ClarityAlliance

HERMETICA USDh SMART CONTRACT AUDIT

Conducted by:
KRISTIAN APOSTOLOV, 0X3B, STORMY

JUNE 17TH, 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.

CONTENTS

ClarityAlliance
Security Review

Hermetica USDh

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

Hermetica USDh

3. Introduction

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

4. About Hermetica USDh

USDh - A synthetic dollar consisting of Bitcoin coupled with a short futures position.
sUSDh - A bitcoin-native bond that generates up to 25% yield from funding rates.

USDh and sUSDh will be issued on Bitcoin L1 via the Runes protocol as well as several Bitcoin L2s, starting with Stacks. Once released, USDh will be available to buy on popular DeFi markets like Magic Eden and Alex. To access the yield, users can stake USDh and immediately receive the yield-bearing token, sUSDh.

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

Hermetica USDh

6. Security Assessment Summary

Review Commit Hash:
83c3cad999b6797e68527ecbccbba34b2bdd2611

Scope

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

- `contracts/traits/sip-010-trait.clar`
- `contracts/protocol/controller-v0-1.clar`
- `contracts/protocol/staking-silo-v0-1.clar`
- `contracts/protocol/minting-v0-1.clar`
- `contracts/protocol/reserve-v0-1.clar`
- `contracts/protocol/minting-otc-v0-1.clar`
- `contracts/protocol/hq-v0-1.clar`
- `contracts/protocol/insurance-fund-v0-1.clar`
- `contracts/protocol/redeeming-reserve-v0-1.clar`
- `contracts/protocol/staking-reserve-v0-1.clar`
- `contracts/protocol/recover-v0-1.clar`
- `contracts/protocol/blacklist-susdh-v0-1.clar`
- `contracts/protocol/staking-v0-1.clar`
- `contracts/tokens/susdh-token-v0-1.clar`
- `contracts/tokens/usdh-token-v0-1.clar`

CONTENTS

7. Executive Summary

Over the course of the security review, Kristian Apostolov, 0×3b, Stormy engaged with Hermetica to review Hermetica USDh. In this period of time a total of **15** issues were uncovered.

Protocol Summary

Protocol Name	Hermetica USDh
Repository	https://github.com/hermetica-fi/hermetica-usdh-contracts
Date	June 17th, 2024
Protocol Type	Delta-Neutral Stablecoin
SLOC	1716

Findings Count

Severity	Amount
Critical	1
High	1
Medium	5
Low	5
QA	3
Total Findings	**15**

CONTENTS

Summary of Findings

ID	Title	Severity	Status
[C-01]	Reversed slippage check renders confirm-redeem unusable	Critical	Resolved
[H-01]	First depositor attack	High	Resolved
[M-01]	reset-mint-window doesn't perform enough validation	Medium	Resolved
[M-02]	Fractional fee structure is not practical	Medium	Resolved
[M-03]	Missing last-oracle-timestamp update in reset-mint-window	Medium	Resolved
[M-04]	Can create mint requests guaranteed to fail	Medium	Resolved
[M-05]	Mint limit DoS	Medium	Resolved
[L-01]	mint-limit-reset-window off-by-one-error	Low	Resolved
[L-02]	Ownership of hq can be permanently lost due to missing safe guard	Low	Resolved
[L-03]	recover-susdh always reverts on a full blacklist when blacklist-enabled is false	Low	Resolved
[L-04]	Missing functionality to mutate soft-blacklist-enabled	Low	Resolved
[L-05]	Soft blacklist bypassed by Sybil transferring sUSDh	Low	Acknowledged
[QA-01]	Redundant logic in minting-otc-v1	QA	Resolved
[QA-02]	confirm-mint and confirm-redeem don't assert asset status	QA	Resolved
[QA-03]	Remove unnecessary variables	QA	Resolved

CONTENTS

8. Findings

8.1. Critical Findings

[C-01] Reversed slippage check renders `confirm-redeem` unusable

Location: minting-v1.clar#L499

Description

Both the minting and redeeming flows allow the user to set a slippage tolerance percentage represented in BPS. That value gets used as the max percentage deviation in the confirmation call.

```
(slippage-tolerance (/ (* price-requested
(get slippage mint-request)) bps-base))
```

The issue here arises due to the bellow slippage check in `confirm-redeem`. It asserts that the price of the asset, which we are redeeming `>=` the requested price.

Thus, the above will revert any `confirm-redeem` executes the user with a price they are comfortable with, and even proceed with prices, which will damage the user financially.

Recommendation

Consider changing the assert in `confirm-redeem` to the following:

```
(asserts! (>= price
(+ price-requested slippage-tolerance)) ERR_SLIPPAGE_TOO_HIGH)
```

CONTENTS

8.2. High Findings

[H-01] First depositor attack

Location: staking-v1.clar#L41

Description

`staking-v0-1::stake` is currently vulnerable to a first depositor attack, where a malicious actor front-runs the first staker and steals 25% of their deposit. This attack vector exploits Clarity's round-down division behavior, where any decimal part of the division result is truncated.

Example:

- The staking vault is empty; no sUSDH has been minted yet

1. Bob stakes 10,000e8 USDh
2. Alice front-runs 1. by staking 1 USDh, receiving 1 sUSDh in return
3. Alice transfers 5,000e8 USDh to the reserve
4. When minting sUSDh, Bob receives only 1 share, worth 7500e8 USDh (not 10000e8); his share output gets truncated
5. Now, there are 2 shares and 15,000e8 USDh in the vault. The Alice can unstake her share and get 7,500e8 in return

Recommendation

Consider minting a certain amount of shares to a dead address during contract deployment (similar to Uniswap V2's implementation). One share (1e8) will be more than enough.

CONTENTS

ClarityAlliance
Security Review

Hermetica USDh

8.3. Medium Findings

[M-01] `reset-mint-window` doesn't perform enough validation

Location: minting-v1.clar

Description

Every time a request is made for minting or redeeming, the system checks the publish time of the decoded price and ensures it is not stale based on `last-oracle-timestap`. `reset-mint-window` is used by a timestamper account to reset `last-mint-limit-reset` .

Currently, the function only checks for staleness based on `last-mint-limit-reset` , which can differ significantly from the current timestamp.

Recommendation

`reset-mint-window` , ensure stale data is not is use by checking for staleness based on the value of `last-oracle-timestap` .

CONTENTS

[M-02] Fractional fee structure is not practical

Location: minting-v1.clar#443 minting-v1.clar#499

Description

Currently, the protocol takes 0.1999% in fees when completing a mint/redeem action. It is broken down into the following:

- mint: 0.1% collateral token fee, 0.1% out of 99.9% = 0.999% out of the USDh minted.
- redeem: 0.1% of the burnedUSDh, 0.1% out of 99.9% = 0.999% out of the collateral token sent to the user.

The issue arises due to there not being a minimum fee amount taken. Since the protocol needs to execute a separate `confirm-[action]` call afterwards, the fees taken should be able to cover for the cost of executing the call. Thus, the current implementation allows for bad actors to submit dust amount actions, in order to slowly drain the trader account out of it's STX reserve, and subsequently DoS it.

Recommendation

Consider implementing a minimum fee, which should at the least be able to cover for transaction costs.

CONTENTS

[M-03] Missing `last-oracle-timestamp` update in `reset-mint-window`

Location: minting-v1.clar#L680

Description

`reset-mint-window` allows a system account to update `current-mint-limit` based a Pyth VAA's timestamp.

The issue arises due to the function only updating `last-mint-limit-reset`, and not `last-oracle-timestamp`. This will allow for mint/redeem requests with Pyth data from before the current minting epoch to be created.

Recommendation

Consider adding the following snippet to `reset-mint-window`:

```
;; reset-mint-window...
(var-set last-oracle-timestamp timestamp) ;; <- @audit
(ok true)
```

CONTENTS

[M-04] Can create mint requests guaranteed to fail

Location: minting-v1.clar#L295

Description

`request-mint` and `request-redeem` both have a `slippage` parameter, which allows the user to set their tolerance for any price deviation that might occur between creating a request and it being processed.

The issue arises due to `slippage` not being verified before getting set in the `mint-requests` entry. This allows requests with >100% slippage to be created.

Any such request confirmations will be reverted due to underflow.

```
(asserts! (>= price
(- price-requested slippage-tolerance)) ERR_SLIPPAGE_TOO_HIGH)
```

Recommendation

Consider implementing proper validation on the `slippage` parameter in both `request-mint` and `request-redeem`.

```
(asserts! (slippage <= bps-base) <ERROR>)
```

CONTENTS

[M-05] Mint limit DoS

Location: minting-v1.clar

Description

The system subtracts from `current-mint-limit` inside of `request-mint` , though some of those created requests will be canceled or unconfirmed.

The above-mentioned logic enables bad actors with access to vast capital to DoS minting by submitting requests no slippage tolerance. Since having zero price movement in the time between creating a request and getting it fulfilled, it is highly unlikely that any such request will be confirmed.

Thus, `current-mint-limit` will be decremented without any minting happening.

Recommendation

Consider implementing a mechanism where `current-mint-limit` incremented until reaching `current-mint-limit` in `cancel-unconfirmed-mint` .

8.4. Low Findings

[L-01] `mint-limit-reset-window` off-by-one error

Location: minting-otc-v1.clar#L209

Description

`confirm-mint` uses `burn-block-height` check to ensure no more than the `mint-limit` can be minted every `n` blocks. However, the current if condition uses `>` instead of `>=`, which effectively increases `mint-limit-reset-window` from six to seven blocks.

Example:

- Current block: 100

1. Block 106: `106 > 100 + 6 -> false`
2. Block 107: `107 > 100 + 6 -> true` ; set `last-mint-reset-window` to `107`

Effectively using seven blocks as our window results in 16.67% less throughput compared to using six blocks.

Recommendation

Consider using `>=` instead of `>`, to more accurately check for the `mint-limit-reset-window` .

CONTENTS

[L-02] Ownership of `hq` can be permanently lost due to missing safe guard

Location: hq-v1.clar#L220-L226

Description

There is no safeguard in `remove-owner` for an owner to remove themselves and leave the system without owners.

Mistakenly calling this function will lead to a permanent DoS of the following functions:

- `set-minting-enabled`
- `request-minting-contract-update`
- `remove-minting-contract`
- `activate-minting-contract`

Recommendation

Add a safeguard to allow a call to `remove-owner` only when the `activate-owner` function was previously set, and reset the safeguard after `remove-owner` is called.

CONTENTS

ClarityAlliance
Security Review

Hermetica USDh

[L-03] `recover-susdh` always reverts on a full blacklist even when `blacklist-enabled` is `false`

Location: recover-v1.clar#L26-L39

Description

The `transfer` function in sUSDh performs a full blacklist check on an address only when `blacklist-enabled` is `true`.

However, this is not the case in `recover-susdh`. Even when `blacklist-enabled` is `false`, the system will still revert on a full blacklist on both addresses when when the function is called.

Recommendation

Check for full blacklist only when `blacklist-enabled` is set to `true`.

CONTENTS

[L-04] Missing functionality to mutate `soft-blacklist-enabled`

Location: blacklist-susdh-v1.clar#L15 blacklist-susdh-v1.clar#L87-L95

Description

There is an if statement in `check-is-not-soft-blacklist` which checks whether `soft-blacklist` is `true`. If so, it ensures that the user is not soft blacklisted.

```
(if (get-soft-blacklist-enabled)
    (asserts! (not (get-soft-blacklist contract)) ERR_SOFT_BLACKLISTED)
    true
)
```

Though notice how `soft-blacklist-enabled` is fixed to `true` on deployment. There is no logic implemented to mutate this variable, making the if statement in `check-is-not-soft-blacklist` irrelevant, as the soft blacklist feature is always enabled.

Recommendation

Implement functionality to mutate `soft-blacklist-enabled`.

CONTENTS

[L-05] Soft blacklist bypassed by Sybil transferring sUSDh

Location: staking-v1.clar#L43-L74 susdh-token-v1.clar

Description

`stake` and `unstake` ensure that a user is not soft blacklisted; otherwise, those functions are temporarily disabled for them.

The issue arises because `transfer` in sUSDh reverts if either the sender or the recipient is fully blacklisted. However, there is no assertion for a soft blacklist.

As a result, the security check preventing soft blacklisted users from staking or unstaking can easily be bypassed by transferring the tokens to another account.

Recommendation

Consider implementing a soft blacklist assertion in `transfer`.

CONTENTS

8.5. QA Findings

[QA-01] Redundant logic in `minting-otc-v1`

Location: minting-otc-v1.clar#L48

Description

The system currently implements logic for changing the asset mint commission in OTC, however the functionality is not used anywhere across that contract.

Recommendation

Remove the following declarations:

- `mint-commission-asset`
- `get-mint-commission-asset`
- `set-mint-commission-asset`

CONTENTS

[QA-02] `confirm-mint` and `confirm-redeem` don't assert asset status

Location: minting-v1.clar#L471-L499 minting-v1.clar#L529-L557

Description

When a requesting mint or redeem, the system makes sure that the given asset is supported by being active.

However this check is missing when confirming a mint or redeem, as a result, the mentioned functions can be evoked with an inactive asset.

Recommendation

Add a an activity assertion for the asset in `confirm-mint` and `confirm-redeem` .

CONTENTS

[QA-03] Remove unnecessary variables

Location: minting-v1.clar

Description

Variables used only once can be inlined to save gas. For example, in `minting-v1::confirm-mint`, the `amount-requested` variable is initialized but only used once inside an assert statement.

It's also recommended to remove variables that are not used at all. Examples include `amount-usdh-requested` and `amount-asset-requested` in the mint and redeem mappings. These values are already emitted, and if they are not used for any off-chain purposes, they can be removed to save gas.

Recommendation

Inline variables that are used only once and remove all unnecessary code.