UTEXO BRIDGE SMART CONTRACTS SECURITY RE-AUDIT REPORT

OXORIO

OCTOBER 2, 2026

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security re-audit conducted by Oxorio for the UTEXO Bridge.

UTEXO Protocol develops a trust-minimized bridge connecting Bitcoin/RGB assets with EVM networks, with the stated design goal that even a compromise of the off-chain operators must not allow assets to be stolen from the bridge.

This re-audit covers the Solidity delta in `UTEXO-Protocol/bridge-smart-contracts` from the last fully audited revision `ea5de9b4` to `95d8e566`. USDT0 contracts are outside this delta. The change introduces an ERC-1967 upgradeable Bridge, a direct emergency guardian, flat commissions, rewritten outflow-bucket references, an RGB pool settlement module, and Bitcoin-relay checkpoint checks.

The audit process involved a comprehensive approach, including manual code review, automated analysis, and extensive testing and simulations of the smart contracts to assess the project's security and functionality. The re-audit covered 38 Solidity files under `ethereum/src`, encompassing 8,148 total lines (4,381 lines of code). The codebase was thoroughly examined against the August 12, 2026 final report and the baseline commit. For an in-depth explanation of the smart contract security audit methodology, please refer to the Security Assessment Methodology section of this document.

The same zero-trust model as the original audit applies: no off-chain actor (TEE, federation, guardian, backend, listener, or host) is assumed correct unless an on-chain quorum or timelock check enforces it.

Of the 47 findings in the August 12, 2026 final report, none of the verified fixes are confirmed regressed on this delta. This re-audit reports 6 new warnings and 9 new informational issues. Seven prior acknowledgements remain accepted residuals.

Throughout the re-audit, a collaborative approach was maintained with UTEXO Protocol to address all concerns identified within the audit's scope. Each issue has been either resolved or formally acknowledged by UTEXO Protocol, contributing to the robustness of the project.

As a result, following a comprehensive review, our auditors have verified that the UTEXO Bridge, as of audited commit `dev@c2a5f213` (bridge-smart-contracts), has met the security and functionality requirements established for this audit, based on the code and documentation provided, and operates as intended within the defined scope.

1.2 SUMMARY OF FINDINGS

The table below provides a comprehensive summary of the audit findings, categorizing each by status and severity level. For a detailed description of the severity levels and statuses of findings, see the Findings Classification Reference section.

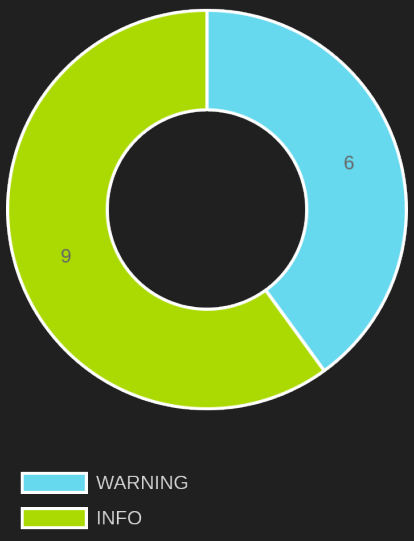
Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Findings Report section for further reference.

Prior findings on this Bridge delta: 40 previously FIXED remain FIXED; 7 previously ACKNOWLEDGED remain ACKNOWLEDGED; 0 confirmed regressions.

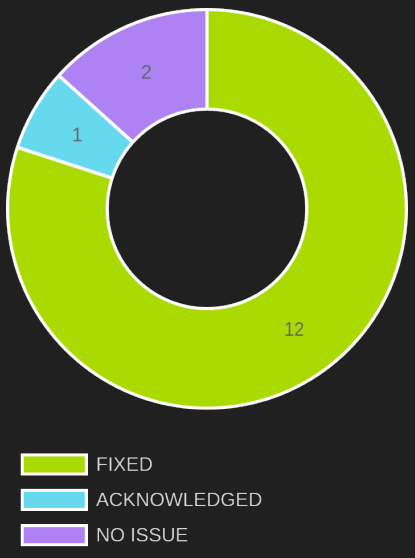
Of the 15 findings in this re-audit, 12 are FIXED, 2 are NO ISSUE, and 1 is ACKNOWLEDGED. None remain open.

All identified issues have been addressed, with UTEXO Protocol fixing them or formally acknowledging their status.

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**0**	0	0	0	0
MAJOR	**0**	0	0	0	0
WARNING	**6**	0	6	0	0
INFO	**9**	0	6	1	2
TOTAL	**15**	**0**	**12**	**1**	**2**



Issue distribution by severity



Issue distribution by status

2 AUDIT OVERVIEW

CONTENTS

1. EXECUTIVE SUMMARY 2

1.1. EXECUTIVE SUMMARY 3

1.2. SUMMARY OF FINDINGS 4

2. AUDIT OVERVIEW 6

2.1. DISCLAIMER 9

2.2. PROJECT BRIEF 10

2.3. PROJECT TIMELINE 11

2.4. AUDITED FILES 12

2.5. PROJECT OVERVIEW 14

2.6. CODEBASE QUALITY ASSESSMENT 15

2.7. FINDINGS BREAKDOWN BY FILE 17

2.8. CONCLUSION 18

3. FINDINGS REPORT 19

3.1. CRITICAL 20

3.2. MAJOR 21

3.3. WARNING 22

W-01 Pool settlement does not bind the released amount to cited mint/burn records in RgbPoolSettlementModule.sol 22

W-02 Emergency guardian can unpause a federation freeze without signatures in MultisigProxy.sol 25

W-03 Guardian pause cycles leave federation emergency signatures valid in MultisigProxy.sol ... 28

W-04 Generic governance proposals remain unbound to a target while StaleBridgeTarget wraps only the upgrade lane in MultisigProxy.sol 31

W-05 fundsOut does not enforce commission conservation after CommissionManager became mutable in Bridge.sol 35

W-06 Upgrade to the current implementation can call fundsOut and skip the enclave in MultisigProxy.sol 38
3.4. INFO 42
I-01 cancelProposal authorization has no nonce and no upper deadline bound in MultisigProxy.sol 42
I-02 minFundsInAmount and minFundsOutAmount are independent in Bridge.sol 45
I-03 fundsOut repeats a zero-net check the shipped manager already makes in Bridge.sol 47
I-04 Production fundsIn does not share MinimalBridge's uint64 cap in Bridge.sol 49
I-05 Upgrade calldata has no length cap in BridgeProxy.sol 51
I-06 Post-upgrade owner() is a single call, not a stable read in BridgeProxy.sol 53
I-07 A 32-byte answer to implementation() is treated as a proxy in BridgeProxy.sol 56
I-08 Federation rotation emits two events that always fire together in MultisigProxy.sol 59
I-09 The zero-registry check in _requireManagedOwnershipTarget is redundant in MultisigProxy.sol 61
4. APPENDIX 63
4.1. SECURITY ASSESSMENT METHODOLOGY 64
4.2. CODEBASE QUALITY ASSESSMENT REFERENCE 66
Rating Criteria 67
4.3. FINDINGS CLASSIFICATION REFERENCE 68
Severity Level Reference 68
Status Level Reference 68
4.4. ABOUT OXORIO 70

2.1 DISCLAIMER

At the request of the client, Oxorio consents to the public release of this audit report. The information contained herein is provided "as is" without any representations or warranties of any kind. Oxorio disclaims all liability for any damages arising from or related to the use of this audit report. Oxorio retains copyright over the contents of this report.

This report is based on the scope of materials and documentation provided to Oxorio for the security audit as detailed in the Executive Summary and Audited Files sections. The findings presented in this report may not encompass all potential vulnerabilities. Oxorio delivers this report and its findings on an as-is basis, and any reliance on this report is undertaken at the user's sole risk. It is important to recognize that blockchain technology remains in a developmental stage and is subject to inherent risks and flaws.

This audit does not extend beyond the programming language of smart contracts to include areas such as the compiler layer or other components that may introduce security risks. Consequently, this report should not be interpreted as an endorsement of any project or team, nor does it guarantee the security of the project under review.

THE CONTENT OF THIS REPORT, INCLUDING ITS ACCESS AND/OR USE, AS WELL AS ANY ASSOCIATED SERVICES OR MATERIALS, MUST NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE. Third parties should not rely on this report for making any decisions, including the purchase or sale of any product, service, or asset. Oxorio expressly disclaims any liability related to the report, its contents, and any associated services, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Oxorio does not warrant, endorse, or take responsibility for any product or service referenced or linked within this report.

For any decisions related to financial, legal, regulatory, or other professional advice, users are strongly encouraged to consult with qualified professionals.

2.2 PROJECT BRIEF

Title	Description
Client	UTEXO Protocol
Project name	UTEXO Bridge
Category	Cross-chain bridge (RGB ↔ EVM)
Website	https://utexo.com
Documentation	https://docs.utexo.com
Repository	bridge-smart-contracts
Initial Commit	`ea5de9b4`
Final Commit	`c2a5f213`
Full diff	ea5de9b4...c2a5f213
Platform	EVM
Network	Ethereum (with Bitcoin/RGB and LayerZero integration)
Languages	Solidity 0.8.35
Lead Auditor	Alexander Mazaletskiy - am@oxor.io
Project Manager	Ales Kuznetsova - ales@oxor.io

2.3 PROJECT TIMELINE

The key events and milestones of the project are outlined below.

Date	Event
May 25, 2026	The audit team initiated work on the project.
June 1, 2026	Submission of the preliminary audit report #1.
June 8, 2026	Submission of the preliminary audit report #2.
June 11, 2026	Submission of the preliminary audit report #3.
June 15, 2026	Submission of the preliminary audit report #4.
June 22, 2026	Submission of the audit report.
July 15, 2026	The team began the re-audit of the client's remediations.
July 21, 2026	Submission of the final audit report after the remediation re-audit.
August 11, 2026	Re-audit of the additional remediation PRs and their integration into `dev`.
August 12, 2026	Submission of the final audit report at `ea5de9b4`.
September 17, 2026	Re-audit of the Bridge delta from `ea5de9b4` to `95d8e566` initiated.
September 22, 2026	Submission of the re-audit report at `95d8e566`.
September 29, 2026	Client began remediation of the 15 findings.
October 1, 2026	Remediation submitted at `c2a5f213`.
October 1, 2026	Re-audit of that remediation.
October 2, 2026	Submission of the final re-audit report incorporating the verified fixes.

2.4 AUDITED FILES

The following table is the finding baseline at commit `95d8e566`. The scc tool was used to count the number of lines and assess complexity of the files. USDT0 contracts are unchanged in this delta and are outside this scope. `RgbPoolSettlementModule.sol` is present in this baseline and is removed by the remediation before `c2a5f213`.

	File	Lines	Blanks	Comments	Code	Complexity
1	ethereum/src/MultisigProxy.sol	1551	196	299	**1056**	23%
2	ethereum/src/Bridge.sol	1248	126	462	**660**	17%
3	ethereum/src/CommissionManager.sol	809	87	345	**377**	30%
4	ethereum/src/interfaces/IMultisigProxy.sol	559	53	179	**327**	0%
5	ethereum/src/btc_relay/structs/StoredBlockHeader.sol	264	31	41	**192**	12%
6	ethereum/src/btc_relay/structs/StoredBlockHeaderTestnet.sol	269	32	54	**183**	11%
7	ethereum/src/btc_relay/BtcRelayTestnet.sol	262	50	46	**166**	12%
8	ethereum/src/btc_relay/BtcRelay.sol	258	50	46	**162**	12%
9	ethereum/src/interfaces/IBridge.sol	479	50	281	**148**	0%
10	ethereum/src/interfaces/ICommissionManager.sol	276	60	77	**139**	0%
11	ethereum/src/libraries/OutflowRateLimiter.sol	135	24	16	**95**	35%
12	ethereum/src/settlement/RgbPoolSettlementModule.sol	128	17	39	**72**	38%
13	ethereum/src/RouteRegistry.sol	169	26	78	**65**	22%
14	ethereum/src/BridgeBaseUpgradeable.sol	90	18	7	**65**	14%
15	ethereum/src/BridgeBase.sol	159	26	73	**60**	15%
16	ethereum/src/libraries/RollingOutflowLimiter.sol	89	15	15	**59**	17%
17	ethereum/src/btc_relay/utils/Difficulty.sol	91	12	21	**58**	16%
18	ethereum/src/settlement/RgbSettlementModule.sol	210	25	129	**56**	36%
19	ethereum/src/settlement/RgbOutboundSettlementModule.sol	154	22	78	**54**	31%
20	ethereum/src/btc_relay/utils/Nbits.sol	63	6	9	**48**	21%
21	ethereum/src/BridgeProxy.sol	76	10	18	**48**	29%
22	ethereum/src/verifiers/RGBVerifier.sol	138	18	77	**43**	16%
23	ethereum/src/btc_relay/btc_utils/Endianness.sol	45	3	2	**40**	8%
24	ethereum/src/interfaces/RouteTypes.sol	103	4	71	**28**	0%
25	ethereum/src/interfaces/IRouteRegistry.sol	121	17	77	**27**	0%
26	ethereum/src/MinimalBridge.sol	78	16	37	**25**	20%
27	ethereum/src/btc_relay/state/BtcRelayState.sol	32	3	9	**20**	10%
28	ethereum/src/btc_relay/structs/CompactBlockHeader.sol	39	6	14	**19**	11%
29	ethereum/src/interfaces/AggregatorV3Interface.sol	26	6	6	**14**	0%
30	ethereum/src/btc_relay/state/Fork.sol	20	3	4	**13**	8%
31	ethereum/src/btc_relay/Events.sol	14	1	1	**12**	0%
32	ethereum/src/settlement/NullSettlementModule.sol	34	3	21	**10**	0%
33	ethereum/src/btc_relay/Constants.sol	21	4	8	**9**	0%
34	ethereum/src/interfaces/IBtcRelayView.sol	27	5	14	**8**	0%

	File	Lines	Blanks	Comments	Code	Complexity
35	ethereum/src/verifiers/NullVerifier.sol	27	2	18	**7**	0%
36	ethereum/src/interfaces/ISettlementModule.sol	47	3	38	**6**	0%
37	ethereum/src/interfaces/IFinalityVerifier.sol	30	2	23	**5**	0%
38	ethereum/src/interfaces/IBridgeProxy.sol	7	1	1	**5**	0%
	Total	**8148**	**1033**	**2734**	**4381**	**17%**

Lines: The total number of lines in each file. This provides a quick overview of the file size and its contents.

Blanks: The count of blank lines in the file.

Comments: This column shows the number of lines that are comments.

Code: The count of lines that actually contain executable code. This metric is essential for understanding how much of the file is dedicated to operational elements rather than comments or whitespace.

Complexity: This column shows the file complexity per line of code. It is calculated by dividing the file's total complexity (an approximation of cyclomatic complexity that estimates logical depth and decision points like loops and conditional branches) by the number of executable lines of code. A higher value suggests greater complexity per line, indicating areas with concentrated logic.

2.5 PROJECT OVERVIEW

The UTEXO Bridge moves value between the Bitcoin/RGB world and EVM networks. On the EVM side, `Bridge` custodies the pooled token and exposes the two value paths: `fundsIn` (inflow) and `fundsOut` (outflow, an enclave-signed release). At `95d8e566` the production `Bridge` is an ERC-1967 proxy (`BridgeProxy`) with implementation storage in `BridgeBaseUpgradeable`. `MinimalBridge` remains the non-upgradeable reference implementation.

Release routing is pluggable. `RouteRegistry` maps a `(sourceChainId, destChainId)` route key to a `FinalityVerifier` and a `SettlementModule`. `RGBVerifier` checks block inclusion through `BtcRelay`. `RgbSettlementModule` records canonical mint/burn inflow. At `95d8e566`, `RgbPoolSettlementModule` was the backing check on the pool route; that module is removed by the remediation. `CommissionManager` computes inbound and outbound fees, now including a flat `baseFee`, against a Chainlink ETH/USD feed. The commission pointer is stored in proxy storage and can be rotated by a timelocked federation operation.

Configuration and the privileged money path are governed by `MultisigProxy`. Federation operations remain timelocked. Enclave quorums still authorize `fundsOut`. This delta adds `emergencyGuardian`, a single key that can pause without signatures. Unpause stays on the federation emergency lane. `UpgradeBridgeImplementation` upgrades the Bridge proxy and rejects a release selector in the initialization calldata. Live-pointer proposals store the signed target and re-check it at execution. The pool settlement module present at `95d8e566` is removed by the remediation.

2.6 CODEBASE QUALITY ASSESSMENT

The Codebase Quality Assessment table offers a comprehensive assessment of various code metrics, as evaluated by our team during the audit, to gauge the overall quality and maturity of the project's codebase. By evaluating factors such as complexity, documentation and testing coverage to best practices, this table highlights areas where the project excels and identifies potential improvement opportunities. Each metric receives an individual rating, offering a clear snapshot of the project's current state, guiding prioritization for refactoring efforts, and providing insights into its maintainability, security, and scalability. For a detailed description of the categories and ratings, see the Codebase Quality Assessment Reference section.

Category	Assessment	Result
Access Control	Federation and enclave quorums remain the core authority model. The guardian is pause-only and cannot lift a federation freeze. A guardian pause consumes `emergencyNonce`. Live-pointer proposals store the signed target and revert `StaleProposalTarget` if it moved. An upgrade to the current implementation is rejected, and `fundsOut` or `rebalanceLiquidity` cannot be the initialization selector. Related: guardian unpause, guardian pause-cycle signatures, live-pointer proposals, upgrade `fundsOut` bypass.	**Good**
Arithmetic	Solidity 0.8 checked arithmetic is used. `fundsOut` now rejects a quote unless `tokenCommission <= amount` and `netAmount == amount - tokenCommission`. A conserved zero net still reverts `ZeroNetAmount`. Related: commission conservation, repeated zero-net check.	**Good**
Complexity	`MultisigProxy` still concentrates proposal dispatch. The proxy and guardian add lanes beside the existing ones. The pool settlement module is removed. Related: no finding is solely a complexity defect.	**Good**
Data Validation	The pool module that cited records without summing them is removed. Inflow and outflow floors cannot be inverted. The `uint64` cap stays on the RGB event path only; non-RGB routes intentionally retain `uint256` accounting. Related: pool settlement binding, independent minima, `uint64` cap.	**Good**

Category	Assessment	Result
Decentralization	Releases remain bounded by the previously audited outflow caps. The guardian can pause, not unpause. Upgrade still requires a federation quorum and a timelock, and cannot skip the enclave through initialization calldata. The acknowledged gap that the federation can install unattested enclave keys remains. Related: guardian unpause, upgrade `fundsOut` bypass.	**Fair**
Documentation	The owner check reads the Ownable slot. The 32-byte `implementation()` heuristic is removed. A candidate is still not identified as a proxy by code hash; UTEXO Protocol accepts that residual. `cancelProposal` documents that authorization is nonce-free and bounded by `MAX_PROPOSAL_LIFETIME`. Related: post-upgrade `owner()`, 32-byte `implementation()` heuristic, `cancelProposal`.	**Good**
External Dependencies	`BtcRelay` rejects heights below the checkpoint and a zero commitment. Chainlink and LayerZero behavior reviewed in the August report is unchanged in this delta. Related: no new external-dependency finding.	**Good**
Error Handling	Custom errors remain consistent. `cancelProposal` rejects a deadline beyond 30 days. The repeated `ZeroNetAmount` check is retained on purpose, after the conservation check. Related: `cancelProposal`, repeated zero-net check.	**Good**
Logging and Monitoring	Federation rotation emits one `FederationSignersUpdated` event that includes the new signer-set version. Related: paired federation events.	**Good**
Low-Level Calls	Token movements still use `SafeERC20`. Upgrade initialization calldata is capped at 4096 bytes on both `MultisigProxy` and `BridgeProxy`. A same-implementation upgrade is rejected, and a release selector in that calldata is rejected. Related: upgrade `fundsOut` bypass, uncapped upgrade calldata.	**Good**
Testing and Verification	The remediation commits add Foundry coverage for the guardian lane, target binding, commission quotes, upgrade guards, floors, calldata limits, the owner slot, and the compatibility marker. The warnings in this report were reproduced on `95d8e566` and checked against the linked remediation commits. Related: no finding is a missing test harness by itself.	**Good**

2.7 FINDINGS BREAKDOWN BY FILE

This table provides an overview of the findings across the audited files, categorized by severity level. It serves as a useful tool for identifying areas that may require attention, helping to prioritize remediation efforts, and provides a clear summary of the audit results.

File	TOTAL	CRITICAL	MAJOR	WARNING	INFO
ethereum/src/MultisigProxy.sol	**9**	0	0	5	4
ethereum/src/Bridge.sol	**4**	0	0	1	3
ethereum/src/BridgeProxy.sol	**3**	0	0	0	3
ethereum/src/CommissionManager.sol	**1**	0	0	0	1
ethereum/src/MinimalBridge.sol	**1**	0	0	0	1
ethereum/src/settlement/RgbPoolSettlementModule.sol	**1**	0	0	1	0
ethereum/test/BridgeRebalance.t.sol	**1**	0	0	1	0

2.8 CONCLUSION

A re-audit was conducted on 38 Solidity files at `95d8e566`, against the August 12, 2026 final report at `ea5de9b4`. No previously fixed finding is confirmed regressed. The new delta does not introduce a critical or major issue. It initially introduced six warnings and nine informational notes, concentrated on the new governance and settlement surfaces: pool backing, a guardian that could lift a federation freeze, unbound live-pointer proposals, commission conservation, and an upgrade that could call `fundsOut`.

Following that report, UTEXO Protocol remediated the findings. The pool settlement module is removed. The guardian is pause-only and consumes the emergency nonce. Live-pointer proposals are bound to the signed target. `fundsOut` checks commission conservation. An upgrade cannot target the current implementation or carry a release selector. Of the 15 findings, 12 are verified fixed, 2 are classified as no issue, and 1 is acknowledged: the compatibility UUID is not treated as proof that a candidate is not a proxy, and a code-hash allowlist is not added. None remain open. The seven acknowledged residuals from the August 12, 2026 report remain accepted.

PRs #151, #152, #153, and #155 landed after `95d8e566` and before the remediation commits. They are included in `c2a5f213`, but they are not the fixes for these 15 findings. Each fixed finding cites its own remediation commit. Those product changes tighten the shared burn identifier and RGB address canonicalization. They do not replace the previously acknowledged limit that `sourceBurnTxId` is still not proven on-chain to be the RGB burn.

As a result, the project has passed our audit. Our auditors have verified that the UTEXO Bridge, as of audited commit `dev@c2a5f213` (bridge-smart-contracts), operates as intended within the defined scope, based on the information and code provided at the time of evaluation. The robustness of the codebase has been significantly improved, meeting the necessary security and functionality requirements established for this audit.

3 FINDINGS REPORT

OXORIO

3.1 CRITICAL

No issues found

3.2 MAJOR

No issues found

3.3 WARNING

W-01	Pool settlement does not bind the released amount to cited mint/burn records in `RgbPoolSettlementModule.sol`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
RgbPoolSettlementModule.sol	contract `RgbPoolSettlementModule` > function `beforeFundsOut`	102
BridgeRebalance.t.sol	production route `97 -> EVM`	162

Description

In the function `beforeFundsOut` of contract `RgbPoolSettlementModule`, the module does not exist at `ea5de9b4`. On the target, `beforeFundsOut` is the backing check for a pool debit. The in-repo production topology registers route `97 -> EVM` with `NullVerifier` and this module. `NullVerifier.verify` is empty, so this function is the only on-chain check on that release.

The function requires each cited id to exist, to equal `amounts[i]`, and to be tagged with the immutable mint/burn chain. It never reads `FundsOutContext.amount`, never sums the cited amounts, never rejects duplicates, and never consumes the record.

```
function beforeFundsOut(FundsOutContext calldata ctx, bytes calldata settlementData)
    external
    view
    override
    onlyRouteRegistry
{
    if (ctx.sourceChainId != poolChainId) {
        revert UnexpectedPoolSource(poolChainId, ctx.sourceChainId);
    }
    (bytes32[] memory operationIds, uint256[] memory amounts) =
```

```
        abi.decode(settlementData, (bytes32[], uint256[]));
    if (operationIds.length != amounts.length) revert SettlementDataLengthMismatch();
    if (operationIds.length == 0) revert EmptySettlementRecords();
    for (uint256 i = 0; i < operationIds.length; i++) {
        uint256 recorded = rgbModule.fundsInRecords(operationIds[i]);
        if (recorded == 0) revert FundsInNotFound(operationIds[i]);
        if (recorded != amounts[i]) revert AmountMismatch(operationIds[i], amounts[i],
recorded);
        uint256 recordChainId = rgbModule.fundsInRecordChainIds(operationIds[i]);
        if (recordChainId != backingRecordChainId) {
            revert FundsInRecordChainMismatch(operationIds[i], backingRecordChainId,
recordChainId);
        }
    }
}
```

A 50-unit mint record satisfies a 100-unit release context. The same 100-unit id listed twice satisfies a 200-unit context. Pool deposits write no canonical record, and a cited record is not consumed, so one historical mint/burn record can be attached to every later pool release. `lockedLiquidity`, the token buckets, and the 20%/24h safety window still bound the outflow. They do not restore a per-release link to the cited records.

This still needs an enclave quorum on `fundsOutCall`. It is not a permissionless call.

Proof of Concept

```
uint256 recorded = 50e6;
uint256 released = 100e6;
_recordCanonical(OPERATION_ID, recorded);

bytes32[] memory operationIds = new bytes32[](1);
operationIds[0] = OPERATION_ID;
uint256[] memory amounts = new uint256[](1);
amounts[0] = recorded;

vm.prank(routeRegistry);
poolModule.beforeFundsOut(_fundsOutContext(released), abi.encode(operationIds, amounts));
```

```
uint256 recorded = 100e6;
_recordCanonical(OPERATION_ID, recorded);

bytes32[] memory operationIds = new bytes32[](2);
operationIds[0] = OPERATION_ID;
```

```
operationIds[1] = OPERATION_ID;
uint256[] memory amounts = new uint256[](2);
amounts[0] = recorded;
amounts[1] = recorded;

vm.prank(routeRegistry);
poolModule.beforeFundsOut(_fundsOutContext(recorded * 2), abi.encode(operationIds, amounts));
```

Recommendation

We recommend to require unique operation ids, require `sum(amounts) == ctx.amount`, and consume cited records if they are exclusive backing:

```
uint256 cited;
for (uint256 i = 0; i < operationIds.length; i++) {
    if (used[operationIds[i]]) revert DuplicateSettlementRecord(operationIds[i]);
    used[operationIds[i]] = true;
    // existing existence / amount / chain checks
    cited += amounts[i];
}
if (cited != ctx.amount) revert BackingAmountMismatch(cited, ctx.amount);
```

Update

Client's response

RgbPoolSettlementModule was removed because the RGB pool flow is no longer supported; RGB now uses mint/burn routes only. Its deployment, pool-route configuration, and associated test fixtures were also removed. The affected pool-release path is therefore absent from the intended deployment, rather than retained with additional backing checks.

RgbOutboundSettlementModule remains for RGB mint/burn rebalances to non-RGB destinations; it is separate from the retired pool flow.

Oxorio's response

Verified in the linked remediation commit. `RgbPoolSettlementModule` and its deploy script are gone, and no production source still references the pool route. The unbound backing check cannot be reached. `RgbOutboundSettlementModule` is a different mint/burn path and is not this finding.

Fixed at: `13b39e99`

W-02	Emergency guardian can unpause a federation freeze without signatures in `MultisigProxy.sol`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
MultisigProxy.sol	contract `MultisigProxy` > function `guardianEmergencyPause`	562
MultisigProxy.sol	contract `MultisigProxy` > function `guardianEmergencyUnpause`	569

Description

In the functions `guardianEmergencyPause` and `guardianEmergencyUnpause` of contract `MultisigProxy`, `emergencyGuardian` does not exist at `ea5de9b4`. On the target, one address can freeze and unfreeze the Bridge with no signatures, nonce, proposal, or timelock. Both functions dispatch the same Bridge calls as the federation emergency path.

```
function guardianEmergencyPause() external {
    if (msg.sender != emergencyGuardian) revert UnauthorizedEmergencyGuardian(msg.sender);
    _emergencyPauseBridge();
    emit GuardianEmergencyPaused(msg.sender);
}

function guardianEmergencyUnpause() external {
    if (msg.sender != emergencyGuardian) revert UnauthorizedEmergencyGuardian(msg.sender);
    _emergencyUnpauseBridge();
    emit GuardianEmergencyUnpaused(msg.sender);
}
```

The constructor requires a non-zero guardian. Disabling it is a later timelocked `SetEmergencyGuardian`. A compromised guardian key can lift a freeze the federation just imposed and make enclave `fundsOut` executable again. The 20%/24h window and the buckets still apply. This matches the README. Under a zero-trust reading of pause control it is a new single-key authority over incident response.

Proof of Concept

```
uint256 nonce = proxy.emergencyNonce();
uint256 deadline = block.timestamp + 1 hours;

bytes32 digest = MultisigHelper.digestEmergencyPause(domainSep, nonce, deadline);
(uint256 bitmap, bytes[] memory sigs) = _fedSigs(digest);
proxy.emergencyPause(nonce, deadline, bitmap, sigs);

assertTrue(bridge.paused(), "federation quorum freezes the bridge");
uint256 nonceAfterFederationPause = proxy.emergencyNonce();
assertEq(nonceAfterFederationPause, nonce + 1, "federation lane consumes its nonce");

vm.prank(emergencyGuardian);
proxy.guardianEmergencyUnpause();

assertFalse(bridge.paused(), "a single unsigned guardian call lifts the federation freeze");
assertEq(
    proxy.emergencyNonce(),
    nonceAfterFederationPause,
    "guardian unpause leaves emergencyNonce untouched, so it is invisible to the federation
lane"
);
```

Recommendation

We recommend to restrict the guardian to pause-only. Leave unpause on the federation emergency lane:

```
function guardianEmergencyUnpause() external {
    revert GuardianCannotUnpause();
}
```

Update

Client's response

The emergency guardian is now pause-only. **guardianEmergencyUnpause** was removed from the contract and interface, together with its event and interaction script. The guardian can still immediately freeze both inflow and outflow without signatures, but cannot lift either its own pause or a federation-imposed freeze.

Resuming through **emergencyUnpause** requires a valid federation quorum and remains available without a timelock. Tests cover rejection of the removed guardian selector, rejection of unsigned unpause attempts, and successful unpause with federation authorization.

Oxorio's response

Verified in the linked remediation commit. `guardianEmergencyUnpause` is removed from the contract and the interface. `guardianEmergencyPause` remains. `emergencyUnpause` still requires a federation quorum. A single guardian key can freeze the bridge and cannot lift that freeze.

Fixed at: `ee2d9074`

W-03	Guardian pause cycles leave federation emergency signatures valid in `MultisigProxy.sol`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
MultisigProxy.sol	contract `MultisigProxy` > function `emergencyPause`	528
MultisigProxy.sol	contract `MultisigProxy` > function `guardianEmergencyPause`	562

Description

In the function `emergencyPause` of contract `MultisigProxy`, federation `emergencyPause` commits to `(nonce, deadline)` and increments `emergencyNonce`. Guardian pause and unpause call `_emergencyPauseBridge` / `_emergencyUnpauseBridge` and do not touch that nonce. `emergencyPause` also has no upper bound on `deadline`. That missing ceiling was already true at `ea5de9b4`. What this delta adds is the guardian path, which can change pause state without invalidating a bundle the federation already signed.

```
function emergencyPause(uint256 nonce, uint256 deadline, uint256 fedBitmap, bytes[] calldata
fedSigs) external {
    if (block.timestamp > deadline) revert Expired();
    if (nonce != emergencyNonce) revert InvalidNonce();
    // verify, then:
    emergencyNonce++;
    _emergencyPauseBridge();
}
```

Sequence:

1. Federation signs `emergencyPause` for the current nonce with `deadline = now + 10 years`.
2. Guardian pauses, then unpauses. The bridge is running. `emergencyNonce` is unchanged.
3. Anyone holding the original signatures submits `emergencyPause`.
4. Inflow and outflow freeze again. The signatures cannot be retracted on-chain except by using them, waiting for `deadline`, or rotating the federation signer set.

Proof of Concept
The signed deadline is `block.timestamp + 10 * 365 days`.

```
uint256 nonce = proxy.emergencyNonce();
// `emergencyPause` only rejects expired deadlines; `MAX_TEE_DEADLINE` is
// not applied here, so a decade-long bundle is accepted below.
uint256 deadline = block.timestamp + 10 * 365 days;

bytes32 digest = MultisigHelper.digestEmergencyPause(domainSep, nonce, deadline);
(uint256 bitmap, bytes[] memory sigs) = _fedSigs(digest);

vm.prank(emergencyGuardian);
proxy.guardianEmergencyPause();
assertTrue(bridge.paused(), "guardian freeze applies");
assertEq(proxy.emergencyNonce(), nonce, "guardian pause does not consume the federation
nonce");

vm.prank(emergencyGuardian);
proxy.guardianEmergencyUnpause();
assertFalse(bridge.paused(), "guardian lifts its own freeze");
assertEq(proxy.emergencyNonce(), nonce, "guardian unpause does not consume the federation
nonce either");

proxy.emergencyPause(nonce, deadline, bitmap, sigs);

assertTrue(bridge.paused(), "pre-signed federation bundle still freezes the bridge after a
guardian cycle");
assertEq(proxy.emergencyNonce(), nonce + 1, "only the federation call advances
emergencyNonce");
```

Recommendation

We recommend to consume `emergencyNonce` on the guardian path, or bind an emergency signature to the current pause state. Cap emergency deadlines with `MAX_TEE_DEADLINE`.

Update

Client's response

Every successful **guardianEmergencyPause** now increments **emergencyNonce**, including repeated calls while the **Bridge** is already paused. Previously signed federation pause and unpause commands therefore fail with `InvalidNonce`. Reverted guardian calls consume no nonce, and the proposal and TEE nonce lanes remain unchanged.

Both federation emergency operations now require `block.timestamp <= deadline <= block.timestamp + MAX_EMERGENCY_DEADLINE`, with a dedicated one-day limit. This bounds the remaining validity at submission, rather than proving the age of the signature.

Oxorio's response

Verified in the linked remediation commit. `guardianEmergencyPause` increments `emergencyNonce` before the pause call, including when the bridge is already paused, because `emergencyPauseAll` does not revert on a second pause. A reverted guardian transaction does not consume the nonce. `MAX_EMERGENCY_DEADLINE` is 1 day, the same bound as `MAX_TEE_DEADLINE`. A previously signed federation bundle no longer matches the nonce after a guardian pause.

Fixed at: `1d23e977`

W-04	Generic governance proposals remain unbound to a target while `StaleBridgeTarget` wraps only the upgrade lane in `MultisigProxy.sol`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
MultisigProxy.sol	contract `MultisigProxy` > function `proposeAdminExecute`	580
MultisigProxy.sol	contract `MultisigProxy` > function `executeProposal`	1075
MultisigProxy.sol	contract `MultisigProxy` > function `_executeByType`	1187
MultisigProxy.sol	contract `MultisigProxy` > function `_executeByType`	1258
MultisigProxy.sol	contract `MultisigProxy` > function `_executeByType`	1334

Description

In the function `proposeAdminExecute` of contract `MultisigProxy`, the EIP-712 payload commits to the selector and the calldata only. It does not name the address the call will land on. Execution in `_executeByType` then uses the live `bridge`, `commissionManager`, `lzAdapter`, or `routeRegistry()` pointer.

```
bytes32 structHash =
    keccak256(abi.encode(_PROPOSE_ADMIN_EXECUTE_TYPEHASH, selector, keccak256(callData),
nonce, deadline));

(bool ok, bytes memory ret) = bridge.call(opData);
```

This delta added a target check, and it wraps only the new upgrade lane. `proposeUpgradeBridgeImplementation` and its execute branch revert `StaleBridgeTarget` if the pointer moved:

```
if (bridgeProxy != bridge) revert StaleBridgeTarget(bridgeProxy, bridge);
```

Generic `AdminExecute`, `AdminExecuteCommissionManager`, `AdminExecuteRouteRegistry`, `AdminExecuteAdapter`, typed commission withdrawals, and `SetRoute` have no such wrap. `executeProposal` is permissionless. It checks that the chosen proposal is `Pending`, that `federationVersion` still matches, that its own `timelockSnapshot` has elapsed, that it is not expired, and that `opData` matches `dataHash`. It does not order independent matured proposals.

`CommissionManager` A holds 10_000 TOKEN in fees. The federation signs two proposals:

1. `WithdrawTokenCommissionCM(TOKEN, 10_000e6)`, meant to withdraw from A.
2. `UpdateCommissionManager(B)`, which repoints both `Bridge` and `MultisigProxy` to B.

The withdrawal digest is `(token, amount, nonce, deadline)`. It never names A. Anyone can execute the rotation first, then the withdrawal. The withdrawal call lands on B, not on A. A keeps its 10_000 TOKEN. If B has a balance, that balance is paid to `commissionRecipient` instead. The withdrawal succeeds, but it drains the wrong contract.

The same ordering on `UpgradeBridgeImplementation` reverts `StaleBridgeTarget`.

The same split applies to `AdminExecute` after `UpdateBridge`, `SetRoute` after `UpdateRouteRegistry` (the registry is read from the live Bridge), and `AdminExecuteAdapter` after `UpdateLZAdapter`. Both proposals already carry a federation quorum. The gap is that `StaleBridgeTarget` was introduced for upgrade and was not applied to the other live pointers.

Proof of Concept

```
bytes memory callData = abi.encodeCall(LiveTargetDummy.ping, (7));
uint256 deadline = block.timestamp + 1 days;

uint256 adminNonce = proxy.proposalNonce();
bytes32 adminDigest =
    MultisigHelper.digestProposeAdminExecute(domainSep, bytes4(callData), callData,
adminNonce, deadline);
(uint256 adminBitmap, bytes[] memory adminSigs) = _fedSigs(adminDigest);
bytes32 adminId = proxy.proposeAdminExecute(callData, adminNonce, deadline, adminBitmap,
adminSigs);

uint256 updateNonce = proxy.proposalNonce();
bytes32 updateDigest =
    MultisigHelper.digestProposeUpdateBridge(domainSep, address(replacementTarget),
updateNonce, deadline);
(uint256 updateBitmap, bytes[] memory updateSigs) = _fedSigs(updateDigest);
bytes32 updateId = proxy.proposeUpdateBridge(
```

```
    address(replacementTarget), updateNonce, deadline, updateBitmap, updateSigs
);

vm.warp(block.timestamp + TIMELOCK + 1);
proxy.executeProposal(updateId, abi.encode(address(replacementTarget)));
proxy.executeProposal(adminId, callData);

assertEq(proxy.bridge(), address(replacementTarget));
assertEq(originalTarget.lastValue(), 0, "signed calldata must not hit the original target
after rotation");
assertEq(replacementTarget.lastValue(), 7, "live pointer substitution applies the matured
admin call");
```

Recommendation

We recommend wrapping every live-pointer lane the same way as upgrade: put the expected address in the signed payload and stored `opData`, and require equality at execution:

```
bytes32 structHash = keccak256(
    abi.encode(_PROPOSE_ADMIN_EXECUTE_TYPEHASH, bridge, selector, keccak256(callData),
nonce, deadline)
);

if (expectedTarget != bridge) revert StaleBridgeTarget(expectedTarget, bridge);
```

Update

Client's response

All eleven governance operations that resolve mutable external targets now include the target address in their EIP-712 signed payload and store it as **expectedTarget**. This covers generic calls to **Bridge**, **CommissionManager**, **RouteRegistry** and the adapter, commission withdrawals, route configuration, commission-manager and registry updates, and inflow pause/unpause.

Before execution, `executeProposal` compares the currently resolved target with the signed snapshot. A mismatch reverts with `StaleProposalTarget`, leaving the proposal pending; successful execution dispatches to the validated address. A pending operation therefore cannot silently execute against a replacement contract. Signatures collected before proposal creation also cannot follow a changed target.

Oxorio's response

Verified in the linked remediation commit. Eleven lanes store `expectedTarget` and put that address in the EIP-712 payload: `AdminExecute`, both commission withdrawals, `AdminExecuteCommissionManager`, `AdminExecuteRouteRegistry`, `AdminExecuteAdapter`, `SetRoute`, `UpdateCommissionManager`, `UpdateRouteRegistry`, `PauseInflow`, and `UnpauseInflow`. `executeProposal` re-reads the live pointer and reverts `StaleProposalTarget` on a mismatch. The withdrawal that used to follow a rotated commission manager now reverts instead.

Fixed at: `342fa317`

W-05	`fundsOut` does not enforce commission conservation after `CommissionManager` became mutable in `Bridge.sol`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Bridge.sol	contract `Bridge` > function `fundsOut`	560
MultisigProxy.sol	contract `MultisigProxy` > `UpdateCommissionManager`	1272

Description

In the function `fundsOut` of contract `Bridge`, at `ea5de9b4`, `Bridge.commissionManager` is `immutable`, and `fundsOut` already forwards `netAmount` without checking `tokenCommission + netAmount == amount`. The target stores the pointer in proxy storage and adds `UpdateCommissionManager`, which only rejects the zero address.

```
} else if (opType == OperationType.UpdateCommissionManager) {
    address newCm = abi.decode(opData, (address));
    if (newCm == address(0)) revert ZeroCommissionManager();
    address old = commissionManager;
    IBridge(bridge).setCommissionManager(newCm);
    commissionManager = newCm;
    emit CommissionManagerUpdated(old, newCm);
}
```

```
(uint256 tokenCommission,, uint256 netAmount) =
commissionManager.calculateFundsOutCommission(
    fundsOutParams.sourceChainId, fundsOutParams.destinationChainId, TOKEN,
fundsOutParams.amount
);
if (netAmount == 0) revert ZeroNetAmount();
// buckets and the 20% window consume fundsOutParams.amount
```

```
if (tokenCommission != 0) _forwardTokenCommission(tokenCommission);
IERC20(TOKEN).safeTransfer(fundsOutParams.recipient, netAmount);
```

The shipped `CommissionManager` still returns `tokenCommission + netAmount == amount` for `TOKEN`, and it rejects `NATIVE` on `FUNDS_OUT` at configuration time. A replacement installed by a federation quorum after the timelock can return `netAmount = amount * 2`. Accounting consumes `amount`. The transfer sends `2 * amount`. The extra tokens come from the shared ERC-20 balance of the other source chains. An honest enclave signature is enough for the payout. The rotation is the same trust class as a malicious upgrade. The new fact is that the money-path invariant is no longer structural.

Proof of Concept

The replacement manager is rotated through the full governance path: federation propose, timelock, `executeProposal` on `UpdateCommissionManager`. It quotes `netAmount = 2 * amount`. The release is a normal enclave-signed `fundsOutCall`.

```
DoublePayoutCommissionManager rogue = new DoublePayoutCommissionManager();
_rotateCommissionManagerThroughGovernance(address(rogue));
assertEq(address(bridge.commissionManager()), address(rogue), "governance rotated the
commission manager");

uint256 lockedBefore = bridge.lockedLiquidity(RGB_CHAIN_ID);
uint256 totalLockedBefore = bridge.totalLockedLiquidity();
uint256 bridgeBalanceBefore = token.balanceOf(address(bridge));

_enclaveFundsOut(AMOUNT);

// Payout follows the manager's `netAmount`; accounting follows `amount`.
assertEq(token.balanceOf(recipient), 2 * AMOUNT, "recipient received twice the released
amount");
assertEq(bridgeBalanceBefore - token.balanceOf(address(bridge)), 2 * AMOUNT, "custody fell
by the net payout");
assertEq(lockedBefore - bridge.lockedLiquidity(RGB_CHAIN_ID), AMOUNT, "chain liquidity fell
by amount only");
assertEq(totalLockedBefore - bridge.totalLockedLiquidity(), AMOUNT, "total liquidity fell by
amount only");
```

Recommendation

We recommend to apply the following change:

```
if (tokenCommission + netAmount != fundsOutParams.amount) revert
CommissionConservationBroken();
```

On rotation, require `newCm.code.length != 0` and `ICommissionManager(newCm).bridgeAddress() == bridge`.

Update

Client's response

`Bridge.fundsOut` now enforces commission conservation independently of the replaceable manager: `tokenCommission <= amount and netAmount == amount - tokenCommission`. This is equivalent to `tokenCommission + netAmount == amount` without risking overflow on a malformed quote. Invalid quotes revert with `CommissionConservationBroken` before commission forwarding or recipient payment.

Commission-manager rotation also requires deployed code and `bridgeAddress() == address(this)`. Tests cover invalid replacement addresses and an inflated quote after a timelocked rotation, verifying that the rejected release preserves balances, liquidity accounting, replay state and the TEE nonce, and can be retried with the same authorization once the quote is corrected.

Oxorio's response

Verified in the linked remediation commit. `fundsOut` reverts `CommissionConservationBroken` unless `tokenCommission <= amount` and `netAmount == amount - tokenCommission`. That is the conservation check, written so a malformed quote cannot overflow the sum. `setCommissionManager` rejects an address with no code and a manager whose `bridgeAddress()` is not this bridge.

Fixed at: `40747b9d`

W-06	Upgrade to the current implementation can call `fundsOut` and skip the enclave in `MultisigProxy.sol`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
MultisigProxy.sol	contract `MultisigProxy` > function `proposeUpgradeBridgeImplementation`	701
MultisigProxy.sol	contract `MultisigProxy` > function `_executeByType`	1331
MultisigProxy.sol	contract `MultisigProxy` > function `_requireNotBridgeReleaseSelector`	1520

Description

In the function `proposeUpgradeBridgeImplementation` of contract `MultisigProxy`, `BridgeProxy` and `UpgradeBridgeImplementation` do not exist at `ea5de9b4`. On the target, `proposeUpgradeBridgeImplementation` rejects a zero-code target and `newImplementation == bridgeProxy`. It does not reject the current implementation, and it does not look at the selector inside `initializationData`.

`_requireNotBridgeReleaseSelector` rejects `fundsOut` and `rebalanceLiquidity` on every generic lane. `ERC1967Utils.upgradeToAndCall` still `delegatecall`s whenever `data.length > 0`, including when the implementation address does not change. During that delegatecall `msg.sender` is `MultisigProxy`, which is `Bridge.owner()`, so `fundsOut` passes `onlyOwner`.

```
} else if (opType == OperationType.UpgradeBridgeImplementation) {
    (address bridgeProxy, address newImplementation, bytes memory initializationData) =
        abi.decode(opData, (address, address, bytes));
    if (bridgeProxy != bridge) revert StaleBridgeTarget(bridgeProxy, bridge);
    if (newImplementation.code.length == 0 || newImplementation == bridgeProxy) {
        revert InvalidBridgeImplementation(newImplementation);
    }
```

```
        IBridgeProxy(bridgeProxy).upgradeToAndCall(newImplementation, initializationData);
    }
```

A federation quorum plus the timelock can sign the upgrade operation with the current implementation and `fundsOut` calldata:

```
UpgradeBridgeImplementation(
    currentImplementation,
    abi.encodeCall(Bridge.fundsOut, (...))
)
```

After execution the implementation address is unchanged, the recipient receives the tokens, and `teeNonce` does not move. The quorum signs the hash of that calldata, so this is not a hidden payload. The operation is named as an upgrade and behaves as a release. `proposeAdminExecute` with the same calldata reverts `ForbiddenBridgeReleaseSelector`.

Proof of Concept

The same `releaseCallData` blob is rejected on the generic lane and accepted as `initializationData`.

```
bytes memory releaseCallData = abi.encodeCall(Bridge.fundsOut, (_fundsOutParams(AMOUNT)));
uint256 deadline = block.timestamp + 1 days;

// Generic lane: the release selector is rejected before any signature check.
uint256 adminNonce = proxy.proposalNonce();
bytes32 adminDigest = MultisigHelper.digestProposeAdminExecute(
    domainSep, IBridge.fundsOut.selector, releaseCallData, adminNonce, deadline
);
vm.expectRevert(
    abi.encodeWithSelector(IMultisigProxy.ForbiddenBridgeReleaseSelector.selector,
IBridge.fundsOut.selector)
);
proxy.proposeAdminExecute(releaseCallData, adminNonce, deadline, 0x3, _fedSigs(adminDigest));

// Upgrade lane: the very same calldata rides in as `initializationData`
// of an upgrade whose target is the implementation already installed.
uint256 upgradeNonce = proxy.proposalNonce();
bytes32 upgradeDigest = MultisigHelper.digestProposeUpgradeBridgeImplementation(
    domainSep, address(bridge), address(implementation), releaseCallData, upgradeNonce,
deadline
);
bytes32 upgradeId = proxy.proposeUpgradeBridgeImplementation(
    address(bridge), address(implementation), releaseCallData, upgradeNonce, deadline, 0x3,
```

```
_fedSigs(upgradeDigest)
);

uint256 teeNonceBefore = proxy.teeNonce(RGB_CHAIN_ID);
uint256 lockedBefore = bridge.lockedLiquidity(RGB_CHAIN_ID);

vm.warp(block.timestamp + TIMELOCK + 1);
proxy.executeProposal(upgradeId, abi.encode(address(bridge), address(implementation),
releaseCallData));

assertEq(bridgeProxy.implementation(), address(implementation), "implementation slot is
unchanged");
assertEq(token.balanceOf(recipient), AMOUNT, "recipient was paid through the initialization
delegatecall");
assertEq(lockedBefore - bridge.lockedLiquidity(RGB_CHAIN_ID), AMOUNT, "the release debited
chain liquidity");
assertEq(proxy.teeNonce(RGB_CHAIN_ID), teeNonceBefore, "no enclave nonce was consumed");
```

Recommendation

We recommend to reject an upgrade whose `newImplementation` equals the current implementation, both in `proposeUpgradeBridgeImplementation` and in `_executeByType`. `ERC1967Utils.upgradeToAndCall` still `delegatecall`s non-empty `initializationData` when the address does not change, so that call is a release under another name.

We also recommend to reject `fundsOut` and `rebalanceLiquidity` inside `initializationData` when the implementation does change. The same-address check does not cover that case. Prefer empty upgrade calldata unless the selector is an explicit reinitializer.

Update

Client's response

MultisigProxy now validates upgrades at both proposal creation and execution. The candidate must differ from the current implementation, and non-empty initialization calldata must contain a complete selector and cannot directly invoke `fundsOut` or `rebalanceLiquidity`. This also rejects a pending upgrade if another proposal has already installed its candidate. Generic governance call paths reject `upgradeToAndCall`, preventing them from bypassing the typed upgrade checks.

Empty calldata and versioned reinitializers remain supported; the upgrade EIP-712 schema is unchanged. These restrictions apply to the federation governance entrypoint and block the direct release-selector bypass described in the finding.

Oxorio's response

Verified in the linked remediation commit. `_requireValidBridgeUpgrade` runs on propose and on execute. It rejects `newImplementation` equal to the current implementation, and it rejects `fundsOut` and `rebalanceLiquidity` as the outer selector of `initializationData`. The same-implementation `fundsOut` bypass no longer passes propose.

Fixed at: `56479026`

3.4 INFO

I-01	`cancelProposal` authorization has no nonce and no upper deadline bound in `MultisigProxy.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
MultisigProxy.sol	contract `MultisigProxy` > function `cancelProposal`	1054

Description

In the function `cancelProposal` of contract `MultisigProxy`, at `ea5de9b4`, `cancelProposal` takes `nonce`, requires `nonce == proposalNonce`, puts that nonce in the digest, and increments `proposalNonce`. It already has no `MAX_PROPOSAL_LIFETIME` check. The target drops the nonce. The digest is `(proposalId, deadline)` only. `_propose` still rejects `deadline > block.timestamp + 30 days`. `cancelProposal` does not.

```
function cancelProposal(bytes32 proposalId, uint256 deadline, uint256 fedBitmap, bytes[]
calldata fedSigs) external {
    if (block.timestamp > deadline) revert Expired();
    Proposal storage p = _proposals[proposalId];
    if (p.status != ProposalStatus.Pending) revert NotPending();
    bytes32 digest = _hashTypedData(keccak256(abi.encode(_CANCEL_PROPOSAL_TYPEHASH,
proposalId, deadline)));
    _verifySignatures(digest, fedBitmap, fedSigs, _federationSigners, federationThreshold);
    p.status = ProposalStatus.Cancelled;
}
```

A cancellation signed with `deadline = now + 10 years` stays executable while the proposal is `Pending`. Unrelated proposals no longer invalidate it. Replay after cancellation is blocked by status `Cancelled`. The quorum cannot retract the signatures on-chain except by executing the proposal, waiting out `deadline`, or rotating the federation signers.

The new part is the removed nonce. The missing deadline ceiling was already there.

Proof of Concept

```
uint256 proposalDeadline = block.timestamp + 1 days;
bytes32 proposalA = _proposePing(1, proposalDeadline);

// A decade-long deadline is rejected on the propose lane ...
uint256 farDeadline = block.timestamp + 10 * 365 days;
bytes memory farCallData = abi.encodeCall(CancelTargetDummy.ping, (3));
uint256 liveNonce = proxy.proposalNonce();
bytes[] memory noSigs = new bytes[](0);
vm.expectRevert(IMultisigProxy.DeadlineTooFar.selector);
proxy.proposeAdminExecute(farCallData, liveNonce, farDeadline, 0, noSigs);

// ... but `cancelProposal` has no MAX_PROPOSAL_LIFETIME guard, so the
// same deadline is signable for a cancel bundle.
bytes32 cancelDigest = MultisigHelper.digestCancelProposal(domainSep, proposalA,
farDeadline);
(uint256 cancelBitmap, bytes[] memory cancelSigs) = _fedSigs(cancelDigest);

uint256 nonceAtSigning = proxy.proposalNonce();
bytes32 proposalB = _proposePing(2, proposalDeadline);
assertEq(proxy.proposalNonce(), nonceAtSigning + 1, "proposal lane advanced after the cancel
bundle was signed");

proxy.cancelProposal(proposalA, farDeadline, cancelBitmap, cancelSigs);

assertEq(
    uint256(proxy.getProposal(proposalA).status),
    uint256(IMultisigProxy.ProposalStatus.Cancelled),
    "stale cancel bundle still cancels proposal A"
);
assertEq(
    uint256(proxy.getProposal(proposalB).status),
    uint256(IMultisigProxy.ProposalStatus.Pending),
    "cancel is bound to one proposalId only"
);
```

Recommendation

We recommend to apply the following change:

```
if (deadline > block.timestamp + MAX_PROPOSAL_LIFETIME) revert DeadlineTooFar();
```

Update

Client's response

`cancelProposal` now enforces `block.timestamp <= deadline <= block.timestamp + MAX_PROPOSAL_LIFETIME`, with the existing 30-day limit. Expired authorizations and deadlines beyond that ceiling are rejected. The ceiling bounds remaining validity at submission, rather than the age of the signature.

Cancellation intentionally remains nonce-free. The EIP-712 authorization is bound to one exact **proposalId**, and cancellation requires that proposal to be **Pending**. A successful call changes it to **Cancelled**, blocking replay; the same signatures cannot cancel a different proposal. Advancing `proposalNonce` through unrelated proposals therefore does not invalidate an already-authorized cancellation.

Oxorio's response

Verified in the linked remediation commit. `cancelProposal` reverts `DeadlineTooFar` when `deadline > block.timestamp + MAX_PROPOSAL_LIFETIME` (30 days), the same ceiling `_propose` already used. The digest stays `(proposalId, deadline)`. That nonce-free shape is what the response describes, and a cancelled proposal cannot be cancelled again because status must be `Pending`.

Fixed at: `6cc22621`

I-02	`minFundsInAmount` and `minFundsOutAmount` are independent in `Bridge.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Bridge.sol	contract `Bridge` > function `setMinFundsInAmount`	344
Bridge.sol	contract `Bridge` > function `setMinFundsOutAmount`	355

Description

In the function `setMinFundsInAmount` of contract `Bridge`, `setMinFundsOutAmount` does not exist at `ea5de9b4`. On the target each setter only rejects zero. Nothing requires `minFundsOutAmount <= minFundsInAmount`.

```
function setMinFundsInAmount(uint256 newMinimum) external override onlyOwner {
    if (newMinimum == 0) revert InvalidMinFundsInAmount();
    minFundsInAmount = newMinimum;
}

function setMinFundsOutAmount(uint256 newMinimum) external override onlyOwner {
    if (newMinimum == 0) revert InvalidMinFundsOutAmount();
    minFundsOutAmount = newMinimum;
}
```

A deposit at `minFundsInAmount` is accepted. After token commission, `netAmount` can sit below `minFundsOutAmount`. `fundsOut` then reverts `AmountBelowMinimum` for a release of that size. The tokens stay in the pool and can leave only inside a larger release. This is an operator footgun, not a third-party loss.

Proof of Concept
Code inspection on `95d8e566`. `setMinFundsOutAmount` is absent from `ea5de9b4`. Neither setter reads the other floor. `fundsOut` enforces only `params.amount >= minFundsOutAmount`.

Recommendation

We recommend to reject a floor update that inverts the two minima, or document that a single minimum deposit is not guaranteed to be withdrawable on its own.

Update

Client's response

Bridge now enforces `minFundsInAmount >= minFundsOutAmount` during initialization and in both setters. Updates that invert this ordering revert with `InvalidAmountFloors`, using the counterpart's current value at execution. Equal minima and different minima with a higher inbound floor remain supported.

Oxorio's response

Verified in the linked remediation commit. `initialize`, `setMinFundsInAmount`, and `setMinFundsOutAmount` all revert `InvalidAmountFloors` when the inbound floor would fall below the outbound floor. A minimum deposit can no longer be configured below the minimum release.

Fixed at: `4d39173a`

I-03	`fundsOut` repeats a zero-net check the shipped manager already makes in `Bridge.sol`
Severity	**INFO**
Status	• NO ISSUE

Location

File	Location	Line
Bridge.sol	contract `Bridge` > function `fundsOut`	564
CommissionManager.sol	contract `CommissionManager` > function `calculateFundsOutCommission`	264

Description

In the function `fundsOut` of contract `Bridge`, `CommissionManager.calculateFundsOutCommission` reverts `ZeroNetAmount(amount, totalFee)` when a `TOKEN` fee consumes the whole amount. `Bridge.fundsOut` then checks `netAmount == 0` again. That Bridge check is not present at `ea5de9b4`. The manager check is.

```
if (totalFee >= amount) revert ZeroNetAmount(amount, totalFee);
// Bridge.fundsOut:
if (netAmount == 0) revert ZeroNetAmount();
```

On the shipped manager the Bridge check is unreachable for `TOKEN`: the quote reverts first. A wrong-side quote returns `(0, 0, amount)`, so the net is not zero either. The Bridge check still matters if a rotated manager returns a zero net without reverting.

Proof of Concept
Code inspection. The two reverts use different errors: `CommissionManager.ZeroNetAmount(uint256,uint256)` and `Bridge.ZeroNetAmount()`.

Recommendation

We recommend to keep the Bridge check until conservation is enforced. `tokenCommission + netAmount == amount` also rejects a zero net. After that, the standalone `netAmount == 0` check is redundant for the shipped manager.

Update

Client's response

The Bridge-level ZeroNetAmount check was intentionally retained. The shipped **CommissionManager** already rejects a fee that consumes the entire amount, but a replacement manager could return `tokenCommission = amount` and `netAmount = 0` without reverting. That quote satisfies the conservation invariant added for W-05.

The checks therefore enforce separate requirements: conservation of the gross release amount and a strictly positive net payout. Keeping the latter in **Bridge** preserves this guarantee independently of the manager implementation.

Oxorio's response

No issue. The conservation check allows `tokenCommission == amount` and `netAmount == 0`. The following `ZeroNetAmount` check is what rejects that quote. Keeping it is stricter than the shipped manager alone, and it is not a defect.

I-04	Production `fundsIn` does not share `MinimalBridge`'s `uint64` cap in `Bridge.sol`
Severity	**INFO**
Status	• NO ISSUE

Location

File	Location	Line
MinimalBridge.sol	contract `MinimalBridge` > function `fundsIn`	50
Bridge.sol	contract `Bridge` > function `_emitRgbFundsIn`	756

Description

In the function `fundsIn` of contract `MinimalBridge`, at `ea5de9b4`, `BaseBridge.fundsIn` (now renamed `MinimalBridge`) emits `uint256` and has no `uint64` check. The target adds the check only on `MinimalBridge`, and on the production Bridge only when a settlement module returns a non-zero RGB op id.

```
// MinimalBridge.fundsIn
if (amount > type(uint64).max) revert AmountExceedsUint64(amount);

// Bridge._emitRgbFundsIn, called only when rgbOpId != 0
if (amount > type(uint64).max) revert AmountExceedsUint64(amount);
emit FundsIn(sender, rgbOpId, uint64(amount));
```

`RgbPoolSettlementModule.onFundsIn` and `NullSettlementModule.onFundsIn` return `0`, so those deposits skip the check and can lock more than `2^64 - 1` units. A mint/burn deposit that does emit `FundsIn` reverts the whole deposit at the event. At USDT0's 6 decimals the bound is about `1.8e13` tokens. This is an inconsistency introduced in the delta, not a realistic DoS.

Proof of Concept
Code inspection of the delta. `git diff ea5de9b4..95d8e566` adds the `uint64` check to `MinimalBridge.fundsIn` and to `Bridge._emitRgbFundsIn` only.

Recommendation

We recommend, if every production asset is `uint64`-shaped, to reject `amount > type(uint64).max` once in `_fundsIn`, including routes that emit no RGB event.

Update

Client's response

The `uint64` bound is intentionally retained on the **RGB** compatibility surface. **MinimalBridge** always emits the uint64-shaped `FundsIn` event, while production **Bridge** emits it only when the settlement module returns a **non-zero RGB OpId**. `_emitRgbFundsIn` checks the amount before casting, so an oversized **RGB** amount reverts the entire operation without truncation. For production deposits, this is the credited netAmount after commission.

Non-RGB routes use `uint256` accounting and the generic `BridgeFundsIn` event. A universal cap would impose an **RGB** encoding constraint on routes that do not use that format, so we kept the restriction route-specific.

Oxorio's response

No issue. `_emitRgbFundsIn` still rejects `amount > type(uint64).max`, and production `fundsIn` calls it only when the settlement module returns a non-zero RGB op id, passing `netAmount`. Routes that emit no RGB event stay on `uint256`. That matches the stated scope of the bound.

I-05	Upgrade calldata has no length cap in `BridgeProxy.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
BridgeProxy.sol	contract `BridgeProxy` > function `upgradeToAndCall`	41
MultisigProxy.sol	contract `MultisigProxy` > function `proposeUpgradeBridgeImplementation`	691

Description

In the function `upgradeToAndCall` of contract `BridgeProxy`, `BridgeProxy` is new. `upgradeToAndCall` forwards `data` with no maximum length. `proposeUpgradeBridgeImplementation` hashes `initializationData` and checks the implementation address, not the size. `Bridge` caps `settlementData` at 4096 bytes. This path has no equivalent cap.

The federation signs the hash, so this is not unsigned arbitrary execution. The `fundsOut` bypass is the WARNING above. What remains is that a signed blob is bounded only by block gas. A revert during execution rolls the transaction back, so the proposal stays `Pending` until its deadline.

Proof of Concept
Code inspection. `proposeUpgradeBridgeImplementation` checks `newImplementation.code.length` and `keccak256(initializationData)`. It does not check `initializationData.length`.

Recommendation

We recommend to cap `initializationData` the same way Bridge caps `settlementData`.

Update

Client's response

Upgrade initialization calldata is now capped at 4096 bytes, including the selector. **MultisigProxy** enforces the limit at proposal creation and execution, and

`BridgeProxy.upgradeToAndCall` independently enforces it before delegatecall. Oversized payloads revert with `UpgradeCallDataTooLong`; rejected proposal creation preserves the nonce, while rejected execution leaves the proposal pending and the implementation unchanged.

The limit applies to the initialization bytes, not the outer ABI-encoded operation. Empty calldata and exactly 4096 bytes remain supported. Tests cover the boundary and oversized rejection.

Oxorio's response

Verified in the linked remediation commit. Both `MultisigProxy._requireValidBridgeUpgrade` and `BridgeProxy.upgradeToAndCall` revert `UpgradeCallDataTooLong` above 4096 bytes. The check runs at propose and at execute, and again inside the proxy before the delegatecall.

Fixed at: `7263af82`

I-06	Post-upgrade `owner()` is a single call, not a stable read in `BridgeProxy.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
BridgeProxy.sol	contract `BridgeProxy` > function `upgradeToAndCall`	51

Description

In the function `upgradeToAndCall` of contract `BridgeProxy`, `BridgeProxy` is new. After the upgrade delegatecall, the proxy checks `owner()` once:

```
ERC1967Utils.upgradeToAndCall(newImplementation, data);
if (IERC5313(address(this)).owner() != msg.sender) {
    revert IncompatibleBridgeImplementation(newImplementation);
}
```

`msg.sender` in this check is the multisig. The shipped Bridge reads the Ownable owner from storage, so the check passes. The check does not read that slot. It trusts whatever `owner()` returns, and it calls `owner()` once.

An implementation the federation already signed can answer with the multisig on that call and with another address after it:

```
uint256 calls;

function owner() external returns (address) {
    if (calls++ == 0) return multisig;
    return other;
}
```

The upgrade sees `multisig` and commits. The next `owner()` returns `other`. The proxy does not look again.

Proof of Concept
Code inspection of `upgradeToAndCall`. The owner check is one external call after `upgradeToAndCall`, with no second read.

Recommendation

We recommend to read the Ownable owner slot instead of calling `owner()`. In the shipped OpenZeppelin layout that slot is `0x9016d09d72d40fdae2fd8ceac6b6234c7706214fd39c1cd1e609a0528c199300`, and `_owner` is its first field:

```
bytes32 private constant OWNABLE_OWNER_SLOT =
    0x9016d09d72d40fdae2fd8ceac6b6234c7706214fd39c1cd1e609a0528c199300;

function upgradeToAndCall(address newImplementation, bytes calldata data) external payable
onlyBridgeOwner {
    _requireCompatibleImplementation(newImplementation);
    ERC1967Utils.upgradeToAndCall(newImplementation, data);

    address storedOwner;
    assembly {
        storedOwner := sload(OWNABLE_OWNER_SLOT)
    }
    if (storedOwner != msg.sender) revert
IncompatibleBridgeImplementation(newImplementation);
}
```

`sload` reads proxy storage. A lying `owner()` does not change it. If the upgrade writes a different address into the slot, the check rolls the transaction back.

Update

Client's response

BridgeProxy now reads the owner directly from the OpenZeppelin Ownable storage namespace using `sload`, extracting the address from the low 160 bits. Both upgrade authorization and the post-upgrade preservation check use this stored value instead of calling the implementation's `owner()` getter.

After the upgrade and reinitializer, the stored owner must still equal the authorized caller. Any change, including setting it to zero, reverts the entire transaction and rolls back the implementation and initialization writes. A misleading or broken getter cannot grant upgrade authority or prevent the stored owner from restoring a reviewed implementation. Future implementations must preserve this `Ownable` layout.

Oxorio's response

Verified in the linked remediation commit. `upgradeToAndCall` no longer calls `owner()`. It `sload`s `0x9016d09d72d40fdae2fd8ceac6b6234c7706214fd39c1cd1e609a0528c199300` and keeps the low 160 bits. A getter that returns the multisig once and another address later does not satisfy the check. A write to that slot that changes the owner rolls the upgrade back.

Fixed at: `485c3ce9`

I-07	A 32-byte answer to `implementation()` is treated as a proxy in `BridgeProxy.sol`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
BridgeProxy.sol	contract `BridgeProxy` > function `_requireCompatibleImplementation`	64

Description

In the function `_requireCompatibleImplementation` of contract `BridgeProxy`, `BridgeProxy` is new. The compatibility check rejects a candidate when `staticcall(implementation())` succeeds and the return data is 32 bytes. The comment targets another ERC-1967 proxy. The predicate is wider than that.

```
(bool exposesProxyApi, bytes memory proxyResult) =
    candidate.staticcall(abi.encodeCall(IBridgeProxy.implementation, ()));
if (exposesProxyApi && proxyResult.length == 32) revert
IncompatibleBridgeImplementation(candidate);
```

A contract that is not a proxy, but whose `implementation()` or fallback returns 32 bytes, is rejected even when `bridgeProxyCompatibilityUUID()` would have been correct. A proxy that does not answer `implementation()` with 32 bytes, and that returns the expected UUID from its own code, is not rejected. A standard ERC-1967 proxy is rejected, including by the direct-call UUID check on `Bridge` (`address(this) != _IMPLEMENTATION_SELF`). The upgrade still binds an address, not a code hash.

Proof of Concept
Code inspection of `_requireCompatibleImplementation`. `BridgeProxy.t.sol` shows standard proxies are rejected. The 32-byte predicate itself does not check that the candidate is an ERC-1967 proxy.

Recommendation

We recommend to identify proxies by the ERC-1967 implementation slot or by code hash. Do not treat any 32-byte return as proof of a proxy.

Update

Client's response

The heuristic that treated any successful 32-byte response to `implementation()` as evidence of a proxy was removed. Compatibility now requires an exact 32-byte UUID from `bridgeProxyCompatibilityUUID()`. Bridge's marker rejects delegatecall execution through its immutable implementation identity, and the proxy itself remains explicitly rejected as a candidate.

This preserves rejection of proxies forwarding the marker to **Bridge** while allowing compatible implementations with unrelated 32-byte fallback responses. Tests cover both cases and malformed, incorrect or reverting marker responses.
We retained compatibility-marker validation rather than introducing a code-hash allowlist. It is not a universal detector for arbitrary proxy bytecode or spoofed markers; candidate review must still verify the marker's direct-call-only behavior and implementation semantics.

Oxorio's response

The 32-byte `implementation()` check is removed at `3d50a9db`. That is only the first part of the recommendation. The candidate is still not rejected for being a proxy.

`bridgeProxyCompatibilityUUID` was already present. It compares the return value with `keccak256("utexo.bridge.proxy.compatibility.v1")` and reverts only when Bridge itself is reached through `delegatecall`. A different contract can return that constant from its own code and still be a proxy. The upgrade accepts it. The removed check would have rejected that candidate when `implementation()` returned 32 bytes. The marker does not replace that check.

We recommend to accept an upgrade only when `extcodehash(candidate)` is an approved implementation hash. A matching compatibility UUID is not proof that the candidate is not a proxy. Another contract's ERC-1967 storage slot cannot be read on-chain, so the enforceable check here is the code hash.

Client's response

The compatibility UUID does not prove that an arbitrary candidate is not a proxy. Reviewing candidate bytecode and delegation behavior remains part of the federation-authorized, timelocked upgrade process.

A fixed code-hash list cannot accommodate unknown future implementations. Supporting them would require an additional governance mechanism for approving, updating and revoking hashes, with new authorization rules. This is a broader change than a local validation check. We retain the existing model and explicitly accept this residual risk.

Oxorio's response

Acknowledged. The compatibility UUID does not prove that a candidate is not a proxy, and a code-hash allowlist is not added. UTEXO Protocol accepts that residual. Review of candidate bytecode stays with the federation-authorized, timelocked upgrade. The 32-byte `implementation()` heuristic remains removed at `3d50a9db`.

I-08	Federation rotation emits two events that always fire together in `MultisigProxy.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
MultisigProxy.sol	contract `MultisigProxy` > function `_executeByType`	1221

Description

In the function `_executeByType` of contract `MultisigProxy`, at `ea5de9b4`, federation rotation emits only `FederationSignersUpdated`. The target also bumps `federationSignerSetVersion` and emits a second event on every successful rotation. Nothing emits either event alone.

```
federationSignerSetVersion++;
emit FederationSignersUpdated(newSigners, newThreshold);
emit FederationSignerSetVersionUpdated(federationSignerSetVersion);
```

Indexers have to subscribe to both for one state change.

Proof of Concept
Code inspection. Both emits are in the same branch, with no other emit site for `FederationSignerSetVersionUpdated`.

Recommendation

We recommend to fold `FederationSignerSetVersionUpdated` into `FederationSignersUpdated`, or document that consumers must read both.

Update

Client's response

Federation rotation now emits a single event containing the complete updated configuration:

```
event FederationSignersUpdated(
    address[] newSigners,
    uint256 newThreshold,
    uint256 indexed newVersion
);
```

`newVersion` is the post-increment signer-set version, including for threshold-only updates. The separate `FederationSignerSetVersionUpdated` event was removed.

Oxorio's response

Verified in the linked remediation commit. The rotation branch emits `FederationSignersUpdated` once, with the new version in the same event. `FederationSignerSetVersionUpdated` is no longer emitted.

Fixed at: `9d98f211`

I-09	The zero-registry check in `_requireManagedOwnershipTarget` is redundant in `MultisigProxy.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
MultisigProxy.sol	contract `MultisigProxy` > function `_requireManagedOwnershipTarget`	1542

Description

In the function `_requireManagedOwnershipTarget` of contract `MultisigProxy`, the function is new. It rejects `target == address(0)` first. The later condition `registry != address(0)` cannot be what rejects a zero target.

```
function _requireManagedOwnershipTarget(address target) private view {
    if (target == address(0)) revert InvalidManagedOwnershipTarget(target);
    if (target == bridge || target == commissionManager || target == lzAdapter) return;
    address registry = IBridge(bridge).routeRegistry();
    if (target == registry && registry != address(0)) return;
    revert InvalidManagedOwnershipTarget(target);
}
```

When `registry` is zero and `target` is not, `target == registry` is already false. The extra comparison does not weaken the allowlist. It is a dead conjunct.

Proof of Concept
Code inspection of the function above. Removing `registry != address(0)` does not change which targets are accepted.

Recommendation

We recommend to drop `registry != address(0)`.

Update

Client's response

The redundant `registry != address(0)` condition was removed from `_requireManagedOwnershipTarget`. The function already rejects a zero target before checking the allowlist, so a non-zero target cannot match a zero registry. The registry check is now simply `target == registry`, with no change to accepted targets or validation at proposal creation and execution.

Oxorio's response

Verified in the linked remediation commit. `_requireManagedOwnershipTarget` still rejects `target == address(0)` first, then accepts `bridge`, `commissionManager`, `lzAdapter`, or the live registry. The dead `registry != address(0)` conjunct is gone. The allowlist is unchanged.

Fixed at: `dde66d5c`

4. APPENDIX

4.1 SECURITY ASSESSMENT METHODOLOGY

Oxorio's smart contract security audit methodology is designed to ensure the security, reliability, and compliance of smart contracts throughout their development lifecycle. Our process integrates the Smart Contract Security Verification Standard (SCSVS) with our advanced techniques to address complex security challenges. For a detailed look at our approach, please refer to the full version of our methodology. Here is a concise overview of our auditing process:

1. Project Architecture Review

All necessary information about the smart contract is gathered, including its intended functionality and dependencies. This stage sets the foundation by reviewing documentation, business logic, and initial code analysis.

2. Vulnerability Assessment

This phase involves a deep dive into the smart contract's code to identify security vulnerabilities. Rigorous testing and review processes are applied to ensure robustness against potential attacks.

This stage is focused on identifying specific vulnerabilities within the smart contract code. It involves scanning and testing the code for known security weaknesses and patterns that could potentially be exploited by malicious actors.

3. Security Model Evaluation

The smart contract's architecture is assessed to ensure it aligns with security best practices and does not introduce potential vulnerabilities. This includes reviewing how the contract integrates with external systems, its compliance with security best practices, and whether the overall design supports a secure operational environment.

This phase involves a analysis of the project's documentation, the consistency of business logic as documented versus implemented in the code, and any assumptions made during the design and development phases. It assesses if the contract's architectural design adequately addresses potential threats and integrates necessary security controls.

4. Cross-Verification by Multiple Auditors

Typically, the project is assessed by multiple auditors to ensure a diverse range of insights and thorough coverage. Findings from individual auditors are cross-checked to verify accuracy and completeness.

5. Report Consolidation

Findings from all auditors are consolidated into a single, comprehensive audit report. This report outlines potential vulnerabilities, areas for improvement, and an overall assessment of the smart contract's security posture.

6. Reaudit of Revised Submissions

Post-review modifications made by the client are reassessed to ensure that all previously identified issues have been adequately addressed. This stage helps validate the effectiveness of the fixes applied.

7. Final Audit Report Publication

The final version of the audit report is delivered to the client and published on Oxorio's official website. This report includes detailed findings, recommendations for improvement, and an executive summary of the smart contract's security status.

4.2 CODEBASE QUALITY ASSESSMENT REFERENCE

The tables below describe the codebase quality assessment categories and rating criteria used in this report.

Category	Description
Access Control	Evaluates the effectiveness of mechanisms controlling access to ensure only authorized entities can execute specific actions, critical for maintaining system integrity and preventing unauthorized use.
Arithmetic	Focuses on the correct implementation of arithmetic operations to prevent vulnerabilities like overflows and underflows, ensuring that mathematical operations are both logically and semantically accurate.
Complexity	Assesses code organization and function clarity to confirm that functions and modules are organized for ease of understanding and maintenance, thereby reducing unnecessary complexity and enhancing readability.
Data Validation	Assesses the robustness of input validation to prevent common vulnerabilities like overflow, invalid addresses, and other malicious input exploits.
Decentralization	Reviews the implementation of decentralized governance structures to mitigate insider threats and ensure effective risk management during contract upgrades.
Documentation	Reviews the comprehensiveness and clarity of code documentation to ensure that it provides adequate guidance for understanding, maintaining, and securely operating the codebase.
External Dependencies	Evaluates the extent to which the codebase depends on external protocols, oracles, or services. It identifies risks posed by these dependencies, such as compromised data integrity, cascading failures, or reliance on centralized entities. The assessment checks if these external integrations have appropriate fallback mechanisms or redundancy to mitigate risks and protect the protocol's functionality.
Error Handling	Reviews the methods used to handle exceptions and errors, ensuring that failures are managed gracefully and securely.
Logging and Monitoring	Evaluates the use of event auditing and logging to ensure effective tracking of critical system interactions and detect potential anomalies.
Low-Level Calls	Reviews the use of low-level constructs like inline assembly, raw `call` or `delegatecall`, ensuring they are justified, carefully implemented, and do not compromise contract security.

Category	Description
Testing and Verification	Reviews the implementation of unit tests and integration tests to verify that codebase has comprehensive test coverage and reliable mechanisms to catch potential issues.

4.2.1 Rating Criteria

Rating	Description
Excellent	The system is flawless and surpasses standard industry best practices.
Good	Only minor issues were detected; overall, the system adheres to established best practices.
Fair	Issues were identified that could potentially compromise system integrity.
Poor	Numerous issues were identified that compromise system integrity.
Absent	A critical component is absent, severely compromising system safety.
Not Applicable	This category does not apply to the current evaluation.

4.3 FINDINGS CLASSIFICATION REFERENCE

4.3.1 Severity Level Reference

The following severity levels were assigned to the issues described in the report:

Title	Description
CRITICAL	Issues that pose immediate and significant risks, potentially leading to asset theft, inaccessible funds, unauthorized transactions, or other substantial financial losses. These vulnerabilities represent serious flaws that could be exploited to compromise or control the entire contract. They require immediate attention and remediation to secure the system and prevent further exploitation.
MAJOR	Issues that could cause a significant failure in the contract's functionality, potentially necessitating manual intervention to modify or replace the contract. These vulnerabilities may result in data corruption, malfunctioning logic, or prolonged downtime, requiring substantial operational changes to restore normal performance. While these issues do not immediately lead to financial losses, they compromise the reliability and security of the contract, demanding prioritized attention and remediation.
WARNING	Issues that might disrupt the contract's intended logic, affecting its correct functioning or making it vulnerable to Denial of Service (DDoS) attacks. These problems may result in the unintended triggering of conditions, edge cases, or interactions that could degrade the user experience or impede specific operations. While they do not pose immediate critical risks, they could impact contract reliability and require attention to prevent future vulnerabilities or disruptions.
INFO	Issues that do not impact the security of the project but are reported to the client's team for improvement. They include recommendations related to code quality, gas optimization, and other minor adjustments that could enhance the project's overall performance and maintainability.

4.3.2 Status Level Reference

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

Title	Description
NEW	Waiting for the project team's feedback.

Title	Description
FIXED	Recommended fixes have been applied to the project code and the identified issue no longer affects the project's security.
ACKNOWLEDGED	The project team is aware of this finding and acknowledges the associated risks. This finding may affect the overall security of the project; however, based on the risk assessment, the team will decide whether to address it or leave it unchanged.
NO ISSUE	Finding does not affect the overall security of the project and does not violate the logic of its work.

4.4 ABOUT OXORIO

OXORIO is a blockchain security firm that specializes in smart contracts, zk-SNARK solutions, and security consulting. With a decade of blockchain development and five years in smart contract auditing, our expert team delivers premier security services for projects at any stage of maturity and development.

Since 2021, we've conducted key security audits for notable DeFi projects like Lido, 1Inch, Rarible, and deBridge, prioritizing excellence and long-term client relationships. Our co-founders, recognized by the Ethereum and Web3 Foundations, lead our continuous research to address new threats in the blockchain industry. Committed to the industry's trust and advancement, we contribute significantly to security standards and practices through our research and education work.

Our contacts:

- oxor.io
- ping@oxor.io
- Github
- Linkedin
- Twitter

THANK YOU FOR CHOOSING

