LIFESTONE LABS

ANYHEDGE

Project Audit Report

AnyHedge

Audit Date: November 5, 2025

Auditor: Kyle Wildeman

Audit Report Disclaimer

Purpose of This Audit

This audit was performed independent of, and without compensation from, the creators of AnyHedge. The purpose is to identify any potential vulnerabilities, logic errors, and security concerns that may exist.

Scope of Analysis

This audit examines the smart contract code for technical vulnerabilities, logic flaws, and potential attack vectors at the contract level. It does not include any analysis of frontend or backend infrastructure. Additionally, it attempts to highlight any trust assumptions or centralization risks that users should be aware of when interacting with the contracts.

Methodology

Manual code review is performed to identify security vulnerabilities, logic errors, and potential attack vectors. This process includes:

- Manual analysis of smart contract code
- Identification of common smart contract vulnerabilities
- Recommendations for security improvements

Limitations and Disclaimer

This audit does not guarantee the security of the audited contracts. Security audits are inherently limited by time, scope, and the possibility of undiscovered vulnerabilities. The absence of identified issues does not imply the absence of vulnerabilities. Users should conduct their own due diligence and consider additional security measures when interacting with smart contracts.

AnyHedge Audit Report

Project Information

Project Name:
AnyHedge

Audit Date:
November 5, 2025

Auditor:
Kyle Wildeman

Language:
CashScript 0.9.2

Project Symbol:
None

Project Website:
https://anyhedge.com/

Project Logo:

Project Description

AnyHedge allows you to take leveraged long positions or hedge your fiat value. Permissionless, non-custodial, and on-chain.

Findings and Severity

HIGH: 2 MEDIUM: 1 LOW: 0 INFO: 0

Transactions and Trust Levels

OPERATOR TRUSTED: 1 FRONTEND TRUSTED: 0 TEMPLATE TRUSTED: 0 TRUSTLESS: 0

Scope Details

This section lists contracts and their functions covered in this audit. Function descriptions are provided by the project.

AnyHedge_v0_12	dynamic addresses	0.9.2

mutualRedeem

Safety feature where Short and Long can exit the contract at any time through mutual agreement.

payout

The contract settles on a predefined maturity date or when a liquidation boundary is hit.

Transactions

This section details the dapps transactions and their level of required trust.

Note: All transactions made by the user still need to be Approved by the user unless they have enabled an auto-Approve feature in their wallet. This means that users do have the opportunity to verify whether a transaction is incorrect or malicious before they approve it. However, we rank the Trust Classification of transactions under the worst-case assumption, which is a user that does not manually verify every transaction request themselves before approving it.

Trust Levels:

Level	Description
OPERATOR TRUSTED	The user must trust the actions of another party (team, admins, external service) to behave correctly. This may include a systemic risk of loss or harm if the party behaves incorrectly.
FRONTEND TRUSTED	The user must trust the frontend interface to build some parts of the transaction correctly since the contracts do not enforce those settings.
TEMPLATE TRUSTED	The user must trust the templates are correct and not maliciously designed.
TRUSTLESS	The contracts enforce all critical transaction building decisions so that no user loss or harm can occur, regardless of the frontend or wallet behaviour.

1. Create Forward Contract

OPERATOR TRUSTED

Two parties enter a BCH settled forward contract. The Short side pays the Long side when the price increases and vice versa. The contract settles on a predefined maturity date or when a liquidation boundary is hit.

Functions
None

Trust Level Explanation:

> Operator level: The user must trust the external Oracle to provide honest and accurate off-chain price data which the contract uses during contract settlement. Frontend-level: The user must trust the frontend to initialize the contract with the correct parameters.

Findings

This section lists potential issues found during the audit.

Severity	Description
HIGH	Findings that can result in significant loss of user funds or cause unintended contract manipulation which undermines the integrity of the project.
MEDIUM	Findings that can result in loss of user funds in a more limited scope or allows contract manipulation which can cause unintended consequences, but doesn't undermine the integrity of the entire project.
LOW	Minor, non-critical findings that allow or cause unintended consequences, such as an edge-case that prevents a user from performing a certain expected transaction, but can be worked around with no harm or loss.
INFO	Minor findings which are unlikely to have any negative impact. Observations of the design's limitations or restrictions.

1. Frontend fully trusted

HIGH

AnyHedge contracts are created on-demand when two users want to enter a forward contract. This requires a trusted frontend/server to handle the contracts parameters which brings the contract into existence.

Impact:

If any of the 13 contract parameters are entered incorrectly by the frontend it will have negative effects:

- shortMutualRedeemPublicKey: unable to call mutualRedeem()
- longMutualRedeemPublicKey: unable to call mutualRedeem()
- enableMutualRedemption: unable to call mutualRedeem()
- shortLockScript: loss of users BCH
- longLockScript: loss of users BCH
- oraclePublicKey: unable to call payout()
- nominalUnitsXSatsPerBch: incorrect payout price calculation
- satsForNominalUnitsAtHighLiquidation: incorrect payout price calculation
- payoutSats: incorrect payout price calculation
- lowLiquidationPrice: incorrect contract price bounds
- highLiquidationPrice: incorrect contract price bounds
- startTimestamp: incorret contract start time
- maturityTimestamp: incorrect contract end time

If one of the first three parameters related to mutualRedeem are entered incorrectly then mutualRedeem() will not be callable. If this occurs along with an incorrect oraclePublicKey then payout() will also not be callable, resulting in permanent locking of funds in the contract.

Recommendation:

> To reduce attack surface a design with persistent contract(s) that lets users create and join forward contracts (as NFTs) would allow the contracts to verify user addresses and remove some trust from the frontend. This would also be easier for alternative frontends to interact with (e.g. if the primary frontend goes down) since the persistent contract would be a known address and all existing forward contracts can be viewed on it, rather than requiring to know each users parameters to rebuild their contracts to determine the addresses. Emergency withdraws would also be simpler to construct since no user signature parameters would be needed with the function.

Status: open

2. No unilateral exit option

HIGH

The Anyhedge contract allows both participants of a forward contract to exit the contract if they both agree, but if one of the users doesn't approve then no exit path can occur.

Impact:

If for some reason the payout function cannot be called (i.e. the contract was setup incorrectly or Oracle issues) and one of the users is missing then the funds will be permanently locked.

Recommendation:

> A timelocked 'exit of last resort' function that makes both users whole well after the initial forward contract expired would prevent user funds from being permanently locked. If adapted to persistent contracts + NFTs the function could also be triggered by anyone.

Status: open

3. Sanity check Oracle data

MEDIUM

The Oracle is already fully trusted, but the contract assumes the Oracles provided data is correct (e.g. all fields exist and are setup correctly)

Impact:

If the Oracle provides malformed data then values used to verify the payout function can be incorrect, resulting in unexpected behaviour.

Recommendation:

> At the most basic a length check would verify that settlementMessage and previousMessage are both 16bytes, limiting the number of ways the Oracle may mistakenly provide data. If possible, additional checks on other fields could be added to verify values are within expected ranges.

Status: open

Report Information

Generated on: November 6, 2025 at 10:46 PM

Total findings: 3

Total observations: 0