Code Audit

for

Streamflow Finance

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

15 Rue des Halles, 75001 Paris, France

Project Information

Project	
Mission	Code audit
Client	Streamflow Finance
Start Date	06/20/2022
End Date	06/29/2022

Document Revision			
Version	Date	Details	Authors
1.0	06/29/2022	Document creation	Thibault MARBOUD Xavier BRUNI
1.1	06/30/2022	Peer review	Baptiste OUERIAGLI
2.0	07/12/2022	Streamflow comments	Thibault MARBOUD
2.1	07/13/2022	Peer review	Baptiste OUERIAGLI

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 using automated and manual testing techniques.

Mission Scope

As defined with Streamflow Finance before the mission, the scope of this assessment was the version 2 of their protocol Solana program. The code source was supplied through the following GitHub repositories:

- https://github.com/streamflow-finance/protocol / 51e6312 (release/v2)

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

Project Summary

Streamflow Finance is building a token vesting application on Solana. It provides all the logic necessary to lock SPL tokens and distribute them to a specific recipient within a given timeframe.

OPCODES team already audited the first version of the Streamflow's program and was therefore already familiar with the codebase prior to this audit. This assessment concerns the version two of the vesting program with new features and business logic. The program now includes two new instructions allowing to pause/resume a vesting. Streamflow engineers also added an optional feature allowing automatic withdrawal.

OPCODES noticed an improvement regarding testing, with the creation of integration tests. Indeed, each instruction is now tested multiple times with both successful and unsuccessful scenarios. OPCODES encourages Streamflow to continue in this direction, as testing is the first

step to security. Moreover, integration tests were useful to develop proof on concept for vulnerabilities identified during the assessment.

Synthesis

Security Level: GOOD

The overall security level is considered as good. Streamflow Finance showed a good understanding of the needed security hygiene. As a result, no funds are at risk.

The assessment demonstrated the presence of one medium vulnerability. This issue would allow anyone to override closed and canceled streams and erase the history of every vesting.

Three minor vulnerabilities have also been reported. They concern an imprecision with the fee computation, a logic bug with the pause functionality and missing ownership checks on associated token accounts.

OPCODES also added six informational issues to this report. They represent possible improvements and do not lead to any exploitable scenario but may enforce bad practices.

Vulnerabilities summary

Total vulnerabilities	10
Critical	0
Major	0
Medium	1
Minor	3
Informational	6

Vulnerabilities & issues table

Identified vulnerabilities

Ref	Vulnerability title	Severity	Remediation effort	Status
#1	Vesting history can be overridden	Medium	Low	Fixed
#2	Missing ATA ownership check	Minor	Medium	Accepted risk
#3	Fees might be null with small deposits	Minor	Low	Accepted risk
#4	Resuming *unstarted* stream increment *pause_cumulative*	Minor	Low	Fixed
#5	Usage and storage of float	Informative	Medium	Accepted risk
#6	Missing returned state	Informative	Low	Fixed
#7	Field *canceled_at* not set	Informative	Low	Fixed
#8	Missing owner check on Mint account	Informative	Low	Fixed
#9	Missing ownership check on Metadata account	Informative	Low	Fixed
#10	Outdated dependencies	Informative	Low	Accepted risk

Identified vulnerabilities

Vesting history can be overridden

Severity	Remediation effort	Status
Medium	Low	Fixed

Description

When a vesting ends, the stream account containing the data is not deleted in order to keep an history of all the vesting on the Solana blockchain. But the stream account can be overridden by a new vesting using the instruction `create_unchecked`.

When the last withdraw occurs or when a stream is canceled, the program calls the function `close_escrow`. It closes the escrow account that was containing the vested tokens and sets the stream field `closed` to the value `true`. Deleting the escrow account makes it possible to call `create_unchecked` because this instruction is expecting the escrow account to be empty to ensure that the stream isn't already existing.

Thanks to the integration tests, it was easy to provide a proof of concept for this issue, it has been included to this report in the annex.

Scope

Streamflow Finance program

Risk

Anyone can override a given closed vesting. This issue could allow a malicious third party to completely erase the onchain history of Streamflow Finance.

Remediation

OPCODES engineers recommend adding a security check inside the `create_unchecked` instruction to ensure that the account is not a closed stream. This can be done by deserializing the account and checking the value of the field *closed*.

Fix

Streamflow Finance followed OPCODES recommendation and patched the issue within commit 047bf7f.

Missing ATA ownership check

Severity	Remediation effort	Status
Minor	Medium	Accepted risk

Description

Streamflow Finance uses *associated token account* (ATA) to send tokens to a recipient. Every instruction using ATAs correctly check the seed of the account to ensure that it is a PDA of the Associated Token Program.

Whenever a new ATA is created, the owner of the ATA is a part of the seed. However, ownership of a token account can change, and streamflow program does not validate that the owner field of the Token Account structure is the same as the one used in the seed.

Scope

Streamflow Finance program

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 Token Account structure is the rightful owner.

Note that the Associated Token Program was updated to prevent the ownership from being changed by default. Unfortunately, this update is not deployed on mainnet yet.

Streamflow Finance comment

Streamflow Finance decided not to address this point for the following reasons:

The whole point of create_unchecked is to remove those checks to reduce amount of accounts in the instruction. I argue that this is a design decision - we are explaining the behavior and expecting users to use with caution. This is not a security concern but a design change that could reflect on users with advanced account usage and no understanding of the concepts.

Fees might be null with small deposits

Severity	Remediation effort	Status
Minor	Low	Accepted risk

Description

Streamflow Finance business model is to take fees from the vested tokens. Fees are paid using the same token as the vesting. If a vesting is using wrapped BTC, Streamflow will receive wrapped BTC. The fee calculation uses a classic percentage computation that will end up returning zero if the amount of vested token is small and/or the percentage is too low.

programs/protocol/src/utils.rs (L93)

```
/// Given amount and percentage, return the u64 of that percentage.
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 //[...]
                                                                //[...]
}
```

Scope

Streamflow Finance program

Risk

Streamflow Finance might not earn fees with tokens that have few decimals or no decimals at all.

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. 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

Even if tokens with small decimals are uncommon, OPCODES engineer would recommend fixing this issue by rounding up to the nearest superior number.

If Streamflow Finance does not wish to charge fees for small vesting, the *topup* instruction could be updated. Indeed, this issue could be partially fixed by computing the fees from the total number of vested tokens.

Resuming *unstarted* stream increment *pause_cumulative*

Severity	Remediation effort	Status
Minor	Low	Fixed

Description

A stream can be paused and resumed at any time. While pausing the stream before the release of the tokens and resuming it after is properly handled, the case where the stream is both paused and resumed before the release is not handled.

The *pause_cumulative* variable is incremented but logically shouldn't:

programs/protocol/src/state.rs (L522)

```
self.pause_cumulative.try_add_assign(current_pause_length)?;
```

Scope

Streamflow Finance program

Risk

The *effective_end_time()* function will return an incorrect result as it depends of the *pause_cumulative* value. Down the line, it can affect any calculation depending of *end_time* such as *withdraw_fees()* as well as prevent instructions from properly working like the cancel instruction.

Remediation

OPCODES engineers recommend adding a condition in order to leave the *pause_cumulative* field as it is when the stream is both paused and resumed before the release of the tokens.

Fix

Streamflow Finance followed OPCODES recommendation and patched this issue in commit 047bf7f.

Usage and storage of float

Severity	Remediation effort	Status
Informative	Medium	Accepted risk

Description

Streamflow Finance program uses float to compute the fees and the withdrawable amount available. Float numbers are also stored inside the metadata account in the fields *streamflow_fee_percent* and *partner_fee_percent*.

Scope

Streamflow Finance program

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 percentages, especially for the fees, storing the value multiplied by 10,000 should be enough. Avoiding float operation is also a good practice and should be done when possible.

Streamflow Finance comment

Streamflow Finance decided not address this point as they do not see any impact of excessive compute budget usage due to recent increases in limits

Missing returned state

Severity	Remediation effort	Status
Informative	Low	Fixed

Description

A vesting can be in 4 different states, namely *Scheduled*, *Streaming*, *Paused* and *Closed*.

However, the state function only has code paths for 3 different states:

```
pub fn state(&self) -> Result<ContractState, ProgramError> {
    if self.closed {
        return Ok(ContractState::Closed);
    }
    if self.current_pause_start > 0 {
        return Ok(ContractState::Paused);
    }
    Ok(ContractState::Scheduled)
}
```

Scope

Streamflow Finance program

Risk

This is an informational issue; At the time of writing this report, it does not represent any risk, but may lead to a vulnerability in the future.

Remediation

OPCODES engineers recommend having an additional code path to return a *Streaming* state.

```
    if now > self.ix.unlock_start() {
        return Ok(ContractState::Streaming)
    }
```

Fix

Streamflow Finance removed the *`Streaming`* state from the enumeration in commit 2bf4677.

Field *canceled_at* not set

Severity	Remediation effort	Fixed
Informative	Medium	Fixed

Description

Vesting structure contains a field *canceled_at* that should be set when the stream is canceled.

programs/protocol/src/state.rs (L262)

```
#[derive(BorshSerialize, BorshDeserialize, Clone, Debug)]
#[repr(C)]
pub struct Contract {
    /// Magic bytes
    pub magic: u64,
    /// Version of the program
    pub version: u8,
    /// Timestamp when stream was created
    pub created_at: u64,
    /// Amount of funds withdrawn
    pub amount_withdrawn: u64,
    /// Timestamp when stream was canceled (if canceled)
    pub canceled_at: u64,
    […]
}
```

Scope

Streamflow Finance program

Risk

This is an informational issue; At the time of writing this report, it does not represent any risk, but may lead to a vulnerability in the future.

Remediation

OPCODES engineers recommend setting the field *canceled_at* inside the cancel instruction in order to know when a vesting has been canceled. It might be interesting to also know who called the cancel instruction.

Fix

Streamflow Finance followed OPCODES recommendation and patched this issue in commit 047bf7f.

Missing owner check on Mint account

Severity	Remediation effort	Fixed
Informative	Low	Fixed

Description

Streamflow Finance uses the Mint account in multiple instructions but does not ensure that the account is owned by the Token Program.

programs/protocol/src/utils.rs (L52)

```
/// Unpack mint account from `account_info`
pub fn unpack_mint_account(
    account_info: &AccountInfo,
) -> Result<spl_token::state::Mint, ProgramError> {
    spl_token::state::Mint::unpack(&account_info.data.borrow())
}
```

Scope

Streamflow Finance program

Risk

Upon vesting creation, the Mint account is used in a CPI to the Token Program which implements correct validation. In other instructions the Mint account is compared to one stored inside the vesting structure. Therefore, this issue does not represent any risk, but may lead to a vulnerability in the future.

Remediation

OPCODES would recommend validating the owner of the Mint account to prevent any possible mistake in the future.

Fix

Streamflow followed OPCODES recommendation and patched this issue in commit 047bf7f.

Missing ownership check on Metadata account

Severity	Remediation effort
Informative	Low

Description

Streamflow Finance uses Metadata accounts to store vesting state. The *create*, *topup* and *update* instructions do not ensure that the Metadata account passed in argument is, owned by Streamflow program and not empty.

Note: The update instruction is only missing the data_is_empty() check.

Scope

Streamflow Finance program

Risk

All of Streamflow instructions write to the metadata account. If anyone try to use an illegitimate metadata account, the program will fail. Therefore, this issue does not represent any risk but may lead to a vulnerability in the future.

Remediation

OPCODES would recommend validating the owner of the Metadata account to prevent any possible mistake in the future.

```
if a.metadata.data_is_empty() || a.metadata.owner != pid {
    return Err(SfError::InvalidMetadataAccount.into())
}
```

Fix

Streamflow Finance followed OPCODES recommendation and patched this issue in commit 2290531.

Outdated dependencies

Severity	Remediation effort	Status
Informative	Low	Accepted risk

Description

The following crates could be updated:

Crate	Current version	Latest version
anyhow	1.0.57	1.0.58
spl-associated-token-account	1.0.3	1.0.5

Scope

Streamflow Finance program

Risk

This is an informational issue; At the time of writing this report, it does not represent any risk, but may lead to a vulnerability in the future.

Remediation

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

Conclusion

Streamflow Finance program has improved the previous version with brand new features. Vesting can now be paused and resumed. An automatic withdraw functionality will also be available. Thanks to an update instruction, owners and recipients of existing vesting will be able to enable automatic withdrawals.

Because of all these new features, Streamflow added a lot of logic in the program making its code complexity higher. OPCODES engineers would recommend being careful when developing new features. Making an access control matrix and maintaining a list of needed check for each instruction would at some point be needed to ensure that no vulnerabilities are introduced along with new features.

As always, Streamflow team was proactive in answering questions about the audit and the code logic. They demonstrated a good understanding of the Solana ecosystem and security hygiene. OPCODES engineers noticed an improvement regarding testing, with the addition of integration test. Each instruction is now tested against successful and unsuccessful scenarios.

Nonetheless, the assessment demonstrated the presence of one medium vulnerability. The vulnerability would have allowed anyone to override closed and canceled streams and erase the history of every vesting.

Annex

Proof of Concept – Overriding closed stream

```
#[tokio::test]
async fn test_opcodes_reinit_attack() -> Result<()> {
    // Copied from the close stream test

    let strm_key = Pubkey::from_str(STRM_TREASURY).unwrap();
    let wdrw_key = Pubkey::from_str(WITHDRAWOR_ADDRESS).unwrap();
    let metadata_kp = Keypair::new();
    let alice = Account { lamports: sol_to_lamports(10.0), ..Account::default() };
    let bob = Account { lamports: sol_to_lamports(10.0), ..Account::default() };

    let mut tt = TimelockProgramTest::start_new(&[alice, bob], &strm_key,
&wdrw_key).await;

    let alice = clone_keypair(&tt.accounts[0]);
    let bob = clone_keypair(&tt.accounts[1]);
    let partner = clone_keypair(&tt.accounts[2]);
    let payer = clone_keypair(&tt.bench.payer);

    let strm_token_mint = Keypair::new();
    let alice_ass_token = get_associated_token_address(&alice.pubkey(),
&strm_token_mint.pubkey());
    let bob_ass_token = get_associated_token_address(&bob.pubkey(),
&strm_token_mint.pubkey());
    let strm_ass_token = get_associated_token_address(&strm_key,
&strm_token_mint.pubkey());
    let partner_ass_token =
        get_associated_token_address(&partner.pubkey(), &strm_token_mint.pubkey());

    tt.bench.create_mint(&strm_token_mint, &tt.bench.payer.pubkey()).await;

    tt.bench.create_associated_token_account(&strm_token_mint.pubkey(),
&alice.pubkey()).await;

    tt.bench
        .mint_tokens(
            &strm_token_mint.pubkey(),
            &payer,
            &alice_ass_token,
            spl_token::ui_amount_to_amount(1000000.0, 8),
        )
        .await;

    let alice_ass_account = tt.bench.get_account(&alice_ass_token).await.unwrap();
```

```
    let alice_token_data =
spl_token::state::Account::unpack_from_slice(&alice_ass_account.data)?;
    assert_eq!(alice_token_data.amount, spl_token::ui_amount_to_amount(1000000.0, 8));
    assert_eq!(alice_token_data.mint, strm_token_mint.pubkey());
    assert_eq!(alice_token_data.owner, alice.pubkey());

    let escrow_tokens_pubkey = find_escrow_account(metadata_kp.pubkey().as_ref(),
&tt.program_id).0;

    let clock = tt.bench.get_clock().await;
    let now = clock.unix_timestamp as u64;
    let transfer_amount = 20;
    let amount_per_period = spl_token::ui_amount_to_amount(0.01, 8);
    let period = 1;

    let cancelable_by_sender = true;
    let cliff = now + 40;
    let create_stream_ix = CreateStreamIx {
        ix: 0,
        metadata: CreateParams {
            start_time: now + 5,
            net_amount_deposited: spl_token::ui_amount_to_amount(transfer_amount as
f64, 8),
            period,
            amount_per_period,
            cliff,
            cliff_amount: spl_token::ui_amount_to_amount(transfer_amount as f64 / 2.0,
8),
            cancelable_by_sender,
            cancelable_by_recipient: false,
            automatic_withdrawal: false,
            transferable_by_sender: false,
            transferable_by_recipient: false,
            can_topup: false,
            stream_name: TEST_STREAM_NAME,
            ..Default::default()
        },
    };

    let create_stream_ix_bytes = Instruction::new_with_bytes(
        tt.program_id,
        &create_stream_ix.try_to_vec()?,
        vec![
            AccountMeta::new(alice.pubkey(), true),
            AccountMeta::new(alice_ass_token, false),
            AccountMeta::new(bob.pubkey(), false),
            AccountMeta::new(bob_ass_token, false),
            AccountMeta::new(metadata_kp.pubkey(), true),
            AccountMeta::new(escrow_tokens_pubkey, false),
            AccountMeta::new(strm_key, false),
            AccountMeta::new(strm_ass_token, false),
            AccountMeta::new(wdrw_key, false),
```

```
            AccountMeta::new(partner.pubkey(), false),
            AccountMeta::new(partner_ass_token, false),
            AccountMeta::new_readonly(strm_token_mint.pubkey(), false),
            AccountMeta::new_readonly(tt.fees_acc, false),
            AccountMeta::new_readonly(rent::id(), false),
            AccountMeta::new_readonly(spl_token::id(), false),
            AccountMeta::new_readonly(spl_associated_token_account::id(), false),
            AccountMeta::new_readonly(system_program::id(), false),
        ],
    );
    let transaction = tt
        .bench
        .process_transaction(&[create_stream_ix_bytes], Some(&[&alice, &metadata_kp]))
        .await;
    assert_eq!(transaction, Ok(()));

    let periods_passed = 200;
    let _periods_after_cliff = now + periods_passed - cliff;
    tt.advance_clock_past_timestamp((now + periods_passed) as i64).await;

    let cancel_ix = CancelIx { ix: 2 };

    let cancel_ix_bytes = Instruction::new_with_bytes(
        tt.program_id,
        &cancel_ix.try_to_vec()?,
        vec![
            AccountMeta::new(alice.pubkey(), true), // sender
            AccountMeta::new(alice.pubkey(), false),
            AccountMeta::new(alice_ass_token, false),
            AccountMeta::new(bob.pubkey(), false),
            AccountMeta::new(bob_ass_token, false),
            AccountMeta::new(metadata_kp.pubkey(), false),
            AccountMeta::new(escrow_tokens_pubkey, false),
            AccountMeta::new(strm_key, false),
            AccountMeta::new(strm_ass_token, false),
            AccountMeta::new(partner.pubkey(), false),
            AccountMeta::new(partner_ass_token, false),
            AccountMeta::new_readonly(strm_token_mint.pubkey(), false),
            AccountMeta::new_readonly(spl_token::id(), false),
        ],
    );

    let transaction = tt.bench.process_transaction(&[cancel_ix_bytes],
Some(&[&alice])).await;

    assert_eq!(transaction.is_err(), !cancelable_by_sender);

    let strm_expected_fee_total =
        (0.0025 * spl_token::ui_amount_to_amount(transfer_amount as f64, 8) as f64) as
u64;
    let strm_expected_fee_withdrawn = 2957500;
    let recipient_expected_withdrawn = 1183000000;
```

```
    let metadata_data: Contract =
tt.bench.get_borsh_account(&metadata_kp.pubkey()).await;
    assert_eq!(metadata_data.ix.cancelable_by_sender, cancelable_by_sender);
    assert_eq!(metadata_data.streamflow_fee_percent, 0.25);
    assert_eq!(metadata_data.streamflow_fee_total, strm_expected_fee_total);
    assert_eq!(metadata_data.partner_fee_total, strm_expected_fee_total);
    assert_eq!(metadata_data.streamflow_fee_withdrawn, strm_expected_fee_withdrawn);
    assert_eq!(metadata_data.partner_fee_withdrawn, strm_expected_fee_withdrawn);
    assert_eq!(metadata_data.amount_withdrawn, recipient_expected_withdrawn);
    let bob_tokens = get_token_balance(&mut tt.bench.context.banks_client,
bob_ass_token).await;
    assert_eq!(bob_tokens, recipient_expected_withdrawn);

    // OPCODES
    // Calling create_unchecked instruction

    let clock = tt.bench.get_clock().await;
    let now = clock.unix_timestamp as u64;
    let transfer_amount = 20;
    let amount_per_period = 100000;
    let period = 1;
    let cancelable_by_sender = false;
    let transferable_by_sender = true;
    let cancelable_by_recipient = true;
    let transferable_by_recipient = true;
    let automatic_withdrawal = false;

    let create_unchecked_stream_ix = CreateStreamUncheckedIx {
        ix: 5,
        metadata: CreateParamsUnchecked {
            start_time: now + 5,
            net_amount_deposited: spl_token::ui_amount_to_amount(transfer_amount as
f64, 8),
            period,
            amount_per_period,
            cliff: 0,
            cliff_amount: 0,
            cancelable_by_sender,
            cancelable_by_recipient,
            automatic_withdrawal,
            transferable_by_sender,
            transferable_by_recipient,
            can_topup: false,
            stream_name: TEST_STREAM_NAME,
            recipient: bob.pubkey(),
            partner: strm_key,
            ..Default::default()
        },
    };

    println!("OPCODES BEFORE TX");
```

```
    let create_unchecked_stream_ix_bytes = Instruction::new_with_bytes(
        tt.program_id,
        &create_unchecked_stream_ix.try_to_vec()?,
        vec![
            AccountMeta::new(alice.pubkey(), true),
            AccountMeta::new(alice_ass_token, false),
            AccountMeta::new(metadata_kp.pubkey(), false),
            AccountMeta::new(escrow_tokens_pubkey, false),
            AccountMeta::new(wdrw_key, false),
            AccountMeta::new_readonly(strm_token_mint.pubkey(), false),
            AccountMeta::new_readonly(tt.fees_acc, false),
            AccountMeta::new_readonly(rent::id(), false),
            AccountMeta::new_readonly(spl_token::id(), false),
            AccountMeta::new_readonly(system_program::id(), false),
        ],
    );
    let transaction = tt
        .bench
        .process_transaction(&[create_unchecked_stream_ix_bytes], Some(&[&alice]))
        .await;
    let is_err = transaction.is_err();
    println!("{:?}", transaction);
    assert!(!is_err);
    let metadata_data: Contract =
tt.bench.get_borsh_account(&metadata_kp.pubkey()).await;
    println!("OPCODES:  stream: {:?}", metadata_data);

    Ok(())
}
```