ClarityAlliance

sBTC REWARDS PROGRAM SECURITY REVIEW

Conducted by:
KRISTIAN APOSTOLOV, ALIN BARBATEI (ABA), MARCHEV

JANUARY 5TH, 2025

CONTENTS

ClarityAlliance
Security Review

sBTC Rewards Program

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

sBTC Rewards Program

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

3. Introduction

A time-boxed security review of sBTC Rewards Program, where Clarity Alliance reviewed the scope and provided insights on improving the protocol.

4. About sBTC Rewards Program

The sBTC Rewards Program enables Bitcoin users to earn a 5% return on their holdings, paid in sBTC, without the need to stake or lock their assets. To participate, users can enroll through the sBTC Rewards Dashboard, which allows them to hold sBTC, earn sBTC, and unlock the full potential of their Bitcoin.

The program operates through a Contributor Smart Contract, which manages the delegation of stacking rewards. This contract allows contributors to delegate and revoke their staking rewards to the incentive program. It also enables the contract administrator to manage stack aggregation, including committing, extending, and increasing stack amounts for contributors.

To get started, users can follow the onboarding process outlined in the sBTC Incentive Protocol documentation. This process involves visiting the designated platform, navigating to the Yield section, enrolling to participate, and, upon transaction confirmation, beginning enrollment with the next cycle. Enrollment remains active for all future cycles unless the user opts out of the program.

Source: https://bitcoinismore.org/

CONTENTS

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.

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

6. Security Assessment Summary

The primary emphasis was on deposit functionality, as the system was not yet live for withdrawals at the time of review.

- Initial analysis started at [89cf1de30fe3668ba8196fa1fc3418946d33cbdb89cf1de30fe3668ba8196fa1fc3418946d33cbdb]
- Subsequent analysis was conducted from [a29794a3f16c92359d0c3bf5a29029888a04a00f]

ClarityAlliance
Security Review
sBTC Rewards Program

CONTENTS

7. Executive Summary

Over the course of the security review, Kristian Apostolov, Alin Barbatei (ABA), Marchev engaged with - to review sBTC Rewards Program. In this period of time a total of **18** issues were uncovered.

Protocol Summary

Protocol Name	sBTC Rewards Program
Date	January 5th, 2025

Findings Count

Severity	Amount
Critical	1
High	1
Medium	3
Low	3
QA	10
Total Findings	**18**

CONTENTS

Summary of Findings

ID	Title	Severity	Status
[C-01]	Attacker Can Opt-Out Before Reward Distribution to Block Cycle Advancement	Critical	Resolved
[H-01]	Rewards Can Be Reduced Through Cycle Conclusion Overloading	High	Resolved
[M-01]	Changing Donors Reward Address Leaves Uncommitted Stakes	Medium	Resolved
[M-02]	Entire Holdings System Is Vulnerable to Griefing	Medium	Resolved
[M-03]	Snapshot May Use Nonexistent sBTC Balance	Medium	Resolved
[L-01]	Inability to Stack STX in the Final Block of the Current Reward Cycle	Low	Resolved
[L-02]	Implement an Emergency Withdrawal Mechanism	Low	Acknowledged
[L-03]	Ambiguous Invocation of Finalize Reward Distribution	Low	Resolved
[QA-01]	Typographical Errors	QA	Resolved
[QA-02]	Remove Unused Constants	QA	Resolved
[QA-03]	Inconsistent Donors Error Codes	QA	Resolved
[QA-04]	Restrict Multiple Calls to Set Reward Distribution Flag	QA	Resolved
[QA-05]	Optimization Opportunities in Staking Operations	QA	Resolved
[QA-06]	Inconsistent Event Strings in Yield Contract	QA	Acknowledged
[QA-07]	Computing Snapshot Balance Should Not Be Callable Between Cycle Snapshot Conclusion and New Cycle	QA	Acknowledged
[QA-08]	Reward Distribution Should Not Be Callable After All Participants Have Been Rewarded	QA	Acknowledged
[QA-09]	Yield Contract Contains Outdated or Incorrect Comments	QA	Acknowledged
[QA-10]	Improvements Needed for Yield Contract Admin Update	QA	Acknowledged

ClarityAlliance
Security Review
sBTC Rewards Program

CONTENTS

ClarityAlliance
Security Review

sBTC Rewards Program

8. Findings

8.1. Critical Findings

[C-01] Attacker Can Opt-Out Before Reward Distribution to Blocking Cycle Advancement

Description

During the reward distribution process, the current logic retrieves the participant (holder) state from the time the snapshot was taken but fetches the current reward address.

```
(participant-state (unwrap!
(map-get? participants {address: user}) ERR_NOT_ENROLLED))
```

Retrieving the current reward address from participants may fail if the user is no longer participating. If, during cycle advancement, a user has opted out or has been blacklisted by the admin, retrieving the reward address results in the `ERR_NOT_ENROLLED` error code. This blocks the cycle advancement because the rewarded counter is not incremented.

Without incrementing the rewarded counter, advancing cycles via the `head-to-next-cycle` function reverts with `ERR_NOT_REWARDED_ALL` .

An attacker can exploit this issue by spamming several hundred principals with no sBTC, having them all enroll, and then opting out before the final reward distribution.

The admin would need to call the `enroll-dex` function on several hundred functions at a time and then call `distribute-rewards` afterward. At any point, the attacker can re-opt out, causing a continuous denial of service (DOS).

Recommendation

If the intention is to use the most recent reward address, then, in its absence, default to using the address from the snapshot moment.

This issue was also identified by the `sBTC` yield smart contract developers and resolved before deployment.

CONTENTS

ClarityAlliance
Security Review

sBTC Rewards Program

8.2. High Findings

[H-01] Rewards Can Be Reduced Through Cycle Conclusion Overloading

Description

At the end of a cycle, the snapshot must be concluded before a new one begins. This is achieved through the `yield::conclude-cycle-snapshots` function call, which sets the `concluded-snapshots` flag and updates the current cycle's total balance with the last snapshot balance.

However, the function does not verify whether it has already been called, allowing an attacker to invoke it multiple times. This results in the artificial inflation of the snapshot balance with each call:

```
(var-set current-cycle-total (+ (var-get current-cycle-total)
  (var-get current-snapshot-total)))
```

Although the overall cycle balance increases, users maintain the same absolute balance. Consequently, fewer rewards are distributed because their share of the total amount is artificially reduced.

An attacker can exploit this to deliberately decrease user rewards.

Recommendation

Ensure that when `yield::conclude-cycle-snapshots` is called, it is verified that the function has not already been executed.

ClarityAlliance
Security Review

sBTC Rewards Program

8.3. Medium Findings

[M-01] Changing Donors Reward Address Leaves Uncommitted Stakes

Description

The `pox-4` stacking contract identifies partially stacked STX (after the `delegate-stack-stx` function is invoked) using a lookup key composed of: `(pox-addr, stx-address, reward-cycle)`.

The `donors` contract encapsulates all the `pox-4` functions that manage the delegated stake:

- `donors::admin-stack-extend` → `pox-4::delegate-stack-extend`
- `donors::admin-stack` → `pox-4::delegate-stack-stx`
- `donors::admin-stack-increase` → `pox-4::delegate-stack-increase`

In each instance, these functions are executed with their respective arguments, but they all maintain the same Bitcoin reward address recipient (`rewards-address`).

The `donors` contract also includes functions that commit the partially locked STX to the reward cycle, a necessary step to ensure rewards are provided:

- `donors::admin-aggregation-commit` → `pox-4::stack-aggregation-commit-indexed`
- `donors::admin-aggregation-increase` → `pox-4::stack-aggregation-increase`

Additionally, the Bitcoin reward address can be altered through admin intervention by calling `donors::update-rewards-address`.

A problem arises with the current design: reward cycles may be skipped if the `rewards-address` , which identifies the partially locked delegated STX, is changed without committing the previous rewards via a `pox-4::stack-aggregation-commit-indexed` call.

If the rewards address is changed before a commit call is made, the previous delegation is essentially discarded, leading to no rewards and necessitating re-delegation. Reference.

Recommendation

Do not allow `donors::update-rewards-address` to be called without first committing the existing delegation.

A flag can be implemented, which is set whenever any function influencing delegation is called and is unset only when a commit is executed. Ensure `update-rewards-address` checks this flag and only permits the call if it is unset. This approach prevents the admin from inadvertently changing the reward address before committing the delegation.

[M-02] Entire Holdings System Is Vulnerable to Griefing

Description

The current `sBTC` reward strategy allows holders to enroll without incurring any cost. Although there is no direct cost to the holders, the more participants that enroll, the more expensive it becomes to iterate through every snapshot and cycle. This is because a new cycle or snapshot cannot commence until all participants from the previous ones have been accounted for.

Bulk accounting functions can process up to 900 participants and are permissionless, meaning anyone can call them as long as they are willing to pay the gas fees. However, due to current Stacks block limitations and costs, performing a bulk compute snapshot call for 600 principals already uses up almost the entire block read count (92%), and distributing rewards to the same number of principals uses 84%.

| Function Call (# of Principals Passed) | Write Length | Write Count | Read Length | Read Count | Runtime | | ---------- | ---------- | ---------- | ---------- | ---------- | | `compute-current-snapshot-balances (600)` | 109,256 | 1,202 | 3,018,811 | 13,808 | 39,372,434 | | `distribute-rewards (600)` | 170,438 | 3,001 | 3,010,413 | 12,608 | 37,531,752 |

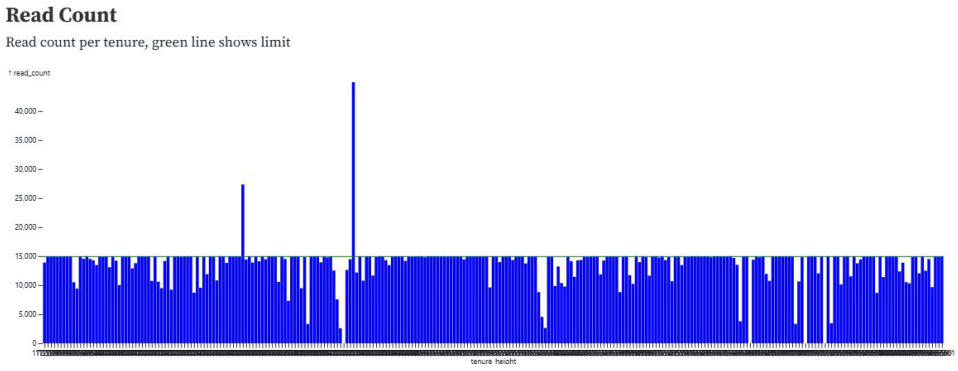
| Function Call (# of Principals Passed) | Write Length | Write Count | Read Length | Read Count | Runtime | | ---------- | ---------- | ---------- | ---------- | ---------- | | `compute-current-snapshot-balances (600)` | 0.73% | 8.01% | 3.02% | 92.05% | 0.79% | | `distribute-rewards (600)` | 1.14% | 20.01% | 3.01% | 84.05% | 0.75% |

The absence of a minimum threshold allows an attacker to:

- Create a large number of principals, e.g., 100,000.
- Transfer just enough STX to enable them to call `enroll`.
- (Optional) Transfer 1 satoshi to keep the reward distribution cost high.

CONTENTS

By doing this, users will be forced to pay very high execution fees just to ensure that the reward cycles continue. This issue is exacerbated because read counts are currently the main bottleneck in processing blocks, with nearly 100% usage in each tenure.



Recommendation

Introduce a minimum `sBTC` amount threshold that must be held when computing stats for the current cycle. This minimum can initially be set to 0 and gradually increased if necessary.

Additionally, implement a bulk blacklist functionality. While admins can manually opt-out users, doing so one transaction at a time is impractical in the event of an attack.

CONTENTS

[M-03] Snapshot May Use Nonexistent sBTC Balance

Description

User `sBTC` snapshots are currently taken using the `sBTC::get-balance` function. This function includes the pending withdrawn balance, which can lead to users being rewarded based on more tokens than they actually possess.

Although this is not an immediate issue since `sBTC` withdrawals will not be activated until the first quarter of 2025, it could result in incorrect reward balances once withdrawals are enabled.

Recommendation

To prevent this issue, use the `sBTC::get-balance-available` function instead of the basic `get-balance`.

8.4. Low Findings

[L-01] Inability to Stack STX in the Final Block of the Current Reward Cycle

Description

When the `donors` contract administrator initiates a STX stack, they invoke the `admin-stack` function, which subsequently calls the `pox-4::delegate-stack-stx` function with specific parameters.

One of these parameters, the start burn height, which indicates when staking should commence, is set to the current burnchain block height plus one.

```
(let ((start-burn-ht (+ burn-block-height u1)))
```

However, the pox-4 implementation mandates that the start-burn-ht must correspond to the next reward cycle when utilized.

```
(let ((first-reward-cycle (+ u1 (current-pox-reward-cycle)))
      (specified-reward-cycle (+ u1
        (burn-height-to-reward-cycle start-burn-ht)))
      (unlock-burn-height (reward-cycle-to-burn-height (+
        (current-pox-reward-cycle) u1 lock-period))))

    ;; the start-burn-ht must result in the next reward cycle, do not allow stackers
  ;; to "post-date" their `stack-stx` transaction
  (asserts! (is-eq first-reward-cycle specified-reward-cycle)
            (err ERR_INVALID_START_BURN_HEIGHT))
```

This creates a specific issue when administrators call `admin-stack` during the last block of the current reward cycle. In such cases, the operation reverts with `ERR_INVALID_START_BURN_HEIGHT`, forcing the administrator to wait until the next cycle to stake, resulting in missed rewards for stakers.

Recommendation

Invoke the `pox-4::delegate-stack-stx` function directly with `burn-block-height` as the `start-burn-ht` parameter.

CONTENTS

[L-02] Implement an Emergency Withdrawal Mechanism

Description

In the event that the smart contract's reward distribution is deemed closed while an `sBTC` balance still exists, there is currently no mechanism to withdraw and transfer the remaining sBTC to a new contract.

Recommendation

Introduce an emergency sBTC withdrawal function to handle any sBTC left in the contract. Consider incorporating this as part of an emergency state, which would temporarily halt contract operations.

While this approach may not align with the principles of decentralization, it would ensure that no Bitcoin is permanently lost in the event of an emergency.

CONTENTS

[L-03] Ambiguous Invocation of Finalize Reward Distribution

Description

Once a cycle is finalized and all participants have been enrolled, the `yield::finalize-reward-distribution` function must be invoked to update the `distribution-finalized-stx-block-height-when-called` map.

There are two scenarios where this function can be called inappropriately:

1. There is no restriction on multiple invocations, allowing the function to be called even after the initial call. The only consequence is altering the `stx-block-height` recorded as the ending block. While not critical, this issue should be addressed.
2. The `finalize-reward-distribution` function can be invoked from the beginning to the end of a new cycle when the number of participants is zero, due to the condition:

```
(asserts! (is-eq (var-get current-cycle-participants-count)
   (var-get rewarded-count)) ERR_NOT_REWARDED_ALL)
```

being satisfied.

Setting `distribution-finalized-stx-block-height-when-called` prematurely impacts external integrators, as anyone relying on `yield::stx-block-height-distribution-finalized` will perceive the cycle as finalized as soon as it begins. However, in reality, the transition to the next cycle does not occur until the tenure reaches the required height.

Recommendation

Eliminate the `finalize-reward-distribution` function and integrate its functionality into the `distribute-rewards` function. Before exiting `distribute-rewards`, in each branch, verify if all participants have been rewarded and set the `distribution-finalized-stx-block-height-when-called` within this function.

CONTENTS

ClarityAlliance
Security Review

sBTC Rewards Program

8.5. QA Findings

[QA-01] Typographical Errors

Description

There are several typographical errors and opportunities for slight wording improvements throughout the codebase within scope:

- `yield`
  - **L401:** `numberof` should be corrected to `number of`.
  - **L530:** `alredy-rewarded-amount` should be corrected to `already-rewarded-amount`

Recommendation

Implement all the indicated changes.

CONTENTS

[QA-02] Remove Unused Constants

Description

There are instances of unused constants within the codebase:

- In `donors` : `err-donors-contract-not-allowed`

Recommendation

Remove the unused constant to enhance code readability, reduce clutter, and slightly decrease runtime read counts and costs. Additionally, after removal, rebase the error codes to eliminate any gaps.

[QA-03] Inconsistent Donors Error Codes

Description

The `donors` contract captures error codes from `pox-4` calls, wraps them, and re-emits them.

Since `pox-4` error codes are integers, not unsigned integers, they are converted to unsigned integers and offset.

The inconsistency arises in the offsetting process. In some cases, the error codes are redundantly offset by multiplying by 1, effectively applying no offset, while in other cases, they are offset by 1000:

```
(define-public (admin-stack (stacker principal) (amount-ustx uint)
  (lock-period uint))
  ;; ... code ...
     error (err (* u1 (to-uint error))))))

;; Private functions
(define-private (donor-delegate-inner (amount-ustx uint)
  (delegate-to principal) (until-burn-ht (optional uint)))
  ;; ... code ...
    error (err (* u1000 (to-uint error))))))

;; Private functions
(define-private (contributor-delegate-inner (amount-ustx uint)
  (delegate-to principal) (until-burn-ht (optional uint)))
(let ((result-revoke
        ;; Calls revoke and ignores result

          (contract-call? 'ST000000000000000000002AMW42H.pox-4 revoke-delegate-stx)))
  ;; Calls delegate-stx, converts any error to uint
  (match
    (contract-call? 'ST000000000000000000002AMW42H.pox-4 delegate-stx amount-ustx dele
    success (ok success)
    error (err (* u1000 (to-uint error))))))
```

This inconsistency reduces code readability, complicates debugging for external integrators in case of errors, and is slightly more costly to execute.

Recommendation

Standardize the error handling by either consistently applying a non-one offset or not offsetting at all, as the error code ranges for the `pox-4` contract do not overlap with those of the `donors` contract.

CONTENTS

[QA-04] Restrict Multiple Calls to Set Reward Distribution Flag

Description

Once a cycle concludes, the function `yield::set-can-distribute-rewards` should be invoked to initiate reward distribution, provided all operations are complete.

This function sets the `can-distribute-rewards` flag, which is essential for commencing the actual reward distribution process.

Currently, the function can be called multiple times without restriction until the cycle is finalized.

Although the function's side effects are deterministic and occur only once, making multiple calls inconsequential, it is generally advisable to prevent such functions from being executed multiple times.

Recommendation

Ensure that `yield::set-can-distribute-rewards` cannot be called if `can-distribute-rewards` is already set to true.

[QA-05] Optimization Opportunities in Staking Operations

Description

Within the `yield` contract, there are several minor optimizations that can enhance execution efficiency and improve code readability:

- In `compute-and-update-balances-one-user` :
  - The `is-enrolled-this-cycle` function is invoked twice for new addresses. Consider storing its result in a `let` declaration and reusing that variable.
  - Reuse `local-current-snapshot-index` for printing instead of repeatedly calling `(var-get current-snapshot-index)` .
  - Create a variable for the `cycle-id` since `(var-get cycle-id)` is called three times.

- In `update-snapshot-for-new-cycle` , the `let` can be eliminated by directly setting `current-snapshot-tenure-height` to the value of `(var-get current-cycle-tenure-height)` .

- Several functions that return `(ok true)` can utilize the result of the preceding command instead of the hardcoded `true` value:
  - In `conclude-cycle-snapshots` , use the return value of `(var-set concluded-snapshots true)`
  - In `initialize-contract` , use the return value of `(var-set is-contract-active true)`
  - In `enroll` and `enroll-dex` , use the return value of `(map-set participants ...`
  - In `check-new-cycle-valid-stacks-block-height` and `check-new-snapshot-valid-stacks-block-height` , use the return value of the previous `asserts!` .
  - In `finalize-reward-distribution` , use the return value of the preceding `map-set` .
  - In `head-to-next-cycle` , use the return value of the preceding `map-set` .
  - In `set-can-distribute-reward` , use the return value of the `(var-set can-distribute-rewards true)` call.

- In `distribute-reward-user` , `(var-get cycle-id)` is called five times. Store it in a `let` variable and reuse it.

Recommendation

Implement the suggested changes.

CONTENTS

ClarityAlliance
Security Review

sBTC Rewards Program

[QA-06] Inconsistent Event Strings in Yield Contract

Description

In the `yield` contract, some functions consistently print events using the `function-name` as the primary entry point name. However, there are three instances where inconsistencies occur:

- The event string associated with `compute-current-snapshot-balances` lacks the leading `s`, appearing as `function-name: "compute-current-snapshot-balance"`.
- The `enroll-defi` function uses the same `"enroll"` message as the `enroll` function, which can cause confusion.
- Similarly, the `opt-out-defi` function shares the `"opt-out"` message with the `opt-out` function, leading to potential misunderstandings.

Recommendation

Adjust the specified event strings to eliminate these inconsistencies.

CONTENTS

ClarityAlliance
Security Review

sBTC Rewards Program

[QA-07] Computing Snapshot Balance Should Not Be Callable Between Cycle Snapshot Conclusion and New Cycle

Description

During the transition of the `yield` contract through various states, there is a point where the current cycle is concluded by invoking the `conclude-cycle-snapshots` function, which sets the `concluded-snapshots` flag.

From this point, there are four additional phases before the next cycle begins. Throughout these phases, the `compute-current-snapshot-balances` function can still be called arbitrarily.

While multiple calls to `compute-current-snapshot-balances` after all users have been accounted for do not produce side effects, it is generally not advisable for state-machine-like systems to permit unnecessary cross-state calls. This practice helps prevent any unforeseen or unaccounted-for situations.

Recommendation

In the `compute-current-snapshot-balances` function, ensure that `current-snapshot-count` is not already equal to `current-cycle-participants-count` .

CONTENTS

ClarityAlliance
Security Review

sBTC Rewards Program

[QA-08] Reward Distribution Should Not Be Callable After All Participants Have Been Rewarded

Description

During the transition of the `yield` contract through various states, rewards for the current cycle are distributed via calls to the `distribute-rewards` function.

Once all rewards have been distributed (i.e., when `rewarded-count` equals `current-cycle-participants-count`), the subsequent and final two phases can proceed.

From that point until a new cycle begins, the `distribute-rewards` function can still be called arbitrarily.

Although multiple calls to `distribute-rewards` after full reward distribution do not cause side effects, it is not advisable for state-machine-like systems to permit unnecessary cross-state calls, as this could lead to unforeseen or unaccounted situations.

Recommendation

In the `distribute-rewards` function, ensure that `rewarded-count` is not already equal to `current-cycle-participants-count`, indicating that this further execution is unnecessary.

CONTENTS

[QA-09] Yield Contract Contains Outdated or Incorrect Comments

Description

The yield contract includes several outdated comments that should be updated or removed:

- `L36` : The comment `;; 8% APR - 0,1 * 10^8` contains an incorrect calculation. It should be `0,08 * 10^8` instead of `0,1` .
- `L38` : The comment `;; Modularize the variables for different networks` is outdated.
- `L48` : The comment `;; For Testnet` is no longer relevant.
- `L103` : The comment mentions `verify at get-burn-block-info state of map when doing calculations` , but in practice, `get-stacks-block-info?` is used instead of `get-burn-block-info` .
- `L232` : The comment `;; enroll-for-rewards` is outdated as the function is now called `enroll` .

Recommendation

Address the identified issues or inconsistencies.

ClarityAlliance
Security Review
sBTC Rewards Program

CONTENTS

ClarityAlliance
Security Review

sBTC Rewards Program

[QA-10] Improvements Needed for Yield Contract Admin Update

Description

In the `yield` contract, the process of changing or updating the admin is carried out through the `update-admin` function. There are two enhancements that can be made to this function:

- Validate that the provided address is appropriate for the current network by using the `is-standard` function.
- Implement a two-step transfer mechanism, where the current admin initiates the update, setting a `pending-admin` principal. In the second step, the `pending-admin` must call a corresponding `accept-admin` function.

Both validations significantly reduce the risk of errors when updating the admin.

Recommendation

Implement a two-step admin transfer mechanism and incorporate a `is-standard` principal check.