One medium vulnerability regarding account confusion. Two minor vulnerabilities that does not lead to any exploitable scenario but may enforce bad practices. Two informational issues that represent possible improvements.

Vulnerabilities summary

Total vulnerabilities	5
Critical	0
Major	0
Medium	1
Minor	2
Informational	2

Vulnerabilities & issues table

Identified vulnerabilities

Ref	Vulnerability title	Severity	Remediation effort
#1	Library design could lead to account confusion	Medium	Medium
#2	Unsafe arithmetic	Minor	Low
#3	PDA seeds could lead to collision	Minor	Low
#4	Use of float operations	Informational	Medium
#5	Unable to request closed/open metadata account	Informational	Low

Identified vulnerabilities

Library design could lead to account confusion

Severity	Remediation effort
Medium	Medium

Description

Implementing the Timelock library inside a Solana program could lead to a vulnerability called “account confusion”. Indeed, the library will create accounts that will be owned by the program it implements. Anyone using the library will have to be very careful to not confuse accounts created by the Timelock library with accounts created by its own program and vice versa.

Scope

Timelock-crate

Risk

The risk only exists when using the library inside a program that add more logic involving accounts. It could lead to account confusion between the library and program accounts.

Remediation

This issue is not easy to remediate as it depends on how the library is implemented. But we think that first, account confusion checks should be done inside Timelock crate. We noticed that the first attribute of *TokenStreamData* structure is an unused *u64* called *magic*. This variable could be used to store an arbitrary value. Below is an example to illustrate our recommended remediation.

```
const METADATA_V1: u64 = 0x4901cee6;

#[allow(clippy::too_many_arguments)]
impl TokenStreamData {
    /// Initialize a new `TokenStreamData` struct.
    pub fn new(
        [...]
    ) -> Self {
        [...]
        Self {
            magic: METADATA_V1,
            created_at,
            withdrawn_amount: 0,
            canceled_at: 0,
            cancellable_at: end_time,
            last_withdrawn_at: 0,
            sender,
            sender_tokens,
            recipient,
            recipient_tokens,
            mint,
            escrow_tokens,
            ix,
        }
    }
```

Every instruction deserializing metadata accounts should check if that arbitrary value is correct.

```
    let mut data = acc.metadata.try_borrow_mut_data()?;
    let mut metadata = match TokenStreamData::try_from_slice(&data) {
        Ok(v) => v,
        Err(_) => return Err(ProgramError::InvalidAccountData),
    };

    if metadata.magic != METADATA_V1 {
        // Account confusion
        return Err(ProgramError::InvalidAccountData);
    }
```

Developers implementing the library may need to add security checks regarding account confusion. A public function allowing to identify if an account is owned by the Timelock library could be added.

Finally, a warning message should be added to Streamflow documentation to prevent developers from making mistake regarding account confusion.

Note: If a modification is done to the `TokenStreamData` structure, we recommend creating a new constant with a different value to identify version of the metadata account.

Unsafe arithmetic

Severity	Remediation effort
Minor	Low

Description

Timelock library uses unsafe arithmetic operation that may lead to integer overflow/underflow.

src/token.rs (L338)

```
metadata.withdrawn_amount += requested;
```

src/token.rs (L476)

```
metadata.withdrawn_amount += available;
let remains = metadata.ix.total_amount - metadata.withdrawn_amount;
```

src/state.rs (L200)

```
    let num_periods = (self.ix.end_time - cliff) as f64 / self.ix.period as f64;
    let period_amount = (self.ix.total_amount - cliff_amount) as f64 / num_periods;
    let periods_passed = (now - cliff) / self.ix.period;
    (periods_passed as f64 * period_amount) as u64 + cliff_amount -
self.withdrawn_amount
```

Scope

Timelock-crate

Risk

The risk is low as most of the unsafe operation are not exploitable because of proper controls and/or are involving input from the user depositing the funds.

Remediation

Keep in mind that integer overflow/underflow are possible in rust. We recommend adding overflow check when performing critical arithmetic operation in a Solana program, especially when user inputs are involved.

Overflow and underflow checks can be done using the following methods of primitive types: *checked_add*, *checked_div*, checked_mul, *checked_neg*, *checked_sub*. The documentation is available here: https://docs.rs/num-traits/0.2.14/num_traits/ops/checked/index.html.

Below is an example of safe addition for the *withdraw* instruction.

```
// metadata.withdrawn_amount += requested;
metadata.withdrawn_amount = metadata
.withdrawn_amount
.checked_add(requested)
.ok_or(ProgramError::InvalidArgument)?;
```

Note: Overflow/underflow checks can be omitted if you are certain that it will not happen.

PDA seed could lead to collision

Severity	Remediation effort
Minor	Low

Description

Timelock library uses Program Derived Address (PDA) to generate an escrow account that will receive and transfer the funds to the stream recipient. PDA are generated using the metadata account public key as a seed.

Scope

Timelock-crate

Risk

Timelock library may be implemented in a program that also uses PDA. In that case, a collision could happen as PDA are often generated with a public key as a seed.

To be exploitable this issue would require some lack of control and security checks.

Remediation

Developers that implement the library should not worry about PDA collision. To avoid that issue, we recommend adding a prefix to the PDA seed as shown in the following example.

```
const PREFIX: &str = "timelock";
...
let (escrow_tokens_pubkey, nonce) =
    Pubkey::find_program_address(&[PREFIX.as_bytes(), acc.metadata.key.as_ref()],
program_id);
```

Use of float operations

Severity	Remediation effort
Informational	Medium

Description

Timelock library is using float operations to compute the funds available for withdraw given a timestamp.

src/state.rs (L209)

```
let num_periods = (self.ix.end_time - cliff) as f64 / self.ix.period as f64;
let period_amount = (self.ix.total_amount - cliff_amount) as f64 / num_periods;
let periods_passed = (now - cliff) / self.ix.period;
(periods_passed as f64 * period_amount) as u64 + cliff_amount - self.withdrawn_amount
```

Scope

Timelock-crate

Risk

As mentioned in Solana documentation, float support is limited and arithmetic operation involving float numbers consume an excessive amount of compute units.

https://docs.solana.com/developing/on-chain-programs/overview#float-support

Remediation

When possible, we recommend using integer operation.

Unable to request closed/open metadata account

Severity	Remediation effort
Informational	Low

Description

When closed, metadata accounts are not deleted in order to keep a history of all the vesting. But no flag is set indicating that the account is closed.

Scope

Timelock-crate

Risk

At the time of the report, the only way to know if a metadata account is closed is to compare the variables *withdrawn_amount* and *total_amount*. In order to get the list of all the closed metadata account, developers will have to retrieve all the accounts data, deserialize them and finally compare the variables.

Remediation

We recommend adding a Boolean attribute inside the *TokenStreamData* structure that indicate if the metadata account is closed or not. This way, it will be easier to request closed/open metadata accounts using one RPC request without any computation.

Conclusion

OPCODES investigation did not produce major or critical severity results and showed a good understanding of needed security checks, thus preventing any harmful scenario.

We recommend patching the medium vulnerability regarding account confusion. It will require some work to be done but will protect projects using Timelock library. Even if minor vulnerabilities are not exploitable, we advise Timelock team to be careful especially with unsafe arithmetic. The informational issues only represent improvements that could be done and do not represent any direct or indirect security risk.