Bridge Wormhole Executor Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Kage

MrPotatoMagic

February 10, 2026

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

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

4 Protocol Summary

The Securitize Bridge protocol previously audited by Cyfrin enables cross-chain interoperability for tokenized real-world assets (RWAs) and USDC stablecoin transfers via Circle's Cross-Chain Transfer Protocol (CCTPv2). The system consists of two primary bridge contracts:

- `SecuritizeBridge` for bridging DS Tokens (digitally securitized tokens representing traditional securities)
- `USDCBridgeV2` for USDC transfers using Wormhole integration with CCTP

`SecuritizeBridge` facilitates the movement of DS Tokens between chains while maintaining regulatory compliance through integration with Securitize's registry service. When users bridge their tokens, the source chain burns the tokens and transmits both the transfer details and the investor's compliance attributes (KYC/AML data, accreditation status, and jurisdiction information) to the destination chain via Wormhole's message-passing infrastructure. Upon receipt, the destination chain mints equivalent tokens to the recipient while simultaneously updating the local registry with the investor's compliance profile, ensuring seamless regulatory adherence across jurisdictions.

The USDC bridge components implement a dual-protocol architecture, combining Wormhole's cross-chain messaging with Circle's native CCTP for secure stablecoin transfers. The V2 implementation integrates with Circle's latest CCTP v2 features, including fast finality options and automated attestation handling. The bridge burns USDC on the source chain through Circle's `TokenMessenger` contract, while Wormhole relays the transfer metadata and CCTP attestation to enable minting on the destination chain. This design aims to provide users with automated, single-transaction cross-chain transfers without requiring manual attestation retrieval or separate destination chain transactions.

Both bridges employ access control with designated bridge callers authorized to initiate transfers, upgradeable proxy patterns for future improvements, and configurable parameters such as gas limits and chain-specific addresses. The protocol's architecture prioritizes security through established infrastructure providers (Wormhole and Circle) while maintaining the regulatory compliance requirements essential for tokenized securities through Securitize's identity and registry systems.

5 Audit Scope

The focus of this audit was the migration from Wormhole's Standard Relayer to its Executor Framework. The audit scope was limited to:

```
// most important commits:
// [https://github.com/securitize-io/bc-securitize-bridge-sc/commit/bfc4a470d7bcd878960dfecb91e2814c5a0
↪  7f060](https://github.com/securitize-io/bc-securitize-bridge-sc/commit/bfc4a470d7bcd878960dfecb91e2
↪  814c5a07f060)
// [https://github.com/securitize-io/bc-securitize-bridge-sc/commit/103ee63fdeb3f16b2443df93702885b87e7
↪  f51da](https://github.com/securitize-io/bc-securitize-bridge-sc/commit/103ee63fdeb3f16b2443df937028
↪  85b87e7f51da)

// affected in-scope files:
contracts/bridge/SecuritizeBridge.sol
contracts/bridge/USDCBridgeV2.sol

// note: these files not in scope, copied from wormhole
// [https://github.com/wormholelabs-xyz/example-messaging-executor/tree/main/evm/src/libraries](https:/
↪  /github.com/wormholelabs-xyz/example-messaging-executor/tree/main/evm/src/libraries)
contracts/wormhole/libraries/ExecutorMessages.sol
contracts/wormhole/libraries/RelayInstructions.sol
```

6 Executive Summary

Over the course of 3 days, the Cyfrin team conducted an audit on the Bridge Wormhole Executor smart contracts provided by Securitize. In this period, a total of 8 issues were found.

Summary

Project Name	Bridge Wormhole Executor
Repository	bc-securitize-bridge-sc
Commit	7ea327463106...
Fix Commit	03cba046c339...
Audit Timeline	Feb 4th - Feb 6th, 2026
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Informational	4
Gas Optimizations	2
Total Issues	8

Summary of Findings

[M-1] Unconfigured CCTP domain mapping defaults to zero, potentially routing USDC to Ethereum instead of intended destination	Resolved
[L-1] Guardian sets can expire when SecuritizeBridge is paused	Acknowledged
[I-1] `USDCBridgeV2::quoteBridge` returns inflated cost estimate by including wormhole core fee that Is never paid in CCTP v2 flow	Resolved
[I-2] Missing nonReentrant modifier on function executeVAAv1	Resolved
[I-3] Incorrect use of $_msgSender()instead of address(this)on quote request$	Resolved
[I-4] Use SafeERC20 approval instead of standard IERC20 approve function	Resolved
[G-1] Unnecessary sequence lookup and request construction in `SecuritizeBridge::_quoteBridge`	Acknowledged
[G-2] Addition and removal functions can be combined into one function	Acknowledged

7 Findings

7.1 Medium Risk

7.1.1 Unconfigured CCTP domain mapping defaults to zero, potentially routing USDC to Ethereum instead of intended destination

Description: The `USDCBridgeV2` contract maintains a mapping from Wormhole chain IDs to Circle CCTP domain IDs via `chainIdToCCTPDomain`.

When a bridge operation is initiated, the contract retrieves the CCTP domain using `USDCBridgeV2::getCCTPDomain`:

```
function getCCTPDomain(uint16 _chain) internal view returns (uint32) {
    return chainIdToCCTPDomain[_chain];  // Returns 0 if not configured
}
```

Solidity mappings return the default value (0) for uninitialized keys. The function does not validate whether the returned domain was explicitly configured or is simply the default zero value. Note that CCTP assigns domain 0 to Ethereum.

Consider the following scenario:

```
- Admin deploys USDCBridgeV2 and configures CCTP domains for some chains but forgets to configure a
↪  specific chain (e.g. Arbitrum)
- BRIDGE_CALLER invokes sendUSDCCrossChainDeposit() with _targetChain set to the arbitrum's Wormhole ID
- getCCTPDomain() returns 0 (default mapping value)
- _transferUSDC() calls circleTokenMessenger.depositForBurn() with destinationDomain = 0
-  In Circle's CCTP, domain 0 corresponds to Ethereum mainnet
- If the source chain's TokenMessenger has Ethereum configured as a valid remote, the transaction
↪  succeeds
- USDC is burned on source chain and minted on Ethereum instead of the intended destination
```

Impact: USDC intended for a recipient on Chain X can be minted on Ethereum instead.

Recommended Mitigation: Consider adding an explicit validation to ensure the CCTP domain has been configured before proceeding with the transfer:

```
function getCCTPDomain(uint16 _chain) internal view returns (uint32) {
      uint32 domain = chainIdToCCTPDomain[_chain];

++      if (domain == 0 && _chain != 2) {
          revert CCTPDomainNotConfigured();
      }

      return domain;
  }
```

Securitize: Fixed in 65369a1 and f92422b.

Cyfrin: Verified.

7.2 Low Risk

7.2.1 Guardian sets can expire when SecuritizeBridge is paused

Description: Any arbitrary caller can call function `executeVAAv1` to execute pending cross-chain messages. When `SecuritizeBridge` is paused, messages cannot be executed.

```
    function executeVAAv1(bytes calldata _encodedVM) external payable whenNotPaused {
        IWormhole _wormholeCore = wormholeCore;
        (IWormhole.VM memory vm, bool valid, ) = _wormholeCore.parseAndVerifyVM(_encodedVM);
        if (!valid) revert InvalidWormholeMessage();
```

In this paused state, inflight `encodedVM`(s) can expire since guardian sets have an expiry time as observed in the Wormhole Core contract.

```
/// @dev Checks if VM guardian set index matches the current index (unless the current set is expired).
        if(vm.guardianSetIndex != getCurrentGuardianSetIndex() && guardianSet.expirationTime <
        ↪  block.timestamp){
            return (false, "guardian set has expired");
        }
```

Impact: This can lead to a scenario where investors have their DS tokens burned on the source chain but never issued on the destination chain. While DS tokens could be manually issued, this could create significant overhead if there are numerous messages in-flight.

Recommended Mitigation: If this is an accepted risk, consider acknowledging the issue and ensure pending in-flight messages (in expired guardian sets) are issued DS tokens manually.

Alternatively, to avoid such a scenario, it is recommended to:

- Remove the bridge address from all source chains
- Allow pending in-flight messages to be fulfilled/executed on the target chain
- Pause the target chain to prevent pending messages from expiring.

Securitize: Acknowledged.

7.3 Informational

7.3.1 `USDCBridgeV2::quoteBridge` returns inflated cost estimate by including wormhole core fee that Is never paid in CCTP v2 flow

Description: `USDCBridgeV2::quoteBridge` estimates the cost of a cross-chain USDC transfer. However, this function returns an inflated estimate that includes a Wormhole Core message fee (`coreFee`) which is never actually paid during the bridge operation.

```
function quoteBridge(uint16 _targetChain) public override view returns (uint256 cost) {
      (uint256 coreFee, uint256 execFee) = _quoteBridge(_targetChain);
      cost = execFee + coreFee;  // @audit coreFee added but is not used in _quoteBridge
  }
```

`USDCBridgeV2::_quoteBridge` calculates both `coreFee` (Wormhole message fee) and `execFee` (Executor fee), but the CCTP v2 flow does not publish a Wormhole VAA and therefore does not require `coreFee`.

```
function _quoteBridge(uint16 _targetChain) private view returns (uint256 coreFee, uint256 execFee) {
      IWormhole _wormholeCore = wormholeCore;

      coreFee = _wormholeCore.messageFee();  // @audit calculated but never used in CCTP v2
      bytes memory request = ExecutorMessages.makeCCTPv2Request();
      bytes memory relayInstructions = RelayInstructions.encodeGas(gasLimit, 0);
      execFee = executorQuoterRouter.quoteExecution(
          _targetChain,
          bytes32(0),
          _msgSender(),
          quoterAddr,
          request,
          relayInstructions
      );
  }
```

In the `USDCBridgeV2::sendUSDCCrossChainDeposit`, we can see that `coreFee` is calculated but never used.

```
function sendUSDCCrossChainDeposit(...) external ... {
      (, uint256 execFee) = _quoteBridge(_targetChain);  // coreFee discarded
      if (address(this).balance < execFee) revert InsufficientContractBalance();
      // ...
      executorQuoterRouter.requestExecution{value: execFee}(...);  // @audit only execFee paid
  }
```

The CCTP v2 flow uses Circle's native messaging infrastructure rather than Wormhole VAAs. No call to `wormholeCore::publishMessage` is made in this flow, so the Wormhole Core message fee is not required.

Impact: External integrations (frontends, other contracts, off-chain systems) querying `USDCBridgeV2::quoteBridge` receive an inflated cost estimate

Recommended Mitigation: Consider removing `coreFee` from `quoteBridge` calculation.

Securitize: Fixed in 73a0a3c.

Cyfrin: Verified.

7.3.2 Missing nonReentrant modifier on function executeVAAv1

Description: `SecuritizeBridge::executeVAAv1` is missing a `nonReentrant` modifier, while this modifier exists in `SecuritizeBridge::bridgeDSTokens`. While this does not pose a risk currently, it is recommended to implement the modifier to maintain consistency.

```
function executeVAAv1(bytes calldata _encodedVM) external payable whenNotPaused {
```

Recommended Mitigation: Consider adding the nonReentrant modifier on function `executeVAAv1`

Securitize: Fixed in a6b287c.

Cyfrin: Verified.

7.3.3 Incorrect use of _msgSender() instead of address(this) on quote request

Description: In the `USDCBridgeV2` contract, function `_quoteBridge` uses the refund address as the `_msgSender()` on the `quoteExecution` call. However in function `sendUSDCCrossChainDeposit`, the refund address is used as `address(this)` on the `requestExecution` external call.

Function `requestExecution`:

```
executorQuoterRouter.requestExecution{value: execFee}(
            _targetChain,
            bytes32(0),
            address(this), // << refund address
            quoterAddr,
            ExecutorMessages.makeCCTPv2Request(),
            RelayInstructions.encodeGas(gasLimit, 0)
        );
```

Function `_quoteBridge`:

```
function _quoteBridge(uint16 _targetChain) private view returns (uint256 coreFee, uint256 execFee) {
        IWormhole _wormholeCore = wormholeCore; // cache storage

        coreFee = _wormholeCore.messageFee();
        bytes memory request = ExecutorMessages.makeCCTPv2Request();
        bytes memory relayInstructions = RelayInstructions.encodeGas(gasLimit, 0);
        execFee = executorQuoterRouter.quoteExecution(
            _targetChain,
            bytes32(0),
            _msgSender(), // << refund address used as msg.sender
            quoterAddr,
            request,
            relayInstructions
        );
    }
```

While this does not affect the quote calculations in the ExecutorQuoterRouter currently, it is recommended to use the accurate refund address to maintain consistency.

Recommended Mitigation: Since the actual refund address is expected to be the `USDCBridgeV2` contract, use `address(this)` instead of `_msgSender()` in function `_quoteBridge`.

Securitize: Fixed in c618dee.

Cyfrin: Verified.

7.3.4 Use SafeERC20 approval instead of standard IERC20 approve function

Description: Use SafeERC20::forceApprove function instead of the standard IERC20 approve function in `USDCBridgeV2._transferUSDC`

```
IERC20(_USDC).approve(address(circleTokenMessenger), _amount);
```

Recommended Mitigation: Consider following the above recommendation.

Securitize: Fixed in d138209.

Cyfrin: Verified.

7.4 Gas Optimization

7.4.1 Unnecessary sequence lookup and request construction in `SecuritizeBridge::_quoteBridge`

Description: `SecuritizeBridge::_quoteBridge` performs several unnecessary operations to construct the request parameter, which is subsequently ignored by the `ExecutorQuoter` contract:

```
function _quoteBridge(uint16 _targetChain) private view returns (uint256 coreFee, uint256 execFee) {
        address targetAddress = bridgeAddresses[_targetChain];
        if (targetAddress == address(0)) revert BridgeAddressNotConfigured();

        IWormhole _wormholeCore = wormholeCore; // cache storage
        uint64 sequence = _wormholeCore.nextSequence(address(this)); //@audit sequence generated

        coreFee = _wormholeCore.messageFee();
        bytes memory request = ExecutorMessages.makeVAAv1Request(_wormholeCore.chainId(),
        ↪  _addressToBytes32(address(this)), sequence);
         //@audit request is computed but this is not used while
        bytes memory relayInstructions = RelayInstructions.encodeGas(gasLimit, 0);
        execFee = executorQuoterRouter.quoteExecution(
            _targetChain,
            _addressToBytes32(targetAddress),
            _msgSender(),
            quoterAddr,
            request,
            relayInstructions
        );
    }
```

The `ExecuteQuoteRouter::quoteExecution` calls the `ExecuteQuote::requestQuote` function that doesn't use the `request` parameter.

```
    //ExecuteQuoteRouter.sol
     function quoteExecution(
        uint16 dstChain,
        bytes32 dstAddr,
        address refundAddr,
        address quoterAddr,
        bytes calldata requestBytes,
        bytes calldata relayInstructions
    ) external view returns (uint256 requiredPayment) {
        requiredPayment =
            quoterContract[quoterAddr].requestQuote(dstChain, dstAddr, refundAddr, requestBytes,
            ↪  relayInstructions);
    }
```

```
    //ExecutorQuoter.sol
    function requestQuote(
        uint16 dstChain,
        bytes32, //dstAddr,
        address, //refundAddr,
        bytes calldata, //requestBytes, //@audit request bytes are unused
        bytes calldata relayInstructions
    ) external view returns (uint256 requiredPayment) {
        ChainInfo storage dstChainInfo = chainInfos[dstChain];
        if (!dstChainInfo.enabled) {
            revert ChainDisabled(dstChain);
        }
        (uint256 gasLimit, uint256 msgValue) = totalGasLimitAndMsgValue(relayInstructions);
        // NOTE: this does not include any maxGasLimit or maxMsgValue checks
        requiredPayment = estimateQuote(quoteByDstChain[dstChain], dstChainInfo, gasLimit, msgValue);

        return requiredPayment;
```

```
    }
```

Recommended Mitigation: Consider removing the unnecessary sequence lookup and request construction, and replace request with empty bytes.

Securitize: Acknowledged.

7.4.2 Addition and removal functions can be combined into one function

Description: Functions `addBridgeCaller` and `removeBridgeCaller` in `USDCBridgeV2` can be combined into one function to save deployment gas.

```
function addBridgeCaller(address _account) external override addressNotZero(_account)
↪  onlyRole(DEFAULT_ADMIN_ROLE) {
        grantRole(BRIDGE_CALLER, _account);
        emit BridgeCallerAdded(_account);
    }

    function removeBridgeCaller(address _account) external override addressNotZero(_account)
    ↪  onlyRole(DEFAULT_ADMIN_ROLE) {
        revokeRole(BRIDGE_CALLER, _account);
        emit BridgeCallerRemoved(_account);
    }
```

Similarly functions `setBridgeAddress` and `removeBridgeAddress` in `SecuritizeBridge` can be combined as well.

```
function setBridgeAddress(uint16 _chainId, address _bridgeAddress) external override onlyOwner
↪  addressNotZero(_bridgeAddress) {
        bridgeAddresses[_chainId] = _bridgeAddress;
        emit BridgeAddressAdded(_chainId, _bridgeAddress);
    }


    function removeBridgeAddress(uint16 _chainId) external override onlyOwner {
        delete bridgeAddresses[_chainId];
        emit BridgeAddressRemoved(_chainId);
    }
```

Recommended Mitigation: Consider combining the functions in the following manner with updated event names:

```
function addBridgeCaller(address _account, bool _status) external override addressNotZero(_account)
↪  onlyRole(DEFAULT_ADMIN_ROLE) {
        if (_status) grantRole(BRIDGE_CALLER, _account);
        else revokeRole(BRIDGE_CALLER, _account);
        emit BridgeCallerUpdated(_account);
    }
```

```
function setBridgeAddress(uint16 _chainId, address _bridgeAddress) external override onlyOwner {
        bridgeAddresses[_chainId] = _bridgeAddress;
        emit BridgeAddressUpdated(_chainId, _bridgeAddress);
    }
```

Securitize: Acknowledged.