GA GUARDIAN

Citrea

Stablecoin Bridge

Security Assessment

October 6th, 2025

Summary

Audit Firm Guardian

Prepared By Robert Reigada, Jakub Zmyslowski, ArnieSec

Client Firm Chainway Labs

Final Report Date October 6, 2025

Audit Summary

Chainway Labs engaged Guardian to review the security of their Citrea Stablecoin Bridge. From the 29th of August to the 3rd of September, a team of 3 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the lack of critical issues detected and minimal code changes following the main review, Guardian assigns a Confidence Ranking of 4 to the protocol. Guardian advises the protocol to consider periodic review with future changes. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 PoC test suite: https://github.com/GuardianOrg/stablecoin-bridge-team1-1756330121613

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. **Recommendation:** Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only Low or Medium-severity issues were discovered. Design patterns are sound, and test coverage is reasonable. Small changes, such as modifying rounding logic, may introduce new vulnerabilities and should be carefully reviewed. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0 High/Critical findings. Varied Low/Medium severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1 High finding and ≥ 3 Medium. Varied Low severity findings.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities (~2/week). Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Overview

Project Summary

Project Name	Citrea Stablecoin Bridge
Language	Solidity
Codebase	https://github.com/chainwayxyz/stablecoin-bridge
Commit(s)	Initial commit: 61690aa44cfcedd0ef91c47f896473ca783aece9 Final commit: 7df209891507f295c24f3f6955230000a02b52dd

Audit Summary

Delivery Date	October 4, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
High	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Low	8	0	0	4	0	4
Info	11	0	0	5	0	6

Audit Scope & Methodology

contract,source,total,comment
stablecoin-bridge/src/DestinationOUSDC.sol,35,69,25
stablecoin-bridge/src/DestinationOUSDT.sol,35,71,26
stablecoin-bridge/src/SourceOFTAdapter.sol,11,15,1
stablecoin-bridge/src/USDCRolesHolder.sol,25,33,1
stablecoin-bridge/script/ConfigSetup.s.sol,179,217,2
stablecoin-bridge/src/for_circle_takeover/DestinationOUSDCForTakeover.sol,43,79,25
stablecoin-bridge/src/for_circle_takeover/SourceOFTAdapterForTakeover.sol,42,55,1
stablecoin-bridge/script/usdt/deploy/01_USDTDeploy.s.sol,33,40,3
stablecoin-bridge/script/usdt/deploy/02_USDTBridgeDeploy.s.sol,46,54,3
stablecoin-bridge/script/usdt/deploy/03_USDTSrcBridgeSetLzConfig.s.sol,37,51,2
stablecoin-bridge/script/usdt/deploy/04_USDTDestBridgeSetLzConfig.s.sol,37,50,2
stablecoin-bridge/script/usdt/deploy/05_USDTSrcBridgeSetPeer.s.sol,21,28,2
stablecoin-bridge/script/usdt/deploy/06_USDTDestBridgeSetPeer.s.sol,21,28,2
stablecoin-bridge/script/usdt/deploy/07_USDTSetBridgeAsMinter.s.sol,18,25,3
stablecoin-bridge/script/usdt/deploy/08_USDTAndBridgeAssignRoles.s.sol,33,40,2
stablecoin-bridge/script/usdc/deploy/02_USDCBridgeDeploy.s.sol,45,53,3
stablecoin-bridge/script/usdc/deploy/03_USDCSrcBridgeSetLzConfig.s.sol,37,50,2
stablecoin-bridge/script/usdc/deploy/04_USDCDestBridgeSetLzConfig.s.sol,37,50,2
stablecoin-bridge/script/usdc/deploy/05_USDCSrcBridgeSetPeer.s.sol,21,28,2
stablecoin-bridge/script/usdc/deploy/06_USDCDestBridgeSetPeer.s.sol,21,28,2
stablecoin-bridge/script/usdc/deploy/07_USDCSetBridgeAsMinter.s.sol,19,25,2
stablecoin-bridge/script/usdc/deploy/08_USDCAndBridgeAssignRoles.s.sol,34,41,2
stablecoin-bridge/script/usdc/for_circle_takeover/01_USDCSrcBridgePrepareTakeover.s.sol,25,31,2
stablecoin-bridge/script/usdc/for_circle_takeover/02_USDCDestBridgePrepareTakeover.s.sol,25,31,2
stablecoin-bridge/script/usdc/for_circle_takeover/03_USDCSrcBridgeSetBlockedMsgLib.s.sol,20,26,2
stablecoin-bridge/script/usdc/for_circle_takeover/04_USDCDestBridgeSetBlockedMsgLib.s.sol,20,26,2
stablecoin-bridge/script/usdc/for_circle_takeover/05_USDCSrcBridgePause.s.sol,20,26,2
stablecoin-bridge/script/usdc/for_circle_takeover/06_USDCDestBridgePause.s.sol,20,26,2
stablecoin-bridge/script/usdc/for_circle_takeover/07_USDCRemoveBridgeAsMinter.s.sol,18,27,5
stablecoin-bridge/script/usdc/for_circle_takeover/08_USDCSrcBridgeSetCircle.s.sol,20,26,2
stablecoin-bridge/script/usdc/for_circle_takeover/09_USDCProxyAdminTransfer.s.sol,18,25,3
stablecoin-bridge/script/usdc/for_circle_takeover/10_USDCTransferOwner.s.sol,22,29,3
stablecoin-bridge/script/usdc/for_circle_takeover/11_USDCRolesHolderSetCircle.s.sol,20,26,2
stablecoin-bridge/script/usdc/for_circle_takeover/unpause/USDCDestBridgeUnpause.s.sol,20,26,2
stablecoin-bridge/script/usdc/for_circle_takeover/unpause/USDCSrcBridgeUnpause.s.sol,20,26,2
source count: {total: 1481, source: 1098, comment: 146, single: 80, block: 66, mixed: 1, empty: 238, todo: 0, blockEmpty: 0, commentToSourceRatio: 0.13296903460837886}

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
L-01	Missing ProxyAdmin Ownership Transfer	Configuration	Low	Acknowledged
L-02	Enabling Peers Before Granting Mint Authorization	Configuration	Low	Resolved
L-03	Pausing _credit During ULN Confirmation Window	Configuration	Low	Acknowledged
L-04	USDC Recipient Blacklist	DoS	Low	Acknowledged
L-05	Zero-Address Recipient	DoS	Low	Resolved
L-06	Reconciled Bridged Supply	Unexpected Behavior	Low	Resolved
L-07	Fee-On-Transfer USDT	Validation	Low	Acknowledged
L-08	Missing Contract Bytecode Verification	Warning	Low	Resolved
I-01	USDC.e Proxy Initialization Can Be Front-Run	Frontrunning	Info	Acknowledged
I-02	Missing Scripts	Configuration	Info	Acknowledged
I-03	Consider Declaring USDC As Immutable	Gas Optimization	Info	Resolved
I-04	Unlicensed Contracts	Best Practices	Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-05	Destination USDC Blacklist Initialized Empty	Warning	Info	Acknowledged
I-06	Manual In-Flight Message Check Is Error-Prone	Warning	Info	Resolved
I-07	Missing __Pausable_init Invocation	Best Practices	Info	Resolved
I-08	Missing Storage Gaps In Upgradeable Contracts	Best Practices	Info	Acknowledged
I-09	Single Step Ownable Module	Best Practices	Info	Resolved
I-10	Discrepancy In The TransparentUpgradeableProxy	Warning	Info	Acknowledged
I-11	setCircle Allows Circle Address To Be Re Set	Warning	Info	Acknowledged

L-01 | Missing ProxyAdmin Ownership Transfer

Category	Severity	Location	Status
Configuration	Low	Global	Acknowledged

Description

The bridge deployment scripts import openzeppelin-contracts/contracts/proxy/transparent/TransparentUpgradeableProxy.sol, which resolves via your remappings to OpenZeppelin v5.2.0. In OZ v5, TransparentUpgradeableProxy always deploys a fresh ProxyAdmin inside its constructor and stores that ProxyAdmin's address as the proxy admin (immutable).

The constructor argument you pass is not the proxy admin itself, but the owner of the newly created ProxyAdmin. In your repo, both the USDC and USDT bridge proxies are created with the v5 TUP:
• script/usdc/deploy/02_USDCBridgeDeploy.s.sol
• script/usdt/deploy/02_USDTBridgeDeploy.s.sol

Your takeover scripts for USDC correctly assume a ProxyAdmin is present and call it:
• script/usdc/for_circle_takeover/01_USDCSrcBridgePrepareTakeover.s.sol
• script/usdc/for_circle_takeover/02_USDCDestBridgePrepareTakeover.s.sol

However, nowhere in the codebase is the ownership of these bridge ProxyAdmin contracts transferred to the intended final controller (e.g., a multisig or Circle). The role handover script for USDC (script/usdc/deploy/08_USDCAndBridgeAssignRoles.s.sol) sets bridge Ownable owners/delegates and master-minter ownership, but does not transfer ownership of the bridge ProxyAdmin. The only ownership transfer you have is for the USDC token proxy (Circle's AdminUpgradeabilityProxy) in script/usdc/for_circle_takeover/09_USDCProxyAdminTransfer.s.sol.

This is a problem because the ProxyAdmin.owner() for the bridge proxies remains with the original EOA, an attacker who compromises that key (or a malicious insider) can front-run Circle's burnLockedUSDC call. For example:
1. Right before Circle calls burnLockedUSDC() on the source adapter, the attacker, in this case the ProxyAdmin's owner upgrades the SourceAdapter(SourceOFTAdapterForTakeover), drains all the USDC it holds, and then upgrades it again to its previous implementation.
2. Circle's burnLockedUSDC is executed. No USDC is burnt, but they think they have burnt the full amount.
3. At this point, all the USDC tokens were stolen by the ProxyAdmin's owner, but Circle thinks they were burnt.

Recommendation

As part of the handover process to Circle, consider transferring the ownerships of the source and destination Adapter's ProxyAdmin to an address controlled by Circle.

Resolution

Chainway Labs Team: Circle's Bridged USDC Standard does not mention that the upgradeability admin of the bridges (SourceOFTAdapter, DestinationOUSDC, DestinationOUSDT) should be transferred to Circle. As such, we don't think this is necessary, additionally, it is unclear that Circle would even accept the responsibility of claiming the upgradeability right of the bridges. On the other hand, if an emergency arises during the Circle takeover process we should have the ability to respond to it by upgrading the bridges. After the takeover process is completed the bridges are no longer relevant, thus we don't think this is an issue.

L-02 | Enabling Peers Before Granting Mint Authorization

Category	Severity	Location	Status
Configuration	● Low	06_USDCDestBridgeSetPeer.s.sol; 07_USDCSetBridgeAsMinter.s.sol; 06_USDTDestBridgeSetPeer.s.sol; 07_USDTSetBridgeAsMinter.s.sol	Resolved

Description

In the current deployment flow for both USDC and USDT, the destination peer is set in `06_*DestBridgeSetPeer.s.sol` before the destination bridge is authorized to mint (USDC) or recognized as the `oftContract` (USDT), which happens in `07_*SetBridgeAsMinter.s.sol`.

As soon as both peers are configured, users can send messages from the source. If the final peer is set while the destination bridge still lacks authorization, any arriving message will revert during `_credit()`. For USDC (`src/DestinationOUSDC.sol`), `_credit` calls `FiatTokenV2_2.mint`, which reverts until the destination bridge proxy is a minter:

```
token_.mint(_to, _amountLD); // reverts until bridge is an authorized minter
```

For USDT (`src/DestinationOUSDT.sol`), `_credit` calls `crosschainMint`, which is guarded by `onlyAuthorizedSender` in `OFTExtension`. Until `oftContract` is set to the destination bridge, calls revert:

```
token_.crosschainMint(_to, _amountLD); // reverts until oftContract = dest bridge
```

The impact is that users send succeed on source (tokens are locked/charged) but deliveries fail on destination until ops finish step 07 and retry.

Recommendation

Reorder or merge steps so the destination is fully authorized before enabling traffic. The safest pattern is: run `07_*SetBridgeAsMinter.s.sol` first, then `06_*DestBridgeSetPeer.s.sol` and set the source peer last (`05_*SrcBridgeSetPeer.s.sol`) to act as the “master switch”. If you keep scripts separate, add a precondition guard to `06_*DestBridgeSetPeer.s.sol`, e.g., for USDC:

```
require(FiatTokenV2_2(destUSDC).isMinter(destUSDCBridgeProxy), "dest bridge not minter");
```

and for USDT:

```
require(TetherTokenOFTExtension(destUSDT).oftContract() = destUSDTBridgeProxy, "oftContract not set");
```

Alternatively, you can also merge “authorize destination” and “set destination peer” into a single script.

Resolution

Chainway Labs Team: The issue was resolved in commit e5d9020.

L-03 | Pausing _credit During ULN Confirmation Window

Category	Severity	Location	Status
Configuration	Low	DestinationOUSDCForTakeover.sol; SourceOFTAdapterForTakeover.sol	Acknowledged

Description

The LayerZero ULN path uses a confirmations parameter (20 blocks set in the testnet config) so DVNs only pass a message to the destination after sufficient finality to mitigate reorgs. In the current takeover implementation, both DestinationOUSDCForTakeover and SourceOFTAdapterForTakeover apply whenNotPaused to both _debit and _credit. This creates a following risk:

1. A user initiates a cross-chain transfer.
2. DVNs wait out the confirmation window on the source chain.
3. During that window, the owner pauses the bridge.
4. When the packet is finally delivered, the destination's _credit (called by lzReceive) hits whenNotPaused and reverts. LayerZero marks the message failed until an explicit retry after unpause.
5. Meanwhile, the user's funds are already locked (or burned in burn-and-mint design). In the takeover flow, this amount is later burned as part of supply finalization, so the user's transfer is effectively stuck until operational recovery. It may be permanently lost to the user if the burn is completed or the MasterMinter role is revoked for the bridge, without minting the corresponding credit.

This behaviour contradicts the migration guidance to "pause bridging activity and reconcile in-flight bridging activity to finalize the total supply of bridged USDC on the destination chain." Pausing may prevent reconciliation by causing in-flight packets to fail right before the settlement.

Recommendation

Remove whenNotPaused from _credit in both takeover contracts. Guard only _debit so already verified packets continue to settle while new sends are blocked, what "pause and reconcile in-flight" requires.

Resolution

Chainway Labs Team: This issue is effectively mitigated as we set the send libraries of the respective bridge ends to BlockedMsgLib and wait for a while until no inflight messages remain. As the result of the fix to I-06, bridges cannot be paused while there are still inflight messages.

L-04 | USDC Recipient Blacklist

Category	Severity	Location	Status
DoS	Low	DestinationOUSDC.sol; SourceOFTAdapter.sol	Acknowledged

Description

USDC token enforces blacklist checks on both the caller and the recipient in mint and transfer functions:

```
function mint(address _to, uint256 _amount)
external
whenNotPaused
onlyMinters
notBlacklisted(msg.sender)
notBlacklisted(_to)
returns (bool)
```

```
function transfer(address to, uint256 value)
external
override
whenNotPaused
notBlacklisted(msg.sender)
notBlacklisted(to)
returns (bool)
```

In the bridge, the source `_debit` path first consumes the user's value (locks via `transferFrom` in an Adapter or burns in a Core). The packet reaches the destination `_credit`, which mints the intended amount of tokens or transfer tokens locked beforehand, to deliver funds. If `_to` is blacklisted on the destination chain at execution time, mint reverts, causing the `LayerZero` receive to fail. The message remains failed pending manual retry after unblacklist, while the user's funds are already locked/burned on the source.

This failure mode occurs on any path where `_credit` mints USDC (e.g. if the source chain uses a Core minter or after takeover when native USDC is the destination token). Thus, any `_credit` that mints/transfer USDC is susceptible to recipient-blacklist reverts. This is particularly risky during the takeover - if supply is finalising (locked source balance is burned) before resolving failed credits, affected users can end up without funds on the destination, even though total supply is conserved.

This also contradicts the intended "lossless delivery" property for cross-chain messages: the application reverts at the destination and forces manual intervention, despite the source having already consumed value.

Recommendation

Make `_credit` non-reverting. Wrap mint and transfer logic implemented by `_credit` functions in try/catch blocks. On failure, queue a pending credit to be claimable after unblacklist. For bridge wrapped by the adapter, override the `_credit` function to introduce the mentioned logic.

Resolution

Chainway Labs Team: If a user is unblacklisted, they can re-try the message to receive their tokens as it wouldn't revert. We think implementing a queue to achieve the same result would unnecessarily complicate the contract.

L-05 | Zero-Address Recipient

Category	Severity	Location	Status
DoS	Low	DestinationOUSDC.sol; DestinationOUSDCForTakeover.sol	Resolved

Description

Both `DestinationOUSDC` and `DestinationOUSDCForTakeover` call `token.mint(_to, _amountLD)` in `_credit` without remapping a zero recipient. In USDC implementations, minting tokens to address(0) reverts. The bridge also performs no preflight check to reject to = 0x0. As a result:

1. The source `_debit` first consumes value (locks via `transferFrom` in Adapter or burns in Core).
2. The LZ packet arrives. Destination `_credit` tries `mint(0x0, amount)` and reverts.
3. The `LayerZero` message is marked failed until manual retry with a corrected recipient, while the source value remains locked/burned.

This is inconsistent with `LayerZero`'s OFT defaults and symmetrical implementation for USDT, which remap 0x0 to 0xdead inside `_credit` to preserve "lossless delivery" and avoid receive-time reverts.

```
function _credit(
address _to,
uint256 _amountLD,
uint32 /*_srcEid*/
) internal virtual override returns (uint256 amountReceivedLD) {
if (_to = address(0x0)) _to = address(0xdead); // _mint(...) does not support address(0x0)
// @dev Default OFT mints on dst.
_mint(_to, _amountLD);
// @dev In the case of NON-default OFT, the _amountLD MIGHT not be = amountReceivedLD.
return _amountLD;
}
```

Recommendation

In `_credit`, remap zero-address recipient to a burn sink (0xdead) before minting. Consider disallowing address-zero transfers preflight.

Resolution

Chainway Labs Team: The issue was resolved in commit cf3edbf.

L-06 | Reconciled Bridged Supply

Category	Severity	Location	Status
Unexpected Behavior	Low	SourceOFTAdapterForTakeover.sol	Resolved

Description

Circle's standard states: "Circle and the third-party team will jointly coordinate to burn an amount of native USDC locked in the bridge contract on the origin chain that equals the supply of bridged USDC."

In the current implementation of `SourceOFTAdapterForTakeover`, the `burnLockedUSDC` burns the entire USDC balance held by the adapter, not the reconciled amount that corresponds to the bridged token supply at the takeover.

```
function burnLockedUSDC() external onlyCircle {
uint256 balance = innerToken.balanceOf(address(this));
FiatTokenV2_2(address(innerToken)).burn(balance);
emit BurnedLockedUSDC(msg.sender, balance);
}
```

Because it uses the raw balance as the burn amount, any extra USDC held by the adapter (e.g. due to incorrect direct transfers) will be burned along with the actual locked collateral that backs the bridged supply.

This violates the standard's requirement to burn exactly the amount equal to the outstanding bridged supply, as more native USDC may be burned than the supply represented on the destination.

Recommendation

Implement explicit internal accounting and burn only the reconciled amount. Increase a counter on each `_debit` call by the actual received delta and decrease it on every reverse unlock/credit by the amount transferred out.

Resolution

Chainway Labs Team: The issue was resolved in commit 4c6c76b. We took a different approach then the one suggested as having an internal counter wouldn't also cover additional edge cases such as a balance discrepancy due to transaction reverting on destination chain. Instead we implemented a privileged setter for the amount to burn source side USDC, and in scripts we read it directly from the destination RPC. This implementation also supports integration of a cross-chain reader as a new role is defined for this purpose, so a fork may choose to implement that without modifying the code.

L-07 | Fee-On-Transfer USDT

Category	Severity	Location	Status
Validation	Low	SourceOFTAdapterForTakeover.sol; SourceOFTAdapter.sol	Acknowledged

Description

On the source chain, the OFT Adapter's `_debit` locks USDT by pulling it from the user. The default implementation assumes lossless ERC-20 and sets `amountReceivedLD` equal to `amountSentLD` from `_debitView`, without verifying what was actually received.

If the underlying USDT token has fee-on-transfer enabled, the adapter on the source chain receives only the requested amount minus the token's transfer fee, but the message still carries an amount equal to the originally requested value.

The destination `_credit` then mints the full requested amount, meaning the tokens minted on the destination exceed the collateral actually locked on the source by the size of the fee.

Repeated transfers accumulate a solvency gap on the adapter. Over time, this gap can leave the adapter under-collateralized, causing return path redemptions to fail.

Recommendation

In the adapter's `_debit` function, calculate the balance delta and use it as the amount to encode into the message. This ensures the destination mints exactly what the adapter actually received.

Resolution

Chainway Labs Team: We acknowledge this risk with a warning in README, added in e8c2001.

L-08 | Missing Contract Bytecode Verification

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

The deployment flow broadcasts implementations and proxies and then persists the resulting addresses to config, but it never verifies the deployed bytecode on explorers nor asserts that the on-chain code matches the expected build.

For example, `script/usdc/deploy/02_USDCBridgeDeploy.s.sol` deploys `SourceOFTAdapter` and a `TransparentUpgradeableProxy` and then writes the proxy address into the TOML, but performs no explorer/source verification (`--verify`, `forge verify-contract`).

The USDT script similarly deploys an implementation via raw bytecode and `assembly { create(...) }` without any follow-up attestation.

Recommendation

Implement explorer verification in the runbook for each network (implementation + proxy + admin path): For example, run with `--verify` (and the per-chain API key) immediately after/bundled with broadcast.

For the scripts that use multichain deployments like `script/usdc/deploy/02_USDCBridgeDeploy.s.sol` consider following Foundry's multichain deployment guide:

https://getfoundry.sh/forge/deploying/#multi-chain-deployments

Resolution

Chainway Labs Team: The issue was resolved in commit 61380ca.

I-01 | USDC.e Proxy Initialization Can Be Front-Run

Category	Severity	Location	Status
Frontrunning	Info	01_USDCDeploy.sh; deploy-fiat-token.s.sol	Acknowledged

Description

The USDC deployment script `deploy-fiat-token.s.sol`, invoked by `01_USDCDeploy.sh` from Circle's stablecoin-evm repository at commit `c8c31b2`, performs the proxy deployment and subsequent multi-version initializations as separate transactions under `vm.startBroadcast(deployerPrivateKey)`, enabling an attacker to frontrun the unpermissioned `initialize` calls on the newly deployed `FiatTokenProxy` after its transaction executes but before the deployer's init transactions are run, allowing the attacker to hijack ownership by calling `initialize` first with malicious parameters.

Specifically, the script deploys or reuses the implementation via `FiatTokenV2_2 fiatTokenV2_2 = getOrDeployImpl(_impl)` which safely initializes the implementation using a temporary upgrader to disable its own reinitialization, then deploys the proxy with `FiatTokenProxy proxy = new FiatTokenProxy(address(fiatTokenV2_2))` in one transaction, followed by separate transactions for `proxy.changeAdmin(proxyAdmin)` and the casts to `FiatTokenV2_2 proxyAsV2_2 = FiatTokenV2_2(address(proxy))` to invoke `proxyAsV2_2.initialize()` which delegates to the implementation's V1 `initializer` setting the owner and other roles and then individual calls to `proxyAsV2_2.initializeV2()`, `proxyAsV2_2.initializeV2_1()` and `proxyAsV2_2.initializeV2_2()`.

Since Foundry broadcasts each of these operations as distinct transactions and the initializers are not permissioned as per Circle's design to allow one-time setup, an attacker monitoring the public mempool can detect the proxy deployment transaction, predict its address using the deployer's nonce and the `CREATE` opcode determinism and submit a competing transaction with higher gas fees to execute first, casting the proxy to `FiatTokenV2_2` and calling `initialize` with the attacker's address as owner to set `initialized = true` in the proxy's storage and claim all roles before the deployer's transactions process.

If successful, the deployer's init calls should revert with "already initialized" errors and would require a new deployment/re-run of the script.

Recommendation

Be aware of this risk. Consider also refactoring the `deploy-fiat-token.s.sol` script to deploy a temporary upgrader contract that atomically deploys the proxy, changes the admin, and calls all initializers in a single transaction, eliminating the multi-tx window.

Resolution

Chainway Labs Team: We acknowledge this risk with a note in README, added in 67838e2.

I-02 | Missing Scripts

Category	Severity	Location	Status
Configuration	Info	Global	Acknowledged

Description

The repository implements the takeover mechanics but does not provide scripts to execute the critical on-chain calls at upgrade time, as required by the project's own auditors_guide.md (which states that every on-chain action in the Circle standard must have a corresponding script).

1. No script calls burnLockedUSDC() on the source bridge: The takeover implementation src/for_circle_takeover/SourceOFTAdapterForTakeover.sol exposes burnLockedUSDC. This is the exact entrypoint Circle requires on the source chain to burn the locked USDC that backs the finalized bridged supply. However, under script/usdc/for_circle_takeover/ there is no script that invokes this function from the configured circle address. Existing takeover scripts cover preparing new impls, pausing/unpausing, removing minter, setting circle and transferring ProxyAdmin, but they never perform the actual burn.

2. No script triggers USDCRolesHolder.transferUSDCRoles(address) (Circle-side finalization).src/USDCRolesHolder.sol correctly exposes transferUSDCRoles. You deploy the roles holder and set Circle with:
• script/usdc/for_circle_takeover/10_USDCTransferOwner.s.sol
• script/usdc/for_circle_takeover/11_USDCRolesHolderSetCircle.s.sol

But there is no script that calls transferUSDCRoles(<Circle owner>) to actually hand over the Implementation Owner to Circle.

Recommendation

Add two minimal, purpose-built takeover scripts and wire them into the documented runbook:
1. 12_USDCSrcBridgeBurnLockedUSDC.s.sol (run on source chain, from the configured circle address):
• Fork srcRPC.
• Call SourceOFTAdapter(srcUSDCBridgeProxy).burnLockedUSDC().
• Recommended pre-checks: ensure the bridge is paused, log innerToken.balanceOf(proxy).

2. 13_USDCRolesHolderTransferUSDCRoles.s.sol (run on destination chain, from the configured circle address):
• Fork destRPC.
• Obtain roles holder from FiatTokenV2_2(destUSDC).owner().
• Call USDCRolesHolder(rolesHolder).transferUSDCRoles(<CIRCLE_IMPLEMENTATION_OWNER>).
• Ensure your separate ProxyAdmin transfer (09_USDCProxyAdminTransfer.s.sol) has already been executed.

Also update the guide to include these steps in the exact sequence (pause → remove minter → set Circle → burnLockedUSDC → transfer ProxyAdmin → transferUSDCRoles → unpause if applicable).

Resolution

Chainway Labs Team: We think Circle has their own scripts as the function signatures are directly taken from Circle's Bridged USDC Standard, we don't think there is a need to implement scripts that call the functions that should be called by Circle.

I-03 | Consider Declaring USDC As Immutable

Category	Severity	Location	Status
Gas Optimization	Info	USDCRolesHolder.sol	Resolved

Description

In the `USDCRolesHolder` contract, the `usdc` state variable is assigned in the constructor to reference the `FiatTokenV2_2` interface at the provided `usdcProxy` address and is never modified thereafter, as there are no setter functions or internal assignments that alter its value post-deployment.

This variable is used solely for read operations, such as in the `transferUSDCRoles` function where it calls `usdc.transferOwnership(_owner)`, making it a prime candidate for immutability.

Declaring it as a regular public variable incurs unnecessary storage reads during runtime, which consume gas, whereas an immutable variable is embedded directly into the bytecode at deployment time, eliminating storage access costs.

Recommendation

Modify the `usdc` declaration to `FiatTokenV2_2 public immutable usdc;` and assign it directly in the constructor without changes to the existing logic, ensuring the value is set at deployment and remains constant thereafter.

Resolution

Chainway Labs Team: The issue was resolved in commit 5ea30e5.

I-04 | Unlicensed Contracts

Category	Severity	Location	Status
Best Practices	Info	Global	Resolved

Description

Several in-scope contracts are marked with

```
// SPDX-License-Identifier: UNLICENSED
```

which grants no rights to use, modify, or distribute the code. This ambiguity can deter integrators and developers from using or contributing to the project.

Additionally, mixed or missing license headers across contract (e.g. some files unlicensed, others permissively licensed) create inconsistency, making it unclear which terms apply to which components and how the code may be combined or redistributed.

Recommendation

Choose and apply a clear open-source license. Update every Solidity file's SPDX header to the chosen identifier (keeping third-party code under its original license). Common options:

• MIT – permissive, minimal restrictions.

• Apache-2.0 – permissive, explicit patent grant.

• GPL-3.0 – copyleft, derivatives must remain open-source.

Resolution

Chainway Labs Team: Resolved.

I-05 | Destination USDC Blacklist Initialized Empty

Category	Severity	Location	Status
Warning	Info	FiatToken.sol	Acknowledged

Description

The USDC deployment script (`script/usdc/deploy/01_USDCDeploy.sh`) force-sets an empty blacklist before running Circle's deployer by writing `[]` into `blacklist.remote.json`: `echo "[]" > blacklist.remote.json`

Circle's `FiatToken` contracts actively maintain a blacklist on production chains (e.g., Arbitrum shows a designated Blacklister at 0x13F2A44FaD26c2cc25d3e3b869364142ce5995Bb and USDC runs at 0xaf88d065e77c8cC2239327C5EDb3A432268e5831).

Initializing the destination chain with an empty blacklist means addresses currently blocked on other chains will not be blocked on the new destination at launch.

During this divergence window, a recipient blacklisted on Arbitrum (or Ethereum) can receive mints and transfer on the destination chain until the list is manually synced, which could be a policy/compliance gap.

Recommendation

Consider checking with the Circle's team how to proceed with the blacklist. Blacklisting all the addresses in the destination chain would imply very high gas costs.

Resolution

Chainway Labs Team: We found no way of initializing a blacklist for USDC. Even if there was such a way, we don't think this it would be significantly cheaper as the same number of storage slots needs to be set whether it is done during initialization or after deployment. Regardless, we couldn't find a way of doing that thus we think this issue is invalid.

I-06 | Manual In-Flight Message Check Is Error-Prone

Category	Severity	Location	Status
Warning	Info	Global	Resolved

Description

The takeover runbook instructs operators to “wait and check LayerZero Scan” to ensure no in-flight messages exist before pausing the bridges. This off-chain, manual verification is easy to skip or perform inconsistently.

If operators pause while messages are still in flight, subsequent deliveries will hit whenNotPaused guards in the takeover implementations (e.g., DestinationOUSDCForTakeover._credit) and revert, creating avoidable failed executions that require retries/unpausing.

The LayerZero Scan API already exposes a per-OApp feed, GET /messages/oapp/{eid}/{address} with optional limit, start, end and nextToken, that returns statuses such as INFLIGHT, CONFIRMING, PAYLOAD_STORED, BLOCKED, FAILED, and DELIVERED. Relying solely on human diligence instead of this endpoint risks pausing with unresolved messages.

Recommendation

Before executing pause scripts, add an automated pre-check that queries GET /messages/oapp/{eid}/{address} for both destination and source bridge proxies and aborts if any non-DELIVERED messages are returned.

Resolution

Chainway Labs Team: The issue was resolved in commit 4822498.

I-07 | Missing __Pausable_init Invocation

Category	Severity	Location	Status
Best Practices	Info	SourceOFTAdapterForTakeover.sol	Resolved

Description

SourceOFTAdapterForTakeover inherits PausableUpgradeable but it's initialize function does not call __Pausable_init.

```
function initialize(address _delegate) public initializer {
__OFTCore_init(_delegate);
__Ownable_init(_delegate);
}
```

In OpenZeppelin's upgradeable contracts pattern, every inherited module's initializer should be invoked from the proxy initializer (e.g. __Pausable_init when inheriting PausableUpgradeable). Omitting it deviates from the recommended initialization procedure for upgradeable contracts.

Recommendation

Call __Pausable_init inside the initialize function.

Resolution

Chainway Labs Team: The issue was resolved in commit ff5f596.

I-08 | Missing Storage Gaps In Upgradeable Contracts

Category	Severity	Location	Status
Best Practices	Info	Global	Acknowledged

Description

Upgradeable contracts in scope do not declare a storage gap.

OpenZeppelin recommends reserving unused storage slots at the end of upgradeable contracts to reduce the risk of storage layout conflicts in future versions (e.g. when adding new state variables, changing inheritance or upgrading dependencies storage).

Without a gap, adding variables later may shift storage and corrupt state in existing proxies.

Recommendation

Add a storage gap to each upgradeable contract even if it currently uses only immutable variables. When adding state later, consume slots from the gap and keep prior layout preserved.

Resolution

Chainway Labs Team: None of the contracts we wrote are base contracts, so we don't think there's a need for storage gaps as it is equivalent to adding new state variables after the existing ones without messing up the order of them.

I-09 | Single Step Ownable Module

Category	Severity	Location	Status
Best Practices	Info	Global	Resolved

Description

The bridge and adapter contracts rely on an owner account to perform sensitive lifecycle actions (configure peers, pause/unpause, set Circle takeover roles, assign/revoke mint/burn permissions). This design allows operators to react quickly and coordinate the Bridged USDC takeover.

However, the contracts use single-step `Ownable` module, where ownership transfer finalize immediately and privileged functions execute without delay.

This creates a potential risk, where a compromised key, mistaken transfer, or rushed action could instantly allow to:

• set peer to a malicious `OApp` (granting mint authority on the destination bridge),
• change takeover addresses (`setCircle`) or call `burnLockedUSDC` at the wrong time,
• push unreviewed `LayerZero` security/library/DVN config changes,
• pause/unpause contracts in an undesirable time.

Because ownership changes take effect immediately, there is no review window for monitoring to catch bad actions and no explicit acceptance step by the new owner.

Recommendation

• Consider implementing `Ownable2Step` module (if feasible and compatible with the Bridged USDC Standard), so that ownership transfers require explicit acceptance by the new owner.
• Process non-emergency privileged calls (`setPeer`, DVN/library config, `setCircle`, minter role changes) through the timelock.
• Set the owner to a multisig behind a timelock.
• Override the `renounceOwnership` function to prevent accidental loss of ownership.
• Implement a timelock on the `setPeer` function and monitor pending changes.

Resolution

Chainway Labs Team: Implemented the Ownable2Step suggestion in the only place we can, in commit 309d8cf. All of the other suggestions regarding timelocks are configuration issues and any of the privileged addresses can be timelocked setters.

I-10 | Discrepancy In The TransparentUpgradeableProxy

Category	Severity	Location	Status
Warning	Info	09_USDCProxyAdminTransfer.s.sol	Acknowledged

Description

The `TransparentUpgradeableProxy` contracts deployed in all the scripts except `script/usdc/deploy/01_USDCDeploy.sh` are a v5+ `Openzeppelin` version: `import "openzeppelin-contracts/contracts/proxy/transparent/TransparentUpgradeableProxy.sol";`.

`TransparentUpgradeableProxy` v5+ use an immutable admin and auto-deploys a `ProxyAdmin` in its constructor.

However, USDC uses Circle's `FiatTokenProxy` / `Openzeppelin`'s v4 `AdminUpgradeabilityProxy`:

https://github.com/circlefin/stablecoin-evm/blob/master/contracts/upgradeability/AdminUpgradeabilityProxy.sol

that uses a mutable admin slot with constructor allowing direct admin swaps via `changeAdmin(newAdmin)` only callable by the current admin.

Recommendation

Merely informative. Be aware that the USDC token address in the source chain is the only contract that is not behind a TUP.

Resolution

Chainway Labs Team: This is a requirement arising from Circle's Bridged USDC Standard as the standard instructs to not modify USDC code.

I-11 | setCircle Allows Circle Address To Be Re Set

Category	Severity	Location	Status
Warning	Info	SourceOFTAdapterForTakeover.sol	Acknowledged

Description

The `setCircle` function allows the owner to set the address for `circle`. However the function does not prohibit the ability to call `setCircle` again once the contract is given the zero allowance `Minter` role.

> According to the standard,
>
> Be only callable by an address that Circle specifies closer to the time of the upgrade. Note that this address will not necessarily be the same address that is specified to call transferUSDCRoles.

The above describes restrictions according to the `burnLockedUSDC` function. This function is callable by the address that is set as `circle` in the `setCircle` function.

Therefore the code does not fully comply with the standard since the owner is allowed to change the `circle` address after it was already set.

Recommendation

Consider changing the logic of `setCircle` to

```
function setCircle(address _circle) external onlyOwner {
require(circle = address(0), "circle already set");
require(_circle = address(0), "zero address");
circle = _circle;
emit CircleSet(_circle);
}
```

Resolution

Chainway Labs Team: We disagree with this as operational mistakes can occur while calling `setCircle` thus making it one time is dangerous. Bridge owners have no incentive to sabotage the takeover process if they are not malicious as takeover is something desired. If they are malicious then there is no point of waiting until the takeover process as they can attack earlier.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
I-01	InflightMsgCheckLzScan.sh Can Silently Succeed	Configuration	Info	Resolved

I-01 | InflightMsgCheckLzScan.sh Can Silently Succeed

Category	Severity	Location	Status
Configuration	Info	InflightMsgCheckLzScan.sh	Resolved

Description

The takeover helper `script/usdc/for_circle_takeover/InflightMsgCheckLzScan.sh` relies on `curl` and `jq` to ensure there are no LayerZero messages in flight before pausing the bridge.

When the HTTP call fails or returns malformed JSON, `jq` exits with a non-zero status and nothing is written to `stdout`, yet the following if:

```
if echo "$response" | jq -e '.data[] | select(.status.name = "INFLIGHT" or .status.name =
"CONFIRMING")' > /dev/null; then
```

treats that failure exactly like “no inflight messages” and the function returns 0, so the script prints “SUCCESS: All checks passed.”

This creates a false sense of safety as subsequent pause scripts will continue even though the bridge’s state is unknown, making it possible that real inflight transfers get frozen mid-upgrade.

Recommendation

Fail fast on tooling errors. Add `set -euo pipefail` to the shell prologue, capture the exit status of both `curl` and `jq` and abort unless both succeed. If either command fails, emit a clear error and return a non-zero exit code so the pausing scripts refuse to continue.

Resolution

Chainway Labs Team: The issue was resolved in commit 7df2098.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits