ClarityAlliance

HERMETICA USDh (UPGRADE) SECURITY REVIEW

Conducted by:
KRISTIAN APOSTOLOV, ALIN BARBATEI (ABA)

SEPTEMBER 4TH, 2025

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

Hermetica USDh
(Upgrade)

CONTENTS

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 Hermetica USDh protocol, where Clarity Alliance reviewed the scope and provided insights on improving the protocol.

4. About Hermetica USDh

Hermetica's USDh is the first Bitcoin-backed synthetic dollar that yields up to 25%.

The Hermetica protocol couples spot BTC with a short perpetual futures position to create a synthetic dollar that is native to Bitcoin L1 and L2s.

Staked USDh, a Bitcoin backed, yield instruments accrues daily yields from perpetual futures funding rates.

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

Scope

The security review covered all updates to the Clarity smart contracts located in the repository:
https://github.com/hermetica-fi/hermetica-contracts

In scope were the following contracts:

- `contracts/protocol/staking-reserve-v1.clar`
- `contracts/protocol/staking-silo-v1-1.clar`
- `contracts/protocol/staking-state-v1.clar`
- `contracts/protocol/staking-v1-1.clar`

Initial Commit Reviewed:
1b9a84f91204ed326952d0d1e0adc464df5c7a52

Final Commit After Remediations:
b33b31af526b74daecf31f87825f46270bbdffb5

CONTENTS

7. Executive Summary

Over the course of the security review, Kristian Apostolov, Alin Barbatei (ABA) engaged with - to review Hermetica USDh. In this period of time a total of **7** issues were uncovered.

Protocol Summary

Protocol Name	Hermetica USDh
Date	September 4th, 2025

Findings Count

Severity	Amount
High	1
Low	1
QA	5
Total Findings	**7**

ClarityAlliance
Security Review

Hermetica USDh (Upgrade)

CONTENTS

Summary of Findings

ID	Title	Severity	Status
[H-01]	Design Flaw in Different Staking Contract Versions Skews USDh/sUSDh Conversion Ratio	High	Resolved
[L-01]	Staking State Contract Authorization Ambiguities	Low	Resolved
[QA-01]	Claims with Zero Tokens Should Not Be Valid	QA	Resolved
[QA-02]	Redundant Tuples with Single Element as Map Key or Value	QA	Aknowledged
[QA-03]	Lack of Affiliate Validation	QA	Aknowledged
[QA-04]	Staking Cannot Be Paused Separately From Unstaking	QA	Resolved
[QA-05]	Unstaking Process Can Be Simplified	QA	Resolved

CONTENTS

8. Findings

8.1. High Findings

[H-01] Design Flaw in Different Staking Contract Versions Skews USDh/sUSDh Conversion Ratio

Description

The newly introduced staking contracts are designed to reuse the existing `sUSDh` token contract, `susdh-token-v1`, for calculating the `USDh/sUSDh` conversion ratio. This design introduces a significant flaw, as both the soon-to-be-deprecated staking contracts (`staking-v1` and `staking-silo-v1`) and the new staking contracts (`staking-v1-1`, `staking-state-v1`, `staking-silo-v1-1`, and `staking-reserve-v1`) rely on the same underlying `susdh-token-v1` token supply for this calculation. However, these contracts are not aware of each other's actual staked USDh tokens.

In the old v1 contracts, the ratio is retrieved via a `staking-v1::get-usdh-per-susdh` call, calculated as the USDh staked balance of the `staking-v1` contract itself divided by the total supply of minted `sUSDh` tokens.

```
(define-read-only (get-usdh-per-susdh)
  (let (
    (total-usdh-staked (unwrap-panic
      (contract-call? .usdh-token get-balance .staking)))
    (total-susdh-supply (unwrap-panic
      (contract-call? .susdh-token get-total-supply)))
  )
    (if (and (> total-usdh-staked u0) (> total-susdh-supply u0))
      (/
        (*
         total-usdh-staked
         usdh-base
        )
        total-susdh-supply
      )
      usdh-base
    )
  )
)
```

In contrast, the new v1.1 staking contract uses a similar `staking-v1-1::get-usdh-per-susdh` function to retrieve the ratio, calculated as the USDh staked balance of the `staking-reserve-v1` contract divided by the total supply of minted `sUSDh` tokens.

```
(define-read-only (get-usdh-per-susdh)
  (let (
    (total-usdh-staked (unwrap-panic
      (contract-call? .usdh-token get-balance .staking)))
    (total-susdh-supply (unwrap-panic
      (contract-call? .susdh-token get-total-supply)))
  )
    (ok (if (and (> total-usdh-staked u0) (> total-susdh-supply u0))
      (/
        (*
          total-usdh-staked
          usdh-base
        )
        total-susdh-supply
      )
      usdh-base
    ))
  )
)
```

Both staking contracts consider the total staked `USDh` amounts as different contract balances but use the same underlying minted staked token, `sUSDh`. The two staking contract versions cannot function simultaneously, as depositing in one increases the `sUSDh` total supply for both, resulting in different conversion ratios being used in each.

This discrepancy can lead to user fund losses, as the staking/unstaking ratio is not synchronized with the actual staked balance. In theory, if all market participants exit the old staking contract and migrate to the new one, the issue would be resolved. However, this is not realistically feasible. The old staking contract currently holds a balance of `2,168,542 USDh` tokens considered staked. There will inevitably be cases of idle users who have left the markets, lost access to their wallets, or other scenarios that will leave significant amounts of staked `USDh` tokens.

Recommendation

To mitigate the issue during product deprecation and migration, create a separate `staking-migration` contract/proposal with `USDh` minting/burning rights. During the upgrade:

1. Remove the `staking-v1` contract from being a protocol contract to prevent new users from joining.
2. Deploy the new staking contracts, but do not set them as protocol contracts until funds are migrated. This ensures no users enter with a skewed conversion ratio.
3. Deploy and execute a `staking-migration` contract to burn the `staking-v1` contract's `USDh` balance and mint it to the new `staking-reserve-v1` contract (equivalent to moving funds).
4. Set the new staking contracts as protocol contracts to allow staking to resume without issues.

For future contract versions, if the new design is maintained, it will only be necessary to deactivate/activate the old/new contract versions, as long as the same staking reserve contract is retained.

Note: The development team was aware of this issue and had a migration routine in place that resolves it.

8.2. Low Findings

[L-01] Staking State Contract Authorization Ambiguities

Description

The `staking-state-v1` contract includes two permissioned setter functions, each employing a distinct method for authorization:

- `set-cooldown-window` :This function uses the `hq:: check-is-protocol` validation on the `tx-sender`, ensuring the contract is invoked by a smart contract associated with the protocol.
- `set-custom-cooldown` :This function uses the `hq::check-is-admin` validation on the `tx-sender`, ensuring the contract is called by one of the protocol's administrators.

Any contract verification that expects to be called from a protocol contract should be validated using `contract-caller` rather than `tx-sender`. It is generally recommended that all validations be performed using `contract-caller`, if the deployment strategy allows.

By relying on `tx-sender`, there is a risk that admins or approved principals who fall victim to phishing scams and interact with malicious contracts could inadvertently engage with the codebase and execute sensitive operations.

For instance, a trusted address might interact with a malicious contract, which could then set the withdrawal cooldown to 0, allowing anyone to unstake without a waiting period.

It is important to note that the above scenario cannot occur without administrative negligence.

Recommendation

If `set-custom-cooldown` is intended to be called from a smart contract context, modify it to use `hq::check-is-protocol` instead of `hq::check-is-admin`. Additionally, use `contract-caller` instead of `tx-sender` in the `staking-state-v1` contract.

CONTENTS

8.3. QA Findings

[QA-01] Claims with Zero Tokens Should Not Be Valid

Description

When an unstake request is executed via `staking-v1-1::unstake`, edge cases involving dust unstaking can lead to a 0 amount `USDh` conversion due to rounding issues in the operation:

```
(amount-usdh (/ (* amount ratio) usdh-base))
```

This process further creates a claim through `staking-silo-v1-1::create-claim`, resulting in a valid claim with a 0 amount. The unstake call itself would then revert in this scenario because it attempts to transfer 0 `USDh` tokens, triggering a generic `(err u3) -- amount to send is non-positive` error, which can cause confusion for integrating projects.

Recommendation

Implement a check when creating a claim via `staking-silo-v1-1::create-claim` to ensure that the claim amount is greater than 0.

CONTENTS

ClarityAlliance
Security Review

Hermetica USDh (Upgrade)

[QA-02] Redundant Tuples with Single Element as Map Key or Value

Description

The codebase contains several instances where tuples with a single element are used in maps, both as keys and values:

- In `staking-silo-v1-1`, the `claims` map uses a tuple containing only a `claim-id` entry, which could be replaced with a direct `uint` key.
- In `staking-state-v1`, `custom-cooldown` map uses a tuple with a single element for both the key (a `principal` entry and the value (a `cooldown-window` entry). This can be replaced with a map of principal to uint.

Recommendation

Replace the tuples with the direct, single-entry element type in the specified instances.

CONTENTS

[QA-03] Lack of Affiliate Validation

Description

The new stacking contract includes an optional `affiliation` parameter, which can be up to 64 `buff` in length.

Currently, there is no validation performed on this parameter if it is provided. This lack of validation allows for the possibility of arbitrary input lengths and spam values being submitted.

Although a true validation cannot be realistically achieved without an on-chain affiliate registry, it is advisable to check the length of the affiliate parameter if it is provided. This would ensure a standardized length, thereby maintaining consistency for off-chain monitoring systems.

Recommendation

In the `staking-v1-1::stake` function, if the `affiliate` parameter is provided, verify that it meets the expected length as defined by the backend (e.g., exactly 64).

CONTENTS

[QA-04] Staking Cannot Be Paused Separately From Unstaking

Description

In the new staking contract, Hermetica lacks the ability to disable staking independently while still permitting unstaking. Currently, by removing the staking contract's principal from the protocol's approved contracts, both staking and unstaking are disabled simultaneously.

However, if there is a need to prevent new users from staking while allowing existing stakers to exit-such as when preparing for a migration or planning to disable staking entirely in the future-there is no option for such specific permission control.

Recommendation

In the `staking-state-v1` contract, introduce a `staking-enabled` variable that can be set by the protocol. This flag should be checked in the `staking-v1-1::stake` function to ensure that staking operations are permitted.

CONTENTS

[QA-05] Unstaking Process Can Be Simplified

Description

When initiating an unstaking process through the `staking-v1-1::unstake` function, a claim is generated by invoking the `staking-silo-v1-1::create-claim` function.

For external integrations, the newly created claim ID is returned by the `unstake` function. However, the `create-claim` function itself does not return the new claim ID. Instead, within the `unstake` function, it is redundantly pre-calculated:

```
(claim-id (+ u1 (contract-call? .staking-silo get-current-claim-id)))
```

This results in an unnecessary, redundant silo call with each unstake operation.

Recommendation

Revise the `staking-silo-v1-1::create-claim` function to return the claim ID directly, and have the `staking-v1-1:: unstake` function return this ID, eliminating the need for local recalculation.