Security Audit Report

CoinDrip streaming
smart contract

by ARDA

on February 10, 2023

Table of Content

Audit Summary

Scope

- **Repository:** https://github.com/CoinDrip-finance/coindrip-protocol-sc
- **Commit:** `97f8cb7be15da79f157a79e5daa97e64a81d7273`
- **Path to Smart contract:** `./`

Report objectives

1. Reporting all issues in smart contract **code** alongside recommendations
2. Reporting all issues in smart contract **test** alongside recommendations
3. Reporting all **other** issues alongside recommendations

Issues

Number of issues reported and issues remaining at last reviewed commit `3934e6e837e94cfa143c18fd6292fbe982f976cf`:

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	0	0	0	0	0	0
Major	0	0	0	0	0	0
Medium	3	1	0	0	0	0
Minor	4	0	0	0	0	0

Code issues & Recommendations

C1: cancel_stream fails if recipient is a non-payable smart contract

Severity: Medium **Status:** Fixed

Location

```
src/coindrip-protocol.rs
    cancel_stream
```

Description

If a sender wants to cancel a stream, but the recipient is a non-payable smart contract, the `cancel_stream` method will fail because the token transfer will fail (as detailed in MultiversX documentation). Therefore the sender's funds will be stuck in CoinDrip's smart contract.

Recommendation

We suggest a modified procedure for cancelling streams, which consists in the following:

- In the struct `Stream`, a new field `balances_after_cancel: Option<BalancesAfterCancel>`, where the struct `BalancesAfterCancel` has 2 `BigUint` fields: `sender_balance` and `recipient_balance`.
- An endpoint `cancel_stream` , which takes a `stream_id` as argument.
- An endpoint `claim_from_stream_after_cancel` , which takes a `stream_id` as argument.

The `cancel_stream` method is the same as the current one, except that it does not send `sender_balance` and `recipient_balance` to the sender and the recipient. Instead, it stores these 2 quantities in the stream's field `balances_after_cancel`.

Once a stream is cancelled, its field `balances_after_cancel` is not `None` anymore. In this case, the methods `claim_from_stream` and `cancel_stream` should fail.

Then, when the sender calls `claim_cancelled_stream`, he receives the amount given in the field `sender_balance` of `balances_after_cancel`, and this field is then set to 0. Similarly when the recipient calls `claim_cancelled_stream`.

At the end of `claim_cancelled_stream`, if both `sender_balance` and `recipient_balance` equal 0, then the stream is removed from storage.

Note: A possible optimisation is to execute `claim_cancelled_stream` at the end of `cancel_stream` to allow for the caller to receive his funds immediately.

C2: Recipient can receive less than expected

Severity: Medium **Status:** Fixed

Location

```
src/coindrip-protocol.rs
    recipient_balance
```

Description

When the recipient claims, the amount of tokens he receives depends on the times at which he previously claimed. This can lead to an unexpected and underestimated amount for the recipient.

Example: Consider a token with 0 decimals. The sender Alice creates a stream of 3 tokens for the recipient Bob. The stream starts at the beginning of week 1 and should be claimed over 3 weeks. If Bob claims at the beginning of every week, he will receive 1 token each time as expected. For instance:

- When Bob claims at the beginning of week 3, he will have received a total of 2 tokens.

However, if Bob first claims at the end of week 2, he receives 1 token, and the stream is then reset: the 2 remaining tokens will be distributed over week 3, and the next time Bob can receive 1 extra token will be at the middle of week 3. In particular:

- When Bob claims at the beginning of week 3, he receives no extra token. Thus, he will have received a total of 1 token while he would have expected to have received a total of 2 tokens.

No matter the times at which Bob previously claimed, he should expect to have received the same total amount of tokens.

Recommendation

We suggest to remove the field `remaining_amount` of the struct `Stream`, and instead to record the amount already claimed by the recipient `total_claimed_amount`. Then, we can modify the way `recipient_balance` computes the amount `amount_to_claim` that can be claimed by the recipient as follows:

1. Compute the total amount the recipient should have claimed at the current time: `total_received_after_claim = min(deposit * (current_time - start_time) / (end_time - start_time), deposit)`.
2. Let `amount_to_claim = total_received_after_claim - total_claimed_amount`.
3. Increase `total_claimed_amount += amount_to_claim`.
4. Return `amount_to_claim`.

C3: recipient_balance returns wrong amount

Severity: Medium **Status:** Fixed

Location

```
src/coindrip-protocol.rs
    balance_of
```

Description

Due to rounding imprecisions, the method `recipient_balance` underestimates the amount of tokens that the recipient can claim. Therefore, `recipient_balance` can't be completely trusted by other endpoints like `sender_balance` and `balance_of`. This, in turn, forces `balance_of` to implement additional logic to make sure that the recipient will receive the entirety of the sender's deposit at the end of the stream.

If the recommendation suggested in Recipient can receive less than expected is implemented, `recipient_balance` would return the accurate amount of tokens that the recipient can claim. Then, `sender_balance` and `balance_of` could safely rely on `recipient_balance` and do not need to introduce extra logic.

Recommendation

If the recommendation suggested in Recipient can receive less than expected is implemented, the methods `sender_balance` and `balance_of` could then safely rely on `recipient_balance` and we can remove the extra logic in `balance_of`.

Otherwise, we suggest computing the accurate amount of tokens that the recipient can claim directly in `recipient_balance`. Then, `sender_balance` and `balance_of` could safely rely on `recipient_balance` and we can remove the extra logic in `balance_of`.

C4: Useless method balance_of

Severity: Minor **Status:** Fixed

Location

```
src/coindrip-protocol.rs
    balance_of
```

Description

The method `balance_of` to compute the balances of the recipient or the sender is not needed, since the methods `recipient_balance` and `sender_balance` are capable of returning these amounts already.

Recommendation

Following the recommendation of recipient_balance returns wrong amount, we can ensure that the methods `recipient_balance` and `sender_balance` return the correct balances for the recipient and sender. Therefore, we can remove the method `balance_of` (and its occurrences), and use these two methods directly.

C5: Unnecessary restriction to fungible tokens

Severity: Minor **Status:** Fixed

Location

```
src/coindrip-protocol.rs
    create_stream
```

Description

In `create_stream`, the requirement that the token nonce is 0 is not necessary and prevents streams of meta-ESDTs like xExchange farm tokens and metastaking tokens.

Recommendation

We suggest to remove the requirement `require!(token_nonce == 0, ERR_STREAM_ONLY_FUNGIBLE)` from the `create_stream` method, and to allow any type of token.

C6: Unused global constants

Severity: Minor **Status:** Fixed

Location

`src/errors.rs`

Description

The constants `ERR_CLAIM_TOO_BIG` and `ERR_NO_STREAM` are never used.

Recommendation

We suggest to remove them.

C7: No sanity check in recipient_balance

Severity: Minor **Status:** Fixed

Location

```
src/coindrip-protocol.rs
    recipient_balance
```

Description

The amount of tokens `amount` that the recipient can claim should never exceed the remaining balance `remaining_balance` in the stream, otherwise the recipient could receive tokens which belong to other users.

Although the overall logic of the smart contract does not currently allow for situations in which `amount > remaining_balance`, there is no explicit requirement that `amount <= remaining_balance`, which would provide a reliable extra protection in case of unanticipated scenarios, for example if future modifications of the smart contract introduce calculation errors and situations in which `amount > remaining_balance`.

This protection would already be implemented if the recommendation suggested in Recipient can receive less than expected is followed, as then the `min` function is used to ensure that the total amount claimed by the recipient never exceeds the sender's deposit.

Recommendation

If the recommendation suggested in Recipient can receive less than expected is implemented, nothing should be done.

Otherwise, in `recipient_balance`, we suggest adding an explicit requirement that the recipient balance is less or equal than `remaining_balance`.

Test issues & Recommendations

T1: No test for verifying recipient receives the right amount at any time

Severity: Medium **Status:** Fixed

Location

`tests/coindrip_protocol_test.rs`

Description

Depending on the precise timestamp when the recipient claims tokens, some rounding imprecisions may occur in the amount he receives.

The existing tests already cover scenarios where the recipient claims tokens, but at particularly convenient timestamps where the rounding imprecisions do not occur. There is no test making sure that the recipient receives the right amount at arbitrary timestamps, no matter the rounding imprecisions.

Recommendation

Add a test where rounding imprecisions occur:

1. The sender creates a stream with an amount of 2 *atoms* (for instance, if the token has 3 decimals, an amount of 2 atoms equals 0.002 tokens).
2. The recipient claims before half the stream period, and should receive nothing.
3. The recipient claims after half the stream period, and should receive 1 atom.
4. The recipient claims at the end of the streaming period, and should receive 1 atom.

Disclaimer

The security audit report makes no statements or warranties, either expressed or implied, regarding the security of the code, the information herein or its usage. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract.

This report does not constitute legal or investment advice. It is for informational purposes only and is provided on an "as-is" basis. You acknowledge that any use of this report and the information contained herein is at your own risk. The authors of this report shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.