Security Audit Report

Pact.fi StableSwap AMM

Delivered: June 6th, 2022

Prepared for Pact.fi by

Table of Contents

runtime verification

Summary

Pact engaged Runtime Verification Inc. to conduct a security audit of their smart contract implementing a StableSwap Automated Market Maker (AMM).

The objective was to review the contract's business logic and implementation in PyTeal and identify any issues that could potentially cause the system to malfunction or be exploited.

Timeline

The audit has been conducted over a period of 4 weeks, from May 9, 2022 to June 3, 2022.

Findings

The audit uncovered two medium severity issues (A1-A2), three low severity issues (B1-B3) and six informative findings (C1-C6). All medium and low severity issues as most of informative findings have been appropriately addressed.

Disclaimer

This report does not constitute legal or investment advice. You understand and agree that this report relates to new and emerging technologies and that there are significant risks inherent in using such technologies that cannot be completely protected against. While this report has been prepared based on data and information that has been provided by you or is otherwise publicly available, there are likely additional unknown risks which otherwise exist. This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system. This report is for informational purposes only and is provided on an "as-is" basis and you acknowledge and agree that you are making use of this report and the information contained herein at your own risk. The preparers of this report make no representations or warranties of any kind, either express or implied, regarding the information in or the use of this report and 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.

Smart contracts are still a nascent software arena, and their deployment and public offering carries substantial risk.

Finally, the possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims which impact a large quantity of funds.

Goals

The goals of the audit were:

- Review the architecture of the StableSwap smart contract based on the provided documentation with a focus on mathematical operations;
- Review the PyTeal implementation of the contract to identify any programming errors;
- Build an executable model of the contract and conduct extensive testing of the contract’s life cycle, leveraging pattern-based fuzzing in a simulated blockchain environment.

The audit focuses on trying to identify issues in the system’s logic and its implementation that could potentially render the system vulnerable to attacks or cause it to malfunction.

Scope

The audit has been conducted on Pact private GitHub Repository of three following versions:

- **V1**. The commit hash is 1155c1e8d993cd06ddb856c77290c2728a638258.
- **V2**. The commit hash is 455caf6b5c54798216e5cb4cf521239822d1ec8e.
- **V3**. The commit hash is 4193a99dd0a9061fe2fcf76b5fc2942b41e136b3.

The audit scope includes the following files:

- The StableSwap smart contract source code in PyTeal: assets/stable_exchange.py.
- Transaction validation code: assets/helpers/validation.py.
- Additionally, we have informally reviewed the pytealext library.

We have not audited any off-chain components of the system and further patches except explicitly mentioned above.

Methodology

Audit process

The audit process has included the following stages:

1. Manual code review of the StableSwap contract, including thorough analysis of math.
2. Integration of the contract’s PyTeal source code into the simulation environment.
3. Testing and pattern-based fuzzing of admin and user functions of the contract.
4. Development of pure pythonic implementations of “get_D” and “get_y” functions based on numpy and Newton’s method. Comparison of these functions with outputs obtained by simulation of “get_D” and “get_y” implementations from the contract’s source code.

Stages 1 and 2 create the foundation for the rest of the audit: the auditors gain in-depth understanding of the contracts’ inner structure and obtain an executable model of the contract. Stage 3 allows testing main functions of the contract and prove or disprove feasibility of attacks. Stage 4 aims at revealing any significant rounding errors that may cause invariant violations.

Findings

A1 Missing an exponentiation while calculating Ann

Severity: Medium

Description

The following StableSwap safety invariant must hold:

$$Ann * \sum x_i + D = Ann * D + \frac{D^{n+1}}{n^n * \Pi\, x_i}$$

Where $Ann = A * n^n$, *n* is the number of assets and *A* is an amplification factor. The version **V1** of the contract misses the exponentiation and uses $A * n$ instead of the forementioned expression. This issue affects all swaps, adding and removing liquidity operations except the first provision of liquidity. Obtained *D* is much less than properly calculated one in a general case. Swaps become more expensive for a user: in most simulation experiments' swaps return 5% less tokens per a token sent by a user. However, the extreme ratio between asset balances may lead to a significant deviation.

Recommendation

We recommend using $Ann = A * n^n$ according to the initial design.

Status

The issue is fixed in **V2**.

A2 Retrieving treasury fees keeps contract’s balances unchanged

Severity: Medium, Difficulty: Low, Probability: High

Description

The contract accumulates fees at user swaps. Accumulated fees are intended to be sent to the treasury account. The contract should not include amounts of these fees in asset balances, but versions **V1** and **V2 of the contract** do not update relevant global variables.

Recommendation

We propose to modify global variables corresponding to asset balances by amounts of fees while processing swaps.

Status

The issue is addressed in version **V3** of the contract.

B1 User can swap indirectly by adding and removing liquidity

Severity: Low, Difficulty: Low, Probability: High

Description

The StableSwap AMM provides a user with add liquidity, remove liquidity and swap operations. The contract enables sending assets in any ratio while depositing liquidity. Removing liquidity is always done in a proportion of contract’s asset balances. These two operations are not charged by the contract. The contract calculates the output amount while swapping according to its balances and other internal parameters. This output value is reduced by an amount of a contract fee.

Swapping should be the exclusive way to exchange assets. However, a user can exchange assets by adding and removing liquidity. Let us consider the following example:

- Contract balances $x_1 = 10^5$, $x_2 = 10^5$
- Amplification factor before scaling: $A = 80$
- Fees: 6%

Sending 10^3 of the primary asset tokens at a swap operation allows a user to get $\Delta x_2 = 939$ tokens of the secondary asset. But, a user can send 2000 primary asset tokens to get 999 liquidity tokens in return. Sending these tokens to the contract doing the remove liquidity operation returns to the user $\Delta x_1 = 1089$ and $\Delta x_2 = 908$ tokens back. The canonical swap in the example provided the user with 0.94 tokens per received token. The indirect swap enabled almost 1:1 exchange.

Amounts of tokens that should be returned at swaps and remove liquidity operations are determined by different formulas. However, the invariant of the contract must hold at any time. Adding liquidity and immediate selling of all earned liquidity tokens result in different asset balances. However, D remains the same because the contract does not imply any fees. The D value after a swap is greater than before, as the calculated output value is decreased by the amount of a fee:

$$\Delta y' = \Delta y \frac{10^5 - fees}{10^5}$$

If the contract has enough liquidity of both kinds and a user possess one of the assets, it is possible to send tokens of this kind and then immediately sell earned liquidity tokens in the same transaction group. It allows getting back assets of both kinds, which we call an indirect swapping. A user can get the following amount of token doing an indirect swap supplying $(\Delta x,\ 0)$ tokens:

$$\Delta y = \frac{(D(x+\Delta x, y) - D(x, y))*D(x, y)}{D(x+\Delta x, y)}$$

Where $D(x + \Delta x,\ y) > D(x,\ y)$ if $\Delta x > 0$

Simulations confirm that a user gains no less than $\frac{fees}{10^2}\%$ of tokens more at indirect swaps than after normal ones. However, normal swaps are more profitable than indirect ones if the pool posses significantly more tokens of the kind which was provided by a user.

Recommendation

We propose introducing fees for sending liquidity operations.

Status

Acknowledged by Pact.

B2 The amplification factor can be decreased more than by ten times

Severity: Low, Difficulty: Low, Probability: Low

Description

The contract allows the admin user changing the amplification factor. However, drastic decrease in the factor can result in liquidity losses for the contract if asset balances are in an imbalanced ratio.

The contract uses the following formula to compare initial and final amplification factor values:

$min(A_1, A_2) * MAXDIFF \geq max(A_1, A_2)$

Where $MAXACHANGE = 10 * A_{precision}$

The initial value of A_1 is expected to be scaled by $A_{precision}$, which is 10^3 . The new amplification factor, which is denoted as A_2, is not required to be scaled, so the difference between them can be more than 10 times but less than $A_{precision}$ to pass the assertion above.

Recommendation

We propose to set $MAXACHANGE = 10$ and scale all values provided by a user by $A_{precision}$ or at least scale constants that restrict the maximum and minimum values of the amplification factor.

Status

Fixed in the **V3** version of the contract.

B3 There is an unnecessary byte division

Severity: Low

Description

The estimation of the new D using the Newton's method is iterative. Each step contains calculation of previous value in power of three. The contract performs this step according to the following formula:

$$Dp_1 = D$$

$$Dp_2 = \frac{Dp_1 * D}{n * x_1}$$

$$D^3 = \frac{Dp_2 * D}{n * x_2}$$

The implementation decreases the precision of the output. For $x_1 = 99$ and $x_2 = 1$ corresponding to primary and secondary balances, the output is 2500. The version with a single division gives 2525. The loss in the example is 25. The error equals to $\sqrt{\frac{D}{2}}$ by the order of magnitude in the general case.

If there is a significant difference between contract balances, the D value can be less than the maximum balance of two assets, which increases the rounding error on each iteration. It might prevent the Newton's method to converge in specific cases. We confirm the latter fact for the following parameters by simulation:

1. $x_1 = 61000, x_2 = 1, A = 80;$
2. $x_1 = 80000, x_2 = 21, A = 5.$

Recommendation

We recommend replacing the expression with calculations of the numerator and denominator and a single division.

Note, that the new formula has a higher risk of an overflow, but it is assumed to be safe considering realistic amounts of tokens that can be stored in the contract.

Status

Acknowledged by Pact.

C1 A few byte operations can be replaced by int64 analogues

Severity: Informative

Description

Byte operations according to the Algorand documentation cost more than int64 ones by an order of magnitude. Despite byte operations are necessary to maintain required level of precision, it is not worth applying them for calculation of constants and values that always fit 64bit integer type.

Recommendation

The following expressions in the code can be replaced with int64 analogues, as results always fit 64 integer value:

- S.store(BytesAdd(Itob(total_primary), Itob(total_secondary)))
- BytesMul(BytesAdd(Itob(number_of_tokens), Itob(Int(1))), D_P.load())
- BytesMul(BytesMinus(Ann.load(), Itob(a_precision)), D.load())
- BytesMinus(D, b)

Status

Acknowledged by Pact.

C2 The first provision of liquidity does not require increasing the opcodes quota

Severity: Informative

Description

The contract uses the following formula to calculate the first amount of liquidity tokens:

$$lt = \sqrt{x_1 * x_2}$$

The rest operations of this kind follow the next formula:

$$lt_2 - lt_1 = \frac{lt_1 * (D_2 - D_1)}{D_1}$$

The implementation of the latter formula requires increasing opcode quota to calculate D. The first formula does not use D, so the increase in opcodes quota is not required.

Recommendation

We recommend moving the relevant code to a specific conditional branch.

Status

Acknowledged by Pact and fixed in **V3**.

C3 The contract does not check senders of non-admin transactions

Severity: Informative

Description

Adding liquidity, swaps and removing liquidity result in sending assets back to the application call transaction sender. However, the contract does not check that the senders of all transactions in a transaction group match.

If the user signs a transaction group with different senders of transactions, he loses his assets. However, we understand that a user can be affected only in case of malformed operation of the off-chain logic.

Recommendation

We recommend adding checks for senders of all transactions in a transaction group.

Status

Acknowledged by Pact.

C4 The number of assets is not a constant

(Severity: Informative)

Description

The implementation of the contract supports swaps between strictly two assets. However, some functions accept the variable number of assets as a parameter. It introduces unnecessary calculations and makes the code more error-prone.

Recommendation

We recommend using a constant for the number of assets and remove corresponding function parameters.

Status

Acknowledged by Pact. Partially addressed. The **V3** version of the contract uses the constant, but few functions still have the number of assets as a parameter.

C5 The contract does not check that swaps have non-zero output

Severity: Informative

Description

The StableSwap contract has a parameter corresponding to the minimum value of tokens received after a swap. This value can be set to null, and the user might receive no assets back as a result if a petite number of tokens is sent in exchange for a single token. Another form of the issue is that a user is allowed to swap nothing for nothing.

Recommendation

We propose to check that both incoming and minimum output asset amounts are positive.

Status

Acknowledged by Pact.

C6 The treasury address cannot be changed

Severity: Informative

Description

The treasury address is used to receive fees accumulated by the contract. However, admin cannot change this address.

Recommendation

We propose to implement an extra application call to allow an admin changing the treasury address.

Status

Acknowledged by Pact.