Code Audit

for

Streamflow Labs

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

15 Rue des Halles, 75001 Paris, France

Project Information

Project	
Mission	Code audit
Client	Streamflow Labs
Start Date	02/08/2022
End Date	02/23/2022

Document Revision			
Version	Date	Details	Authors
1.0	02/21/2022	Document creation	Thibault MARBOUD Théo LEJEUNE
1.1	02/23/2022	Peer review	Xavier BRUNI
2.0	03/02/2022	Public version	Xavier BRUNI

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. Comprehensive testing has been performed utilizing automated and manual testing techniques.

Mission Scope

As defined with Streamflow Labs before the mission, the scope of this assessment was two Solana programs. This report only concerns the Protocol program, another one has been edited for the Partner oracle program. The code source was supplied through the following GitHub repositories:

- https://github.com/streamflow-finance/protocol / 7fc6d89 (main)
- https://github.com/streamflow-finance/partner-oracle / 50a145e (master)

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

At the end of the audit, Streamflow Labs patched multiple reported vulnerabilities. These patches have been reviewed by OPCODES up to commit cf12851. The issues table list the status of each vulnerability (Fixed / Accepted risk / Rejected) and the vulnerability write-up contains a link to the resolution commit when applicable.

Note: OPCODES engineers audited the main branch of the protocol repository and master branch of the partner oracle repository.

Project Summary

As a reminder, streamflow is building a token vesting library/application on Solana. It provides all the logic necessary to lock SPL tokens and distribute them to a specific recipient within a given timeframe.

The Protocol repository is a more advanced token vesting with a new instruction called *topup*, allowing users to add funds to an existing stream. More logic has been added regarding fees, the program now supports sharing them with an identified set of partners.

Regarding testing, the program is not widely covered. There are too few tests that do not pass at the current commit. OPCODES strongly encourage Streamflow to improve the testing and coverage of the program.

The application is developed without any framework. If possible, it should be considered to use the Anchor framework. This allows for easier testing using the tools provided by Anchor ecosystem. From a security point of view, the vulnerability found could have been avoided with Anchor.

Synthesis

Security Level: GOOD

The overall security level is considered as good. OPCODES investigation only produced 1 major severity result which has been fixed.

In total 9 vulnerabilities have been discovered during the assessment.

One major vulnerability regarding a missing account validation on the cancel instruction. This issue allowed a recipient to completely bypass the vesting logic and claims the token right after the stream creation.

One minor vulnerability that concerns the validation of the *owner* field of the associated token account.

OPCODES also reported three other minor vulnerabilities that do not lead to any exploitable scenario but may enforce bad practices. Four informational issues have been created; they represent possible improvements.

Vulnerabilities summary

Total vulnerabilities	9
Critical	0
Major	1
Medium	0
Minor	3
Informational	5

Vulnerabilities & issues table

Identified vulnerabilities

Ref	Vulnerability title	Severity	Remediation effort	Status
#1	Missing sender account validation on cancel instruction	Major	Low	Fixed
#2	Missing owner validation on associated token account	Minor	Medium	Accepted risk
#3	Loss of precision in fee calculation	Minor	Low	Accepted risk
#4	*Close* attribute is not used nor validated	Minor	Low	Fixed
#5	Missing user input validation inside topup	Informational	Low	Fixed
#6	Users that do not have ownership over their ATA anymore can't use Streamflow	Informational	Medium	Accepted risk
#7	Unsafe arithmetic	Informational	Low	Fixed
#8	Usage of float	Informational	Low	Accepted risk
#9	Outdated dependencies	Informational	Low	Accepted risk

Identified vulnerabilities

Missing sender account validation on cancel instruction

Severity	Remediation effort	Status
Major	Low	Fixed

Description

The *cancel* instruction allows the sender and/or recipient (depending on the stream configuration) to cancel the vesting. When calling the instruction, part of the funds remaining in the escrow account will be sent back to the sender. To transfer the funds, the sender account must be passed in argument but is not validated against the sender account inside the metadata structure.

Scope

Protocol

Risk

When the recipient has the right to cancel the stream, he could provide a wrong sender account that will receive the remaining funds. Thus, bypassing the token vesting logic and claiming all the tokens.

Remediation

The program should validate the accounts *sender* and *sender_token* of the *CancelAccounts* structure against the *sender* and *sender_token* stored inside the metadata account. The following code can be used as an example to patch the vulnerability.

```
fn metadata_sanity_check(acc: CancelAccounts, metadata: Contract) -> ProgramResult {
    // Compare that all the given accounts match the ones inside our metadata.
-    if acc.recipient.key != &metadata.recipient ||
+    if acc.sender.key != &metadata.sender ||
+        acc.sender_tokens.key != &metadata.sender_tokens ||
+        acc.recipient.key != &metadata.recipient ||
        acc.recipient_tokens.key != &metadata.recipient_tokens ||
        acc.mint.key != &metadata.mint ||
        acc.escrow_tokens.key != &metadata.escrow_tokens ||
        acc.streamflow_treasury.key != &metadata.streamflow_treasury ||
        acc.streamflow_treasury_tokens.key != &metadata.streamflow_treasury_tokens ||
        acc.partner.key != &metadata.partner ||
        acc.partner_tokens.key != &metadata.partner_tokens
    {
        return Err(SfError::MetadataAccountMismatch.into())
    }

    // Passed without touching the lasers
    Ok(())
}
```

Fix

This issue has been discussed and fixed with commit [40a341a](40a341a) during the audit.

Missing owner validation on associated token account

Severity	Remediation effort	Status
Minor	Medium	Accepted risk

Description

The *account_sanity_check* function of the create instruction validate the different accounts passed by the user. It also checks the different *associated token account (*ATA) *b*ut these checks are not enough as the authority of an *associated token account* can be changed. It is possible to have an ATA with a different owner than the one used in the seed.

Scope

Protocol

Risk

Upon withdrawal or stream cancellation, tokens might get transferred to an ATA that is no longer owned by the original user.

Remediation

Each *associated token account* should be deserialized to ensure the owner field of the SPL Account structure is the rightful owner.

Note that the ATA program is getting updated to prevent authority from being changed by default and add a new instruction called *initialize_immutable_owner* to create ATA: https://github.com/solana-labs/solana-program-library/pull/2854

Anchor has also updated its constraint on AssociatedToken for version 0.20.0 to ensure that the ATA is still owned by the original owner: https://github.com/project-serum/anchor/pull/1240/commits/676a3d8255129424cecb055765cee132986bad36

Streamflow comment

At this point, we do not have a viable remediation strategy for this case. We think that the user should be responsible for token account management and simply raising an error if recipient is not owner of the ATA will result in a stuck state of the contract which we wish to avoid. We will iterate on this and look for a way to protect our customers without inducing significant protocol changes in the future.

Loss of precision in the fee calculation

Severity	Remediation effort	Status
Minor	Low	Accepted risk

Description

Because of a division, the function *calculate_fee_from_amount* inside *topup* instruction leads to a loss of precision that the user could exploit in order to avoid Streamflow's fees.

Scope

Protocol

Risk

By sending a small amount, the result of the fee's calculation would be zero, thus bypassing Streamflow's fees.

The risk of this vulnerability could increase if Solana transaction fees decrease. It is even more relevant as Solana is planning an upgrade of his fee system which will be based on the total number of compute units consumed by the transaction.

As the price for sending low CU consuming transaction decreases, a potential attack vector might become exploitable whereas an attacker would call the *topup* instruction multiple times with a lot of very small amount.

Remediation

We think that the fee should be calculated using the *try_ceil_div* function to avoid any loss of fees. Below is an example of the patch we propose.

```
pub fn calculate_fee_from_amount_patch(amount: u64, percentage: f32) -> Result<u64,
ProgramError>  {
    if percentage <= 0.0 { return 0 }

    let fees = (percentage * 100.0) as u64;
    return amount.try_mul(fees)?.try_ceil_div(10000)?;
}
```

A warning message could be shown to the user to warn them about the minimal amount to top up.

Note: Another vulnerability has been created regarding the use of float.

Streamflow comment

We accept the risk of not charging fees for small amount streams and will address this in future releases. The proposed remediation introduces new negative consequences such as fee precision lost beyond 2 decimals, potential integer overflow for large token amounts etc..

Close attribute is not used nor validated

Severity	Remediation effort	Status
Minor	Low	Fixed

Description

When closing a stream, the *close* field of the metadata account is not set to the value *true*. Furthermore, it is never validated inside the other instructions.

Scope

Protocol

Risk

It is possible to interact with a “closed stream” using other instructions such as *transfer.*

Remediation

We recommend setting the *closed* field of the metadata account to *true* when closing a stream. Moreover, the status of the stream should be checked inside each instruction to ensure that the program is not interacting with a closed stream.

Note: A closed stream acts as a history, the closed field should act like a frozen attribute and no changes should be made to a closed stream.

Fix

This issue has been fixed in commit cf12851.

Missing user input validation inside topup

Severity	Remediation effort	Status
Informational	Low	Fixed

Description

Unlike all the other instructions, the *topup* instruction does not have an *account_sanity_check* and *metadata_sanity_check* functions.

First, anyone can call the *topup* instruction for any stream, which may be an intended behavior. We think that only a set of identified users should be able to interact with a specific stream. This may help to limit any malicious users from tampering with others' stream and we see no clear benefits to allow anyone to topup a stream they don't have authority over.

Second, the mint account is not verified against the mint inside the metadata account but is deserialized and used to print a debug message.

Lastly, no checks are done regarding the field *is_writable* of the accounts.

Scope

Protocol

Risk

None of these issues represent a real security risk, but it enforces bad practice and could lead to a missing validation in the future.

Remediation

We recommend adding both *account_sanity_check* and *metadata_sanity_check* functions inside the *topup* instruction. This will help keep consistency across your codebase and other instruction.

Fix

This issue has been fixed in commit cf12851.

Users that do not have ownership over their ATA anymore can't use Streamflow

Severity	Remediation effort	Status
Informational	Medium	Accepted risk

Description

The protocol program only accepts associated token account as source and destination of the tokens. Each user can only create a single associated token account for each token and given that ATA ownership can be transferred, a user might at some point be in a situation where he doesn’t own his ATA and does not have the possibility to create a new one. In this case, he is unable to use streamflow.

Scope

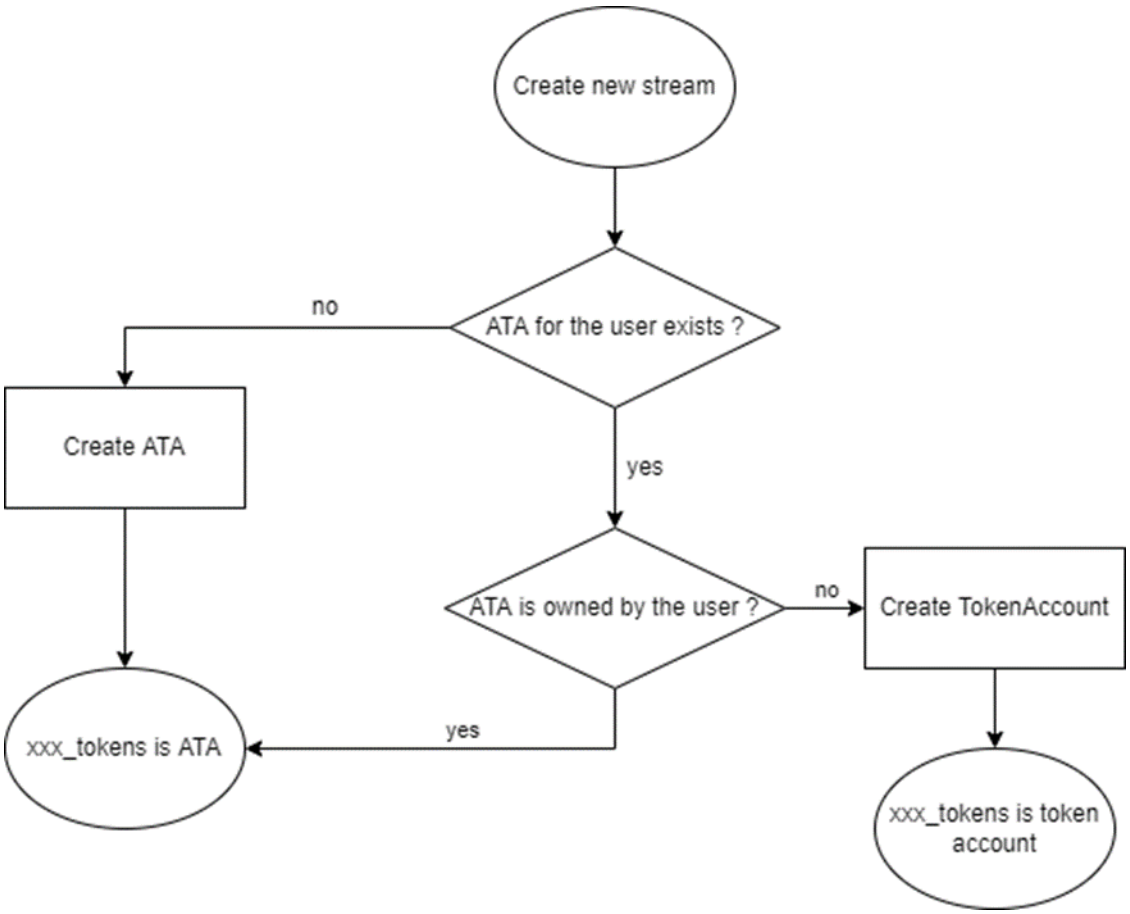
Protocol

Risk

Some users might not be able to use streamflow.

Remediation

We recommend to implement the flow diagram below



Flow for the withdraw and cancel instruction should also be changed to allow users to recover the tokens they are entitled even if the token account ownership changed after the creation of the stream.

Streamflow comment

Reason specified in the "Missing owner validation on associated token account" section since it's closely related.

Unsafe arithmetic

Severity	Remediation effort	Status
Informational	Low	Fixed

Description

The program uses unsafe arithmetic operations that may lead to integer overflow/underflow.

programs/protocol/src/utils.rs (L45)

```
pub fn pretty_time(t: u64) -> String {
    let seconds = t % 60;
    let minutes = (t / 60) % 60;
    let hours = (t / (60 * 60)) % 24;
    let days = t / (60 * 60 * 24);

    format!("{} days, {} hours, {} minutes, {} seconds", days, hours, minutes, seconds)
}
```

programs/protocol/src/utils.rs (L89)

```
pub fn calculate_fee_from_amount(amount: u64, percentage: f32) -> u64 {
    if percentage <= 0.0 {
        return 0
    }
    let precision_factor: f32 = 1000000.0;
    let factor = (percentage / 100.0 * precision_factor) as u128;
    (amount as u128 * factor / precision_factor as u128) as u64
}
```

programs/protocol/src/create.rs (L220)

```
let mut metadata_struct_size = metadata_bytes.len() + METADATA_PADDING;
while metadata_struct_size % 8 > 0 {
    metadata_struct_size += 1;
}
```

Scope

Protocol

Risk

The risk is low as most of the unsafe operations are not exploitable because of proper controls and/or are not involving user inputs.

Remediation

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

Fix

This issue has been fixed in commit cf12851 (additional test cases were added, *pretty_time* was removed).

Usage of float

Severity	Remediation effort	Status
Informational	Low	Accepted risk

Description

Protocol program uses float to compute the fees and the withdrawable amount available. Float numbers are also stored inside the metadata account with the attributes *streamflow_fee_percent* and *partner_fee_percent*.

programs/protocol/src/utils.rs (L54)

```
pub fn calculate_available(now: u64, end: u64, ix: CreateParams, total: u64, withdrawn: u64, fee_percentage: f32)
```

programs/protocol/src/utils.rs (L89)

```
pub fn calculate_fee_from_amount(amount: u64, percentage: f32)
```

programs/protocol/src/state.rs (L15)

```
pub const STRM_FEE_DEFAULT_PERCENT: f32 = 0.25;
```

programs/protocol/src/state.rs (L149)

```
pub streamflow_fee_percent: f32,
```

programs/protocol/src/state.rs (L149)

```
pub partner_fee_percent: f32,
```

Scope

Protocol

Risk

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

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

Remediation

We recommend avoiding the storage of float numbers inside any structure. When it comes to percentage, especially for the fees, storing the percentage multiplied by 100 should be enough. Avoiding float operation is also a good practice and should be done when possible.

Streamflow comment

We propose not to change the type of fee_percentage variables to integer at this point, because we think benefits do not outweigh the costs of migrating and upgrading the accounts as we did not yet develop a secure process of state account upgrades. The effort of switching fee_percentage to an integer value will be much cheaper once we enable state upgrades that are planned after Q12022, at which point we will switch to using integer for these fields.

Outdated dependencies

Severity	Remediation effort	Status
Informational	Low	Accepted risk

Description

The following crates could be updated:

Crate	Current version	Latest version
anyhow	1.0.53	1.0.54
solana-program	1.9.5	1.9.8
solana-program-test	1.9.5	1.9.8
solana-sdk	1.9.5	1.9.8
spl-token	3.2.0	3.3.0
anchor-lang	0.19.0	0.22.0
anchor-spl	0.19.0	0.22.0

Scope

Protocol

Risk

This is an informational issue, it does not represent any risk but may enforce bad practice.

Remediation

It is considered a good practice to update your dependencies when possible.

Streamflow comment

We fully support the attitude that dependencies should be updated regularly, but we would like to accept current dependency tree versions in place for the time being. Some of the

dependencies introduce structural changes that take considerable amount of time to adapt and test and we do not feel confident doing this at his stage of the audit process. This will be addressed in future releases.

Conclusion

Overall, the Protocol program seems less mature than the Timelock-crate and work is still needed. OPCODES strongly recommend improving the integration tests and the coverage of the program.

The major vulnerability regarding the missing account validation has already been patched during the audit.

We advise Streamflow team to be careful with minor vulnerabilities and remediate them when possible. The informational issues only represent improvements that could be done and do not represent any direct or indirect security risk.

In the future, it should be considered to switch the program to Anchor which drastically reduces the number of account checks required and improves code readability.