THREE SIGMA

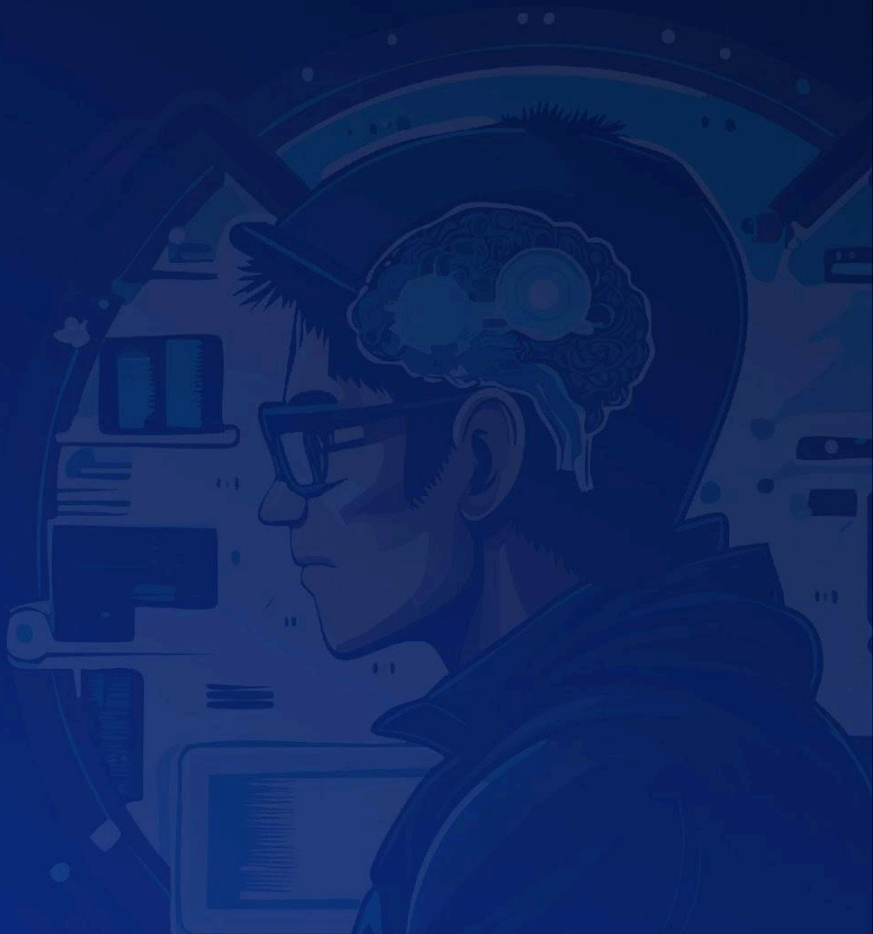
Felix Lending Protocol

Security Review

Disclaimer

Security Review

Felix Lending Protocol



Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Security Review

Felix Lending Protocol

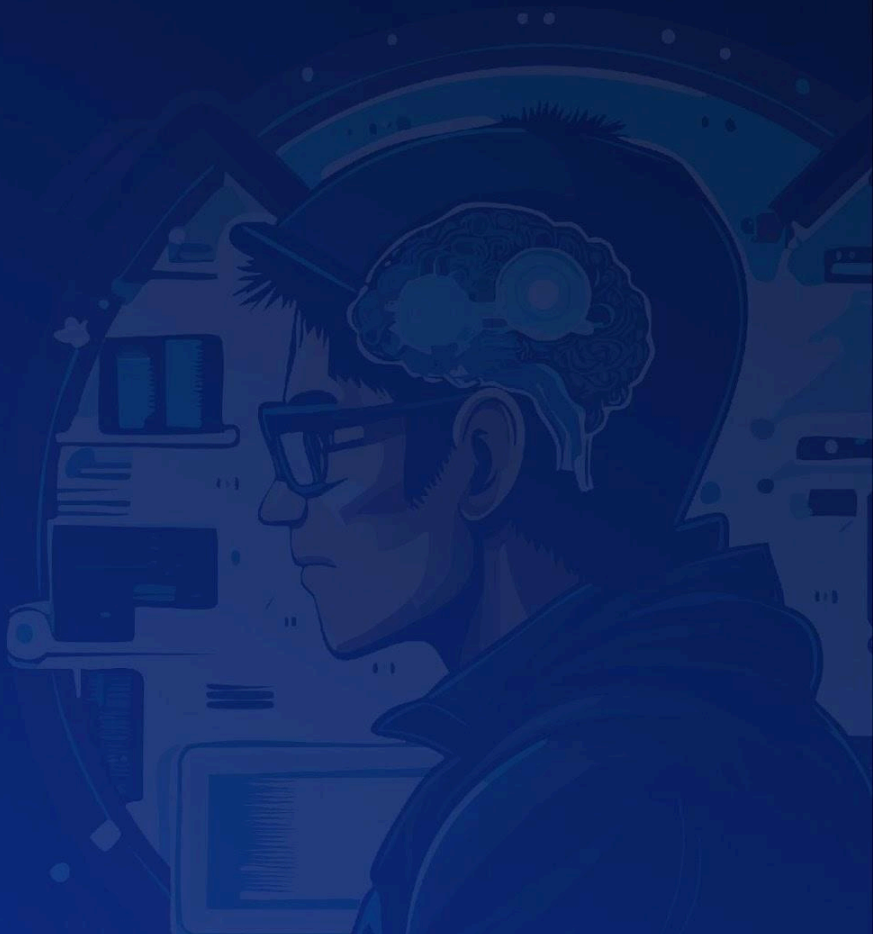
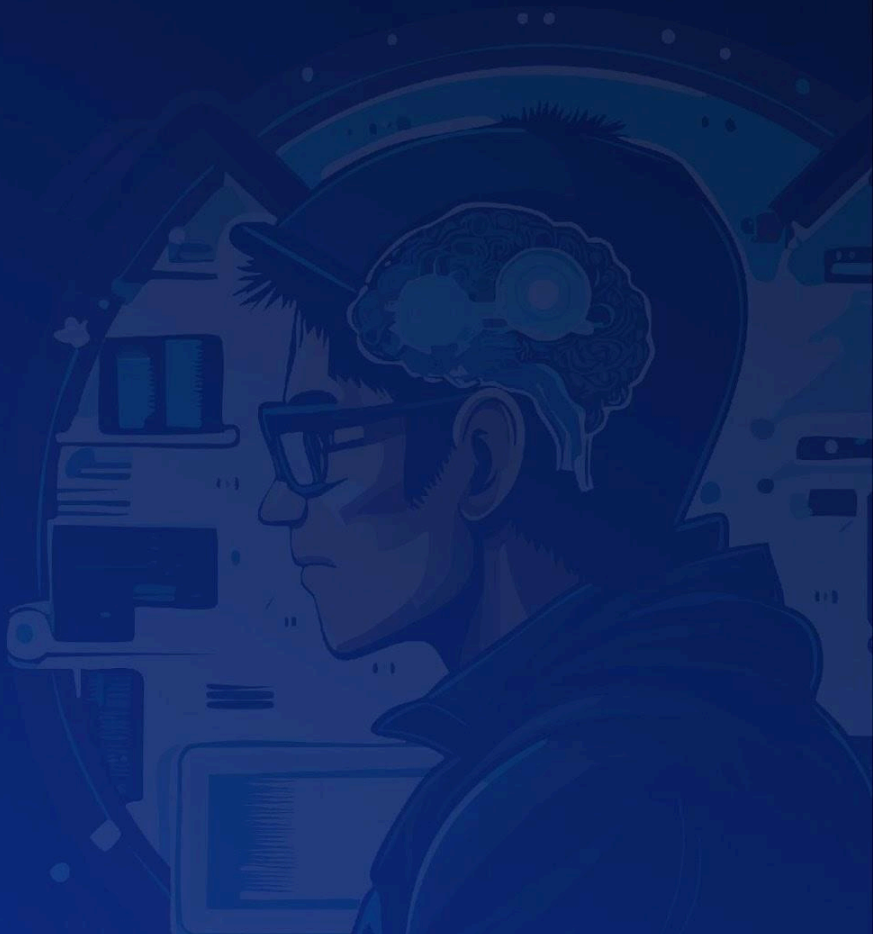


Table of Contents

Summary

Security Review

Felix Lending Protocol



Summary

Three Sigma audited Felix in a 1.2 person week engagement. The audit was conducted from 23/07/2025 to 25/07/2025.

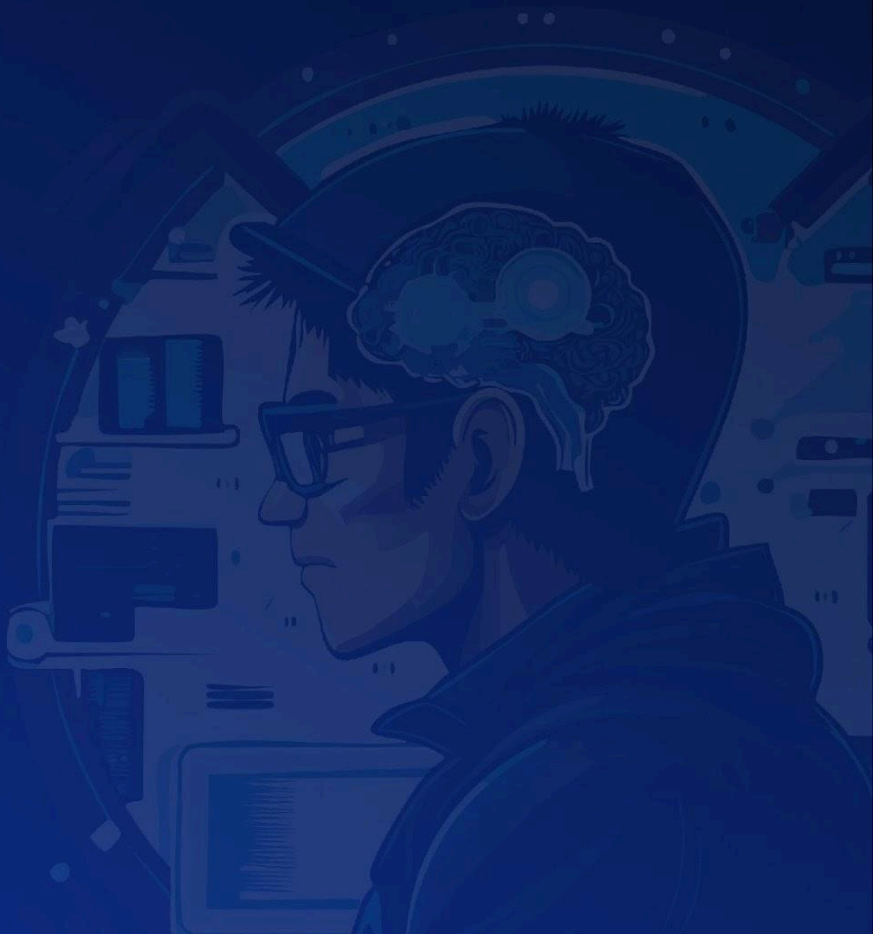
Protocol Description

Felix is a suite of on-chain borrowing and lending products on Hyperliquid L1, centered on a CDP market (feUSD) and variable-rate “Vanilla” pools to unlock liquidity or earn yield securely and with low friction. Users leverage it for margin trades, carry trades (HLP/SP), spot buys, leverage looping, stablecoin yield, and DEX LPing across volatile and stable-swap pairs.

Scope

Security Review

Felix Lending Protocol



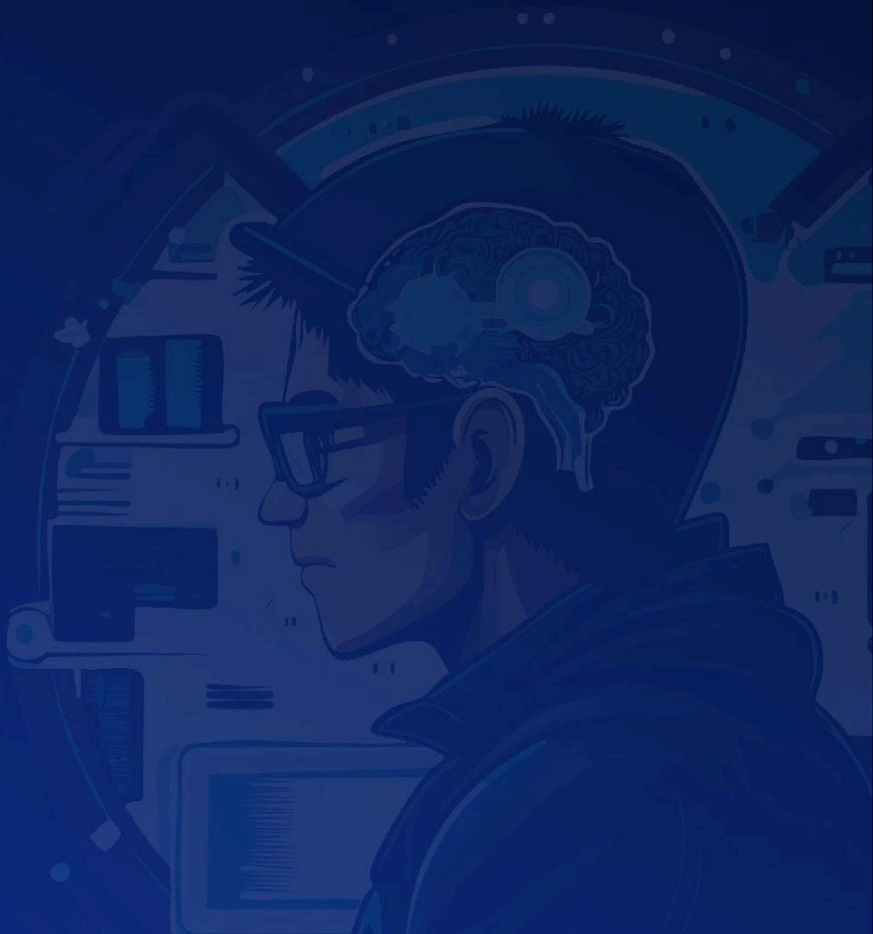
Scope

Filepath	nSLOC
src/PriceFeeds/KHYPEPriceFeed.sol	119
src/PriceFeeds/WSTHYPEPriceFeed.sol	79
Total	**198**

Methodology

Security Review

Felix Lending Protocol



Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

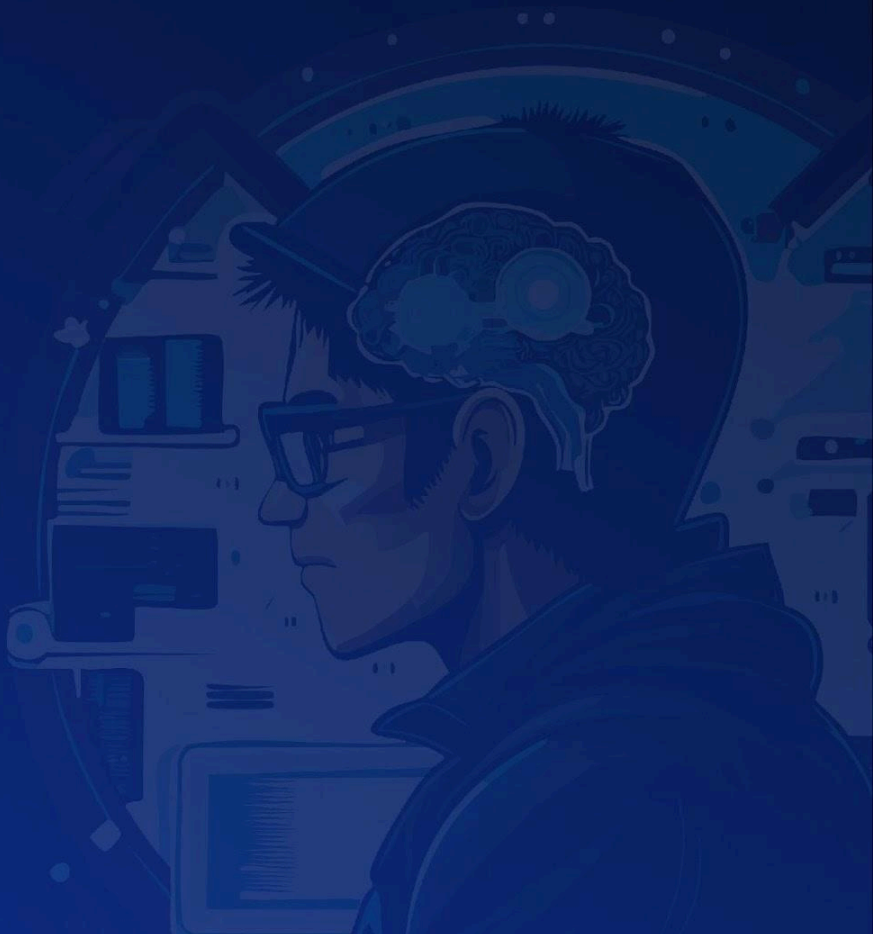
In this audit, we classify findings based on Immunefi's Vulnerability Severity Classification System (v2.3) as a guideline. The final classification considers both the potential impact of an issue, as defined in the referenced system, and its likelihood of being exploited. The following table summarizes the general expected classification according to impact and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Impact / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Security Review

Felix Lending Protocol



Project Dashboard

Application Summary

Name	Felix
Repository	https://github.com/felixprotocol/felix-contracts
Commit	f8a8b02
Language	Solidity
Platform	Hyperliquid

Engagement Summary

Timeline	23/07/2025 to 25/07/2025
№ of Auditors	2
Review Time	1.2 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	2	2	0
Low	0	0	0
None	0	0	0

Category Breakdown

Suggestion	0
Documentation	0
Bug	2
Optimization	0
Good Code Practices	0

Risk Section

Security Review

IndexTokenStaking Staking

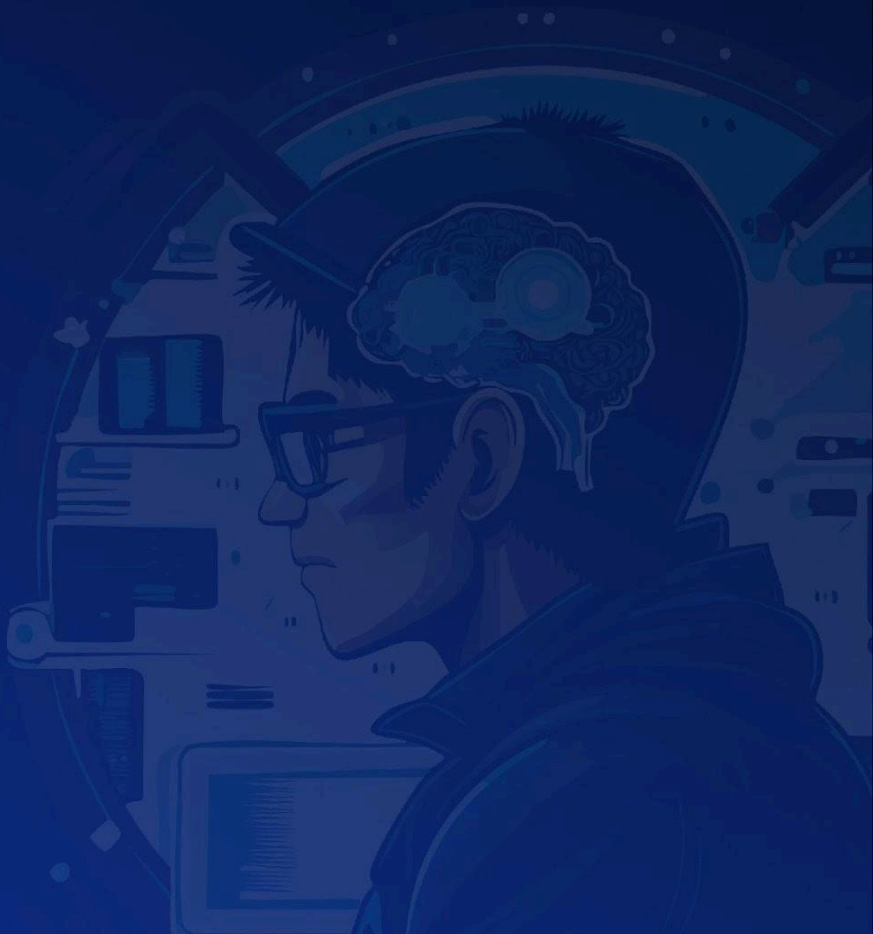
Risk Section

No risks were identified.

Findings

Security Review

Felix Lending Protocol



Findings

3S-Felix-M01

Inaccurate Hype/USD price due to dependency on USDC peg

Id	3S-Felix-M01
Classification	Medium
Impact	High
Likelihood	Low
Category	Bug
Status	Addressed in #398ca73.

Description:

The **KHYPEPriceFeed** and **WSTHYPEPriceFeed** contracts compute the USD prices of kHype and wstHype based on the price of Hype in USD.

```
// Calculate the market KHYPE-USD price: USD_per_KHYPE = USD_per_HYPE * HYPE_per_KHYPE
uint256 kHypeUsdMarketPrice = (hypeUsdPrice * kHypeHypePrice) / 1e18;
// Calculate the canonical LST-USD price: USD_per_KHYPE = USD_per_HYPE * HYPE_per_KHYPE
uint256 kHypeUsdCanonicalPrice = (hypeUsdPrice * kHypePerHype) / 1e18;
```

```
if (_isRedemption && _withinDeviationThreshold(stHypeUsdPrice, hypeUsdPrice, STHYPE_USD_DEVIATION_THRESHOLD)) {
    // If it's a redemption and within 1%, take the max of (STHYPE-USD, HYPE-USD) to mitigate unwanted redemption arb and convert to WSTHYPE-USD
    wstHypeUsdPrice = LiquityMath._max(stHypeUsdPrice, hypeUsdPrice) * stHypePerWstHype / 1e18;
}
```

However, the Redstone oracle does not provide a native Hype/USD feed—only a Hype/USDC feed. This introduces an implicit dependency on the USDC/USD peg being exactly 1.0.

In the event of a USDC depeg (e.g., during market stress or a banking crisis), any logic relying on Hype/USD calculations becomes unreliable. This may lead to incorrect valuations in redemption operations, liquidations, or other critical mechanisms, and consequently to loss of funds. Notably, even if the Hype/USDC rate remains constant, the actual USD-denominated value of Hype may diverge significantly.

Recommendation:

To reduce reliance on the stability of the USDC peg, the implementation should explicitly incorporate the USDC/USD feed and compute the Hype/USD price as (Hype/USDC) * (USDC/USD). This will ensure accurate pricing even during periods of depeg, and prevent financial discrepancies across the system.

3S-Felix-M02

Incorrect deviation thresholds used in **KHYPEPriceFeed** and **WSTHYPEPriceFeed**

Id	3S-Felix-M02
Classification	Medium
Impact	High
Likelihood	Low
Category	Bug
Status	Addressed in #398ca73.

Description

The **KHYPEPriceFeed** contract uses an incorrect deviation threshold of 2% (**KHYPE_HYPE_DEVIATION_THRESHOLD = 2e16**) when determining whether to apply oracle frontrunning mitigation during redemptions. According to the audit findings, the actual deviation threshold of the Redstone KHYPE-HYPE oracle feed is 0.5%, but the contract incorrectly uses 2% because it was forked from the RETH-ETH price feed implementation whose deviation threshold is 2%.

This incorrect threshold affects the price calculation logic in the **_fetchPricePrimary** function:

```
// If it's a redemption and canonical is within 2% of market, use the max to mitigate unwanted redemption oracle arb
if (
    _isRedemption &&
    _withinDeviationThreshold(
        kHypeUsdMarketPrice,
        kHypeUsdCanonicalPrice,
        KHYPE_HYPE_DEVIATION_THRESHOLD // Using 2% instead of correct 0.5%
    )
) {
    kHypeUsdPrice = LiquityMath._max(
        kHypeUsdMarketPrice,
        kHypeUsdCanonicalPrice
    );
} else {
    kHypeUsdPrice = LiquityMath._min(
        kHypeUsdMarketPrice,
```

```
        kHypeUsdCanonicalPrice
    );
}
```

Similarly **WSTHYPEPriceFeed** is also using 1% deviation, but the actual deviation threshold for Redstone stHYPE-USD oracle's deviation threshold is 0.5%.

KHYPEPriceFeed: When price differences are between 0.5% and 2%, users receive worse redemption prices due to incorrect application of oracle frontrunning protection

WSTHYPEPriceFeed: When price differences are between 0.5% and 1%, users receive worse redemption prices for the same reason

Oracle frontrunning mitigation is applied too broadly, beyond the actual oracles' deviation thresholds

---

Recommendation

Update both deviation threshold constants to reflect the actual 0.5% deviation threshold of their respective oracles:

```
uint256 public constant KHYPE_HYPE_DEVIATION_THRESHOLD = 5e15; // 0.5%
uint256 public constant STHYPE_USD_DEVIATION_THRESHOLD = 5e15; // 0.5%
```

These changes will ensure that oracle front-running mitigation is only applied when price differences are within the actual oracles' update thresholds.

3S-Felix-N01

Informational Findings

Id	3S-Felix-N01
Classification	None
Category	Suggestion

Description:

According to the Kinetiq docs one of the risks associated with kHype is a market price risk:

Users may face a situation where the exchange price of kHYPE falls below its inherent value due to withdrawal restrictions on Kinetiq, preventing arbitrage and risk-free market-making. Kinetiq is committed to minimizing these risks and, where possible, eliminating them entirely. However, they may still persist, and it is our responsibility to ensure they are clearly communicated.

The reason for this is that users have to wait 7 days before they can unstake. If someone wants to perform arbitrage and, for example, buy discounted kHYPE for USD, they would have to wait 7 days before receiving HYPE and potentially selling it at a profit. In a downtrend, this is even riskier for the arbitrageur.

In the KHYPEPriceFeed is implemented the following protection against the oracle frontrunning attack described in the docs of Liquity:

```
// If it's a redemption and canonical is within 2% of market, use the max to mitigate unwanted redemption oracle arb
if (
    _isRedemption &&
    _withinDeviationThreshold(
        kHypeUsdMarketPrice,
        kHypeUsdCanonicalPrice,
        KHYPE_HYPE_DEVIATION_THRESHOLD
    )
```

```
) {
    kHypeUsdPrice = LiquityMath._max(
        kHypeUsdMarketPrice,
        kHypeUsdCanonicalPrice
    );
} else {
    // Take the minimum of (market, canonical) in order to mitigate against upward market price manipulation.
    // Assumes a deviation between market <> canonical of >2% represents a legitimate market price difference.
    kHypeUsdPrice = LiquityMath._min(
        kHypeUsdMarketPrice,
        kHypeUsdCanonicalPrice
    );
}
```

In short, the idea is that during redemption, if the difference between the oracle price and the canonical price is within the feed's threshold, the higher value is returned in order to prevent an oracle frontrunning attack.

Let's consider the following example scenario:

1. The price of Hype is 10, the oracle price of kHype is 9.8, and the canonical price of kHype is 10.1.
2. A user notices the oracle price of kHype rising to 9.9 and frontruns the transaction that updates the price. At that moment, the difference between the canonical price and the oracle price is greater than 0.5% because, due to market conditions, the price of kHype is lower.
3. The protocol uses the lower of the two prices – 9.8 – in the redemption operation, even though this is an oracle frontrunning attack. As a result, the user receives more tokens than if the real market price were used.
4. The user sells the received tokens and locks in a profit, effectively extracting value from the protocol.

5. The oracle price of kHype remains more than 0.5% lower than the price of Hype, and upon the next price increase, the same process can be repeated to lock in another profit.

In the described scenario, there was a depeg of kHype, which led to a significant difference between the oracle price and the canonical price. What would happen if there were no depeg of kHype in this scenario? The answer is that the increase in the oracle price of kHype would very likely have been accompanied by an increase in the price of Hype, and consequently in the canonical price of kHype. This would have reduced the chance of a large price discrepancy and could have triggered the described mitigation, which is the intended design in Liquity.

Therefore, the susceptibility to depegging — due to the lack of easy arbitrage on kHype's market price — is the main reason why using kHype carries additional risk compared to rETH and wstETH, which have similar feeds in Liquity.

Finally, I want to add this quote from the Liquity documentation:

However, we don't expect oracle frontrunning to be a significant issue since these LST-ETH feeds are historically not volatile and rarely deviate by significant amounts: they usually update based on the heartbeat (mininum update frequency) rather than the threshold.

where it states that significant issues with oracle frontrunning are not expected because LST-ETH pairs have historically been stable. However, this is not the case at all for kHype/Hype, and therefore the risk of oracle frontrunning attacks is significantly higher.

Recommendation:

There is no good solution to this problem because it stems from the way kHype works and its underlying mechanics.