Kinetiq: kntq solo review

Security Review

Cantina Managed review by:
Optimum, Lead Security Researcher

January 15, 2026

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**
- 2.1 Scope . . . 3

3 Findings **4**
- 3.1 Informational . . . 4
  - 3.1.1 `DeploySKNTQ`: batched initialization calldata is not directly observable from script output . . . 4
  - 3.1.2 Using `CREATE2` in forge scripts can be front-run causing unexpected reverts . . . 4
  - 3.1.3 Avoid hardcoded `CREATE2` deployer address magic constant . . . 4
  - 3.1.4 `ERC4626AsynchronousUpgradeable`: unused _transferIn function should be removed . . . 5
  - 3.1.5 `ERC4626` inflation mitigation: isolate initial shares from vault-held assets . . . 5
  - 3.1.6 Storage layout inconsistency: `SKNTQ` uses linear storage while base contracts use `ERC7201` . . . 5

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

A security review is a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While the review endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that a security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Kinetiq is a protocol that scores validators and distributes staking allocations based on performance, utilizing HYPE tokens to manage delegation.

From Jan 4th to Jan 7th the security researchers conducted a review of kntq on commit hash 5005c71c. A total of **6** issues were identified:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	6	0	6
Total	**6**	**0**	**6**

2.1 Scope

The security review had the following components in scope for kntq on commit hash 5005c71c:

```
src
├── base
│   └── ERC4626AsynchronousUpgradeable.sol
├── Create2Factory.sol
├── KNTQ.sol
├── lib
│   ├── L1Read.sol
│   ├── L1Write.sol
│   └── PauserRegistry.sol
├── PlaceholderImplementation.sol
└── sKNTQ.sol
```

3 Findings

3.1 Informational

3.1.1 `DeploySKNTQ`: batched initialization calldata is not directly observable from script output

Severity: Informational

Context: DeploySKNTQ.s.sol#L235

Description: When `batchCalls` execution is skipped in `forge script` (e.g. due to precompile revert), the script logs each individual target, calldata, and value in a loop.

This does not reflect the actual on-chain execution, which is a single call to `CallBatcher.batchCalls`, and requires manual reconstruction of calldata, which is error-prone.

As a result, the exact transaction that must be sent on-chain is not directly observable from the script output.

Recommendation: Log the fully encoded calldata for `CallBatcher.batchCalls(targets, data, values)` so it can be submitted directly via `cast send`.

Kinetiq: Acknowledged.

Optimum: Acknowledged.

3.1.2 Using `CREATE2` in forge scripts can be front-run causing unexpected reverts

Severity: Informational

Context: DeploySKNTQ.s.sol#L88

Description: Foundry deployment scripts rely under the hood on the CREATE2 deployer at. `0x4e59b44847b379578588920cA78FbF26c0B4956C`, whose fallback function performs CREATE2 deployments.

Because this is a public contract, a third party can front-run a deployment transaction and deploy a contract with the same salt and bytecode first. When the script's transaction executes afterward, the `CREATE2` call reverts, causing the deployment script to fail.

Foundry does not currently detect or gracefully handle this case, so the revert can be unexpected and hard to diagnose.

Impact: Deployment scripts using `new {salt: ...}` may revert non-deterministically under adversarial or congested conditions, even when the script logic itself is correct.

> **Note:** While this can be worked around by avoiding `new` and using a low-level call that tolerates pre-existing deployments, this is heavy-handed. At minimum, this behavior should be documented to raise awareness for users encountering unexplained deployment reverts.

Kinetiq: Acknowledged.

Optimum: Acknowledged.

3.1.3 Avoid hardcoded `CREATE2` deployer address magic constant

Severity: Informational

Context: DeploySKNTQ.s.sol#L172

Description: Some deployment scripts rely on the CREATE2 deployer at the fixed address. `0x4e59b44847b379578588920cA78FbF26c0B4956C`, embedding it directly as a literal in the code.

Using this address as a magic constant is brittle and obscures where the value comes from, making scripts harder to reason about, audit, and maintain. It also tightly couples user code to Foundry internals without an explicit abstraction or reference.

Recommendation: Avoid hardcoding the CREATE2 deployer address as a literal in deployment scripts. Prefer a named constant, helper, or documented reference provided by Foundry.

See Foundry's documentation on deterministic deployments for context.

Kinetiq: Acknowledged.

Optimum: Acknowledged.

3.1.4 `ERC4626AsynchronousUpgradeable`: unused _transferIn function should be removed

Severity: Informational

Context: ERC4626AsynchronousUpgradeable.sol#L169

Description: The function `_transferIn(address from, uint256 assets)` is defined in `ERC4626AsynchronousUpgradeable` but is never used by the contract.

Deposits rely on the inherited ERC4626 implementation, and this override is not referenced anywhere in the codebase, making it dead code.

Recommendation: Remove `_transferIn` if it is not intended to be used, or explicitly document and integrate it into the deposit flow if customization is planned.

Kinetiq: Acknowledged.

Optimum: Acknowledged.

3.1.5 ERC4626 inflation mitigation: isolate initial shares from vault-held assets

Severity: Informational

Context: DeploySKNTQ.s.sol#L223

Description: The initial shares minted for ERC4626 inflation-attack mitigation are currently sent to the `skntq` contract itself. This is functionally correct because the vault already holds the corresponding token balance, and the accounting remains consistent.

However, this design couples the ”burned” shares with a contract that actively participates in vault accounting. While correct by construction, this reduces isolation between live vault state and permanently inaccessible shares.

Recommendation: Consider sending the initial shares to a burn address:

```
0x000000000000000000000000000000000000dEaD
```

to fully isolate unspendable shares from the vault's operational balance. This improves conceptual clarity and reduces the chance of confusion during audits or future changes.

> **Note:** This is a best-practice suggestion for clarity and isolation, not a correctness issue.

Kinetiq: Acknowledged.

Optimum: Acknowledged.

3.1.6 Storage layout inconsistency: `SKNTQ` uses linear storage while base contracts use `ERC7201`

Severity: Informational

Context: sKNTQ.sol#L26

Description: The base contracts follow the ERC7201 namespaced storage pattern, while `SKNTQ` uses a traditional linear storage layout.

While not incorrect, this inconsistency increases cognitive overhead and raises the risk of storage collisions or upgrade fragility in a system composed of multiple upgradeable components.

Recommendation: Consider migrating `SKNTQ` to the ERC7201 storage pattern to align with the base contracts and ensure consistent, collision-resistant storage layouts across the codebase.

Kinetiq: Acknowledged.

Optimum: Acknowledged.