dMakers

Public report

dMakers

Security audit of HAI protocol

Foreword

Status of a vulnerability or security issue

Clarity is a rare commodity. That is why for the convenience of both the client and the reader, we have introduced a system of marking vulnerabilities and security issues we discover during our security audits.

No issue

Let's start with an ideal case. If an identified security imperfection bears no impact on the security of our client, we mark it with the label.

Fixed

The fixed security issues get the label that informs those reading our public report that the flaws in question should no longer be worried about.

Addressed

In case a client addresses an issue in another way (e.g., by updating the information in the technical papers and specification) we put a nice tag right in front of it.

Acknowledged

If an issue is planned to be addressed in the future, it gets the tag, and a client clearly sees what is yet to be done. Although the issues marked "Fixed" and "Acknowledged" are no threat, we still list them to provide the most detailed and up-to-date information for the client and the reader.

Severity levels

We also rank the magnitude of the risk a vulnerability or security issue pose. For this purpose, we use 4 "severity levels" namely:

1. Minor
2. Medium
3. Major
4. Critical

More details about the ranking system as well as the description of the severity levels can be found in Appendix 1. Terminology

Table of contents

Introduction

Scope of work

I. Security Audit and Review

1.1 Conduct a comprehensive security review and audit of existing smart contracts to identify and mitigate potential vulnerabilities and risks. The source code is located at https://github.com/hai-on-op/hai/tree/v0.1.2-rc.2.
Auditor will focus on the contracts folder, the rest of the repository will be used to get a better understanding of the system.

1.2 Provide recommendations and guidance for enhancing security based on audit findings in the form of a report.

1.3 Verify fixes of the audit findings.

II. Consulting Services

2.1 Provide consultation and mentorship services to developers of the team, fostering a culture of learning and continuous improvement.

2.2 Advise on best practices, emerging trends, and innovative solutions in the blockchain and DApp development space.

Security assessment methodology

The smart contract's code is scanned both manually and automatically for known vulnerabilities and logic errors that can lead to potential security threats.

The conformity of requirements (e.g., White Paper) and practical implementations are reviewed as well on a consistent basis.

Discovered issues

Findings:

Critical issues

- Unauthorized removing safe using LiquidationEngine.protectSAFE
    - Description: Attacker can remove saviour from other user safe by passing zero address as _saviour.
    - Status: Fixed Link

Major issues

- Surplus Auction reverts if started with _initialBid != 0
    - Description: If _initialBid is not 0, then this line will revert because the initialBid is assigned to the msg.sender up on auction creation however the startAuction method does not transfer protocol token from the creator. Consider validating the _initialBid properly. The same issue exists in the PostSettlementSurplusAuctionHouse.
    - Status: Fixed Link
- DOS using chosenSAFESaviour.
    - Description: There could be a Saviour that always leads to this revert. The reason could be a bug or an attack.
    - Status: Fixed Link
- Sweep vs withdraw
    - Description: Since isAutorized allows any authorized account to call the withdraw, then sweep does not make sence.
    - Status: Fixed Link

Medium issues

- HaiProxy does not validate the target contract
    - Description: HaiProxy.sol: delegateCall does not revert of contract does not exist. Consider reference implementation.
    - Status: Fixed Link

Discovered issues

- Reduntant approve.
  - Description: The CoinJoin contract does not need approve to burn.
  - Status: Fixed Link
- SystemCoin does not have permit functionality.
  - Description: Consider adding the permit functionality to allow users to enjoy Gasless approve functionality.
  - Status: Fixed Link
- HaiProxyRegistry bypass.
  - Description: HaiProxyFactory.build can be called directly, so HaiProxyRegistry wont have the proxy address.
  - Status: Fixed Link
- UniV3Relayer.
  - Description: Consider requiring slot0.observationCardinality > quotePeriod / 13 seconds to check that Uniswap pool has enough history of swaps. To increase it call the increase0bservationCardinality Next method.
  - Status: Acknowledged
  - Comment: The team and DAO will do check that offchain.
- DOS by abusing openSAFE functionality
  - Description: openSAFE method allows to open safe on behalf of other user. Also it's possible to open safe for yourself and transferSAFEOwnership o another user. These two functionalities could be used to overload the victim UI and get advantage of it (force a user to a safe liquidation for example).
    To prevent possible spam the UI should filter the safes by (1) collateral value (2) opened for yourself, (3) opened by others, (4) opened by others and sent to you. Services like theGraph allow to reduce load on the UI.
  - Status: Acknowledged
  - Comment: Will be fixed on UI level.
- Using hardcoded nonce
  - Description: "using 0 nonce" could be avoided. draft-IERC20Permit.sol has nonces(_owner) method.
  - Status: Fixed Link

Discovered issues

- CREATE2 instead of CREATE
  - Description: Using create2 may benefit users if there will be deployment on different networks. Accidentially sent coins for example or to avoid potential front-run issues.
  - Status: Acknowledged
  - Comment: Added target address to Salt

- Encoding.sol does not check data.length.
  - Status: Acknowledged
  - Comment: For a future release

- DelayedOracle.Feed is always valid. Even when priceSource.getResultWithValidity() returns false. Consider writing Feed(_priceFeedValue, **false**);
  - Status: Fixed Link

Minor issues

- Own implementation of Ownable.sol instead of reusing standards
  - Description: consider using Openzeppeling reference implementation for the utils/Ownable.sol
  - Status: Fixed Link

- Inconsistent type cast approach.
  - Description: This cast can be implemented using Math.toInt() for consistency.
  - Status: Fixed Link

- Usage uint256 instead of bool
  - Description: consider using the Bool type for safeSaviours mapping.
  - Status: Fixed Link

- Sending tokens before adjusting storage
  - Description: Consider delete _auctions[_id]; before sending tokens. 1,2.(Another auctions as well)
  - Status: Fixed Link

Discovered issues

- Lack of address validation
  - Description: consider address assertNonNull for the delegatee address.
  - Status: Fixed Link
- Usage uint256 instead of bool
  - Description: consider using bool type for the _ok param of the safeCan mapping.
  - Status: Fixed Link
- Depricated functionality
  - Description: ETHJoin is depricated. The system uses WETH, so consider removing the contract from repository.
  - Status: Fixed Link
- _validateClaim can reuse the _canClaim functionality. It would look better as modifier.
  - Status: Fixed Link
- Why not Bool?
  - Status: Acknowledged
  - Comment: Not using bools in packed structs as it affects test environment (until std.store supports it)
- _validateDelayedOracle. Check magic value to validate support of specific interface.
  - Status: Fixed Link
- Redundant argument _amountToBuy
  - Status: Fixed Link
- “Deprecated” functions.
  - Description: Consider removing deprecated functionality.
  - Status: Fixed Link
- SafeEngine gas optimisation
  - Description: The SafeEngine address used in multiple contracts as Storage value (1, 2, 3 and so on). It could be marked as immutable to safe Gas to users.
  - Status: Acknowledged
  - Comment: Since the Gas it so cheap on Optimism and other reasons we are living it as is.

Security assumptions

UI ensures accurate transaction data for users

The application relies on the use of a proxy contract, specifically HAIProxy.sol, to manage and route user interactions with the underlying system.
One of the critical functions within this proxy contract, execute, takes an address _target and a bytes input _data which dictates the operation to be performed via a delegate call.

A challenge arises due to the intrinsic opacity of the _data bytes input.
In conventional transactions, where operations like approve or transfer are directly invoked, modern wallets can interpret the transaction's intent and provide user-friendly prompts or warnings. However, when transacting via the HAIProxy.sol proxy, the user relies heavily on the application's UI to craft the correct byte data, potentially obfuscating the transaction's true intent.

Users become over-reliant on the application's user interface to interpret and display the true action behind the transaction, creating a vector for phishing or malicious intent if the UI is compromised or if they are directed to a fake UI.

Deployment Procedures

Verification of Authorized Accounts:

• Distinct Authorization Practices: For each account within the HAI platform, admin/owner keys will be deliberately distinguished from deployer keys.
This differentiation ensures that there isn't a single point of vulnerability, bolstering HAI's security framework.

• Dynamic Authorization Management: Whenever HAI's contracts, especially pivotal ones like auctions and the LiquidationEngine, undergo modifications, prior authorizations will be programmatically revoked or updated. This proactive approach ensures that outdated permissions are negated, preventing potential security backdoors.

Parameter Verification:

• System Parameter Scrutiny: Prior to any mainnet deployment, HAI's developers will systematically review every system parameter. This verification process ensures that all configurations are optimized, guaranteeing both operational efficiency and heightened security.

Automated Deployment and Testing:

• Deployment Scripts: To streamline and secure the deployment process, HAI's developers utilize automated scripts. These scripts inherently follow the essential security checks, minimizing the possibility of oversights or human-induced errors.

• End-to-End Testing on Local Fork: Following deployment, HAI release is subjected to end-to-end tests on a local fork of the network used for deployment. This simulation affords the team an additional layer of assurance, validating that HAI operates seamlessly and securely in a production-like environment.

Reliance on External Job Performers

The HAI system is architecturally designed to depend significantly on external entities executing specific tasks, referred to as Jobs. These actors play a crucial role in the efficient functioning and responsiveness of the platform. The HAI team has undertaken the responsibility to ensure MEV searchers and all interested parties are fully informed about the opportunities and nuances associated with HAI's operation.

Reliance on Oracle Providers:

The security and accuracy of the HAI system are deeply connected to the reliability of the Oracles providing real-time price feeds. HAI utilizes both Chainlink oracles and the Uniswap v3 TWAP to source prices for all collaterals in the system, as well as the HAI token itself.

Just as with the external job performers, the trustworthiness and timely performance of these oracle providers are paramount. Any inaccuracies or deliberate manipulations in the price feeds can introduce vulnerabilities, potentially affecting the collateralization ratios and the overall stability of the HAI system.

Limitations of Saviours in Liquidation

The HAI system permits users to designate specific 'saviours' for their safes, intended to intervene during liquidation scenarios. However, it's crucial for users to recognize that saviours can only provide protection against abrupt fluctuations in collateral market prices. They offer no safeguard against Oracle failures. In events of Oracle malfunction, liquidators can potentially seize both the user's safe and saviour funds.

Considerations During Global Settlement

During a Global Settlement phase, processing all safes within the HAI system is of paramount importance. It's pertinent to note that for a segment of users, possibly a significant portion, invoking the processSAFE function might not be beneficial, particularly if their safe is undercollateralized. It is incumbent upon the DAO and the HAI team to manage and address this, ensuring smooth and equitable system-wide settlement.

Implications of Insufficient stabilityFeeTreasury Funds

If the stabilityFeeTreasury lacks adequate tokens for rewards, the pullFunds function will fail, resulting in the system not updating as expected. Direct interventions would then be required, specifically targeting accountingEngine, liquidationEngine, pidRateSetter, and oracleRelayer.

It's crucial to ascertain the readiness of the Job software to handle such scenarios, particularly concerning the liquidationEngine. In its absence, assets might not transition to auction. The responsibility for addressing these issues lies squarely with the HAI team and DAO.

Oracle Delay Implications

The inclusion of a 1-hour delay for the Oracle offers users a window to either exit the system or bolster their collateral. However, this delay also opens the door for potential manipulations.

An attacker, privy to market movements, can exploit this time lag. For instance, if an attacker discerns a 50% drop in collateral value that the system has yet to reflect, they might acquire this devalued collateral off the market at a cheaper rate. Post-acquisition, they can introduce this collateral into the system and artificially generate an inflated amount of HAI. Subsequently, they can trade a fraction of this HAI for more collateral and settle their debt.

While this can be viewed as mere arbitrage, it does highlight a potential avenue for exploitation rooted in the Oracle's delay mechanism.

UI Preparedness for Global Settlement Event

At the time of our audit, the HAI user interface (UI) does not accommodate the Global Settlement event.

This limitation means that in the event of such a settlement, a vast majority of users may be hindered from accessing and executing essential functions like processSAFE, prepareCoinsForRedeeming, redeemCollateral, and freeCollateral.

It is the responsibility of the HAI team to ensure that all requisite preparations are undertaken and that the UI is adequately equipped to handle a Global Settlement event seamlessly.

Airdrop Considerations for TokenDistributor.sol

In the forthcoming airdrop of governance tokens, the current design of TokenDistributor.sol does not accommodate scenarios where a single user address qualifies for multiple airdrops. As a result, one of the airdrops would be lost.

However, the HAI team has provided assurances that this limitation will be addressed during the construction of the Merkle tree for TokenDistributor.sol. By ensuring that no two leaves correspond to a single address, they intend to mitigate the risk of any user losing out on their rightful airdrop.

System Configuration and Parameter Modification

The HAI system's operational efficiency and security are underpinned by its intricate configuration across multiple smart contracts, each with multiple parameters. While the deployed version of the system is subject to comprehensive end-to-end testing, any subsequent parameter changes, potentially initiated by the DAO, necessitate exhaustive testing.

Responsibility lies both with the proposer introducing the change and the DAO overseeing it. Both entities must ensure that any parameter modification aligns with the best interests of HAI and doesn't inadvertently introduce vulnerabilities or inefficiencies.

Additionally, these checks and evaluations could be conducted off-chain or be integrated within the proposal mechanism itself, ensuring that the implications of any change are understood before its implementation.

Understanding the System Coin Limit

The HAI system incorporates a "System Coin Limit" parameter. This defines a collective cap on the total amount of coins, equivalent to debt, that can be auctioned across all collateral types. When this limit is reached, no further debt auctions can be initiated, regardless of the specific collateral type.

Purpose of the System Coin Limit:
The HAI team instituted this limit primarily as a safeguard. The key concern is a scenario in which the value of a collateral type plummets, possibly to zero. Such a drastic drop could trigger an uncontrolled issuance of the governance token, potentially causing inflationary pressures. The System Coin Limit acts as a barrier against this unbounded token production

Potential Implications and Team's Perspective:
However, this measure can present challenges. If one collateral type becomes highly volatile and consumes the entire debt limit through a multitude of auctions, it can effectively block other collaterals from being auctioned. This could hinder system liquidity and stability.

There could be a specific situation where a collateral's sharp drop in value causes all its associated auctions to fail. Such failures could fill the available space for liquidatable coins, preventing the liquidation of other collaterals. The only solutions then would be administrative interventions like raising the limit or prematurely ending all active auctions.

Dependency on DAO

The HAI system's security is intricately tied to the robustness of the DAO controlling it. If the DAO is compromised, the ramifications could be severe. A takeover of the DAO could lead to a significant loss of collateral for all users. Additionally, there's the risk of a massive inflationary surge in the HAI token.

Initial Uniswap Pool Creation

The inception of the HAI/ETH pool on Uniswap presents certain challenges, notably the "egg and chicken" conundrum. The team's approach to this challenge is to integrate the pool creation process within the deployment script, thereby automating and standardizing the procedure.

For ensuring security and stability during this critical phase, the team plans to fix the HAI redemption price to 1 USD for an initial 24-hour period. This strategy aims to offer a controlled environment for the pool's launch, minimizing potential price manipulations or volatile fluctuations.

To protect against price manipulation, it's also vital that the pool starts with strong liquidity. This helps absorb big trades and prevents drastic price changes. It's up to the project team and DAO to ensure the pool has enough funds from the beginning.

If executed correctly and monitored closely, this methodology can offer a smooth and secure start for the HAI/ETH pool on Uniswap.

Improvement suggestions:

Enhancing User Experience

• Bridge Optimization from L1 to L2:
Streamline the process for users to deposit collateral directly from the mainnet (L1) to the Layer 2 (L2) platform in a single transaction. This not only reduces transaction costs but also provides a smoother user experience.

• Unified Transaction for HAIProxy.sol and Collateral Deposit:
Enable users to initiate the creation (or interaction) of the HAIProxy.sol contract and simultaneously deposit collateral in one cohesive transaction. This minimizes the steps a user needs to take, thus simplifying the process.

• Automated HAI Purchase for Debt Repayment:
Introduce a mechanism for automatic HAI purchases in the open market when users need to repay debt. This can be achieved by utilizing HAI flash loans, ensuring users don't need to buy HAI tokens manually, further improving the overall user experience.

Conclusion

The HAI team has successfully developed a framework for stablecoin issuance, demonstrating a high level of proficiency. All known security issues have been addressed effectively. However, it is essential for users to acknowledge the complexity of this system, which relies on third-party data providers and the community's participation. For further details, please see the Security Assumptions section.

Appendix 1. Terminology

Severity

Assessment the magnitude of an issue.

		Severity		
Impact	Major	Medium	Major	Critical
	Medium	Minor	Medium	Major
	Minor	None	Minor	Medium
		Minor	Medium	Major
		Likelihood of exploitation		

Minor

Minor issues are generally subjective in nature or potentially associated with topics like “best practices” or “readability”. As a rule, minor issues do not indicate an actual problem or bug in the code. The maintainers should use their own judgment as to whether addressing these issues will improve the codebase.

Medium

Medium issues are generally objective in nature but do not represent any actual bugs or security problems. These issues should be addressed unless there is a clear reason not to.

Major

Major issues are things like bugs or vulnerabilities. These issues may be unexploitable directly or may require a certain condition to arise to be exploited. If unaddressed, these issues are likely to cause problems with the operation of the contract or lead to situations which make the system exploitable.

Critical

Critical issues are directly exploitable bugs or security vulnerabilities. If unaddressed, these issues are likely or guaranteed to cause major problems or, ultimately, a full failure in the operations of the contract.