CHAINSECURITY

PUBLIC

Code Assessment
of the kAssets Smart Contracts

November 10, 2026

Produced for

kraken



Contents

1	Executive Summary	3
2	Assessment Overview	4
3	System Overview	5
4	Trust Model	7
5	System Considerations	10
6	Terminology	11
7	Findings	12
8	Limitations and use of report	16

1 Executive Summary

Dear Kraken Team,

Thank you for trusting ChainSecurity with the review of kAssets.

As a globally trusted security partner for digital asset platforms, ChainSecurity provides specialized, independent technical analysis for token systems and cross-chain integrations.

We greatly appreciate the opportunity to collaborate with Kraken, a leading global cryptocurrency exchange and an established pioneer in the digital asset industry.

Kraken implements a smart contract framework for its wrapped digital assets. The contracts support controlled token minting and burning, enforce blacklist and pause restrictions, and integrate with Chainlink CCIP for cross-chain transfers. A timelock governs upgrades to the token and blacklist contracts.

We have conducted a thorough review of the scope. The most important areas covered in our review are:

1. **Bridge integration:** Cross-chain transfers rely on CCIP to mint tokens on the destination chain. We identified an issue where minting to new holders might exceed the default token pool gas limit, causing automatic execution to fail and requiring manual execution. See **Bridged Transfers to New Holders May Require Manual Execution**.

2. **Upgrade liveness:** The token and blacklist contracts rely on the timelock for future upgrades. We identified an issue where renouncing all proposer roles can permanently prevent new operations from being scheduled, disabling upgrades unless a pending operation can restore proposer access. See **Renouncing All Proposer Roles Can Permanently Disable Upgrades**.

All issues raised have been addressed.

We completed the initial review and the review of the subsequent fixes. In its current state, the codebase provides a good level of security. We ask Kraken to carefully review **System Overview**, **Trust Model** and **System Considerations**.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,
ChainSecurity

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the kAssets repository based on the documentation files. The code versions relevant to this report and when they were received are indicated below.

Version	Date	Commit Hash	Note
1	20 October 2026	8b52f530665d9e830fc18fcf5b1d5a47bb97701b	Initial review
2	4 November 2026	d4e228a23a1f7fe11dbd6b3308be8327b9bda0d6	Version with fixes

The code was delivered as ZIP archives rather than through a repository. **Version 1** was received as an archive with the MD5 hash `60f61d0e3d2c839dd6397f47514e651c`. **Version 2** was received as an archive with the MD5 hash `1401efe68116046e4aabc5277f49c6cb`. Each commit referenced above is the import of the corresponding archive into a local repository.

For the Solidity smart contracts, the compiler version `0.8.30` was chosen, targeting the `cancun` EVM version.

The following files are in the scope of this review:

```
src/evm/src/
├── BurnMintPoolDeployer.sol
├── IKAsset.sol
├── IKAssetBlacklist.sol
├── KAsset.sol
├── KAssetBlacklist.sol
└── KAssetTimelock.sol
```

2.1.1 Excluded from scope

Any file not explicitly listed above is out of the scope of this review. Third-party libraries like `openzeppelin-contracts`, `openzeppelin-contracts-upgradeable`, `chainlink-ccip` and `chainlink-evm` are out of the scope of this review and expected to work as intended. The on-chain and off-chain components of the CCIP infrastructure and protocol are out of the scope of this review.

3 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

At the end of this report section, we have added subsections for each of the changes according to the versions.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Kraken provides shared contracts for wrapped assets called kAssets. Each asset is an ERC-20 token that can be minted and burned by authorized parties, paused, and restricted through a chain-specific blacklist. Cross-chain transfers use Chainlink CCIP burn-and-mint pools. The configured EVM target chains are `ethereum`, `ink`, `optimism`, and `unichain`.

On each configured chain, the system consists of a `KAssetTimelock`, a `KAssetBlacklist` implementation and proxy, and a `KAsset` implementation. Each asset has a token proxy and a CCIP pool. All token proxies on a chain refer to the chain's shared blacklist proxy. Each transparent proxy has a `ProxyAdmin` owned by the chain timelock.

3.1 KAsset

`KAsset` implements ERC-20 balances and allowances, EIP-2612 permits, pause controls, issuance and burns, and blacklist checks.

3.1.1 Transfers and approvals

Transfers, minting, and burning revert while the token is paused or when the caller, source, or destination is blacklisted.

`approve()` and `permit()` also revert while paused or when the caller, holder, or spender is blacklisted. Allowance spending through `transferFrom()` is still subject to the transfer checks above.

3.1.2 Minting and burning

Authorized callers (owner or `MINTER_ROLE`) can mint without a token-level cap.

Any holder can burn its own tokens. An authorized caller (owner or `BURNER_ROLE`) can burn another address's tokens without allowance.

3.1.3 Pause, ownership, and token administration

Authorized callers (owner or `EMERGENCY_PAUSE_ROLE`) can pause the token, blocking balance updates and new approvals. Only the owner can call `unpause()`.

Only the owner can grant or revoke token roles. The owner can also recover ERC-20 tokens held by the token through `rescueERC20()`. Ownership renunciation is disabled, and ownership transfer uses the inherited two-step process. The blacklist address and token decimals have no ordinary setter. Changing the blacklist reference requires an implementation upgrade and migration.

3.2 KAssetBlacklist

One `KAssetBlacklist` proxy is expected to be shared by all kAssets on a chain. The owner or a blacklist operator can use `setBlacklisted()` to block or unblock nonzero addresses in a batch; only the owner can grant or revoke roles. The owner can recover ERC-20 tokens held by the blacklist. Ownership renunciation is disabled.

`KAsset` checks the shared blacklist during transfers, minting, burning, approvals, and permits, so a blacklist change affects every token proxy on the chain.

3.3 CCIP pool and cross-chain transfers

Each asset has a Chainlink CCIP `BurnMintTokenPool`. It burns tokens for outbound transfers and mints them on inbound transfers.

`BurnMintPoolDeployer` creates the pool and proposes ownership transfer to the asset owner, who must accept it.

The pool checks its router, configured remote pools, RMN curse state, and rate limits. Its owner can change bridge configuration. See External Components in the Trust Model for CCIP assumptions.

3.4 Timelock and upgrades

`KAssetTimelock` extends OpenZeppelin `TimelockController`. Its constructor and `updateDelay()` require a delay of at least two days. With open execution, any address can execute a scheduled operation once its delay passes. The timelock rejects any executed call whose function selector is `renounceOwnership()`.

The timelock owns the `ProxyAdmin` for the blacklist proxy and each token proxy. Proxy upgrades and optional migration calls execute through the timelock after its delay.

3.5 Changes in Version 2

Version 2 introduces the following changes:

- `KAssetTimelock` disables `renounceRole()`. No timelock role can be renounced, including by the timelock itself through a scheduled operation. Roles can still be removed through `revokeRole()`.
- The CCIP pool is upgraded from `BurnMintTokenPool` v1.6.1 to v2.0.0. `BurnMintPoolDeployer` passes an optional advanced pool hooks contract to the pool instead of an allowlist.
- The v2.0.0 pool lets its owner configure per-lane token transfer fees and destination gas, fast-finality transfers with separate rate limits, and advanced pool hooks that can add transfer checks and required cross-chain verifiers. Fast-finality transfers are disabled and no fees are configured by default.

4 Trust Model

4.1 Introduction

On each configured chain, a `KAssetTimelock` controls the `ProxyAdmin` contracts for the token and shared blacklist proxies. In the intended deployment, the owner is a signatory-service-managed EOA that owns the token, blacklist, and CCIP pools and is the timelock's sole proposer. Deployment scripts configure this layout on `ethereum`, `ink`, `optimism`, and `unichain`, but the contracts do not enforce that the same address holds every role.

4.2 Roles and Permissions

Asset Owner

- Trust Level: Fully trusted.
- Permissions: Controls token roles, can mint, burn from any account, pause and unpause, and rescue tokens. Controls blacklist entries and roles. Owns and configures CCIP pools. Registers as each token's CCIP administrator in the `TokenAdminRegistry` and sets its pool. In the intended deployment, schedules and cancels timelock operations.
- Worst-case: Can mint without a cap, burn holders' balances, halt token operations, blacklist/unblacklist addresses across all tokens on the chain, redirect bridge configuration, or grant privileged roles. It can replace token or blacklist logic through a delayed upgrade.

Timelock Proposer and Canceller

- Trust Level: Fully trusted.
- Permissions: Schedules operations, including proxy upgrades, and cancels pending operations. The deployment assigns these roles to the asset owner. The timelock administers its own roles through scheduled calls.
- Worst-case: Can schedule upgrades to proxies administered by the timelock, or changes to timelock roles or delay. The standard delay is seven days, with a two-day minimum. Executed changes can cause permanent loss or denial of service. No independent party can cancel malicious proposals or force removal of the proposer, which also holds the canceller role, so a malicious proposer cannot be evicted.

Timelock Executor

- Trust Level: Untrusted.
- Permissions: Anyone can execute a ready, scheduled operation because in the deployment script, `address(0)` is granted the executor role.
- Worst-case: Can trigger a scheduled operation after its delay. Cannot change its payload or execute it early.

`MINTER_ROLE` Holder

- Trust Level: Fully trusted.
- Permissions: Calls `KAsset.mint()` without a token-level cap. The launch flow grants this role to the CCIP pool. The pool burns its own balance through permissionless `burn(uint256)`, so it does not need `BURNER_ROLE`.
- Worst-case: Can mint arbitrary supply to an eligible account. The owner can revoke the role, but cannot reverse tokens already transferred or redeemed.

`BURNER_ROLE` Holder

- Trust Level: Fully trusted.
- Permissions: Calls `KAsset.burn(address,uint256)` to burn another account's balance without allowance.
- Worst-case: Can irreversibly destroy any eligible holder's balance, up to that account's full balance.

`EMERGENCY_PAUSE_ROLE` Holder

- Trust Level: Partially trusted.
- Permissions: Calls `pause()`. Only the owner can call `unpause()`.
- Worst-case: Can stop transfers, minting, burning, and approvals until the owner unpauses the token.

`BLACKLIST_OPERATOR_ROLE` Holder

- Trust Level: Partially trusted.
- Permissions: Calls `KAssetBlacklist.setBlacklisted()` to blacklist or unblacklist batches of nonzero addresses.
- Worst-case: Can immediately block/unblock selected accounts from using every token connected to the chain's shared blacklist. The owner can revoke the operator's role and correct the entries.

`DEFAULT_ADMIN_ROLE` Holder on `KAsset`

- Trust Level: Fully trusted.
- Permissions: Not granted in the intended deployment. If granted before a CCIP administrator is established by the owner, the holder can register as the token’s CCIP administrator through the access-control flow and set or replace its pool. The role does not let the holder grant or revoke roles on `KAsset`, as those functions are owner-only.
- Worst-case: Can disrupt or redirect the token’s CCIP pool configuration.

`DEFAULT_ADMIN_ROLE` Holder on `KAssetBlacklist`

- Trust Level: Untrusted.
- Permissions: Not granted in the intended deployment. The owner can grant this role, but holding it does not let an account grant or revoke roles, as those functions are owner-only.
- Worst-case: No additional authority by itself. Any other roles held determine the holder’s permissions.

Token Holders and Other Callers

- Trust Level: Untrusted.
- Permissions: Use ERC-20 and EIP-2612 functions and burn their own tokens. Privileged calls require a role or ownership.
- Worst-case: Can transfer their own tokens or spend approved allowances, and burn only their own balance.

4.3 Collusion

The intended deployment places token ownership, blacklist ownership, pool ownership, and timelock proposal and cancellation under one signatory EOA. This gives it immediate control over issuance, burns, pauses, blacklist entries, and bridge configuration, plus delayed upgrade authority. The delay applies to upgrades and changes to timelock roles or configuration. Delegated role holders can separately mint, burn, pause, or blacklist within their permissions.

4.4 Upgradeability

`KAsset` and `KAssetBlacklist` use transparent proxies whose `ProxyAdmin` contracts are owned by the timelock. `KAssetTimelock` enforces a two-day minimum delay. The deployment script configures open execution and no external default administrator.

4.5 External Components

Underlying asset custody. The model assumes Kraken holds sufficient corresponding assets to back aggregate outstanding kAsset supply across chains and honors redemptions. The contracts neither custody nor verify these reserves. A shortfall or failure to redeem leaves holders with unbacked tokens.

Shared blacklist. Each token relies on its configured `KAssetBlacklist` for transfer and approval checks. An incorrect or unavailable blacklist can block token operations.

Kraken signatory service. The intended owner EOA is managed through Kraken's signatory service, which signs owner transactions prepared by the deployment tooling. The service and its approval controls are outside the reviewed contracts. Compromise or misuse could exercise the EOA's immediate token, blacklist, and pool-owner powers and propose or cancel timelock operations. Service unavailability prevents owner-signed administration and timelock proposals or cancellations.

Chainlink CCIP. `BurnMintTokenPool` from Chainlink CCIP v1.6.1 depends on the configured Router, OnRamp and OffRamp authorization, RMN proxy, remote pool registrations, and rate limits. These components must authenticate messages and enforce their checks correctly. Failure or compromise can halt transfers or allow invalid bridge messages. The pool owner can change its router and configuration.

CCIP configuration. The TokenAdminRegistry and RegistryModuleOwnerCustom must correctly register tokens and pools.

5 System Considerations

We leverage this section to highlight findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require a modification inside the project. Instead, they should raise awareness in order to improve the overall understanding for users and developers.

SC1 CCIP Administrator Does Not Follow KAsset Ownership

System Consideration | Version 1

The CCIP TokenAdminRegistry administrator is assigned separately from `KAsset` ownership. Transferring `KAsset` ownership does not update the registry, so the former owner retains pool administration until the registry role is transferred separately.

SC2 Role Administration Is Owner-Controlled

System Consideration | Version 1

`KAsset` and `KAssetBlacklist` override `grantRole()` and `revokeRole()` with `onlyOwner`. Although the contracts use OpenZeppelin `AccessControl`, role administration does not follow the roles' configured admin-role hierarchy: only the contract owner can grant or revoke roles. Role holders can still perform their assigned operational actions.

6 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

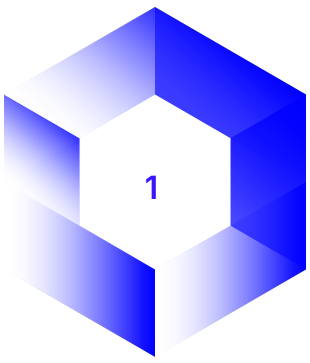
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	**Impact**		
	High	Medium	Low
High	**Critical**	**High**	**Medium**
Medium	**High**	**Medium**	**Low**
Low	**Medium**	**Low**	**Low**

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

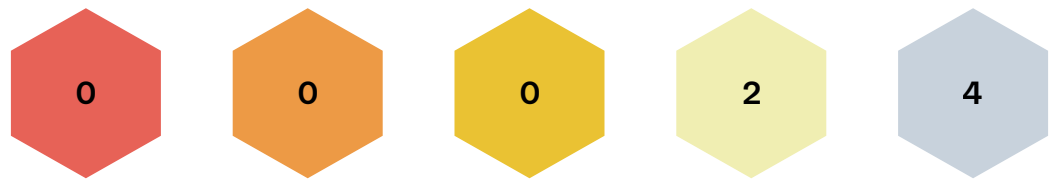
7 Findings

In this section, we describe the findings identified during the course of the engagement. The findings are categorized by severity as explained in the Terminology section. Below we provide an overview of the total number of findings per severity, as well as the number of open issues.



Open Issues

Total number of findings



Critical | High | Medium | Low | Informational

7.1 Overview

The following table lists all identified findings along with their severity and current status. A finding is considered resolved once it has been fully corrected or the specification has been changed accordingly. Non-informational findings with any other status, including partial fixes, acknowledgements, and accepted risks, remain open.

Low	#001	Bridged Transfers to New Holders May Require Manual Execution	Code Corrected
Low	#002	Renouncing All Proposer Roles Can Permanently Disable Upgrades	Code Part. Corrected / Risk Accepted
Info	#003	Context Initialization Should Be Included for Completeness	Acknowledged
Info	#004	Excessive Timelock Delay Can Block Future Upgrades	Acknowledged
Info	#005	Users Cannot Revoke Allowances to Blacklisted Spenders	Acknowledged
Info	#006	Users Cannot Revoke Spender Allowances While Paused	Acknowledged

7.2 Descriptions

#001 Bridged Transfers to New Holders May Require Manual Execution

Design | Low | Version 1 | Code Corrected

The configured lanes have recently been upgraded to CCIP 2.0.0 and remain compatible with the project's `BurnMintTokenPool` v1.6.1. Chainlink states that "the combined cost of these three calls must fit within the default limit of 90,000 gas" for the OffRamp's pre-mint `balanceOf`, `releaseOrMint`, and post-mint `balanceOf` calls (default and custom token pool gas limits). The delivery costs more gas than for a standard ERC-20 because each `KAsset` mint queries the shared blacklist proxy. A sampled automatic-delivery transaction was type 2 with an empty access list. Fork tests measured about 97,000 gas when minting to a new holder and 115,000 gas for the first mint on a chain. First-time recipients commonly have no balance on the destination chain, so these routine transfers may fail automatic execution and need manual execution. Raising the limit for a v1.x pool requires a Chainlink Labs configuration change. A v2.0 pool supports self-service configuration.

Code Corrected

In **Version 2**, `BurnMintPoolDeployer` deploys Chainlink's `BurnMintTokenPool` v2.0.0. This version allows the pool owner to configure the destination gas overhead per lane through `applyTokenTransferFeeConfigUpdates()`, without requiring a Chainlink Labs configuration change.

#002 Renouncing All Proposer Roles Can Permanently Disable Upgrades

Design | Low | Version 1 | Code Partially Corrected / Risk Accepted

The timelock is its own sole admin, so adding a proposer requires a scheduled timelock operation. If all proposers renounce `PROPOSER_ROLE` (the deployment script configures only one), no new operations can be scheduled. Unless a pending operation can restore proposer access, upgrades to the blacklist and token contracts controlled by the timelock are permanently disabled.

Note that the same reasoning applies to `EXECUTOR_ROLE`. The deployment script grants this role to `address(0)`, allowing anyone to execute ready operations. While `address(0)` cannot renounce the role, revoking it without retaining another executor would permanently disable execution.

Code Partially Corrected

In **Version 2**, `KAssetTimelock` overrides `renounceRole()` to always revert. No role can be renounced anymore, including by the timelock itself through a scheduled operation. The risk remains that a scheduled operation revokes all proposers or executors, or grants `DEFAULT_ADMIN_ROLE` to another address that can then revoke them without any delay.

Risk Accepted

Kraken accepts the remaining risk and states:

> Removing a proposer now takes a timelocked revokeRole. The executor case is accepted: revoking address(0) as executor takes a timelocked operation from the trusted proposer, visible and cancelable for the full delay.

#003 Context Initialization Should Be Included for Completeness

Informational **Version 1** **Acknowledged**

In `KAsset.initialize()` and `KAssetBlacklist.initialize()`, `__Context_init()` is omitted from the inherited initialization chain. In the pinned OpenZeppelin 5.5.0 dependency, it is empty, as are the called `__AccessControl_init()` routines and `KAsset`'s `__Pausable_init()`, so the omission has no current runtime effect. Calling the context initializer would complete the inherited setup for consistency.

Acknowledged

Kraken acknowledges the issue and states:

> The omitted initializers are empty in the pinned OpenZeppelin 5.5.0, so there is no runtime effect.

#004 Excessive Timelock Delay Can Block Future Upgrades

Informational **Version 1** **Acknowledged**

`KAssetTimelock` enforces a minimum delay of two days in its constructor and `updateDelay()`, but sets no upper bound. An excessively large delay can therefore block newly scheduled operations, including contract upgrades, for an impractically long time.

Reducing the delay also requires a timelocked operation subject to the current delay, so a misconfiguration cannot be promptly corrected. Existing token operations remain unaffected.

Acknowledged

Kraken acknowledges the issue and states:

> A delay change is itself a timelocked operation from the trusted proposer, visible for the current delay and cancelable before it executes.

#005 Users Cannot Revoke Allowances to Blacklisted Spenders

Informational **Version 1** **Acknowledged**

In `KAsset`, `approve()` and `permit()` revert when the spender is blacklisted, including when a holder tries to set an existing allowance to zero. The allowance remains active and becomes usable if the spender is later unblacklisted.

Acknowledged

Kraken acknowledges the issue and states:

> Intended by the specification, which gates approvals by the blacklist. The allowance cannot be spent while the spender is blacklisted, since transfers check the caller, and the holder can revoke it once the spender is removed. Removal is a deliberate compliance decision.

#006 Users Cannot Revoke Spender Allowances While Paused

Informational **Version 1** **Acknowledged**

In `KAsset`, `approve()` and `permit()` revert while the token is paused, including when a holder tries to set an existing allowance to zero. The allowance remains active and can be spent after the token is unpaused.

Acknowledged

Kraken acknowledges the issue and states:

> Intended by the specification, which blocks all approvals while paused. The allowance cannot be spent while paused, and the holder can revoke it after unpause.

8 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.