Crypto Legacy Audit Report

Prepared by kamensec

Version 1.0

Lead Auditors

kamensec

June 27, 2025

Contents

1 About Kamensec

Kamensec is an independent auditor dedicated to web3 security.

The goal is to increase deployment readiness and reduce attack surface area of smart contract. To see past work go to kamensec.xyz.

2 Disclaimer

Kamensec makes all effort but holds no responsibility for the findings of this security review. Kamensec does not provide any guarantees relating to the function of the components in scope. Kamensec makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

3 Risk Classification

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

4 Protocol Summary

Summary of the core protocol functionality and particular areas of interest within the audit scope.

5 Audit Scope

Summary of the audit scope and any specific inclusions/exceptions.

6 Executive Summary

Over the course of 11 days, the kamensec conducted an audit on the Crypto Legacy smart contracts provided by kamensec. In this period, a total of 8 issues were found.

Summary of the audit findings and any additional executive comments.

Summary

Project Name	Crypto Legacy
Repository	cryptolegacy-contracts
Commit	4558f24ad5c9...
Audit Timeline	Jun 9th - Jun 23rd
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	2
Low Risk	4
Informational	2
Total Issues	8

Summary of Findings

Permanent Pause Denial of Service	Verified Fix
Missing Multisig Threshold Validation	Verified Fix
Unfair Distribution of Negative Rebasing Losses	Acknowledged
Fee Collection Gap in Recovery and Direct Claim Paths	Acknowledged
Missing Error Handling in Fee Withdrawal	Acknowledged
Redundant NFT Transfer Path	Acknowledged
Division by Zero in Vesting Calculation	Verified Fix
Unbounded Beneficiary Array	Acknowledged

7 Findings

7.1 Medium

7.1.1 Permanent Pause Denial of Service

Context: CryptoLegacyBasePlugin.setPause()

Description: If the owner calls `setPause(true)` after a `initiateChallenge()` request but when `block.timestamp < cls.distributionStartAt + cls.challengeTimeout` the contract becomes permanently paused with no recovery mechanism.

Once distribution is ready, all `onlyOwner` functions revert, including the ability to unpause. This blocks all beneficiary claims and token distributions permanently.

Recommendation: Implement one of the following solutions:

1. Add a mechanism for beneficiaries to unpause after distribution starts
2. Automatically unpause when distribution becomes ready
3. Add a time-limited pause that automatically expires

CryptoCustoms: Resolved in commit fefb375c by preventing setPaused when challenge period has started.

7.1.2 Missing Multisig Threshold Validation

Context: LibSafeMinimalBeneficiaryMultisig._setConfirmations(), LibCryptoLegacy._setCryptoLegacyToBeneficiaryRegistry(),CryptoLegacyBasePlugin._setBeneficiaries()

Description: While `LibSafeMinimalBeneficiaryMultisig._setConfirmations()` validates that `_requiredConfirmations <= _voters.length`, the functions `LibCryptoLegacy._setCryptoLegacyToBeneficiaryRegistry()` and `CryptoLegacyBasePlugin._setBeneficiaries()` do not perform this validation when modifying beneficiaries. This can lead to states where the required confirmations exceed the number of available voters, making multisig proposals impossible to execute.

Recommendation: Add validation in all functions that modify beneficiaries to ensure the multisig threshold remains achievable:

```
require(multisigStorage.requiredConfirmations <= cls.beneficiaries.length(), "Confirmations exceed
↪  voters");
```

CryptoCustoms: Resolved in commit 62107dca with a default upper bound on confirmations set to the beneficiary length.

7.2 Low

7.2.1 Unfair Distribution of Negative Rebasing Losses

Context: LibCryptoLegacy._tokenPrepareToDistribute()

Description: When partial token distribution occurs and negative rebasing happens between beneficiary claims, losses are not socialized fairly.

For example, with 1000 tokens total but only 500 initially distributed to 2 equal beneficiaries: if one claims 250 tokens, then 100% negative rebasing occurs, the second beneficiary receives nothing while the first beneficiary can later claim their share from the remaining 500 tokens when they become available.

Despite having 1000 tokens, only 750 where claimed, disproportionately between 2 users, where the first gets 500, and the latter gets 250.

Recommendation: Ensure that deposits occur in completion into the legacy contract prior to negative rebasing calculations being processed.

CryptoCustoms: Won't fix - acknowledged that manipulative rebasing tokens will always present distribution challenges. Document this limitation clearly or implement a mechanism to track and fairly distribute rebasing losses across all beneficiaries.

7.2.2 Fee Collection Gap in Recovery and Direct Claim Paths

Context: LegacyRecoveryPlugin.lrWithdrawTokensFromLegacy()

Description: Outside of initial build fee collection, fees are only collected through update fees through `update()` calls or during `beneficiaryClaim()`. If an owner uses the recovery mechanism to withdraw funds without ever calling `update()`, the BuildManager never receives its intended periodic fees.

Recommendation: Consider adding fee collection to recovery paths.

CryptoCustoms: Won't fix - acknowledged as an acceptable trade-off to avoid blocking emergency recovery.

7.2.3 Missing Error Handling in Fee Withdrawal

Context: FeeRegistry.withdrawAccumulatedFee()

Description: The `withdrawAccumulatedFee()` function attempts to send fees to all beneficiaries in a loop. If any beneficiary's low-level call reverts (e.g., due to a malicious receiver contract or gas costs changes causing reverts), the entire withdrawal fails, potentially locking accumulated fees permanently.

Recommendation: Implement try-catch error handling or a pull-based withdrawal pattern to prevent a single failing beneficiary from blocking all fee withdrawals.

CryptoCustoms: Won't fix - fee beneficiaries are trusted entities set by the owner, if something like this occurs that beneficiary can be removed.

7.2.4 Redundant NFT Transfer Path

Context: NftLegacyPlugin.transferNftTokensToLegacy()

Description: The `beneficiaryClaimNft()` function can transfer NFTs directly from any approved address to the beneficiary after the claim delay, making `transferNftTokensToLegacy()` redundant. Beneficiaries can bypass the intended two-step process (transfer to legacy, then claim) by waiting for the claim delay and calling `beneficiaryClaimNft()` directly. This may lead to unnecessary gas consumption by users.

Recommendation: Either enforce that NFTs must be held by the legacy contract before claiming, or remove the redundant `transferNftTokensToLegacy()` function to simplify the codebase.

CryptoCust: Won't fix - acknowledged as intentional flexibility in the design.

7.3 Informational

7.3.1 Division by Zero in Vesting Calculation

Context: LibCryptoLegacy._getVestedAndClaimedAmount()

Description: The vesting calculation in `_getVestedAndClaimedAmount()` performs division by `bc.vestingPeriod` without checking if it's zero. When a beneficiary is configured with `vestingPeriod = 0` (intended for immediate vesting), the following calculation causes a division by zero:

```
    vestingBps = uint64(block.timestamp) > _endDate ? LibCryptoLegacy.SHARE_BASE :
    ↪  (uint64(block.timestamp) - _startDate) * LibCryptoLegacy.SHARE_BASE / bc.vestingPeriod;
```

Note that in it's current state a division by zero is only possible when `block.timestamp == _endDate`, for the immediate timestamp the division by zero will prevent claims from finalizing.

Recommendation: Add a check for zero vesting period and handle it as immediate vesting:

```
        if (_startDate > uint64(block.timestamp)) {
            vestingBps = 0;
        } else if (uint64(block.timestamp) >= _endDate) {
            vestingBps = LibCryptoLegacy.SHARE_BASE;
        } else {
            vestingBps = uint64(block.timestamp) > _endDate ? LibCryptoLegacy.SHARE_BASE :
            ↪  (uint64(block.timestamp) - _startDate) * LibCryptoLegacy.SHARE_BASE / bc.vestingPeriod;
        }
```

CryptoCustoms: Resolved in commit d54e989c.

7.3.2 Unbounded Beneficiary Array

Context: CryptoLegacyBasePlugin._setBeneficiaries()

Description: There's no maximum limit on the number of beneficiaries that can be added. While `_setBeneficiaries()` itself may run out of gas with too many beneficiaries, plugins with more gas-intensive operations could fail before the base function does, creating a denial of service where beneficiaries cannot be modified.

Recommendation: Implement a reasonable upper bound on the number of beneficiaries to ensure all operations remain within gas limits.

CryptoCust: Won't fix - plugins are replaceable and upgradeable, allowing future versions to handle larger beneficiary sets if needed.