SMART CONTRACT AUDIT

Certification

STABLE SWAP CONTRACT

xaudits

SMART CONTRACT AUDITS

Jan 24th, 2024 / v.0.2

Audited source code version:

b419a336cec96c7b638502cb29aa139ede661329

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. Loss of funds

Not applicable / CRITICAL

Description: Let's imagine the scenario where the SC is deployed with only two stable tokens, A and B, and is currently holding 10 tokens A (10A) and 10 tokens B (10B). The user, unaware of the pool reserves, wants to acquire 10B. An error occurs in the frontend (either made by the same team or an external one) calculus and the user is asked to pay 20A initially. The user seems confused at first, since the tokens are both stablecoins. The MultiversX user, used to the flow of xExchange, accepts and signs the transaction. He does this because he expects the contract to make 'the right thing' and give him his 10B and 'the change' (~10A), as this is what xExchange smart contracts would do (Link). Instead, the smart contract 'eats' all the A tokens given as input and returns the 10B. This is the case even if the user (by habit or by mistake) sends 10 million A initially.

! Possible fix to research

```
Even if the team is designing the frontend, another frontend made by some other
team that uses the same underlying smart contract can make wrong assumptions or
simply a mistake that would end up in lost funds for the user. In conclusion,
in order to protect the user funds, the needs for avoiding this case should be
implemented directly in the smart contract.
```

! Response

```
Not applicable. Swaps are always with fixed input. xExchange example seems to
be for fixed output swaps only, where “the change” is returned (ie. user sent
amount_in + max accepted slippage)
```

! Status

```
Accepted & Closed. We recommend renaming the ‘swap’ endpoint to ‘swapFixedInput’ or
something similar in order to induce the idea that no ‘change’ will be given back.
```

2. Loss of funds

Not applicable / HIGH

Description: It is not clear if the contract may be upgraded in the future. In case it is, let's imagine the scenario where the contract upgrade goes wrong. Users that want to add liquidity or swap during this time will be protected, while the user that panics and tries to remove liquidity will not. The check for whether the contract is paused is not there for **'removeLiquidity'** endpoints and views.

! Possible fix to research

```
Add a check for the ‘paused’ state of the contract in the ‘removeLiqudity’
endpoints and views.
```

! Response

```
Not applicable. Users must always be able to remove their liquidity. That’s why
pausing the smart contract does not prevent liquidity removal. The SC is not
supposed to be upgraded.
```

! Status

```
Accepted & Closed
```

3. Static Amplification Factor

Not applicable / MEDIUM

Description: It is not clear whether the number of tokens in a deployed contract will change (the owner would add or remove tokens). However, if this happens, the **'amp_factor'** might be left unchanged, since it is independent of the number of tokens, and this might lead to different (slightly, but still different) swap rates.

! Possible fix to research

```
Make it dynamic, depending on the number of stablecoins. This way, the rates
will be left unchanged at any time, even if more instances of stablecoins are
added or removed from the contract.
```

! Response

```
Not applicable. It will no longer be possible to change de list of tokens ->
amplification factor can remain static. The SC is not supposed to be upgraded.
The new upgrade function will be implemented, preventing SC owner to change the
amplification factor and list of tokens, as it would be a critical change in the
purpose of the contract. We would advise the pool owner to create a new pool.
```

! Status

```
Accepted & Closed
```

4. Division by zero

Fixed / LOW

Description: The contract assumes in multiple occurrences that the number of tokens is strictly greater than 1, while there's no check for it.

! Possible fix to research

```
When setting the number of tokens, add a check for its value (must be 2 or
higher).
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed
```

5. Unchecked bounds for amplification factor

Fixed / LOW

Description: The contract does not check the value given by the owner when configuring the amplification factor.

! Possible fix to research

```
Add checks with min / max values allowed or at least the sanity zero check.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed
```

6. Identical token identifiers when setting stablecoins

Fixed / LOW

Description: The contract does not check for duplicates in the vector containing **'tokenIdentifiers'** given by the owner. Configuring the same token twice by mistake will break the contract logic.

! Possible fix to research

```
Add checks in order to make sure that the given token id vector has only unique
values.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed
```

7. Redundant calculus

Fixed / LOW

Description: Calculating **'BigUint::from(10u32).pow(LP_TOKEN_DECIMALS))'** each time (slightly) increases the cost unnecessary.

! Possible fix to research

```
Make a constant for the number (since it can be represented on 64 bits, for
example: ‘const ONE_LP_TOKEN: u64 = 1_000_000_000_000_000_000u64’ and use
directly ‘BigUint::from(ONE_LP_TOKEN)’ in order to lower gas consumption.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed
```

8. Unsolved TODO in the code

Fixed / LOW

Description: The code contains a TODO. Checking the Curve code (Link), the current version of the code seems to be correct but it is the developer's decision.

! Possible fix to research

```
Solve the TODO, maybe taking the Curve's code as reference / starting point.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed
```

9. Missing zero checks on user input

Fixed / INFORMATIONAL

Description: The contract does not check the 'min' values received on the user endpoints. In case the user sends 'zero' as 'min' and the calculated 'out' amount is min, all the checks pass and the send function will fail.

! Possible fix to research

```
Add checks for zero values in order to exit early if a user expects 'zero'
amount in return for his actions in all user endpoints and views (add liqudity,
swap, remove liqudity).
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed
```

10. Missing basic checks on user endpoint

Fixed / INFORMATIONAL

Description: The contract does not check if the LP token is issued and roles were set in all user endpoints and views.

! Possible fix to research

```
Add checks in order to exit early if a user wants to make actions when the
contract is not fully configured.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed
```

11. The smart contract has an admin

Not applicable / INFORMATIONAL

Description: The smart contract has an admin, the owner, who can, at any time, pause the contract (which disallows users to add liquidity and swap). In most cases, it is not a problem, since the users in this ecosystem tend to approve of this technique trusting the owner of the smart contract. However, this might be a problem in case, for example, the owner's private key gets compromised).

! Possible fix to research

```
There’s no fix, it’s just the way the SC is designed and the users must be aware
of it.
```

! Response

```
Not applicable. Won’t do. Contract owner can pause the contract for various
reasons (end-of-life, migration, ...). Users are always able to withdraw their
```

! Status

```
Accepted & Closed
```

Verification Conditions

1 **Owner functions are marked using either 'only_owner' macro attribute.**

```
#[only_owner]
#[endpoint]
fn pause(&self) {
    self.do_pause();
}
```

2 **The minimum 'output' value is respected.**

```
require!(amount_out >= min_amount, “Max slippage exceeded”);
```

3 **Valid payments are checked on input.**

```
require!(nb_valid_payments == payments.len(), “Invalid payment token”);
```

4 **Actions are taken when the contract is not paused (except for the ones indicated in the Issues section).**

```
self.require_not_paused();
```

Suggestions (Optional)

1. Tests written in rust are easier to extend and more comprehensible. Might be a good idea to port the mandos tests now because they might grow in the future and more 'depth' will be accumulated around them (harder to follow, harder to comprehend, harder to write new tests, etc).

2. Annotations. There's an unwritten rule that annotations order is: access annotations, payable annotations, and endpoint annotations. Also, if the name of the function name is identical with the name given by the endpoint annotation, the last should be left unspecified.

3. Remove the **'An empty contract. [...]'** comment at the beginning of the contract.

```
Response: Fixed.

Status: Accepted & Closed.
```

4. Remove the **'issueLpToken'** and **'enableMintBurn'** function and do those actions outside of the contract. Since they are done only once, they grow the contract in size and hence any interaction with the contract will be a little more expensive in gas.

5. Edit the error message **'i = j'** and make it more comprehensible. Might not be intuitive what the error means at first.

```
Response: Fixed.

Status: Accepted & Closed.
```

6. When finding a token in **'get_token_index'**, add a **'break'** in order to lower gas consumption.

```
Response: Fixed.

Status: Accepted & Closed.
```

7. Add **'#[view([...])]'** on each storage. Might be useful later to be able to read the values.

```
Response: Fixed.

Status: Accepted & Closed.
```

8. Avoid **'double iterating'** by hand, using an iterator and an index **'i'** incremented manually and use **'enumerate()'** instead.

```
Response: Fixed.

Status: Accepted & Closed.
```

9. Avoid comparing references to references when it is not needed. For example this: **'require!(&amount_out >= &min_amount, "Max slippage exceeded");'** can be written more simply: **'require!(amount_out >= min_amount, "Max slippage exceeded");'**.

```
Response: Fixed.

Status: Accepted & Closed.
```

10. Avoid initialising a new variable with a value and returning it immediately. Simply just return the value directly.

```
Response: Fixed.

Status: Accepted & Closed.
```

11. Avoid using **'return'** keyword where it is redundant (check **'clippy'** output).

```
Response: Fixed.

Status: Accepted & Closed.
```

12. Rewrite **'const FEE_DENOMINATOR: u64 = 1_000000u64;'** to **'const FEE_DENOMINATOR: u64 = 1_000_000u64';**

```
Response: Fixed.

Status: Accepted & Closed.
```

13. In order to make **'PairStatus'** deserialization more readable, you might want to make a custom method **'to_managed_buffer_array'** that converts the object into an array of managed buffers.

14. Try having few to none hardcoded values, for example in **'for i in 0u64..=6u64'**.

15. Not long ago, MultiversX team has added an update in which init is called only at smart contract deploy, so there's no need for all the **'if_empty'** check. You can safely assume that the storage is all empty. In order to upgrade, you will need an **'upgrade'** function. Is an unwritten rule that every upgradable contract has an **'upgrade** function even if the content is empty, in order to remember that the code executed when upgrading the SC must be placed inside **'upgrade** and not inside **'init'** as it was until one of the latest network upgrades. Check the docs for more info about the **'upgrade** function.

```
Response: Fixed.

Status: Accepted & Closed.
```

Test results

```
Scenario: add_liquidity.scen.json ...   ok
Scenario: add_liquidity_and_remove_full.scen.json ...   ok
Scenario: add_liquidity_and_swap_and_remove_full.scen.json ...   ok
Scenario: add_liquidity_and_swap_and_remove_part.scen.json ...   ok
Scenario: add_liquidity_and_swap_big.scen.json ...   FAIL: Check state: mismatch for account "address:user2":
  for token: BUSD-000000, nonce: 0: Bad balance. Want: "99999999999999999999000000000". Have: "0"
Scenario: add_liquidity_balanced.scen.json ...   ok
Scenario: add_liquidity_balanced_unordered.scen.json ...   ok
Scenario: add_liquidity_invalid_token.scen.json ...   ok
Scenario: add_liquidity_max_slippage_exceeded.scen.json ...   ok
Scenario: add_liquidity_no_payment.scen.json ...   ok
Scenario: add_liquidity_paused.scen.json ...   ok
Scenario: deploy.scen.json ...   ok
Scenario: enable_mint_burn_not_owner.scen.json ...   ok
Scenario: estimate_add_liquidity_256.scen.json ...   ok
Scenario: estimate_add_liquidity_paused.scen.json ...   ok
Scenario: estimate_amount_out_256.scen.json ...   ok
Scenario: estimate_amount_out_paused.scen.json ...   ok
Scenario: estimate_amount_out_unbalanced_pool.scen.json ...   ok
Scenario: estimate_remove_liquidity_256.scen.json ...   ok
Scenario: estimate_remove_liquidity_single_256.scen.json ...   ok
Scenario: init.scen.json ...   ok
Scenario: issue_lp_token_already_issued.scen.json ...   ok
Scenario: issue_lp_token_not_owner.scen.json ...   ok
Scenario: remove_liquidity.scen.json ...   ok
Scenario: remove_liquidity_invalid_token.scen.json ...   ok
Scenario: remove_liquidity_max_slippage_exceeded.scen.json ...   ok
Scenario: remove_liquidity_one_token.scen.json ...   ok
Scenario: remove_liquidity_one_token_invalid_token.scen.json ...   ok
Scenario: remove_liquidity_one_token_slippage.scen.json ...   ok
Scenario: set_swap_fee.scen.json ...   ok
Scenario: set_swap_fee_not_owner.scen.json ...   ok
Scenario: swap_balanced_pool.scen.json ...   ok
Scenario: swap_paused.scen.json ...   ok
Done. Passed: 32. Failed: 1. Skipped: 0.
ERROR: some tests failed
```

Audited source code version
6eb706bb7a1a989f2d6bc791692c6c70cdf89c7f

After the second review:

```
Scenario: scenarios/add_liquidity.scen.json ...   ok
Scenario: scenarios/add_liquidity_and_remove_full.scen.json ...   ok
Scenario: scenarios/add_liquidity_and_swap_and_remove_full.scen.json ...   ok
Scenario: scenarios/add_liquidity_and_swap_and_remove_part.scen.json ...   ok
Scenario: scenarios/add_liquidity_and_swap_big.scen.json ...   ok
Scenario: scenarios/add_liquidity_balanced.scen.json ...   ok
Scenario: scenarios/add_liquidity_balanced_unordered.scen.json ...   ok
Scenario: scenarios/add_liquidity_invalid_token.scen.json ...   ok
Scenario: scenarios/add_liquidity_max_slippage_exceeded.scen.json ...   ok
Scenario: scenarios/add_liquidity_no_payment.scen.json ...   ok
Scenario: scenarios/add_liquidity_paused.scen.json ...   ok
Scenario: scenarios/deploy.scen.json ...   ok
Scenario: scenarios/enable_mint_burn_not_owner.scen.json ...   ok
Scenario: scenarios/estimate_add_liquidity_256.scen.json ...   ok
Scenario: scenarios/estimate_add_liquidity_paused.scen.json ...   ok
Scenario: scenarios/estimate_amount_out_256.scen.json ...   ok
Scenario: scenarios/estimate_amount_out_paused.scen.json ...   ok
Scenario: scenarios/estimate_amount_out_unbalanced_pool.scen.json ...   ok
Scenario: scenarios/estimate_remove_liquidity_256.scen.json ...   ok
Scenario: scenarios/estimate_remove_liquidity_single_256.scen.json ...   ok
Scenario: scenarios/init.scen.json ...   ok
Scenario: scenarios/issue_lp_token_already_issued.scen.json ...   ok
Scenario: scenarios/issue_lp_token_not_owner.scen.json ...   ok
Scenario: scenarios/remove_liquidity.scen.json ...   ok
Scenario: scenarios/remove_liquidity_invalid_token.scen.json ...   ok
Scenario: scenarios/remove_liquidity_max_slippage_exceeded.scen.json ...   ok
Scenario: scenarios/remove_liquidity_one_token.scen.json ...   ok
Scenario: scenarios/remove_liquidity_one_token_invalid_token.scen.json ...   ok
Scenario: scenarios/remove_liquidity_one_token_slippage.scen.json ...   ok
Scenario: scenarios/set_swap_fee.scen.json ...   ok
Scenario: scenarios/set_swap_fee_not_owner.scen.json ...   ok
Scenario: scenarios/swap_balanced_pool.scen.json ...   ok
Scenario: scenarios/swap_paused.scen.json ...   ok
Done. Passed: 33. Failed: 0. Skipped: 0.
SUCCESS
```

Hash: b419a336cec96c7b638502cb29aa139ede661329

* The audited source code version is the hash of the last commit of the received code repo.