MixBytes()

CryptoLegacy Security Audit Report

JULY 15, 2025

Table of Contents

1. Introduction

1.1 Disclaimer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

1.2 Executive Summary

CryptoLegacy is a multichain dApp with a minimalist architecture, designed to secure crypto inheritance in the event of unforeseen circumstances. It uses individual smart contracts that do not store funds but provide maximum security by transferring assets from primary wallets to designated beneficiaries in predefined proportions. The system also supports recovery options via assigned guardians and recovery addresses.

The audit was conducted over 8 days by 4 auditors, involving a thorough manual code review of the scope and analysis via automated tools.

We analyzed the following potential attack vectors:

- **Parameters Misconfiguration**
  We checked whether updates to protocol parameters (e.g., beneficiary list, distribution config, plugin list) could create configurations that unintentionally lock the system or make future updates impossible due to revert conditions or excessive gas costs.

- **Fee-Model Bypass**
  We assessed whether the mechanism enforcing time-based fees could be bypassed by participants who maintain control or delay actions without incurring the intended costs.

- **Stale-Quorum Voting**
  We reviewed whether outdated or removed members could continue to affect quorum or influence voting outcomes due to stale state or lack of revocation enforcement.

- **Percentage-Math Overflow**
  We verified whether any arithmetic using percentages or basis points could overflow, underflow, or lead to unexpected rounding behavior that affects system correctness.

- **Gas-Limited ETH Transfers**
  We assessed ETH transfers that rely on the 2,300 gas stipend to see if they could fail when sent to contracts with fallback logic that requires more gas.

- **Fixed-Gas External Call**
  We examined external calls made with fixed gas amounts to determine whether those limits could cause failures, especially under upstream upgrades or increased gas costs.

- **Unbounded-Loop Exhaustion**
  We looked for loops over dynamic or growing arrays that may lead to out-of-gas reverts, potentially rendering some protocol paths unusable.

- **Silent-Failure Masking**
  We reviewed code paths that catch low-level call failures or use unchecked call results to identify if critical failures could be silently ignored.

- **Shared-Slot Collision**
  We assessed storage layout across delegate-call-based modules to determine if storage slot collisions could occur, leading to state corruption.

- **Selector Shadowing**
  We evaluated whether duplicate function selectors across facets could unintentionally override each other, causing unexpected behavior.

- **Raw-Call Accounting Drift**
  We checked if low-level call results were accurately tracked to prevent marking failed calls as successful, especially in proxy or execution contexts.

- **Sentinel-Index Misuse**
  We checked for patterns where lookups return sentinel values (e.g., max uint256 or 0) and whether those could be misinterpreted by downstream logic.

- **Bulk-Transfer Failures**
  We analyzed whether bulk-transfer methods could fail entirely due to a single failure within the batch, resulting in partial execution or DoS.

- **Re-entrancy**
  We examined if calls across plugins could re-enter state-changing logic in ways that bypass reentrancy protections.

- **Unvetted-Plugin Injection**
  We reviewed logic for facet/plugin registration to assess whether untrusted or misconfigured code could be introduced via upgrades by users.

- **Deployment Attack**
  We verified that the deployment procedures do not allow an attacker to intervene or perform malicious actions.

- **Correct Handling of Beneficiary Switch Procedure**
  We verified the correct handling of the procedure for switching the beneficiary to another hash, in the context of fair token claiming and quorum integrity.

- **Handling of Non-Standard Token Behaviors and Unsolicited Transfers**
  We verified that the system correctly handles rebaseable and fee-on-transfer tokens, as well as unsolicited donations, ensuring accurate accounting and protocol stability. We also confirmed fair distribution among beneficiaries, including vesting, under these conditions.

- **Hashed Address Obfuscation**
  We examined the system's use of Keccak-256 hashes for address obfuscation and assessed whether this approach protects the privacy of users.

Besides these, other vectors from our internal security checklist were also reviewed as part of the assessment.

Additionally, the audited codebase interacts with external contracts such as CryptoLegacyBuildManager.sol and BeneficiaryRegistry.sol, which were outside the scope of this audit and therefore were not reviewed.

To enhance the overall robustness and quality of the codebase, we suggest considering the following improvements:

- Use **ERC-7201** for storage layout naming instead of hashing plain strings, to improve consistency and maintainability.
- Replace magic numbers with named constants to enhance code readability and simplify future maintenance.
- Optimize loops to reduce gas costs. Specific cases and recommendations are outlined in the corresponding finding below.
- Implement a function to modify the cls.defaultFuncDisabled variable. Currently, CryptoLegacyBasePlugin.beneficiaryClaim cannot be disabled. It's important to ensure this configurability is added in future versions.
- Remove unused and duplicate import statements to improve code readability.

These and other code quality improvements are also reflected in the findings section.

1.3 Project Overview

Summary

Title	Description
Client Name	CryptoLegacy
Project Name	CryptoLegacy
Type	Solidity
Platform	EVM
Timeline	22.04.2025–25.06.2025

Scope of Audit

File	Link
contracts/plugins/ TrustedGuardiansPlugin.sol	TrustedGuardiansPlugin.sol

File	Link
contracts/plugins/CryptoLegacyBasePlugin.sol	CryptoLegacyBasePlugin.sol
contracts/plugins/LegacyRecoveryPlugin.sol	LegacyRecoveryPlugin.sol
contracts/libraries/LibCryptoLegacy.sol	LibCryptoLegacy.sol
contracts/libraries/LibCryptoLegacyPlugins.sol	LibCryptoLegacyPlugins.sol
contracts/libraries/LibTrustedGuardiansPlugin.sol	LibTrustedGuardiansPlugin.sol
contracts/libraries/LibSafeMinimalMultisig.sol	LibSafeMinimalMultisig.sol
contracts/SignatureRoleTimelock.sol	SignatureRoleTimelock.sol
contracts/CryptoLegacyOwnable.sol	CryptoLegacyOwnable.sol
contracts/CryptoLegacy.sol	CryptoLegacy.sol
contracts/CryptoLegacyDiamondBase.sol	CryptoLegacyDiamondBase.sol
contracts/libraries/LibCreate3.sol	LibCreate3.sol
contracts/libraries/LibCryptoLegacyDeploy.sol	LibCryptoLegacyDeploy.sol
contracts/Create3Factory.sol	Create3Factory.sol
contracts/CryptoLegacyFactory.sol	CryptoLegacyFactory.sol

Versions Log

Date	Commit Hash	Note
22.04.2025	e409929501d5b78941bc78411f1fa3da36249a94	Initial commit
27.05.2025	4558f24ad5c9aecf41751643e7e2ce9b1686092e	Re-audit commit

Date	Commit Hash	Note
29.05.2025	137d597231a13c1d4951dfb847581d7ba577d369	Second re-audit commit
09.06.2025	36c75d8528ac3685bf18196a03a01d5072d33644	Third re-audit commit
23.06.2025	d3ae300a8e07374b425408e84e1101a18a39d4f4	Fourth re-audit commit
24.06.2025	d79ade5f5a9f41314780ee99a8e38ff8351bf75f	Fifth re-audit commit
25.06.2025	fc6242e7ab7d8c24a0bf7e426ca2c8d5cc1f3c1a	Sixth re-audit commit
30.06.2025	df587b4f3efb4878f047cf4d8e5982933567ea7d	Deploy commit

Mainnet Deployments

File	Address	Blockchain
CryptoLegacyFactory.sol	0x05Fd6C...1C888462	Ethereum Mainnet
CryptoLegacyBasePlugin.sol	0x75B8e5...42453D4b	Ethereum Mainnet
TrustedGuardiansPlugin.sol	0x39C755...4e9d6462	Ethereum Mainnet
LegacyRecoveryPlugin.sol	0x416A80...92995ab3	Ethereum Mainnet
SignatureRoleTimelock.sol	0x7Fa59e...46f82b50	Ethereum Mainnet
CryptoLegacyFactory.sol	0x05Fd6C...1C888462	Arbitrum One
CryptoLegacyBasePlugin.sol	0x75B8e5...42453D4b	Arbitrum One
TrustedGuardiansPlugin.sol	0x39C755...4e9d6462	Arbitrum One
LegacyRecoveryPlugin.sol	0x416A80...92995ab3	Arbitrum One
SignatureRoleTimelock.sol	0x7Fa59e...46f82b50	Arbitrum One
CryptoLegacyFactory.sol	0x05Fd6C...1C888462	Optimism

File	Address	Blockchain
CryptoLegacyBasePlugin.sol	0x75B8e5...42453D4b	Optimism
TrustedGuardiansPlugin.sol	0x39C755...4e9d6462	Optimism
LegacyRecoveryPlugin.sol	0x416A80...92995ab3	Optimism
SignatureRoleTimelock.sol	0x7Fa59e...46f82b50	Optimism
CryptoLegacyFactory.sol	0x05Fd6C...1C888462	Base
CryptoLegacyBasePlugin.sol	0x75B8e5...42453D4b	Base
TrustedGuardiansPlugin.sol	0x39C755...4e9d6462	Base
LegacyRecoveryPlugin.sol	0x416A80...92995ab3	Base
SignatureRoleTimelock.sol	0x7Fa59e...46f82b50	Base
CryptoLegacyFactory.sol	0x05Fd6C...1C888462	Linea
CryptoLegacyBasePlugin.sol	0x75B8e5...42453D4b	Linea
TrustedGuardiansPlugin.sol	0x39C755...4e9d6462	Linea
LegacyRecoveryPlugin.sol	0x416A80...92995ab3	Linea
SignatureRoleTimelock.sol	0x7Fa59e...46f82b50	Linea

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	**Project Architecture Review:** · Review project documentation · Conduct a general code review · Perform reverse engineering to analyze the project's architecture based solely on the source code · Develop an independent perspective on the project's architecture · Identify any logical flaws in the design OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.
	Code Review with a Hacker Mindset: · Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities. · Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead. · Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws. · Review test cases and in-code comments to identify potential weaknesses. OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.
	Code Review with a Nerd Mindset: · Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits. · Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.

Stage	Scope of Work
	Consolidation of Auditors' Reports: · Cross-check findings among auditors · Discuss identified issues · Issue an interim audit report for client review OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.
Re-audit	**Bug Fixing & Re-Audit:** · The client addresses the identified issues and provides feedback · Auditors verify the fixes and update their statuses with supporting evidence · A re-audit report is generated and shared with the client OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.
Final audit	**Final Code Verification & Public Audit Report:** · Verify the final code version against recommendations and their statuses · Check deployed contracts for correct initialization parameters · Confirm that the deployed code matches the audited version · Issue a public audit report, published on our official GitHub repository · Announce the successful audit on our official X account OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likehood: High	Critical	High	Medium
Likehood: Medium	High	Medium	Low
Likehood: Low	Medium	Low	Low

Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

Likelihood

- **High** – The event has a 50–60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10–20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	1
Medium	5
Low	28

Findings Statuses

ID	Finding	Severity	Status
H-1	Rebaseable Tokens Cause Unfair Vesting and Claim Failures	High	Fixed
M-1	Incorrect Handling of Guardian Votes in TrustedGuardiansPlugin	Medium	Fixed
M-2	Front-running beneficiarySwitch Can Block Beneficiary Removal (DoS)	Medium	Fixed
M-3	Owner Can Reset Challenge Timer and Evade Update Fee	Medium	Fixed
M-4	Integer Overflow in Default Guardian Confirmations Calculation	Medium	Fixed
M-5	Incorrect lastBalance update in treasury token transfer for rebasing tokens	Medium	Fixed
L-1	Manual Role Renounce Leads to SignatureRoleTimelock Inconsistency	Low	Fixed
L-2	Unnecessary renounceOwnership Function	Low	Fixed
L-3	transfer method is used instead of call	Low	Fixed
L-4	Execution Order Race Condition in SignatureRoleTimelock	Low	Acknowledged

L-5	TrustedGuardiansPlugin._setGuardiansConfig should have a below bound check for guardiansChallengeTimeout	Low	Fixed
L-6	Lack of check for arrays length	Low	Fixed
L-7	Gas Inefficiency Due to Linear Searches in View Functions	Low	Acknowledged
L-8	Weak Privacy Due to Hash-Only Identity Obfuscation	Low	Fixed
L-9	lastUpdateAt Can Be Updated by Non-Owner	Low	Acknowledged
L-10	Missing Sanity Check for guardiansThreshold	Low	Fixed
L-11	Redundant Casting in Function Selector Retrieval	Low	Fixed
L-12	Missing Two-Step Ownership Transfer	Low	Fixed
L-13	Unnecessary Condition Inversion	Low	Fixed
L-14	Hard-Coded Gas Limits Risk Future Compatibility	Low	Fixed
L-15	Unused or Duplicate Imports Increase Deployment Cost	Low	Fixed
L-16	Silent catch Blocks Hide Runtime Errors	Low	Fixed
L-17	Hard-coded Values Reduce Readability	Low	Fixed
L-18	Minor Gas Inefficiencies via Redundant Operations	Low	Fixed
L-19	Timelock Duration Check Missing, Allows Near-Zero Delay	Low	Acknowledged
L-20	Unvalidated _refShare Can Cause Reverts	Low	Fixed
L-21	Standard ReentrancyGuard Inheritance Risks Diamond Storage Collision	Low	Fixed
L-22	Incorrect Handling of Beneficiary Votes in TrustedGuardiansPlugin	Low	Acknowledged
L-23	Beneficiary Can Avoid Paying Protocol Fees	Low	Acknowledged

L-24	Operations Allowed on Proposals with NOT_EXIST Status	Low	Fixed
L-25	Inconsistent Native Value Handling in LegacyRecoveryPlugin	Low	Fixed
L-26	Premature Owner Update in Beneficiary Registry During Ownership Transfer	Low	Fixed
L-27	Improper Handling of Payable Proposals	Low	Fixed
L-28	Incorrect handling of fee-on-transfer and rebasing tokens penalizes single beneficiary	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

H-1	Rebaseable Tokens Cause Unfair Vesting and Claim Failures		
Severity	High	**Status**	Fixed in 36c75d85

Description

This issue has been identified within the `_claimTokenWithVesting` function of the **CryptoLegacyBasePlugin** contract, specifically related to its interaction with rebaseable ERC-20 tokens.

The function calculates the `amountToDistribute` at the time of each claim by summing the contract's *current* token balance and the `totalClaimedAmount`. This method leads to incorrect outcomes when the token balance changes due to rebasing events occurring *between claims* by different beneficiaries or between partial claims by the same beneficiary.

Two primary problems arise:

1. **Unfair Distribution:** If a positive rebase occurs after partial claims, the calculation based on the current balance unfairly distributes the rebase gains. Depending on the claim timing relative to the rebase, this can disproportionately benefit or penalize beneficiaries compared to a scenario where gains are distributed across all original shares.
   - For example, if A and B have 50% shares each of 1000 tokens, A claims 250.
   - Later, the token exchange rate doubles, meaning the remaining 750 tokens rebase to 1500, and claimed 250 tokens by A rebase to 500.
   - When B claims, `amountToDistribute` becomes `1500 + 250 = 1750`. B gets `50% of 1750 = 875`. A claims the rest and gets `1750 * 50% - 250 = 625`.
   - A received `625 + 500 = 1125` total, B received `875` total.
2. **Failed Claims:** If a negative rebase occurs *after* an early claim, the recalculated `amountToDistribute` might imply a remaining entitlement exceeding the actual tokens held by the contract. This will cause subsequent `safeTransferFrom` calls for other beneficiaries to revert due to insufficient balance.

Moreover, it was identified that if `lrWithdrawTokensFromLegacy` in **LegacyRecoveryPlugin** is called to remove tokens from the contract after some tokens have already been partially claimed by beneficiaries, it may lead to issues when those beneficiaries attempt to claim their remaining tokens. In such cases, the vestedAmount might be less than the claimedAmount, potentially causing a temporary underflow.

The issue is classified as **High** severity because it breaks the principle of fair proportional distribution for vested rebaseable tokens and can lead to claim failures.

Recommendation

We recommend calculating entitlements based on fixed shares applied to the current contract balance at the time of claim, ensuring proportional distribution of rebases across all originally vested tokens.

Client's Commentary:

Client: The failed claims scenario caused by negative rebases has been fully resolved. Specifically, we addressed the case where a rebase could reduce the token balance below the calculated entitlement, leading to unclaimable or locked tokens. As for the unfair distribution scenario involving positive rebases, we believe this does not require changes, based on the following considerations: The observed behavior is a consequence of using rebaseable tokens in a pull-based, asynchronous claim system. Any such system would experience similar effects when token balances change between claims. CryptoLegacy makes no assumptions about token supply stability and simply reflects the current state of the token at the time of claim, ensuring deterministic behavior without overengineering around external token mechanics. Most beneficiaries interact with standard ERC-20 tokens. Designing for perfect rebase fairness would introduce complexity and constraints irrelevant for the vast majority of use cases. Claims remain accurate relative to the contract's live balance, and all entitlements are correctly enforced within the rules of the vesting schedule and distribution logic. This behavior is consistent with how other decentralized systems handle dynamic tokens and does not compromise the correctness, safety, or reliability of the platform. 4558f24a

MixBytes: Another problem that arises is that with a minimal rebase (down to 1e9), an underflow may occur during claiming. This happens when the actual balance becomes lower than `td.amountToDistribute`, due to the second condition: `(balanceDiff * 1 gwei / bal) < 1e3`.

2.3 Medium

M-1	Incorrect Handling of Guardian Votes in TrustedGuardiansPlugin		
Severity	Medium	**Status**	Fixed in 4558f24a

Description

The TrustedGuardiansPlugin does not correctly handle cases where a guardian who has already voted is later removed from the guardian set.
Currently, the plugin simply checks if the number of votes (guardiansVoted.length) meets the required threshold, without verifying whether those votes came from currently active guardians. As a result, votes from removed guardians may still be counted, potentially causing quorum to be reached prematurely.
In addition, public view functions that expose the list of votes or voting guardians may return outdated or misleading data, since they don't exclude votes from removed guardians. This may cause incorrect assumptions when the data is consumed off-chain.

Recommendation

We recommend updating the vote validation logic so that only votes from guardians who are active at the time of resolution are taken into account when checking for quorum.
Additionally, we suggest reviewing any public view functions that expose voting data, and considering adjustments to ensure they return only valid, up-to-date votes.

Client's Commentary:

This issue has been fully resolved in the smart contract: removed trusted guardians are now automatically cleared from guardiansVoted, ensuring their votes are no longer counted in quorum calculations or shown in public state.

M-2	Front-running beneficiarySwitch Can Block Beneficiary Removal (DoS)		
Severity	Medium	**Status**	Fixed in 4558f24a

Description

This issue has been identified within the _setBeneficiaries function of the **CryptoLegacyBasePlugin** contract.

A malicious beneficiary can exploit the beneficiary removal process. If the owner submits a transaction to remove a beneficiary by setting their shareBps to 0, the targeted beneficiary can observe this transaction in the mempool and front-run it by calling beneficiarySwitch. This moves their beneficiary configuration to a different hash identifier *before* the owner's transaction is processed.

When the owner's setBeneficiaries transaction executes, it attempts to adjust the share sum based on the *original* beneficiary hash. Since the beneficiary has switched their hash, the internal accounting for the total shareSum becomes incorrect. This causes the check ensuring the sum equals SHARE_BASE to fail and revert the transaction.

The issue is classified as **Medium** severity because it allows a malicious beneficiary to prevent their own removal via front-running, thus disrupting a core governance mechanism.

Recommendation

We recommend modifying setBeneficiaries to identify beneficiaries for removal using a stable identifier (e.g., original hash) that is resistant to changes via beneficiarySwitch.

Client's Commentary:

This issue has been fully resolved — we implemented internal protections to ensure that front-running via beneficiarySwitch no longer disrupts the setBeneficiaries logic or causes unintended reverts.

M-3	Owner Can Reset Challenge Timer and Evade Update Fee		
Severity	Medium	**Status**	Fixed in 4558f24a

Description

This issue has been identified within the resetGuardianVoting function of the **TrustedGuardiansPlugin** contract.

The owner can call resetGuardianVoting to reset the distributionStartAt timer, effectively cancelling a pending challenge initiated by beneficiaries due to owner inactivity.

Crucially, this reset does not require the owner to pay the update fee normally associated with resolving such a challenge. This allows the owner to repeatedly avoid inactivity penalties and associated fees.

The issue is classified as **Medium** severity because it undermines the economic incentive designed to ensure owner liveness, breaking a core part of the challenge mechanism.

Recommendation

We recommend requiring the update fee payment within resetGuardianVoting if a beneficiary challenge is active, or decoupling the challenge state management between TrustedGuardiansPlugin and CryptoLegacyBasePlugin.

Client's Commentary:

This issue has been fully resolved — we have added update fee enforcement to the resetGuardianVoting function when it is called during an active challenge period, ensuring alignment with the intended incentive structure.

M-4	Integer Overflow in Default Guardian Confirmations Calculation		
Severity	Medium	**Status**	Fixed in 137d5972

Description

An integer overflow can occur when computing the default number of confirmations required for guardian votes. The code unconditionally casts the length of the guardian set (a `uint256`) to `uint8` before passing it into `LibSafeMinimalMultisig._calcDefaultConfirmations`:

```
uint8(
    LibSafeMinimalMultisig._calcDefaultConfirmations(
        uint8(
            guardians.length())))
```

If there are 256 or more guardians (or, when no explicit guardians are set, beneficiaries), the `uint8` cast wraps the length modulo 256 (e.g. 256→0, 257→1), causing `_calcDefaultConfirmations` to return an incorrect threshold (e.g. for 0 → 1 confirmation required). This breaks the intended multisig-majority logic.

For example:

- With **256 guardians**, `uint8(guardians.length())` becomes `0`, so the threshold becomes `(0 / 2) + 1 = 1`;
- With **257 guardians**, the length wraps to `1`, resulting in `(1 / 2) + 1 = 1`.

In both cases, only a **single signature** is enough to meet the threshold — defeating the security purpose of a multisig setup.

While having ≥256 guardians or beneficiaries is a highly unlikely scenario in most real-world deployments, it is theoretically possible, and nothing in the current code prevents it.

Recommendation

We recommend refactoring the confirmation calculation logic to use a data type that can safely accommodate a sufficiently large number of participants—such that the practical limit on guardians or beneficiaries can never cause overflow. We recommend using at least `uint32` for both the input and output of the threshold calculation:

```
function _calcDefaultConfirmations(uint32 _voterCount)
    internal
    pure
    returns (uint32)
{
    return (_voterCount / 2) + 1;
}
```

M-5	Incorrect lastBalance update in treasury token transfer for rebasing tokens		
Severity	Medium	**Status**	Fixed in d79ade5f

Description

In the LibCryptoLegacy library's _transferTreasuryTokensToLegacy function, td.lastBalance is updated by adding the difference between the post-transfer balance and the prior balance. This logic miscalculates lastBalance under rebasing tokens, leading to incorrect negative rebase handling.

For example, assume after the previous claim td.lastBalance is 1,000. A positive rebase increases the contract's token balance to 1,100. Then, a treasury transfer of 1,000 tokens raises the balance to 2,100. The code sets td.lastBalance = 1000 + (2100 − 1100) = 2000 instead of the actual 2,100. If a subsequent 0.5% negative rebase reduces the actual balance to 2,089.5, the stale td.lastBalance (2,000) is below the new balance, causing _tokenPrepareToDistribute to detect a negative rebase and incorrectly increase the recorded totalClaimed, reducing future claimable amounts for beneficiaries.

This misallocation violates vesting invariants when rebasing tokens are involved, warranting **Medium** severity because it corrupts distribution without enabling direct asset theft.

Recommendation

We recommend reverting the lastBalance update to the previous implementation:

```
-    td.lastBalance += IERC20(_tokens[i]).balanceOf(address(this))
-                      - balanceBefore;
+    td.lastBalance = IERC20(_tokens[i]).balanceOf(address(this));
```

This change ensures lastBalance always matches the actual post-transfer balance and prevents erroneous claim adjustments caused by rebasing.

2.4 Low

L-1	Manual Role Renounce Leads to SignatureRoleTimelock Inconsistency		
Severity	Low	**Status**	Fixed in 4558f24a

Description

A role is granted to a user during the SignatureRoleTimelock.setRoleAccounts call. However, the user can call AccessControl.sol#L179-L183, which creates two issues:

1. The admin cannot remove this user from roleAccounts using SignatureRoleTimelock.setRoleAccounts due to a SignatureRoleTimelock.sol#L169-L170 check. This causes a temporary denial of service in role management. The admin must first call AccessControl.sol#L144-L146 before the user can be removed.
2. The admin may **mistakenly** add a user to the **roleAccounts** after they have renounced their role, leading to duplicated entries. Removing the user clears only one instance, requiring extra steps to fully clean up.

Recommendation

We recommend restricting users from calling AccessControl.renounceRole.

Client's Commentary:

4558f24a

L-2	Unnecessary renounceOwnership Function		
Severity	Low	**Status**	Fixed in 4558f24a

Description

The CryptoLegacyBasePlugin contract includes a standard CryptoLegacyBasePlugin.sol#L177-L179 function, allowing the owner to transfer ownership to the zero address.

In the context of **CryptoLegacy**, where ownership governs sensitive operations such as beneficiary configuration, fee management, and recovery logic, removing the owner can lead to an irrecoverable loss of control over critical features. From a business logic perspective, it is unlikely that a legitimate owner would intentionally relinquish these capabilities, as doing so would undermine the purpose of maintaining control over their digital legacy.

Moreover, when a CryptoLegacy instance is initialized, the **owner** is LibCryptoLegacy.sol#L395-L396 in the BeneficiaryRegistry contract. However, when the owner calls CryptoLegacyBasePlugin.renounceOwnership, their ownership is CryptoLegacyOwnable.sol#L31-L35 from the BeneficiaryRegistry.cryptoLegacyByOwner, resulting in an inconsistency between the plugin's internal state and the registry.

Recommendation

We recommend removing the renounceOwnership function.

Client's Commentary:

4558f24a

L-3	transfer method is used instead of call		
Severity	Low	**Status**	Fixed in 4558f24a

Description

This issue has been identified in Ether-sending functions like payInitialFee and _sendFeeByTransfer (within **CryptoLegacyBasePlugin** / **LibCryptoLegacy**).
Using transfer forwards only 2300 gas. This fixed limit is often insufficient for recipient smart contracts (e.g., multisigs, bridges, contracts with complex receive logic), causing the Ether transfer to fail.
The issue is classified as **Low** severity as it affects interoperability and robustness rather than causing direct asset loss.

Recommendation

We recommend using the low-level call method instead of transfer and apply reentrancy protection.

Client's Commentary:

4558f24a

L-4	Execution Order Race Condition in SignatureRoleTimelock		
Severity	Low	**Status**	Acknowledged

Description

The SignatureRoleTimelock.executeCallList function allows anyone to SignatureRoleTimelock.sol#L333-L337 scheduled calls after their timelock period expires. However, there is no mechanism to enforce execution order of related calls. If multiple dependent calls are scheduled with overlapping timelock periods, this can lead to execution in an unintended order, potentially compromising security or causing logical errors if calls were designed to be executed in a specific sequence.

Recommendation

We recommend restricting execution to authorized users only and enforcing the correct execution order of scheduled calls.

Client's Commentary:

We acknowledge the finding, but we do not consider this a vulnerability. The executeCallList function is intentionally designed to allow flexible execution after timelock expiry, without enforcing a strict call order. This is by design: CryptoLegacy contracts do not rely on sequential execution of time-locked operations. All privileged actions are intended to be independent and safe to execute in any order. If a future use case requires enforcing execution order (e.g., composable governance flows), this can be easily achieved by assigning non-overlapping timelock durations to dependent calls. Therefore, the system offers maximum flexibility without hardcoding assumptions about call relationships. As such, we treat this as expected behavior, not a security concern.

L-5	TrustedGuardiansPlugin._setGuardiansConfig should have a below bound check for guardiansChallengeTimeout		
Severity	Low	Status	Fixed in 4558f24a

Description

In the TrustedGuardiansPlugin contract, the guardiansChallengeTimeout parameter lacks a lower bound check and only enforces an TrustedGuardiansPlugin.sol#L241-L243. As a result, the owner may not have sufficient time to react if the timeout is set to zero.

Recommendation

We recommend adding a check to enforce a reasonable lower bound for the guardiansChallengeTimeout parameter.

Client's Commentary:

4558f24a

L-6	Lack of check for arrays length		
Severity	Low	Status	Fixed in 4558f24a

Description

The CryptoLegacyBasePlugin.update function lacks a check to ensure that the CryptoLegacyBasePlugin.sol#L294 **_lockToChainIds** and **_crossChainFees** passed by the owner have the same length.

Recommendation

We recommend adding a validation to check that the lengths of **_lockToChainIds** and **_crossChainFees** are equal before processing them.

Client's Commentary:

4558f24a

L-7	Gas Inefficiency Due to Linear Searches in View Functions		
Severity	Low	**Status**	Acknowledged

Description

This issue has been identified within several view or pure functions across multiple contracts and libraries, including:

- `_isVoterAllowed`, `_getConfirmedCount`, `_isMethodAllowed` in **LibSafeMinimalMultisig**;
- `_getAddressIndex`, `_getBytes4Index` in **SignatureRoleTimelock**;
- `_findFacetBySelector` in **LibCryptoLegacyPlugins**;
- and `_isGuardianVoted` in **TrustedGuardiansPlugin**.

These functions use inefficient O(n) loops to search through arrays (e.g., lists of voters, allowed methods, guardians, facets). For large arrays, these searches consume excessive gas, potentially making data retrieval costly or impossible for other contracts or off-chain scripts calling them.

Recommendation

We recommend refactoring these linear searches to use more efficient data structures. Mappings (`mapping(key => value)` or `mapping(key => bool)`) provide O(1) lookup complexity. For scenarios requiring iteration, consider OpenZeppelin's `EnumerableSet` library for O(1) lookups and enumeration capabilities.

Client's Commentary:

Due to the small and limited number of elements, it was decided not to waste extra gas on writing to the mapping, since doing so can cost more than iterating over the small number of elements. In `_findFacetBySelector`, special caching occurs in the mapping whenever possible. Recording all selectors in the mapping when deploying a contract is too expensive.

L-8	Weak Privacy Due to Hash-Only Identity Obfuscation		
Severity	Low	**Status**	Fixed in 4558f24a

Description

To preserve user privacy, the system obscures Ethereum addresses by storing only their Keccak-256 hashes. While in theory it is computationally infeasible to recover a random 160-bit address from its Keccak hash, this assumption overlooks a critical **practical limitation**. In practice, there is no value in protecting addresses that have never interacted with the blockchain. Realistically, privacy is only relevant for **active addresses** – and the global set of such addresses is relatively small and enumerable (e.g., all addresses that ever sent or received transactions). An attacker can precompute the Keccak hashes of all such addresses and perform a **lookup attack**, potentially deanonymizing participants.
This undermines the intended privacy goal: hashes alone **do not provide sufficient anonymity** for active accounts.

Recommendation

Instead of using `keccak256(address)` as a standalone identifier, derive user identity from a combination of address and a secret salt:

```
keccak256(abi.encodePacked(address, salt))
```

The salt should be generated client-side and stored off-chain. It is only revealed at the moment of authorization or voting. Without knowledge of the salt, an attacker cannot match a hash to a known address, even if the address is on a precomputed list.
This approach provides **semantic privacy**: even active addresses cannot be linked to their hashed identity without voluntary disclosure.

Client's Commentary:

4558f24a

L-9	lastUpdateAt Can Be Updated by Non-Owner		
Severity	Low	**Status**	Acknowledged

Description

In the CryptoLegacyBasePlugin.payInitialFee function, CryptoLegacyBasePlugin.sol#L214 updates the **lastUpdateAt** timestamp.

However, it is executed unconditionally, regardless of who calls the function. This allows any external address to call CryptoLegacyBasePlugin.payInitialFee and refresh the **lastUpdateAt** value, even if the caller is not the owner of the legacy.

This behavior is problematic because **lastUpdateAt** is presumably used as a liveness indicator. If any account can update this value, it undermines the reliability of death detection mechanisms that rely on it.

Recommendation

We recommend restricting the CryptoLegacyBasePlugin.payInitialFee function to the contract owner:

```
if (msg.sender != owner()) revert NotOwner();
```

Do not use the onlyOwner modifier here, as its internal _checkOwner() implementation checks whether lastFeePaidAt != 0, which would conflict with this function's own lastFeePaidAt == 0 requirement.

Client's Commentary:

The lastUpdateAt timestamp can only be updated once after contract creation and before it is actively used. Because of this one-time update constraint—combined with the fact that our front-end does not allow token approvals to crypto legacy address before first payment—there is no ongoing vulnerability regarding repeated updates or spoofing of liveness checks.

L-10	Missing Sanity Check for guardiansThreshold		
Severity	Low	Status	Fixed in 4558f24a

Description

TrustedGuardiansPlugin._setGuardiansConfig TrustedGuardiansPlugin.sol#L244 the **guardiansThreshold** without validating whether the threshold is consistent with the number of currently assigned guardians.

As a result, the threshold can be set to a value higher than the number of guardians, temporarily making it impossible to reach quorum in guardian-based processes such as recovery approvals.

However, since the owner retains the ability to update both the guardian list and the threshold, this situation does **not result in a permanent lockout** and can be corrected by the owner.

Recommendation

We recommend adding a validation to ensure **guardiansThreshold** is within valid bounds at the time it is set.

Client's Commentary:

4558f24a

L-11	Redundant Casting in Function Selector Retrieval		
Severity	Low	**Status**	Fixed in 4558f24a

Description

In the `getSigs` functions, selectors are retrieved using an explicit cast to the contract's type:

```
sigs[0] = TrustedGuardiansPlugin(address(this)).initializeGuardians.selector;
```

This cast is unnecessary. In Solidity 0.8.x and above, it is valid and more concise to use:

```
sigs[0] = this.initializeGuardians.selector;
```

Both forms produce the exact same `bytes4` function selector. The cast to `TrustedGuardiansPlugin(address(this))` adds no functional benefit and reduces readability. The `this.` prefix already performs an external call context and resolves the selector properly.

Recommendation

Simplify the expression by removing the explicit cast:

```
sigs[0] = this.initializeGuardians.selector;
```

Apply this replacement consistently for all selector assignments.

Client's Commentary:

4558f24a

L-12	Missing Two-Step Ownership Transfer		
Severity	Low	**Status**	Fixed in 4558f24a

Description

The owner of **CryptoLegacy** might make a mistake when transferring ownership. It's safer to use a two-step ownership transfer process.

Recommendation

We recommend using Ownable2Step from OpenZeppelin.

Client's Commentary:

4558f24a

L-13	Unnecessary Condition Inversion		
Severity	Low	**Status**	Fixed in 4558f24a

Description

In the LibCryptoLegacy._checkDistributionReady LibCryptoLegacy.sol#L147-L152, the condition is logically equivalent to the negation of the LibCryptoLegacy.sol#L63 in LibCryptoLegacy._isDistributionStarted due to De Morgan's laws.

Recommendation

To improve readability and reduce the chance of future errors, consider refactoring the code as follows:

```
function _checkDistributionReady(
    ICryptoLegacy.CryptoLegacyStorage storage cls
) internal view {
    if (!_isDistributionStarted(cls)) {
        revert ICryptoLegacy.TooEarly();
    }
}
```

Client's Commentary:

4558f24a

L-14	Hard-Coded Gas Limits Risk Future Compatibility		
Severity	Low	**Status**	Fixed in 137d5972

Description

This issue has been identified in multiple functions using hard-coded gas limits for external calls.

These fixed limits (e.g., 1e5) may become insufficient if Ethereum gas costs change or if the external contracts require more gas in the future. This could cause these calls to fail, breaking core functionality.

The issue is classified as **Low** severity as it represents a future compatibility risk rather than an immediate flaw.

Recommendation

We recommend replacing hard-coded gas limits for external calls with forwarding all available gas, or alternatively, using configurable gas stipends.

Client's Commentary:

MixBytes: In `LibCryptoLegacy._isLifetimeActiveAndUpdate`, the gas limit for the call to `isLifetimeNftLockedAndUpdate` was reduced from `6e5` to `2e5`. We recommend reverting it to the previous value.
Client: `137d5972`

L-15	Unused or Duplicate Imports Increase Deployment Cost		
Severity	Low	**Status**	Fixed in 4558f24a

Description

This issue has been identified across multiple contracts:

- **CryptoLegacyBasePlugin** imports Pausable.sol but does not appear to use its functionality.
- **LegacyRecoveryPlugin** imports LibDiamond.sol and IERC721.sol, which seem unused within the plugin's logic.
- **CryptoLegacyOwnable** includes duplicate imports for LibCryptoLegacy.sol.

Importing unused contracts/libraries or duplicating imports increases the contract bytecode size and deployment gas costs unnecessarily.

Recommendation

We recommend removing all unused and duplicate import statements.

Client's Commentary:

4558f24a

L-16	Silent `catch` Blocks Hide Runtime Errors		
Severity	Low	**Status**	Fixed in 4558f24a

Description

This issue has been identified within `try...catch` blocks in **LibCryptoLegacy**, particularly around external calls (e.g., registry updates).
Empty `catch {}` blocks are used, which silently ignore any errors occurring during the `try` block. This practice hides potential runtime problems and makes debugging difficult.

Recommendation

We recommend replacing empty `catch {}` blocks with logic that emits an event containing error information, allowing failures to be logged and diagnosed.

Client's Commentary:

4558f24a

L-17	Hard-coded Values Reduce Readability		
Severity	Low	Status	Fixed in 4558f24a

Description

This issue involves the direct use of literal numeric values ('magic numbers') instead of named constants in various contracts:

- Time durations (e.g., 7 days, 30 days) in **SignatureRoleTimelock.sol** and **TrustedGuardiansPlugin.sol**.
- Basis points denominator (10000) in **LibCryptoLegacy.sol** and **CryptoLegacyBasePlugin.sol**.
- An array length limit (100) in **LibCryptoLegacy.sol**.

Using magic numbers makes the code harder to read, as the purpose of the number isn't immediately clear. It also increases the risk of errors during maintenance, as changing a value requires finding every instance.

Recommendation

We recommend replacing magic numbers with named constant for improved clarity and easier updates.

Client's Commentary:

4558f24a

L-18	Minor Gas Inefficiencies via Redundant Operations		
Severity	Low	**Status**	Fixed in 4558f24a

Description

This issue identifies minor inefficiencies in several functions:

- In **SignatureRoleTimelock** `_getAddressIndex` and `_getBytes4Index` continue iterating even after the index is found.
- In **LibCryptoLegacy**, the `_tokenPrepareToDistribute` function contains logic for calculating `amountToDistribute` involving balance and allowance checks that can be simplified to a single assign `td.amountToDistribute = bal + td.totalClaimedAmount;`.
- In **LibCryptoLegacyPlugins**, the `_findFacetBySelector` function iterates through all facets even if a match is found early instead of returning immediately.
- In **LibCryptoLegacy** `_transferTreasuryTokensToLegacy` uses a nested `for` loop. It's better to skip the iteration if the balance is zero and avoid unnecessary transfers.

Recommendation

We recommend refactoring the identified code sections for minor gas savings, primarily by returning early from search loops once an item is found.

Client's Commentary:

4558f24a

L-19	Timelock Duration Check Missing, Allows Near-Zero Delay		
Severity	Low	**Status**	Acknowledged

Description

This issue has been identified within the `_addSignatureRole` function of the **SignatureRoleTimelock** contract.

The function allows setting a role's timelock duration without enforcing a minimum value. This permits setting a timelock of just 1 second, undermining the intended security purpose of providing a meaningful delay before execution.

The issue is classified as **Low** severity as it's a configuration weakness exploitable only by privileged users, but it bypasses an intended safeguard.

Recommendation

We recommend adding a `require` statement in `_addSignatureRole` to enforce a sensible minimum timelock duration:

```
require(_timelock >= 1 hours, "Timelock too short");
```

Client's Commentary:

We intentionally allow the main administrator to set near-zero timelocks to enable quick role configuration when operational needs require it. This flexibility is restricted to privileged users only, ensuring that any short timelock usage remains under controlled circumstances.

L-20	Unvalidated _refShare Can Cause Reverts		
Severity	Low	**Status**	Fixed in 4558f24a

Description

This issue has been identified within the _takeFee function of the **LibCryptoLegacy** library. The function uses _refShare to calculate a referral fee but doesn't check if _refShare exceeds SHARE_BASE (10000). An invalid _refShare (> 10000) can cause an arithmetic underflow during the owner's share calculation, leading to transaction reverts.

Recommendation

We recommend adding validation before calculating fees involving _refShare.

```
require(_ref == address(0) || _refShare <= SHARE_BASE, "Invalid ref share");
```

Client's Commentary:

4558f24a

L-21	Standard ReentrancyGuard Inheritance Risks Diamond Storage Collision		
Severity	Low	**Status**	Fixed in d3ae300a

Description

This issue applies to Diamond plugins **LegacyRecoveryPlugin**, **CryptoLegacyBasePlugin**, **TrustedGuardianPlugin** inheriting the standard ReentrancyGuard.

Standard ReentrancyGuard uses storage slot 0 for its **status** variable. In a Diamond proxy, storage slots are shared across all facets. If another facet deployed to the Diamond also uses slot 0, their storage variables will collide, leading to corrupted state or potentially disabled re-entrancy protection.

The issue is classified as **Low** severity as it represents a latent compatibility risk within the Diamond pattern, which could cause critical issues if conflicting facets are added later.

Recommendation

We recommend using ReentrancyGuardUpgradeable from OpenZeppelin's upgradeable contracts library, as it is designed to avoid storage collisions in proxy patterns like Diamond.

Client's Commentary:

MixBytes: `ReentrancyGuardUpgradeable` was introduced, however `__ReentrancyGuard_init` function is never called. We recommend initializing via this function.

Client: `36c75d85`

MixBytes: `SignatureRoleTimelock` is not an upgradeable contract. However, it inherits from `ReentrancyGuardUpgradeable` and calls `__ReentrancyGuard_init` in the constructor. We recommend using the non-upgradeable version of `ReentrancyGuard` instead, and removing the `__ReentrancyGuard_init` call.

L-22	Incorrect Handling of Beneficiary Votes in TrustedGuardiansPlugin		
Severity	Low	**Status**	Acknowledged

Description

When no guardians are explicitly configured in the TrustedGuardiansPlugin, the plugin defaults to using the list of CryptoLegacy beneficiaries as guardians.
In this setup, a beneficiary can call `beneficiarySwitch` to change their address and potentially vote multiple times. Fortunately, the contract correctly filters out votes from stale (previous) beneficiary addresses when evaluating quorum, so quorum triggering is not affected.
However, public view functions that return voting information may still include these outdated votes, making the results counterintuitive and misleading. For example, a vote cast via a stale address may initially appear in the list of votes, only to disappear later when someone else votes — leading to confusion about the actual state of voting.
Although this behavior does not impact the integrity of quorum calculation, it may cause off-chain clients or observers to misinterpret the voting status.

Recommendation

We recommend reviewing public view functions that expose voting data, and considering adjustments to ensure they return only valid and up-to-date votes — i.e., votes associated with the currently active guardian or beneficiary addresses.

Client's Commentary:

MixBytes: `TrustedGuardiansPlugin.getGuardiansData` still behaves unexpectedly in some conditions: A beneficiary votes, and then `getGuardiansData` returns that the beneficiary voted. The beneficiary then makes a switch, but `getGuardiansData` still returns that the beneficiary voted (under their old hash). After that, another beneficiary votes, and `getGuardiansData` returns that the beneficiary who made the switch did not vote — i.e., the number of voters remains unchanged after the new vote appears because the old vote was removed. This does not affect consensus, because quorum is triggered after the vote and cleanup occurs before that; however, the `getGuardiansData` getter currently returns inconsistent data. Additionally, the `checkGuardiansVotedAndGetGuardiansData` function has been introduced to provide accurate and up-to-date voting data, addressing this issue without altering the existing `getGuardiansData` view function.

L-23	Beneficiary Can Avoid Paying Protocol Fees		
Severity	Low	**Status**	Acknowledged

Description

There are two issues in the fee payment mechanism for beneficiaries when calling CryptoLegacyBasePlugin.beneficiaryClaim:

1. Beneficiaries can completely avoid paying fees when claiming tokens if the distribution has already started. This can happen because, when LibCryptoLegacy.sol#L210-L212 returns **true**, the function updates lastFeePaidAt, and inside LibCryptoLegacy._sendFeeByTransfer, there's an early exit if LibCryptoLegacy.sol#L270-L272. As a result, beneficiaries can call CryptoLegacyBasePlugin.beneficiaryClaim with zero payment and still receive their tokens.
2. Even when beneficiaries do provide payment, they can redirect 100% of the fee to a referral address instead of the protocol by using the maximum **_refShare** value (SHARE_BASE = 10000).

Recommendation

We recommend making the following changes:

1. Add proper fee validation in the LibCryptoLegacy._sendFeeByTransfer function to ensure that beneficiaries cannot claim without paying the required fee.
2. Prevent beneficiaries from manually providing a referral address and share percentage. The referral mechanism should be fully managed by the system to prevent abuse or fee redirection.

Client's Commentary:

We intentionally designed the beneficiaryClaim fee as a voluntary donation rather than a strict requirement. This ensures that beneficiaries can always access their tokens, especially during critical inheritance scenarios. We consider the severity of this issue to be Low, not Medium, for the following reasons: The contract avoids reverts related to fee logic to guarantee that beneficiaries can claim their assets without risk of failure. The early exit on msg.value == 0 is an intentional mechanism to provide fault-tolerant behavior in cases where users or interfaces fail to include a fee. Our frontend includes the appropriate fee and referral parameters by default, so under standard usage, the DAO still receives the expected contribution. Users are free to omit the fee or assign 100% of it to a referral address — this is consistent with the donation-based model and does not impact protocol correctness or security. The fee mechanism exists to support protocol sustainability, not to enforce payment through mandatory contract-level constraints. Given that this behavior is intentional, carries no functional risk, and enhances accessibility in sensitive use cases, we consider the issue to be Low severity.

L-24	Operations Allowed on Proposals with NOT_EXIST Status		
Severity	Low	**Status**	Fixed in 36c75d85

Description

The LibSafeMinimalMultisig._getPendingProposalForVoter function does not validate whether the provided _proposalId falls within the bounds of the proposals array. The function only checks whether the proposal status is not EXECUTED or CANCELED, but still allows operations on proposals with the NOT_EXIST status, which is the default value for uninitialized entries.

As a result, the following problematic scenarios may occur:

- Invoking LegacyRecoveryPlugin.lrConfirm with an arbitrary proposal ID that does not exist in the array may result in the creation and immediate execution of a phantom proposal, especially if the threshold is set to 1.
- Combining a call to LegacyRecoveryPlugin.lrConfirm followed by LegacyRecoveryPlugin.lrCancel may allow setting the status of a non-existent proposal to CANCELED, manipulating internal state and potentially causing inconsistencies.

Recommendation

We recommend allowing confirmation and cancellation operations only for proposals that exist and have a PENDING status.

L-25	Inconsistent Native Value Handling in LegacyRecoveryPlugin		
Severity	Low	**Status**	Fixed in 36c75d85

Description

The `LegacyRecoveryPlugin.lrResetGuardianVoting` function is marked as `payable`, indicating it expects to receive native value. However, the `LegacyRecoveryPlugin.lrPropose` and `LegacyRecoveryPlugin.lrConfirm` functions do not support native value transfers. As a result, for `lrResetGuardianVoting` to receive funds, the value must be sent directly to the `CryptoLegacy` contract address beforehand. This leads to a fragmented funding model where some operations require pre-funding the contract.

In cases where `lrResetGuardianVoting` is the target selector in a proposal, the execution of `LegacyRecoveryPlugin.lrPropose` and `LegacyRecoveryPlugin.lrConfirm` may fail if there is no native value present on the `CryptoLegacy` contract.

Recommendation

We recommend implementing a consistent approach that allows sending native value with function calls, without requiring prior funding of the contract.

Client's Commentary:

Client: 36c75d85

MixBytes: The issue was fixed. Two functions — `LegacyRecoveryPlugin.lrPropose` and `LegacyRecoveryPlugin.lrConfirm` — were marked as payable; however, this change introduced a new issue, described in finding L-27.

L-26	Premature Owner Update in Beneficiary Registry During Ownership Transfer		
Severity	Low	**Status**	Fixed in 36c75d85

Description

In the CryptoLegacyBasePlugin.transferOwnership function the _updateOwnerInBeneficiaryRegistry function is CryptoLegacyBasePlugin.sol#L183 before ownership is actually transferred. This may lead to a state inconsistency within the BeneficiaryRegistry contract, where it reflects an owner different from the actual one defined by the ownership logic.

Recommendation

We recommend moving the call to LibCryptoLegacy._updateOwnerInBeneficiaryRegistry into the CryptoLegacyOwnable.acceptOwnership function to ensure that the beneficiary registry reflects the effective owner only after ownership is accepted.

L-27	Improper Handling of Payable Proposals		
Severity	Low	**Status**	Fixed in d3ae300a

Description

An issue has been identified in the LegacyRecoveryPlugin.

The LegacyRecoveryPlugin.lrPropose and LegacyRecoveryPlugin.lrConfirm functions are marked as payable. This design was implemented for consistency, allowing voters to transfer funds to execute proposals that require payment, such as those using the lrResetGuardianVoting selector.

However, this implementation has several flaws:

1. If requiredConfirmations > 1, multiple voters can send value for the same proposal, but only the msg.value from the final confirming transaction is actually forwarded during execution. Funds from earlier payments remain locked in the contract.
2. If a proposal that required payment is subsequently cancelled, the transferred funds remain locked in the contract with no withdrawal mechanism.
3. Voters can send funds when proposing operations that do not require payment (e.g., lrTransferTreasuryTokensToLegacy, lrWithdrawTokensFromLegacy).

Recommendation

We recommend the following:

1. Implement a mechanism that allows voters to reclaim their funds if a proposal has been cancelled.
2. Remove the payable modifier from the lrConfirm function.
3. Modify the proposal struct to store the payer's address and the paid amount.
4. Allow only one payment per proposal (i.e., a single actual fee).
5. Add checks to disallow payments for proposals whose selector does not require payment.

L-28	Incorrect handling of fee-on-transfer and rebasing tokens penalizes single beneficiary		
Severity	Low	**Status**	Fixed in fc6242e7

Description

In the CryptoLegacyBasePlugin contract's internal _claimTokenWithVesting function, when balanceExpected (post-transfer balance minus claimed amount) is less than balanceBefore, for instance due to a negative rebase inside the transfer or a fee-on-transfer token, the code increments only the single claiming beneficiary's tokenAmountClaimed by the deficit and then calls _tokenPrepareToDistribute. This adjustment wrongly assigns the entire shortfall to one beneficiary rather than distributing it proportionally across all beneficiaries, resulting in unfair vesting.

Here, fee-on-transfer tokens refer to tokens that deduct an extra fee from the sender's balance in addition to the transferred amount. Such tokens are already unsupported by _transferTreasuryTokensToLegacy (which retrieves full balances or allowances, causing such fee-on-transfer transfers to revert), so adding special handling here complicates the logic unnecessarily.

Furthermore, because the code increments bv.tokenAmountClaimed before invoking _tokenPrepareToDistribute, the subsequent balanceDiff calculation within _tokenPrepareToDistribute always evaluates to zero. As a result, the function returns without changing any storage, rendering this adjustment logic effectively a no-op and adding unnecessary complexity without any state change.

Recommendation

We recommend removing the deficit-adjustment block to restore original behavior and suggesting to users to rely on wrapped variants of nonstandard tokens if needed.

Specifically, delete:

```
-         if (balanceExpected < balanceBefore) {
-           bv.tokenAmountClaimed[_token] += (balanceBefore - balanceExpected);
-           LibCryptoLegacy._tokenPrepareToDistribute(cls, _token);
-         }
```

Reverting this change does not risk losing tokens for users: even if fee-on-transfer or rebasing tokens remain unsupported, any extra fees or rebasing during transfer will reduce total distributable tokens, but assets will not be lost and will still be distributed—albeit with minor fairness variations. Given these token behaviors are atypical, this tradeoff is acceptable.

3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

Contact Information

https://mixbytes.io/

https://github.com/mixbytes/audits_public

hello@mixbytes.io

https://x.com/mixbytes