ClarityAlliance

STACKING DAO SECURITY REVIEW

Conducted by:
KRISTIAN APOSTOLOV, ARABADZHIEV, STORMY

NOVEMBER 6TH, 2024

CONTENTS

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.

ClarityAlliance
Security Review
Stacking DAO

CONTENTS

ClarityAlliance
Security Review
Stacking DAO

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 the Stacking DAO implementation, where Clarity Alliance reviewed the scope, whilst simultaneously building out a testing suite for the protocol.

4. About Stacking DAO

A liquid stacking protocol that gives users an auto-compounding tokenised representation of stacked STX (stSTX).

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

CONTENTS

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

ClarityAlliance
Security Review
Stacking DAO

6. Security Assessment Summary

Review Commit Hash:
4291d8ff8da0a8d2f69c69ba101af1e527a0bba1

Scope

The following contracts were in the scope of the security review:

- `/version-2/commission-v2`
- `/version-2/data-core-v1`
- `/version-2/data-direct-stacking-v1`
- `/version-2/data-pools-v1`
- `/version-2/delegates-handler-v1`
- `/version-2/direct-helpers-trait-v1`
- `/version-2/direct-helpers-v1`
- `/version-2/protocol-arkadiko-v1`
- `/version-2/rewards-trait-v1`
- `/version-2/rewards-v1`
- `/version-2/stacking-dao-core-v2`
- `/version-2/stacking-delegate-1`
- `/version-2/stacking-delegate-trait-v1`
- `/version-2/stacking-pool-payout-v1`
- `/version-2/stacking-pool-signer-v1`
- `/version-2/stacking-pool-v1`
- `/version-2/strategy-v2`
- `/version-2/strategy-v3-algo-v1`
- `/version-2/strategy-v3-delegates-v1`
- `/version-2/strategy-v3-pools-v1`
- `/version-2/strategy-v3`
- `/version-3/sdao-token`
- `/version-3/staking-v1`

CONTENTS

ClarityAlliance
Security Review
Stacking DAO

7. Executive Summary

Over the course of the security review, Kristian Apostolov, Arabadzhiev, Stormy engaged with StackingDAO to review their core protocol source code. In this period of time a total of **19** issues were uncovered.

Protocol Summary

Protocol Name	Stacking DAO
Repository	https://github.com/StackingDAO/StackingDAO
Date	November 6th, 2024
Protocol Type	Liquid Staking Token

Findings Count

Severity	Amount
High	2
Medium	4
Low	5
QA	8
Total Findings	**19**

CONTENTS

Summary of Findings

ID	Title	Severity	Status
[H-01]	PoX Rewards Can Be Maliciously Locked	High	Resolved
[H-02]	update-direct-stacking Inconsistency Leading to Pool Preparation DoS	High	Resolved
[M-01]	Inability to Complete Withdrawal at Maturing Block	Medium	Resolved
[M-02]	Incorrect STX Amount Delegated Upon Cancelling Withdrawal	Medium	Resolved
[M-03]	Infinite total-direct-stacking Inflation Leading to Numerous Inconsistencies	Medium	Partially Resolved
[M-04]	update-direct-stacking Lacks Adequate Access Control	Medium	Resolved
[L-01]	ststx-withdraw-nft Does Not Fully Comply with SIP-9	Low	Acknowledged
[L-02]	Potential Read-Only Reentrancy When Buying NFT	Low	Resolved
[L-03]	Vulnerability to Sandwich Attacks in buy-in-ustx	Low	Acknowledged
[L-04]	Use contract-caller instead of tx-sender for admin actions	Low	Acknowledged
[L-05]	Incorrect Commission Rounding	Low	Acknowledged
[QA-01]	Sale of Matured Withdrawals	QA	Acknowledged
[QA-02]	English Dialect Inconsistencies	QA	Acknowledged
[QA-03]	Returning Response Types in Read-Only Functions is an Antipattern	QA	Acknowledged
[QA-04]	Add Conditional Error Handling	QA	Acknowledged
[QA-05]	Transfer Function Simplification	QA	Acknowledged
[QA-06]	Return Type Restructuring	QA	Acknowledged
[QA-07]	Use Constant for BPS Across All Contracts	QA	Acknowledged
[QA-08]	NFT Listing Remains in a Corrupted State After Withdrawal Completion	QA	Resolved

CONTENTS

ClarityAlliance
Security Review
Stacking DAO

8. Findings

8.1. High Findings

[H-01] PoX Rewards Can Be Maliciously Locked

Description

The `rewards-v1` contract is responsible for storing the PoX rewards for a specific cycle. Once the cycle concludes, the `process-rewards` function transfers the reward STX to `reserve-v1`. This function accepts three parameters:

- `commission-contract` - the current whitelisted commission contract implementation.
- `staking-contract` - the current whitelisted staking contract implementation.
- `reserve` - the current whitelisted reserve contract implementation.

All three parameters are validated as whitelisted protocol contracts using `dao.check-is-protocol`.

```
(try! (contract-call? .dao check-is-protocol reserve))
    (try! (contract-call? .dao check-is-protocol
      (contract-of commission-contract)))
    (try! (contract-call? .dao check-is-protocol
      (contract-of staking-contract)))
```

The issue arises because only the `reserve` parameter is validated as a whitelisted protocol contract. Since the only validation performed is to ensure it is a protocol contract, any protocol contract can be passed instead of `reserve-v1`. If a different protocol contract is passed, the entire cycle reward amount can be permanently locked, disrupting core protocol functionality and resulting in the loss of all stackers' funds.

Recommendation

Revise the `process-rewards` function as follows:

```
(define-public (process-rewards
  (commission-contract <commission-trait>)
  (staking-contract <staking-trait>)
  (reserve <reserve-trait>) ;; @audit
)
 (begin
  (try! (contract-call? .dao check-is-enabled))
  (try! (contract-call? .dao check-is-protocol reserve))
  (try! (contract-call? .dao check-is-protocol
    (contract-of commission-contract)))
  (try! (contract-call? .dao check-is-protocol
    (contract-of staking-contract)))
  (asserts! (> burn-block-height (var-get rewards-unlock))
    (err ERR_CAN_NOT_PROCESS_YET))

  (if (> (var-get total-commission) u0)
    (try! (as-contract
      (contract-call? commission-contract add-commission staking-contract (var-get t
    u0
  )
  (if (> (var-get total-rewards-left) u0)
    (try! (as-contract (stx-transfer? (var-get total-rewards-left) tx-sender
      (contract-of reserve)))) ;; @audit
    false
  )

  (var-set total-commission u0)
  (var-set total-rewards-left u0)
  (print { action: "process-rewards", data: { cycle: (
    print{action:"process-rewards",
    data:{cycle:
  ), commission-amount: (var-get total-commission
  (ok true)
 )
)
```

[H-02] `update-direct-stacking` Inconsistency Leading to Pool Preparation DoS

Description

The `update-direct-stacking` function is designed to address irregularities in direct stacking amounts caused by transferring `stSTX`. It calculates the amount of `stSTX` owned by the user, either directly or stored within a protocol, and reduces their direct stacked amount if it exceeds the amount the user currently owns.

```
(if (> diff u0)
    (begin
        (try! (as-contract (subtract-direct-stacking user diff)))
        true
    )
    false
)
```

The issue with this function is that it calculates `diff` as the `stSTX` representation of the user's virtual-real balance, rather than in `STX`. The snippet below will function incorrectly because the `stSTX:STX` ratio is not 1:1, resulting in less `STX` being subtracted from the user's balance.

```
(diff (if (> stacking-ststx balance-ststx)
   (- stacking-ststx balance-ststx)
   u0
))
```

A more significant issue arises from how `direct-stacking-ststx` (which is unwrapped into `stacking-ststx`) is calculated in `calculate-direct-stacking-info`.

```
(direct-stacking-ststx (/ (* direct-stacking DENOMINATOR_6) ratio))
```

This calculation truncates, resulting in at least one satoshi of excess in the user's virtual directly stacked balance, thereby failing to fulfill the function's intended purpose.

This inconsistency causes the pool preparation flow in `strategy-3` to be subject to a Denial of Service (DoS) due to an underflow in the calculation within `strategy-v3-pools-v1::calculate-new-amounts`, as the virtual direct balance is higher than the actual balance.

```
(new-total-normal-stacking (-
(+ total-stacking total-idle) total-withdrawals new-total-direct-stacking))
```

Consequently, this issue disables pool preparation before a stacking cycle, unless the team manually adjusts the state before the cycle begins.

Recommendation

Change `update-direct-stacking:diff` to be denominated in `STX`.

8.2. Medium Findings

[M-01] Inability to Complete Withdrawal at Maturing Block

Description

The `unlock-burn-height` is intended to be the block height at which a withdrawal can be finalized. However, the current contract includes the following assertion in `stacking-dao-core-v2.withdraw`:

```
(asserts! (> burn-block-height unlock-burn-height) (err ERR_WITHDRAW_LOCKED))
```

This assertion prevents users from completing their withdrawal at the `unlock-burn-height` and also from canceling it using `stacking-dao-core-v2.cancel-withdraw`, effectively locking the user's funds for a longer period than necessary. The `stacking-dao-core-v2.cancel-withdraw` function contains the following assertion:

```
(asserts! (< burn-block-height unlock-burn-height) (err ERR_WITHDRAW_CANCEL))
```

Recommendation

Modify the assertion in `stacking-dao-core-v2.withdraw` to:

```
(asserts! (>= burn-block-height unlock-burn-height) (err ERR_WITHDRAW_LOCKED))
```

CONTENTS

[M-02] Incorrect STX Amount Delegated Upon Cancelling Withdrawal

Description

When a user initiates a stSTX withdrawal, their direct stacking amount is reduced accordingly. If they choose to cancel their withdrawal, the amount they cancel should be added back to their direct stacking amount for the selected pool. However, the current implementation executes the following:

```
(try!
  (contract-call? direct-helpers add-direct-stacking tx-sender pool stx-amount))
```

This adds the `stx-amount` tokens based on their value at the time the withdrawal was initiated. Since at least one new cycle must pass before a withdrawal can be completed, the value of the stSTX to be withdrawn will exceed the original `stx-amount`. Therefore, using `stx-amount` as the new direct stacking amount will disrupt the user's direct/general staking ratio and may lead to unexpected side effects.

Recommendation

Consider calculating the current value of the stSTX amount and using this value in the `add-direct-stacking` function instead.

CONTENTS

ClarityAlliance
Security Review
Stacking DAO

[M-03] Infinite `total-direct-stacking` Inflation Leading to Numerous Inconsistencies

Description

The protocol increases a user's directly stacked balance when they call `deposit` or `cancel-withdraw`, based on the specified pool in the function call. There are no other entry points for users to increase their `direct-stacking-user` balance.

When a user receives `stSTX` that they haven't directly stacked themselves, an issue arises with the `virtual:real directly stacked` token balance. This is because the transfer is not reflected in the balances of either the sender or the receiver. To address these discrepancies in direct stacking amounts caused by transferring `stSTX`, the `update-direct-stacking` function was created.

The problem with this functionality arises from how `stop-direct-stacking` operates within the system. It only removes up to the `direct-stacking-user` balance of a particular user when adjusting the `direct-stacking-pool-amount` and `total-direct-stacking` balances. Consequently, when a user initiates a withdrawal with tokens they haven't minted themselves, they enter `stop-direct-stacking` from `subtract-direct-stacking` due to the following condition:

```
(if (>= amount current-direct-amount)
        (begin
                (try! (as-contract (stop-direct-stacking user)))
                true
        )
        ;; ...
```

This condition allows for a scenario where `total-direct-stacking` can seemingly increase up to the unsigned 128-bit integer limit of `2 ** 128 - 1`, causing overflows throughout the system.

A realistic example of this inflation algorithm is as follows: `Contract[0]` takes a flash loan of 10M `stSTX` and calls `init-withdraw`, then transfers the withdrawal NFT to `Contract[1]`, which calls `cancel-withdraw` and assigns some pool to place their direct stake under. `Contract[1]` then sends the received 10M `stSTX` to repay the flash loan, resulting in a `10M * stx/stSTX ratio` more `STX` in direct stacking accounting.

This scenario can also be manipulated to create an arbitrary number of addresses holding an invalid direct stacked amount, increasing the difficulty of counteracting this inflation through `update-direct-stacking` for the protocol.

CONTENTS

Recommendation

The only current safeguard against the described issue is to pass a governance proposal to disable the protocol, which would lead to a denial of service (DoS) for all protocol users.

Consider adding an `original-creator` attribute to the `withdrawals-by-nft` map, and then using it in `cancel-withdraw` to deduct the `stx-amount` from their balance instead of from the `withdraw`'s `tx-sender`.

```
{
        unlock-burn-height: uint,
        stx-amount: uint,
        ststx-amount: uint,
+       original-creator: principal
}
```

Stacking DAO Team: Behavior is expected by design. The current implementation relies on an offchain script.

CONTENTS

[M-04] `update-direct-stacking` Lacks Adequate Access Control

Description

The `update-direct-stacking` function in `direct-helpers-v1` currently does not have any access control logic inside of it, which means that it can be called by anyone. Furthermore, it also doesn't have any validation on its `protocols` input argument. What those two things combined together will lead to is that anyone will be able to call this function for any given `user` with improper input data (`protocols` list with duplicate entries), in turn, making it possible to artificially reduce the direct staking data of all users.

This can be used to change the directly staked STX of any user to normally staked STX without their permission, which will make it possible for anyone to tamper with the STX distribution proportions for any future PoX cycle at any moment.

Recommendation

Add access control to the `update-direct-stacking` function and/or add input validation on its `protocols` input parameter

CONTENTS

8.3. Low Findings

[L-01] ststx-withdraw-nft Does Not Fully Comply with SIP-9

Description

The `ststx-withdraw-nft` ststx-withdraw-nft contract is designed to implement `SIP-9` as it serves as an NFT withdrawal receipt for a specific amount of stSTX. However, the contract's `get-last-token-id` function currently returns `last-id`, which is then used as the token ID for minting during the next withdrawal request. This is problematic because `get-last-token-id` should return the ID of the last minted token, not the next one to be minted. According to the standard's definition:

> Takes no arguments and returns the identifier for the last NFT registered using the contract. The returned ID can be used as the upper limit when iterating through all NFTs.

Recommendation

It is recommended to modify the function so that it returns `last-id - 1` when `last-id > 0`, and `none` otherwise. This adjustment will ensure that the function returns the last minted token ID, aligning with the standard's definition.

CONTENTS

[L-02] Potential Read-Only Reentrancy When Buying NFT

Description

The `list-in-ustx` function allows a `<commission-trait>` contract to be specified during the listing process. This arbitrary contract is then invoked after the buyer sends the funds but before the NFT is transferred or the listing is deleted:

```
(try! (contract-call? commission-contract pay id price))
```

This sequence permits the seller to receive a callback with an updated balance before other states are cleared, potentially leading to a form of read-only reentrancy. Although exploiting this issue would require a very specific scenario involving other code that directly integrates and reads state from `ststx-withdraw-nft`, it remains a concern worth noting.

Recommendation

Consider executing the `<commission-trait>.pay` call after the NFT has been transferred and the listing state has been updated.

CONTENTS

[L-03] Vulnerability to Sandwich Attacks in `buy-in-ustx`

Description

The protocol permits stSTX withdrawal receipts to be sold and "cashed-out" at any chosen price in STX before the maturation timestamp. The issue arises because the `buy-in-ustx` function lacks a `price` parameter, leading to the NFT being purchased at the current offer price. This creates a vulnerability where the seller can frontrun the buyer by re-listing the NFT at a higher price, causing the buyer to overpay. Currently, this risk is only partially mitigated by post-conditions that may be implemented on the frontends.

Recommendation

It is advisable to add a `price` parameter to the `buy-in-ustx` function and ensure it matches the current price through an assertion.

CONTENTS

[L-04] Use contract-caller instead of tx-sender for admin actions

Description

The function `ststx-token.set-token-uri` currently uses `tx-sender` to authorize an admin action. This is inconsistent with most other admin actions in the protocol, which utilize `contract-caller`.

Recommendation

It is recommended to use `contract-caller` instead of `tx-sender`.

```
(try! (contract-call? .dao check-is-protocol contract-caller))
```

CONTENTS

[L-05] Incorrect Commission Rounding

Description

The `commission-amount` is calculated using a round-down precision, which results in the protocol not aligning with itself during reward distribution calculations.

```
(commission-amount (/ (* stx-amount commission) DENOMINATOR_BPS))
```

Recommendation

Consider rounding up for the protocol in the calculation above.

CONTENTS

ClarityAlliance
Security Review
Stacking DAO

8.4. QA Findings

[QA-01] Sale of Matured Withdrawals

Description

The `buy-in-ustx` function does not verify if the NFT being purchased has matured. This oversight permits the sale of matured NFTs, potentially leading to unintended consequences.

Recommendation

Implement a verification step in `buy-in-ustx` to confirm that the NFT being purchased has not yet matured.

CONTENTS

[QA-02] English Dialect Inconsistencies

Description

A common best practice is to use a single language and dialect to ensure consistency, readability, and maintainability. Within the codebase, there is an instance where both British and American dialects are used.

The British spelling “authorised” with an “s” is used in `strategy-4.clar::ERR_UNAUTHORISED`, while the American spelling "authorized" with a "z" is used in the `ststx-token` contract.

Recommendation

Consider maintaining consistency by using only American English.

CONTENTS

[QA-03] Returning Response Types in Read-Only Functions is an Antipattern

Description

Currently, some read-only functions within the protocol return `Response` types instead of direct values. For example, in `ststx-withdraw-nft.clar` :

```
(define-read-only (get-base-token-uri)
  (var-get base-token-uri)
)
```

```
(define-read-only (get-last-token-id)
  (ok (var-get last-id))
)
```

This approach is redundant and can cause confusion when unwrapping the results.

Recommendation

Consider returning direct values instead of `Response` types in all read-only functions.

CONTENTS

[QA-04] Add Conditional Error Handling

Description

The `unlist-in-ustx` function currently succeeds even when a listing for a specific NFT does not exist.

Recommendation

Modify the function to include the following line, which addresses the semantic issue:

```
(try! (is-eq (map-get? market id) true) (err ERR_NO_LISTING))
```

CONTENTS

[QA-05] Transfer Function Simplification

Description

The `ststx-token.transfer` function currently includes a redundant `match` statement, which complicates the logic and results in suboptimal gas usage.

Recommendation

Consider rewriting the function as follows:

```
(define-public (transfer (amount uint) (sender principal)
  (recipient principal) (memo (optional (buff 34))))
  (begin
    (asserts! (is-eq tx-sender sender) (err ERR_NOT_AUTHORIZED))

    (try! (ft-transfer? ststx amount sender recipient))
    (print memo)
    (
      print{action:"transfer",
      data:{sender:tx-sender,
      recipient:recipient,
      amount:amount,
      block-height:block-height}}
    )
    (ok true)
  )
)
```

CONTENTS

[QA-06] Return Type Restructuring

Description

The `withdraw` function currently returns the `stx-amount` as a single value. This can be misleading because it does not reflect the fee that has been deducted.

Recommendation

It is recommended to return `(ok (stx-user-amount, stx-fee-amount))` instead of `(ok stx-amount)`.

CONTENTS

[QA-07] Use Constant for BPS Across All Contracts

Description

Currently, there are several instances where 100% in BPS is hardcoded instead of utilizing the defined `DENOMINATOR_BPS` constant.
Example:

```
(stx-fee-amount (/ (* (get-unstack-fee) stx-amount) u10000))
```

Recommendation

It is advisable to define BPS as a constant across all relevant contracts and replace all hardcoded occurrences with this constant.

```
(define-constant DENOMINATOR_BPS u10000)
```

CONTENTS

ClarityAlliance
Security Review
Stacking DAO

[QA-08] NFT Listing Remains in a Corrupted State After Withdrawal Completion

Description

When the `stacking-dao-core-v2.withdraw` function is called, the withdrawal receipt NFT is burned. However, its listing remains, resulting in an unnecessary state.

Recommendation

It is advisable to include a `unlist-in-ustx` call before burning the NFT in the `stacking-dao-core-v2.withdraw` function to ensure the NFT listing is removed.