Securitize Bridge Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Hans

October 28, 2024

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

SecuritizeBridge is a cross-chain bridge system designed to transfer DS tokens between different blockchain networks using Wormhole's message passing protocol. It handles both token transfers and investor compliance data synchronization across chains.

4.0.1 Actors and Roles

1. **Actors**:
    - Users: Can initiate token bridge transfers if they hold DS tokens (called "investors")
    - Owner: Can configure bridge addresses for different chains and pause/unpause the contract
    - Wormhole Relayer: Wormhole-deployed relayers provide a mechanism for contracts on one blockchain to send messages to contracts on another without requiring off-chain infrastructure. (Trusted)
2. **Roles**:
    - Owner has administrative privileges.
    - Can pause/resume the bridge.
    - Can upgrade the bridge.
    - Can add/change/remove the bridge contract address on target chains.
    - Technically bridge operations require proper KYC and compliance status.
    - If `msg.sender` is not a valid investor, bridge function will revert because the `availableBalanceForTransfer` will be zero.

4.0.2 Key Components

1. SecuritizeBridge: Main contract handling cross-chain token transfers and compliance data
2. WormholeRelayer: Handles cross-chain message delivery. Trusted.
3. Compliance Services: Validates transfer restrictions and lock periods.

4.0.3 Token Bridge Flow

1. Source Chain Operations:
    - Validate user balance and compliance status
    - Burn tokens from sender
    - Send cross-chain message with investor details
2. Target Chain Operations:
    - Verify message source and relayer
    - Update investor registry data (add if not exists)
    - Mint tokens to receiver

4.0.4 General Security Considerations

1. Centralization Risks:
    - Owner controls bridge address configuration
    - Reliance on Wormhole relayer network
2. Data Integrity:
    - Synchronized investor data across chains

4.0.5 Integration Points

1. Wormhole Protocol: For cross-chain message passing
2. DS Token System:
    - Registry Service
    - Compliance Service
    - Token Contract Cyfrin team referenced `@securitize/digital_securities 3.1.3`.

5 Audit Scope

All Solidity files inside the contracts directory except mock contracts are included in the audit scope. The audit scope includes the following:

```
contracts/
 bridge/
    SecuritizeBridge.sol
    ISecuritizeBridge.sol
 utils/
    BaseContract.sol
    Proxies.sol
```

6 Executive Summary

Over the course of 5 days, the Cyfrin team conducted an audit on the Securitize Bridge smart contracts provided by Securitize. In this period, a total of 3 issues were found.

Summary

Project Name	Securitize Bridge
Repository	bc-securitize-bridge-sc
Commit	0db2d1a9f9ad...
Audit Timeline	Oct 21st - Oct 25th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Informational	1
Gas Optimizations	0
Total Issues	3

Summary of Findings

[M-1] Allow message value to be more than the quote cost	Resolved
[L-1] Make the gas limit configurable	Resolved
[I-1] Add a validation to check the message sender and the token value	Resolved

7 Findings

7.1 Medium Risk

7.1.1 Allow message value to be more than the quote cost

Description: The `SecuritizeBridge` contract's `bridgeDSTokens()` function requires users to provide an exact value that matches the quote obtained from `quoteBridge()`. This strict matching requirement creates issues because the actual cost can change between when a user checks the quote and when they submit their transaction.

```
    function bridgeDSTokens(uint16 targetChain, uint256 value) public override payable whenNotPaused {
        uint256 cost = quoteBridge(targetChain);
        require(msg.value == cost, "Transaction value should be equal to quoteBridge response");
...
    }
```

The cost calculation depends on multiple factors as shown in Wormhole's `DeliveryProvider` contract here, including gas prices on the target chain and asset conversion rates. These values can fluctuate frequently based on network conditions.

```
    function quoteEvmDeliveryPrice(
        uint16 targetChain,
        Gas gasLimit,
        TargetNative receiverValue
    )
        public
        view
        returns (LocalNative nativePriceQuote, GasPrice targetChainRefundPerUnitGasUnused)
    {
        // Calculates the amount to refund user on the target chain, for each unit of target chain gas
        ↪  unused
        // by multiplying the price of that amount of gas (in target chain currency)
        // by a target-chain-specific constant 'denominator'/('denominator' + 'buffer'), which will be
        ↪  close to 1

        (uint16 buffer, uint16 denominator) = assetConversionBuffer(targetChain);
        targetChainRefundPerUnitGasUnused = GasPrice.wrap(gasPrice(targetChain).unwrap() *
        ↪  (denominator) / (uint256(denominator) + buffer));

        // Calculates the cost of performing a delivery with 'gasLimit' units of gas and
        ↪  'receiverValue' wei delivered to the target contract

        LocalNative gasLimitCostInSourceCurrency = quoteGasCost(targetChain, gasLimit);
        LocalNative receiverValueCostInSourceCurrency = quoteAssetCost(targetChain, receiverValue);
        nativePriceQuote = quoteDeliveryOverhead(targetChain) + gasLimitCostInSourceCurrency +
        ↪  receiverValueCostInSourceCurrency;

        // Checks that the amount of wei that needs to be sent into the target chain is <= the 'maximum
        ↪  budget' for the target chain

        TargetNative gasLimitCost = gasLimit.toWei(gasPrice(targetChain)).asTargetNative();
        if(receiverValue.asNative() + gasLimitCost.asNative() > maximumBudget(targetChain).asNative()) {
            revert ExceedsMaximumBudget(targetChain, receiverValue.unwrap() + gasLimitCost.unwrap(),
            ↪  maximumBudget(targetChain).unwrap());
        }
    }
```

When the cost changes even slightly between the quote check and transaction submission, the transaction fails. This creates a poor user experience where transactions frequently revert despite users attempting to pay the

correct amount.

A malicious actor could worsen this issue by manipulating network conditions to cause price fluctuations, effectively preventing other users from successfully bridging their assets.

Impact: Users face failed transactions when attempting to bridge assets causing frustration. In extreme cases, attackers could temporarily prevent specific users from bridging assets by manipulating conditions to cause price fluctuations.

Recommended Mitigation: Modify the function to accept value that exceed the current quote and automatically refund any excess amount back to the user. This approach provides flexibility to handle minor price fluctuations while ensuring users don't overpay.

Securitize: Fixed in commit d3b97a and 221759.

Cyfrin: Verified.

7.2 Low Risk

7.2.1 Make the gas limit configurable

Description: The `SecuritizeBridge` contract currently uses a fixed (hardcoded) gas limit of 2,500,000 for all cross-chain message transactions through the Wormhole protocol. This value represents the maximum computational units (gas) allowed for the execution of the transaction on the target chain.

While this value works under current implementation, having it as a hardcoded constant makes it difficult to adjust if future upgrades or changes to the contract's functionality require different gas consumption. For example, if the contract's logic is upgraded and requires more computational steps, the current gas limit might become insufficient, requiring a full contract redeployment just to adjust this value.

Recommended Mitigation: Make the gas limit configurable by adding an owner-controlled function to update the value. This would allow the protocol administrators to adjust the gas limit if future contract upgrades require different gas consumption, without requiring a full contract redeployment.

Replace:

```
uint256 public constant GAS_LIMIT = 2500_000;
```

with:

```
uint256 public gasLimit;

function setGasLimit(uint256 _gasLimit) external onlyOwner {
    gasLimit = _gasLimit;
}
```

Securitize: Fixed in commit 525d86.

Cyfrin: Verified.

7.3 Informational

7.3.1 Add a validation to check the message sender and the token value

Description: The `SecuritizeBridge` contract has a potential concern in its token bridging functionality. While the contract is designed to work with compliance-verified investors, there's a gap in the validation process:

In the current Implementation:

- The contract checks if users have enough tokens to bridge (`balanceOf` check).
- It validates if tokens are not locked (`validateLockedTokens` check). However, it doesn't explicitly verify if the sender is a valid investor.

As a result, when a user attempts to bridge 0 tokens, both validation checks will pass. This means non-validated investors could successfully execute bridge transactions with 0 tokens. While this doesn't result in any token transfer, it creates unnecessary cross-chain messages and potentially create noise in system monitoring and event logs

Securitize: Fixed in commit 6529fe.

Cyfrin: Verified.