Security Audit Report

PulsarMoney smart-payments
smart contract

by Arda

on October 4, 2023

Table of Content

Audit Summary

Scope

- **Repository:** https://github.com/astrarizon/pulsar-contract
- **Commit:** `a69e7234c717e2392f33659c53cbd51ffb9416db`
- **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 `9ecc80bbb7c2db10eed321d12f262f755049dc66`:

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

Code issues & Recommendations

C1: No user control over max total tax paid

Severity: Critical **Status:** Fixed

Location

`src/pulsar_payment.rs`
`create`

Description

A user cannot currently revert his `create` transaction if the total tax effectively paid is higher than a maximum total tax the user is willing to pay.

This is particularly important because:

1. The owner of the contract can change the fee percentage to any value between the moment a user signs a `create` transaction and the moment this transaction is executed by the blockchain.

2. The rounding errors that might occur at payment creation are taken by the protocol as part of the fee, and in some cases represent more than 50% of the payment amount!

For example, if we assume the protocol fee is 0% and a user creates a payment of 50,000 USDC (6 decimals) to 163 receivers over 5 years splitted monthly, then:

```
amount_per_interval = 50000 * 10^6 / (60*60*24*30*60) / 163 *
60*60*24*30
                    = 2592000
```

And so the 163 receivers will receive a total of:

```
total_received = amount_per_interval * 60 * 163
               = 25349760000
```

which corresponds to 25,349 USDC. So the total fee is 49% of the total payment amount.

Recommendation

Adding a `max_fee: BigUint` parameter to `create` endpoint, that would represent the maximum fee percentage the user would be willing to pay.

Then requiring the total tax paid by the payment creator (including protocol tax and rounding error tax) to be lower than the maximum fee percentage multiplied by the total payment amount.

C2: amount_per_interval formula can introduce a significant total fee

Severity: Major **Status:** Fixed

Location

```
src/pulsar_payment.rs
    create
```

Description

The formula to compute `amount_per_interval` is currently the following:

```
let amount_per_interval = amount_post_tax / interval_seconds.clone()
/ BigUint::from(receivers.len()) * release_request.interval_seconds;
```

In the example given in issue No user control over max total tax paid, the rounding error introduced by the formula would introduce a total fee of 49%.

Because of this and assuming issue No user control over max total tax paid is fixed, many `create` transactions would probably fail because the effective fee would be higher than the max fee allowed by the user.

This would make the creation of payment very difficult.

Recommendation

Rewritting `amount_per_interval` formula to only have one division:

```
let amount_per_interval = release_request.amount.clone() *
(FEE_DENOMINATOR - self.fee().get()) *
release_request.interval_seconds / (BigUint::from(FEE_DENOMINATOR) *
interval_seconds.clone() * BigUint::from(receivers.len()));
```

There will still be a small rounding error, but it will be limited to 0.001% of the release request amount.

C3: Receiver might lose up to 1% depending on how he claims

Severity: Medium **Status:** Fixed

Location

`src/pulsar_payment.rs`
`claim_release`

Description

If a sender sends 100,999 tokens to a receiver, the receiver receives 1000 payment tokens.

If he sends the 1000 payment tokens to the `claim_release` method, he would receive all the 100,999 tokens.

If he sends only 1 payment token to the `claim_release` method and repeats this 1000 times, he will receive `1000 x 100,999 / 1000 = 100,000` tokens and will lose 999 tokens, which represent ~1% of the total amount of tokens he should have received.

Recommendation

Requiring **at payment creation** that the `release.amount` of the following formula:

```
let claimable_amount = amount * claimable_intervals *
release.amount.clone() / BigUint::from(ONE_PAYMENT_TOKEN);
```

is divisible by `ONE_PAYMENT_TOKEN` (1000).

C4: Payment creation fails if fee is 0

Severity: Medium **Status:** Fixed

Location

`src/pulsar_payment.rs`
`create`

Description

When creating a payment, if the fee percentage is set to 0, then the total fee (tax) will be 0, and so the tax transfer will fail.

Recommendation

Only transfer the tax if it is greater than 0.

C5: Payment and Cancel tokens identifiers can be changed

Severity: Medium **Status:** Fixed

Location

`src/pulsar_payment.rs`
`init`

Description

Cancel and Payment tokens identifiers are set at contract deploy but can be changed on contract upgrade.

If these identifiers were to change, that would make all previously minted payment or cancellation tokens unusable.

Recommendation

The auditor recommends to use the `set_if_empty` method of `payment_token_id` and `cancel_token_id` storages in `init` method:

```
#[init]
fn init(&self, payment_token_id: TokenIdentifier, cancel_token_id:
TokenIdentifier, fee: u64) {
    self.payment_token_id().set_if_empty(&payment_token_id);
    self.cancel_token_id().set_if_empty(&cancel_token_id);
...
```

C6: Payment and Cancel token identifiers can take invalid values

Severity: Minor **Status:** Fixed

Location

`src/pulsar_payment.rs`
`init`

Description

In `init` method, nothing prevents to pass token identifiers in an incorrect format (e.g. “ABC”) to `payment_token_id` and `cancel_token_id` parameters.

If incorrectly formatted token identifiers were to be passed, users would not be able to create payments.

Recommendation

The auditor recommends to use `is_valid_esdt_identifier` from `TokenIdentifier` to check whether payment and cancel token identifiers are in the correct format.

```
#[init]
fn init(&self, payment_token_id: TokenIdentifier, cancel_token_id:
TokenIdentifier, fee: u64) {
    require!(payment_token_id.is_valid_esdt_identifier(), "Wrong
format for payment token id");
    require!(cancel_token_id.is_valid_esdt_identifier(), "Wrong
format for cancel token id");
...
```

C7: Fee value can exceed the maximum tolerated

Severity: Minor **Status:** Fixed

Location

```
src/pulsar_payment.rs
    set_fee
```

Description

`amount_post_tax` is computed with this formula:

```
release_request.amount.clone() * (BigUint::from(1000u64 -
self.fee().get())) / BigUint::from(1000u64);
```

Currently, there is no constraint on the value stored in `fee` storage. It could be higher than 1000, which would make the creation of payment fail, or higher than 100 (10%), the maximum value tolerated for the fee.

Recommendation

The auditor recommends to introduce a `set_fee` method that would check the correctness of the fee. This method would be used in `init`:

```
#[init]
fn init(&self, payment_token_id: TokenIdentifier, cancel_token_id:
TokenIdentifier, fee: u64) {
self.payment_token_id().set(&payment_token_id);
    self.cancel_token_id().set(&cancel_token_id);
    self.set_fee(fee);
}

fn set_fee(&self, fee: u64) {
    require!(fee <= 100, "Fee out of range. Must be between 0 (no
fee) and 100 (10%)" );
    self.fee().set(fee);
}
```

C8: Unnecessarily complex logic to get token attributes

Severity: Minor **Status:** Fixed

Location

```
src/pulsar_payment.rs
    decode_token_attributes
```

Description

The `decode_token_attributes` function uses an unnecessarily complex logic to get token attributes as there is already a `get_token_attributes` built-in method in the MultiversX Rust smart contract framework.

Recommendation

Using the `get_token_attributes` built-in method:

```
self.blockchain().get_token_attributes::<Release>(token_id,
token_nonce)
```

C9: Duplication of the fee denominator value

Severity: Minor **Status:** Fixed

Location

```
src/pulsar_payment.rs
    create
```

Description

The fee denominator value (`1000`) used to compute tax is duplicated:

```
let amount_post_tax = release_request.amount.clone() *
(BigUint::from(1000u64 - self.fee().get())) /
BigUint::from(1000u64);
```

This could lead to errors if it were to be updated in one part of the formula but not in the other part.

Recommendation

The auditor recommends to first introduce a constant `FEE_DENOMINATOR` and to use it in the formula:

```
const FEE_DENOMINATOR: u64 = 1_000u64;

...

let amount_post_tax = release_request.amount.clone() *
(BigUint::from(FEE_DENOMINATOR - self.fee().get())) /
BigUint::from(FEE_DENOMINATOR);
```

Test issues & Recommendations

T1: Mandos tests fail

Severity: Minor **Status:** Fixed

Location

`src/scenarios/*.json`

Description

The path to the wasm file is wrong in some mandos tests: it is `file:../output/pulsar-payment-contract.wasm` instead of `file:../output/pulsar-contract.wasm`.

Because of this, 59 failed tests are currently failing.

Recommendation

Update `contractCode` attribute from `file:../output/pulsar-payment-contract.wasm` to `file:../output/pulsar-contract.wasm` in each failed test.

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.