ClarityAlliance

BITFLOW STABLESWAP STAKING SECURITY REVIEW

Conducted by:
KRISTIAN APOSTOLOV, ABA, MARCHEV

SEPTEMBER 30TH, 2024

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

1. About Clarity Alliance

Clarity Alliance is a team of expert whitehat hackers specialising in securing protocols on Stacks.

They have disclosed vulnerabilities that have saved millions in live TVL and conducted thorough reviews for some of the largest projects across the Stacks ecosystem.

Learn more about Clarity Alliance at clarityalliance.org.

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

2. Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Clarity Alliance to perform a security assessment.

This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Clarity Alliance’s position is that each company and individual are responsible for their own due diligence and continuous security. Clarity Alliance’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree
to analyze.

The assessment services provided by Clarity Alliance are subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third parties. Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Clarity Alliance does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

3. Introduction

A time-boxed security review of the Bitflow Stableswap Staking implementation, where Clarity Alliance reviewed the scope, whilst simultaneously building out a testing suite for the protocol.

4. About Bitflow Stableswap

Bitflow StableSwap is the first protocol designed to enable users to efficiently swap stable assets, including stablecoins, within the Bitcoin ecosystem. It operates on the Stacks layer, a platform specifically designed to facilitate smart contracts and decentralized applications on Bitcoin.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1 Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

CONTENTS

5.2 Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

6. Security Assessment Summary

Review Commit Hash:
e949a74b25bbe280708685cd7b40aa1b756166d1

- `stableswap-emissions-stx-ststx-stx-v-1-1.clar`
- `stableswap-staking-stx-ststx-v-1-1.clar`

CONTENTS

7. Executive Summary

Over the course of the security review, Kristian Apostolov, ABA, Marchev engaged with Bitflow to review Bitflow Stableswap Staking. In this period of time a total of **24** issues were uncovered.

Protocol Summary

Protocol Name	Bitflow Stableswap
Repository	https://github.com/BitflowFinance/bitflow-stableswap
Date	September 30th, 2024
Protocol Type	Stableswap AMM Staking Module

Findings Count

Severity	Amount
High	1
Medium	4
Low	7
QA	12
Total Findings	**24**

ClarityAlliance
Security Review

Bitflow Stableswap Staking

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

Summary of Findings

ID	Title	Severity	Status
[H-01]	Early Unstaking Erases User Unclaimed Rewards	High	Resolved
[M-01]	Early Unstaking Applies Fees to Already Matured Staked LPs	Medium	Resolved
[M-02]	Expired Rewards Can Be Claimed	Medium	Resolved
[M-03]	Incorrect Reward Distribution Due to Possible Cycle Desynchronization	Medium	Acknowledged
[M-04]	Potential Exploitation of Staking Mechanism	Medium	Resolved
[L-01]	Invalid Unstaking Fee Can Be Set	Low	Resolved
[L-02]	Missing Validation for Early Unstake Fee Address	Low	Resolved
[L-03]	Missing Standard Validation for Rewards Withdrawal Receiver	Low	Resolved
[L-04]	Emission Rewards Not Validated Against Available Balance	Low	Resolved
[L-05]	Reward Expiration Delta Should Not Be Allowed to Be 0	Low	Resolved
[L-06]	Inner Contract Calls Must Be Updated for Mainnet	Low	Resolved
[L-07]	Avoid Using tx-sender for Caller Identification	Low	Acknowledged
[QA-01]	Undocumented Staking Particularities	QA	Acknowledged
[QA-02]	Improvement of Staking Contract Filter Functions	QA	Resolved
[QA-03]	Improper Function Naming Reduces Code Readability	QA	Resolved
[QA-04]	Unnecessary Processing of Future Cycles in unstake-lp-tokens	QA	Resolved
[QA-05]	Inconsistent Use of Caller Declaration	QA	Resolved
[QA-06]	The cycles-to-unstake Field in the fold-early-unstake-per-cycle Accumulator is Unnecessary	QA	Resolved
[QA-07]	Simplification of the stake-lp-tokens Function	QA	Resolved
[QA-08]	Improve Error Handling in get-cycle-from-height	QA	Resolved
[QA-09]	Simplification of Emission User-Data-at-Cycle Map	QA	Resolved

CONTENTS

Summary of Findings

ID	Title	Severity	Status
[QA-10]	Misleading Variable Name: current-cycle-data	QA	Resolved
[QA-11]	Use Errors Instead of Panicking	QA	Resolved
[QA-12]	Simplification of fold-cycles-to-unstakeable-cycles	QA	Resolved

ClarityAlliance
Security Review

Bitflow Stableswap Staking

CONTENTS

8.1. High Findings

[H-01] Early Unstaking Erases User Unclaimed Rewards

Description

Stakers have the option to unstake their LP tokens early at any time, albeit with an additional fee applied to the unstaked amount. However, if a user initiates early unstaking, their entire staking history is erased. While this does not affect future cycles, the history of staked LPs that have already matured is also deleted.

By removing this historical data, when a user attempts to claim their staking rewards via the `stableswap-emissions-stx-ststx-stx-v-1-1::claim-rewards` function, they will receive fewer rewards. This is because the reward calculation relies on the same mapped data that is deleted during early withdrawal, which is retrieved by the `get-external-user-data` function from the same contract.

```
(define-public (claim-rewards (cycle uint))
  (let (
    ;; ... code ...
    (user-data-external (try! (get-external-user-data tx-sender cycle)))
    ;; ... code ...
    (user-lp-staked (unwrap!
      (get lp-staked user-data-external) ERR_NO_EXTERNAL_USER_DATA))
    (cycle-lp-staked (unwrap! cycle-data-external ERR_NO_EXTERNAL_CYCLE_DATA))
    (user-rewards (/ (*
      (get total-rewards current-cycle-data) user-lp-staked) cycle-lp-staked))
    ;; ... code ...
)
```

Thus, early unstaking effectively deletes the accounting needed for users to claim rewards, resulting in a loss of funds.

Another important related issue is that, at the end of the current, incorrect deletion of all user cycle data, another user cycle data deletion is seemingly arbitrarily performed:

```
(map-delete user-data-at-cycle {user: caller, cycle: (+
  (len user-cycles-staked) u1)})
```

This deletion appears to be a leftover from development that must be removed, as it deletes the user cycle data that is one unit greater than the length of the initial user cycle stored array.

Once the system is corrected to only delete user data from future cycles, this arbitrary deletion may coincide with an old user staking cycle and, once again, delete reward data.

CONTENTS

Recommendation

In the `fold-early-unstake-per-cycle` function from the `stableswap-staking-stx-ststx-v-1-1` contract, do not delete the `user-data-at-cycle` entry unless it is from a cycle that is strictly greater than the current cycle (`current-cycle-static`).

Another possible solution is to filter the `user-cycles-staked` list before passing it to the `fold-early-unstake-per-cycle` function, excluding any cycles less than or equal to the current cycle.

If this second solution is implemented, then the `current-cycle` does not need to be passed to the `early-unstake-per-cycle` function anymore.

Remove the `map-delete` from the end of the `early-unstake-lp-tokens` function.

ClarityAlliance
Security Review

Bitflow
Stableswap
Staking

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

8.2. Medium Findings

[M-01] Early Unstaking Applies Fees to Already Matured Staked LPs

Description

Stakers have the option to unstake their LP tokens early at any time, but this incurs an additional fee on the unstaked amount.

However, if a user initiates early unstaking without first unstaking their already matured LP stakes, the early unstaking fee is incorrectly applied to all LPs, regardless of whether they have reached maturity.

If a user has staked several LP amounts over time, with some reaching maturity, and they call `early-unstake-lp-tokens`, the fee will be applied to all LPs. This occurs even though the user has the right to unstake some without any fee, resulting in a loss of funds.

Recommendation

Modify the `early-unstake-lp-tokens` function in the `stableswap-staking-stx-ststx-v-1-1` contract to exclude the staking fee on already matured staked LP amounts. Alternatively, directly call the `unstake-lp-tokens` function as the first action within `early-unstake-lp-tokens`.

This approach will ensure that any matured amounts are unstaked without fees before proceeding with the early unstaking logic.

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

[M-02] Expired Rewards Can Be Claimed

Description

Within the `stableswap-emissions-stx-ststx-stx-v-1-1` contract, there is logic implemented to manage a reward expiration block. This logic includes setting the value, retrieving the value, and clearing any rewards that have expired. A reward (per cycle) is considered expired if at least `rewards-expiration (uint)` cycles have passed since then. Admins are required to call the `clear-expired-rewards` function on an expired cycle to account for it internally.

However, there is currently no validation to ensure that the cycle users are claiming rewards for has not already expired.

As a result, if admins, for any reason, fail to call the `clear-expired-rewards` function, users will still be able to withdraw rewards. Additionally, a user could potentially front-run a `clear-expired-rewards` call with their `claim-rewards` and successfully claim rewards in an expired session.

Recommendation

Implement a check in the `claim-rewards` function to validate that the targeted claim cycle has not expired.

CONTENTS

[M-03] Incorrect Reward Distribution Due to Possible Cycle Desynchronization

Description

The `stableswap-staking-stx-ststx-stx-v-1-1` and `stableswap-emmissions-stx-ststx-stx-v-1-1` smart contracts both rely on a shared cycle schedule based on the `burn-block-height`. However, the Stacks blockchain does not support deploying multiple contracts within a single transaction, which disrupts the assumption of atomic deployment for these two contracts.

As a result, the contracts may be deployed in different blocks, leading to desynchronized cycle schedules. This desynchronization can cause incorrect reward calculations and distributions.

Recommendation

To address this issue:

1. Deploy the `stableswap-staking-stx-ststx-stx-v-1-1` contract first.
2. Initialize the `stableswap-emissions-stx-ststx-stx-v-1-1` contract's `DEPLOYMENT_HEIGHT` by calling the staking contract's `get-deployment-height` function. This ensures that both contracts reference the same start height.

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

[M-04] Potential Exploitation of Staking Mechanism

Description

The current staking mechanism presents several scenarios where users can exploit the reward system without contributing their fair share.

Two key factors to consider:

- The staking mechanism allows users to stake for a minimum of 1 cycle (approximately 24 hours). While the minimum staking period is 1 cycle, unstaking can only occur after an additional cycle.
- Rewards are allocated by administrators per cycle, and the higher the percentage of staked LPs a user has for that cycle, the greater their reward percentage.

The potential exploitation scenarios include:

1. *Exploiting Advance Reward Allocations*

If rewards are allocated starting from a future cycle, for example, 10 cycles (10 days) ahead, there is no incentive for users to stake until 9 cycles from the present.

2. *Exploiting Reward Cycle Gaps*

If rewards are allocated in cycles 10 and 12, a staker can:

- Stake at the end of cycle 9 for 1 cycle (approximately 10 minutes of staking).
- Unstake at the beginning of cycle 11 (staked for 24 hours + 10 minutes in the new cycle).
- Restake at the end of cycle 11 (instantly).
- Unstake at the beginning of cycle 13 (staked for an additional 24 hours).

Thus, due to the absence of a minimum stake period and the allocation of reward cycles with gaps, a staker can stake for only 10 minutes instead of a full 24-hour cycle, skipping the unrewarded cycle while benefiting from the full 2 cycles of rewards.

3. *Exploiting Uneven Reward Allocations*

If rewards are not allocated evenly, users can calculate and determine that it is more cost-effective to stake for a shorter period while maximizing their gains. For example, if rewards are allocated per cycle as follows: cycle 1 (10%), 2 (10%), 3 (10%), 4 (30%), 5 (40%), users will be incentivized to stake only from cycle 4 for 2 cycles, as they would receive 70% of the rewards, whereas if they staked during the first 3 cycles, they would only receive 30%.

CONTENTS

Recommendation

Several changes are recommended, both on-chain and in the off-chain logic:

- Implement a changeable minimum and maximum staking duration within the `stableswap-staking-stx-ststx-stx-v-1-1` contract, with defaults as they currently are `[1 - 120]`.
- Ensure rewards are evenly distributed, or that the differences between cycles are not significant. This means always using `set-rewards-multi` and distributing the rewards as evenly as possible.
- Add rewards with minimal notice to the community. Expired rewards should be collected and re-added to the pool by administrators.

CONTENTS

8.3. Low Findings

[L-01] Invalid Unstaking Fee Can Be Set

Description

In the `stableswap-staking-stx-ststx-v-1-1` contract, stakers who wish to withdraw their initial stake before maturation are required to pay an early unstaking fee. This fee is a percentage of the unstaked amount and can be modified by administrators using the `set-early-unstake-fee` function. However, there is a risk that this fee could mistakenly be set to exceed 100%.

If this occurs, early unstaking would effectively be blocked.

Recommendation

Ensure that when setting the early unstaking fee, it is validated to be less than the equivalent of 100% (`MATH_NUM_10000`).

CONTENTS

[L-02] Missing Validation for Early Unstake Fee Address

Description

The `stableswap-staking-stx-ststx-v-1-1::set-early-unstake-fee-address` function currently lacks validation to ensure that the new fee address is a standard principal address.

This oversight could potentially lead to the redirection of fees to unintended addresses, posing a risk of fund loss.

Recommendation

Implement validation for the `early-unstake-fee-address` by using the following assertion:

```
(asserts! (is-standard address) ERR_INVALID_PRINCIPAL)
```

ClarityAlliance
Security Review

Bitflow Stableswap Staking

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

[L-03] Missing Standard Validation for Rewards Withdrawal Receiver

Description

In the `withdraw-rewards` function of the `stableswap-emissions-stx-ststx-stx-v-1-1` contract, there is no verification to ensure that the new receiver is a valid standard principal address.

Mistakenly sending the rewards to an invalid address could result in a loss of funds for the protocol.

Recommendation

Introduce an `ERR_INVALID_PRINCIPAL` error in the `stableswap-emissions-stx-ststx-stx-v-1-1` contract and trigger it if the `recipient` parameter is not a standard principal (fails the `is-standard` check).

CONTENTS

[L-04] Emission Rewards Not Validated Against Available Balance

Description

Adding rewards to the emission contract involves two steps:

1. The actual reward tokens are directly transferred to the `stableswap-emissions-stx-ststx-stx-v-1-1`
2. The internal accounting of the contract is updated to reflect that rewards are allocated to a specific cycle. This is done by admins calling the `set-rewards` function.

Since these two steps are not atomic, a potential issue may arise if step (2) is completed before step (1). In such a case, if users claim rewards between these steps, their transfers could be reverted due to insufficient funds.

Additionally, when an admin wishes to remove reward tokens, they call the `withdraw-rewards` function. This function does not take a cycle parameter and simply transfers a specified amount of reward tokens from the emission contract to the admin.

Due to the lack of proper validation in this function, more tokens may be removed than necessary to cover the `total-unclaimed-rewards` amount, which also risks causing a revert on user reward claims.

Recommendation

In both the `set-rewards` and `withdraw-rewards` functions of the `stableswap-emissions-stx-ststx-stx-v-1-1` contract, ensure that the emission contract itself has a reward token balance **greater than or equal to** the `total-unclaimed-rewards`. This will ensure that the calls revert if an unbacked reward balance remains.

ClarityAlliance
Security Review

Bitflow Stableswap Staking

CONTENTS

[L-05] Reward Expiration Delta Should Not Be Allowed to Be 0

Description

When setting a reward expiration cycle time delta via the `set-rewards-expiration` function in the `stableswap-emissions-stx-ststx-stx-v-1-1` contract, there is no validation to prevent the expiration from being set to an arbitrarily low number, specifically 0.

If, by mistake, the expiration is set to a few blocks, such as 1 to 3, or even 0, users may lose their rewards.

Recommendation

Implement a minimum value for the expiration. At the very least, it should not be allowed to be 0.

CONTENTS

[L-06] Inner Contract Calls Must Be Updated for Mainnet

Description

Throughout the codebase, there are several instances where contracts call one another. Currently, the full contract name is used for each call. For example:

```
(define-private (transfer-lp-token (amount uint) (sender principal)
  (recipient principal))
  (let (
    (call-a (unwrap! (contract-call?
                                    'ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM.
                  amount sender recipient none) ERR_TOKEN_TRANSFER_FAILED))
  )
    (ok call-a)
  )
)
```

The issue is that all the set addresses are testnet addresses (starting with `ST`), which need to be updated for deployment on the mainnet.

Recommendation

If the same principal will be used to deploy all the contracts, then use the short contract name for all instances. For example, in the `transfer-lp-token` function:

```
(define-private (transfer-lp-token-aba (amount uint) (sender principal)
  (recipient principal))
  (ok (unwrap!
    (contract-call? .stableswap-pool-stx-ststx-v-1-1 transfer amount sender recipient
)
```

If the contracts will be deployed from different accounts, acknowledge this issue and ensure the addresses are updated appropriately before deployment.

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

[L-07] Avoid Using `tx-sender` for Caller Identification

Description

Throughout the codebase, there are instances where `tx-sender` is used instead of `contract-caller` or passing the caller's address. This practice can lead to security vulnerabilities, as users who fall victim to phishing scams and interact with malicious contracts may inadvertently execute sensitive operations within the codebase.

For example, a user might interact with a malicious contract, which could then initiate an early unstaking on their behalf via the `stableswap-staking-stx-ststx-v-1-1::early-unstake-lp-tokens` function call.

Recommendation

Use `contract-caller` instead of `tx-sender` in all instances, except within the SIP-10 `transfer` function and contract-deployer type variables.

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

8.4. QA Findings

[QA-01] Undocumented Staking Particularities

Description

The current staking logic includes some important specific details that are not documented but must be known by users and integrators:

1. *Staked Cycles Beyond Maturity Do Not Yield Rewards*

Users who stake for N cycles and do not unstake at the end of this period might mistakenly believe they will continue earning rewards. This is incorrect. Once the specified N staked cycles have passed, no further rewards will be attributed to the staker.

2. *Current Cycle Does Not Count as Staked*

When staking, the current cycle is not counted as staked. This discourages users from starting to stake at the beginning of the current cycle, as they will miss out on a cycle's worth of potential rewards.

Given the overall system design, this implementation is appropriate to prevent gaming issues, but it needs to be explicitly stated and documented.

Recommendation

Thoroughly document all staking particularities in user-facing APIs and documentation.

On-chain, two comments can be added to avoid cluttering the codebase, which also describe these limitations:

```
;; Staked cycles beyond maturity do not yield rewards
;; Current cycle does not count as staked
(define-public (stake-lp-tokens (amount uint) (cycles uint))
```

[QA-02] Improvement of Staking Contract Filter Functions

Description

In the `stableswap-staking-stx-ststx-v-1-1` contract, several `filter-*` functions are utilized. These functions currently employ `if` statements that return only boolean values. These can be simplified for better readability and efficiency.

Recommendation

Eliminate the `if` statements from the `filter-cycles-list`, `filter-next-cycles-list`, and `filter-unstaked-cycles-list` functions. Instead, directly return the boolean result (or its negation, if necessary) of the evaluated expression.

Example:

```
(define-private (filter-cycles-list (value uint))
- (if (<= value (var-get helper-value))
-   true
-   false
- )
+  (<= value (var-get helper-value))
 )
(define-private (filter-next-cycles-list (value uint))
-  (if (is-some (index-of (var-get helper-list) value))
-   false
-   true
-  )
+  (not (is-some (index-of (var-get helper-list) value)))
 )
(define-private (filter-unstaked-cycles-list (value uint))
-  (if (is-eq value (var-get helper-value))
-   false
-   true
-  )
+  (not (is-eq value (var-get helper-value)))
 )
 (define-private (map-filtered-cycles-list (value uint))
```

CONTENTS

[QA-03] Improper Function Naming Reduces Code Readability

Description

The following function names are overly generic and do not clearly convey their purpose, which hampers code readability:

- `filter-cycles-list`
- `filter-next-cycles-list`
- `filter-unstaked-cycles-list`
- `map-filtered-cycles-list`

These ambiguous names make it difficult for developers to understand the functions' intent, leading to potential misuse and an increased likelihood of bugs.

Recommendation

Select more descriptive names for the functions listed above. For example, consider renaming them as follows:

- Rename `filter-cycles-list` to `filter-values-lte-helper-value`
- Rename `filter-next-cycles-list` to `filter-out-values-contained-in-helper-list`
- Rename `filter-unstaked-cycles-list` to `filter-out-values-eq-to-helper-value`
- Rename `map-filtered-cycles-list` to `sum-with-helper-value`

CONTENTS

[QA-04] Unnecessary Processing of Future Cycles in `unstake-lp-tokens`

Description

The `unstake-lp-tokens` function calculates the amount of LP tokens to unstake using `fold-cycles-to-unstakeable-cycles`, by passing the `user-cycles-to-unstake` argument. However, `user-cycles-to-unstake` includes all user cycles to be unstaked, even those in the future:

```
(current-user-data (unwrap! (map-get? user-data tx-sender) ERR_NO_USER_DATA))
(user-cycles-to-unstake (get cycles-to-unstake current-user-data))
```

Processing future user cycles results in additional, unnecessary iterations, leading to increased transaction costs for users.

Recommendation

Filter the `user-cycles-to-unstake` list to include only cycles that are less than or equal to `get-current-cycle` before passing it to `fold-cycles-to-unstakeable-cycles`. This will help reduce the transaction costs associated with `unstake-lp-tokens` for users.

Since `current-cycle` is already redundantly stored in `unstake-lp-tokens`, consider using it as suggested, or remove it from the function along with the redundant helper-value set.

ClarityAlliance
Security Review

Bitflow Stableswap Staking

CONTENTS

[QA-05] Inconsistent Use of Caller Declaration

Description

Within the codebase, most functions begin with a `let` block where several variables are declared. The last variable is typically the exact caller of the function, set as `(caller tx-sender)`.

However, there are instances where `tx-sender`, which semantically represents the caller, is used before the caller declaration, leading to a somewhat redundant implementation. Examples can be found in the `stableswap-staking-stx-ststx-v-1-1::stake-lp-tokens` contract functions:

```
(define-public (stake-lp-tokens (amount uint) (cycles uint))
  (let (
    (current-user-data (map-get? user-data tx-sender))
    ;; ... code ...
    (caller tx-sender)
```

Recommendation

In all specified locations, move the `caller` declaration to the beginning of the `let` block and use it instead of reusing `tx-sender`.

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

[QA-06] The `cycles-to-unstake` Field in the `fold-early-unstake-per-cycle` Accumulator is Unnecessary

Description

In the `fold-early-unstake-per-cycle` function, the accumulator tuple includes a field named `cycles-to-unstake`, which holds the cycles in which the caller has staked. However, this field is not utilized during the folding process to compute the final tuple value. It is merely read and returned unchanged in each fold iteration, making it redundant.

This unnecessary field increases transaction costs and reduces code readability.

Recommendation

Remove the `cycles-to-unstake` field from the accumulator tuple in the `fold-early-unstake-per-cycle` function.

CONTENTS

[QA-07] Simplification of the `stake-lp-tokens` Function

Description

The `stake-lp-tokens` function in the `stableswap-staking-stx-ststx-v-1-1` contract contains several redundancies that can be streamlined:

1. Simplification of Addition: The expression `(cycle-to-unstake (+ u1 (+ current-cycle cycles)))` can be simplified to `(cycle-to-unstake (+ u1 current-cycle cycles))` .

2. Use of Helper Intermediary Variable Declaration: The data retrieval operation `(map-get? user-data-at-cycle {user: caller, cycle: cycle-to-unstake})` is repeated three times. This can be optimized by introducing a `let` variable, such as `user-data-at-unstaking-unstaking-cycle` , to store the result and reuse it.

   Similarly, the calculation of the new user total LP staked amount, `(+ amount (default-to u0 (get lp-staked current-user-data)))` , is repeated twice. This can be improved by using a `let` variable, such as `new-user-lp-staked`, to store the result and reuse it.

Recommendation

Implement the suggested changes to enhance the function's efficiency and readability.

CONTENTS

[QA-08] Improve Error Handling in `get-cycle-from-height`

Description

The `get-cycle-from-height` function in both the `stableswap-staking-stx-ststx-v-1-1` and `stableswap-emissions-stx-ststx-stx-v-1-1` contracts can unexpectedly revert if the provided height is less than the contract deployment height.

This reversion results in a runtime underflow error, which can be confusing for external integrators.

Recommendation

Implement a custom error message for cases when `get-cycle-from-height` is called with an invalid height.

ClarityAlliance
Security Review

Bitflow
Stableswap
Staking

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

[QA-09] Simplification of Emission User-Data-at-Cycle Map

Description

In the `stableswap-emissions-stx-ststx-stx-v-1-1` contract, the map used to track whether a user has claimed at a specific cycle is currently defined as:

```
(define-map user-data-at-cycle {user: principal, cycle: uint} {
  claimed: bool
})
```

The value is unnecessarily stored as a tuple, even though it contains only a single element, `claimed`. This design choice increases the operational cost, as the value must be packed and unpacked from the map.

Recommendation

To enhance clarity and efficiency, both the map's name and its value type should be revised. A more straightforward example would be:

```
(define-map user-claimed-at-cycle {user: principal, cycle: uint} bool)
```

Additionally, update the associated getters to have more descriptive names.

CONTENTS

ClarityAlliance
Security Review

Bitflow
Stableswap
Staking

[QA-10] Misleading Variable Name: `current-cycle-data`

Description

In the `stableswap-emissions-stx-ststx-stx-v-1-1` contract, contract, several functions declare a variable named `current-cycle-data` as follows:

```
(current-cycle-data (unwrap! (map-get? cycle-data cycle) ERR_NO_CYCLE_DATA))
```

The name `current-cycle-data` is misleading because, in the same context, the actual current cycle is represented by `(current-cycle (get-current-cycle))` . This variable accurately reflects the current cycle, whereas `current-cycle-data` pertains to the data of the targeted cycle.

Recommendation

Rename `current-cycle-data` to a more descriptive name, such as `target-cycle-data`.

CONTENTS

[QA-11] Use Errors Instead of Panicking

Description

Throughout the codebase, there are instances where `unwrap-panic` is used instead of `unwrap!` with a custom error message:

The `get-user-rewards-at-cycle` function from the `stableswap-emissions-stx-ststx-stx-v-1-1` contract contains two `unwrap-panic` calls. If triggered, these could confuse external integrators regarding the nature of the issue.

```
(user-lp-staked (unwrap-panic (get lp-staked user-data-external)))
(cycle-lp-staked (unwrap-panic cycle-data-external))
```

In contrast, the `claim-rewards` function, which includes similar logic to the `get-user-rewards-at-cycle` function, uses distinct error codes for each scenario.

```
(user-lp-staked (unwrap!
  (get lp-staked user-data-external) ERR_NO_EXTERNAL_USER_DATA))
(cycle-lp-staked (unwrap! cycle-data-external ERR_NO_EXTERNAL_CYCLE_DATA))
```

Additionally, in the `stableswap-core-v-1-1` contract, `unwrap-panic` is used in several places with standard `SIP-10` functions such as `get-symbol` and `get-decimals`, as well as when splitting strings. While the string splitting implementation cannot revert, the SIP-10 function calls may revert on non-standard tokens.

Ending execution with a panic results in a runtime error. Runtime errors cannot be handled by the caller and do not provide meaningful information about the execution, thus they are discouraged.

Recommendation

Modify the `get-user-rewards-at-cycle` function to return `ERR_NO_EXTERNAL_USER_DATA` and `ERR_NO_EXTERNAL_CYCLE_DATA` in the specified situations.

In the `stableswap-core-v-1-1` contract, replace the `unwrap-panic` calls applied to the `get-symbol` and `get-decimals` functions with a custom `NON_SIP_10_STANDARD_TOKEN` error.

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Staking

[QA-12] Simplification of fold-cycles-to-unstakeable-cycles

Description

The `fold-cycles-to-unstakeable-cycles` function in the `stableswap-staking-stx-ststx-v-1-1` contract can be simplified to enhance code clarity and efficiency.

The current `user-cycle-data` declaration is as follows:

```
(user-cycle-data (match
  (map-get? user-data-at-cycle {user: tx-sender, cycle: cycle})
  unwrapped-value
    unwrapped-value
  {lp-staked: u0, lp-to-unstake: u0}
))
```

This can be modified to:

```
(user-cycle-data (default-to {lp-staked: u0, lp-to-unstake: u0}
(map-get? user-data-at-cycle {user: caller, cycle: cycle})))
```

This change not only simplifies the code but also aligns it with the implementation in the `fold-user-data-per-cycle` function.

Recommendation

Implement the suggested modification.