SMART CONTRACT AUDIT

Certification

STREAMING ORDERS CONTRACT

SMART CONTRACT AUDITS

Feb 9th, 2024 / v.0.2

Audited source code version:
90792711236392c32674751321a73f1c930ea339

Structure and Organization of the Document

Some sections are more important than others. The most critical areas are at the top, and the less critical sections are at the bottom. The issues in these sections have been fixed or addressed and will show by the "Resolved" or "Unresolved" tags. Each case is written so you can understand how serious it is, with an explanation of whether it is a risk of exploitation or unexpected behavior.

CRITICAL

These issues can have a dangerous effect on the ability of the contract to work correctly.

HIGH

These issues significantly affect the ability of the contract to work correctly.

MEDIUM

These issues affect the ability of the contract to operate correctly but do not hinder its behavior.

LOW

These issues have a minimal impact on the contract's ability to operate.

INFORMATIONAL

These issues do not impact the contract's ability to operate.

Issues

1. Commented code

Fixed / LOW

Description: There are storages that are not set when the init is called (the code is commented) and the contract depends on them.

! Possible fix to research

```
Uncomment the code in the init function.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

2. Sanity check on input

Fixed / LOW

Description: There are no boundaries checks for the 'fee_percentage' when setting the value (at the moment - the set is commented out, but it does not contain the boundaries check)

! Possible fix to research

```
Add boundaries checks. Greater or equal to 0 and lesser or equal to ‘MAX_PERCENT’
should do it.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

3. Sanity check on input

Fixed / LOW

Description: There is no check if **'_token_nonce'** is zero when the payment is received in **'createStreamingSwap'**.

! Possible fix to research

```
Make sure no SFT/NFT touches this contract and exit early in case it does by
adding a 'require(_token_nonce == 0, "payment nonce should be 0")' after the
payment is read.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

4. Sanity check on input

Unresolved / LOW

Description: There is no check if the payment matches the first token in **'path'** in **'createStreamingSwap'** endpoint.

! Possible fix to research

```
Add a check in order to exit early in case the two values differ.
```

! Response

```
Fixed.
```

! Status

```
The introduced function that does the checking, 'check_path', has a bug, in case
the input is 'EGLD' the line '(token_in.clone().unwrap_esdt()' will crash.
```

5. Sanity check on input

Fixed / LOW

Description: There is no check if the first token in path differs from the last (trying to swap one token for the same one - after several hops). This would be allowed as a nice feature but since the contract relies on checking balances while doing external calls, having the same 'token_in' and **'token_out'** may break logic. For example in **'let final_balance = token_balance_out - token_balance_in;'**, where the contract assumes that the balance after the external call has to be greater or equal than the balance before the call - if the **'token_in'** and **'token_out'** are the same, the assumption may not be valid (one can pay 10A and receive back 9A after the external 'swap' call).

! Possible fix to research

```
Add a check to make sure the 'token_in' differs from 'token_out'. However, the flow
is meant to be allowed, other checks must be introduced or enforced.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

6. Sanity check on input

Fixed / LOW

Description: There is no check on the **'token_out'** in the 'manage_swap' function. The contract checks if unwrapping is required, and if so - sets the **'token_out'** value equal to **'EGLD'**.

! Possible fix to research

```
Require 'token_out' in 'path' to be 'WEGLD' if unwrapping is required. Otherwise
(if unwrapping is required but token_out is not 'WEGLD') it wouldn't make sense.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

7. EGLD & wEGLD mangling

Fixed / LOW

Description: In **'streamSwap'** endpoint, there are two places where the **'stream_swap.token_in'** and 'token_in' are compared and used to determine the value of **'swap_token_in'**. Besides ambiguity, the **'swap_token_in'** can, in the end, have a different value from that of the payment, when the creation of the streaming swap was done.

! Possible fix to research

All swap contracts manage ESDT tokens and not EGLD directly. All **'stream_swap.token_in'**, **'token_in'** and **'swap_token_in'** should have the same value, since they represent the same information **'the value of the received payment'**. There should be checks that enforce this constraint and disallow for differences between them. Even if, theoretically EGLD and wEGLD are the same, practically they are not, one is the native token of the blockchain and the other is an ESDT and they cannot be used interchangeably.

! Response

Fixed.

! Status

Accepted & Closed.

8. Lack of tests

Not Addressed / INFORMATIONAL

! Status

Not Addressed.

9. Lack of interaction scripts

Not Addressed / INFORMATIONAL

! Status

Not Addressed.

10. Lack of documentation

Not Addressed / INFORMATIONAL

! Status

Not Addressed.

Verification Conditions

1 **Only the owner of the streaming swap can call 'cancelStreamingSwap'.**

```
require!(my_streaming_swap.user_address == self.blockchain().get_caller(),
“Streaming Swap does not belong to you”);
```

2 **Only the owner can update the contract configuration state (active/inactive, collectors of fees).**

3 **Only the owner can call 'removeStreamingSwap'.**

```
#[only_owner]
#[endpoint(removeStreamingSwap)]
```

4 **Only the pre-configured processor can call 'streamSwap' endpoint.**

```
require!(self.processor().get() == self.blockchain().get_caller(), “You are not
allowed”);
```

Suggestions (Optional)

1. Update to latest framework version (best practice when deploying a new contract).

2. Write tests (Rust Testing Framework is recommended, documentation (at least a readme) and interaction scripts.

3. Remove unnecessary comments (for example there are functions that are commented entirely).

```
Response: Fixed.

Status: Accepted & Closed.
```

4. Format the code using 'cargo fmt' and solve the warnings (check them using 'cargo clippy').

5. Add '#[view]' on top of storages for better transparency / visibility of the configured values.

Test results

There are no tests.

Audited source code version
B909a123817370b4b57c4061157b77b376f6a783

Second audited source code version
907927112363 92c32674751321a73f1c930ea339