Securitize DSTokenClassSwap Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Hans

January 10, 2025

Contents

1 About Cyfrin **2**

2 Disclaimer **2**

3 Risk Classification **2**

4 Protocol Summary **2**

5 Audit Scope **2**

6 Executive Summary **2**

7 Findings **4**

7.1 Medium Risk 4

7.1.1 Meta transactions will not work due to direct msg.sender usage in validateLockedTokens . . 4

7.2 Informational 6

7.2.1 Missing zero address validation in initialize function 6

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

Securitize and its subsidiaries have a global footprint with offices worldwide, giving them a unique perspective and the opportunity to lead the evolution of digital asset securities (security tokens) in and across multiple markets.

The contract in scope (DSTokenClassSwap.sol) is a token swap system that handles the conversion between different classes of security tokens while maintaining compliance and registry requirements.

5 Audit Scope

The contract `DSTokenClassSwap.sol` and its dependencies are audited.

6 Executive Summary

Over the course of 5 days, the Cyfrin team conducted an audit on the Securitize DSTokenClassSwap smart contracts provided by Securitize. In this period, a total of 2 issues were found.

The security review of DSTokenClassSwap protocol revealed two findings:

1. A Medium severity issue was identified where meta transactions would not function properly due to direct usage of `msg.sender` in the `validateLockedTokens` function instead of `_msgSender()`. This limitation would prevent the protocol from supporting gasless transactions through relayers.
2. An Informational finding highlighted the lack of zero address validation in the initialize function for `_sourceDSToken` and `_targetDSToken` parameters. While not critical, implementing these checks would improve the contract's robustness.

The findings primarily focus on improving meta transaction support and implementing best practices for contract initialization. The core token swap functionality appears to be well-implemented with proper compliance and registry service integrations.

Summary

Project Name	Securitize DSTokenClassSwap
Repository	bc-dstoken-class-swap-sc
Commit	f2471e696d8e. . .
Fix Commit	b26a167524df. . .
Audit Timeline	Jan 6th - Jan 10th
Methods	Manual Review

Issues Found

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

Summary of Findings

[M-1] Meta transactions will not work due to direct msg.sender usage in validateLockedTokens	Resolved
[I-1] Missing zero address validation in initialize function	Resolved

7 Findings

7.1 Medium Risk

7.1.1 Meta transactions will not work due to direct msg.sender usage in validateLockedTokens

Description: The protocol makes use of `_msgSender()` in several parts and it is understood the protocol team considers possible support of meta transactions where relayers will handle the transactions that are signed by the investors. But the `validateLockedTokens` function uses `msg.sender` directly to check the available balance for transfer. This prevents the contract from supporting meta transactions since the actual token holder's address would be different from the relayer's address (msg.sender) in a meta transaction context.

```
function validateLockedTokens(string memory investorId, uint256 value, IDSRegistryService registryService) private view {
    IDSComplianceService complianceService = IDSComplianceService(sourceServiceConsumer.getDSService(sourceServiceConsumer.COMPLIANCE_SERVICE()));
    IDSComplianceConfigurationService complianceConfigurationService = IDSComplianceConfigurationService(sourceServiceConsumer.getDSService(sourceServiceConsumer.COMPLIANCE_CONFIGURATION_SERVICE()));

    string memory country = registryService.getCountry(investorId);
    uint256 region = complianceConfigurationService.getCountryCompliance(country);

    // lock/hold up validation
    uint256 lockPeriod = (region == US) ? complianceConfigurationService.getUSLockPeriod() : complianceConfigurationService.getNonUSLockPeriod();
    uint256 availableBalanceForTransfer = complianceService.getComplianceTransferableTokens(msg.sender, block.timestamp, uint64(lockPeriod));//@audit-issue msg.sender can be different from _msgSender
    require(availableBalanceForTransfer >= value, "Not enough unlocked balance");
}
```

Note that in the function `ComplianceServiceRegulated::getComplianceTransferableTokens()`, the first parameter `_who` is used to get investor info by `getRegistryService().getInvestor(_who);`.

```
@securitize\digital_securities\contracts\compliance\ComplianceServiceRegulated.sol
    function getComplianceTransferableTokens(
        address _who,
        uint256 _time,
        uint64 _lockTime
    ) public view override returns (uint256) {
        require(_time != 0, "Time must be greater than zero");
        string memory investor = getRegistryService().getInvestor(_who);

        uint256 balanceOfInvestor = getLockManager().getTransferableTokens(_who, _time);

        uint256 investorIssuancesCount = issuancesCounters[investor];

        //No locks, go to base class implementation
        if (investorIssuancesCount == 0) {
            return balanceOfInvestor;
        }

        uint256 totalLockedTokens = 0;
        for (uint256 i = 0; i < investorIssuancesCount; i++) {
            uint256 issuanceTimestamp = issuancesTimestamps[investor][i];

            if (uint256(_lockTime) > _time || issuanceTimestamp > (_time - uint256(_lockTime))) {
                totalLockedTokens = totalLockedTokens + issuancesValues[investor][i];
            }
        }
        //there may be more locked tokens than actual tokens, so the minimum between the two
        uint256 transferable = balanceOfInvestor - Math.min(totalLockedTokens, balanceOfInvestor);

        return transferable;
    }
```

In other parts, `msg.sender` and `_msgSender()` are being used correctly to handle the meta transactions.

Impact: For meta transactions, `getComplianceTransferableTokens` will return incorrect value because `msg.sender` is not necessarily the investor. Users would always need to have ETH to pay for gas, which defeats one of the main benefits of meta transactions where users could have their transactions relayed by others.

Recommended Mitigation: Use `_msgSender()` instead of using `msg.sender` in the specific part as belows.

```
    function validateLockedTokens(string memory investorId, uint256 value, IDSRegistryService
    ↪  registryService) private view {
        IDSComplianceService complianceService = IDSComplianceService(sourceServiceConsumer.getDSServic ⌋
        ↪  e(sourceServiceConsumer.COMPLIANCE_SERVICE()));
        IDSComplianceConfigurationService complianceConfigurationService =
        ↪  IDSComplianceConfigurationService(sourceServiceConsumer.getDSService(sourceServiceConsumer. ⌋
        ↪  COMPLIANCE_CONFIGURATION_SERVICE()));

        string memory country = registryService.getCountry(investorId);
        uint256 region = complianceConfigurationService.getCountryCompliance(country);

        // lock/hold up validation
        uint256 lockPeriod = (region == US) ? complianceConfigurationService.getUSLockPeriod() :
        ↪  complianceConfigurationService.getNonUSLockPeriod();//@audit-info assume these values are
        ↪  representing time duration in seconds
--        uint256 availableBalanceForTransfer =
↪   complianceService.getComplianceTransferableTokens(msg.sender, block.timestamp, uint64(lockPeriod));
++        uint256 availableBalanceForTransfer =
↪   complianceService.getComplianceTransferableTokens(_msgSender(), block.timestamp,
↪   uint64(lockPeriod));

        require(availableBalanceForTransfer >= value, "Not enough unlocked balance");
    }
```

Securitize: Fixed in commit b26a16.

Cyfrin: Verified.

7.2 Informational

7.2.1 Missing zero address validation in initialize function

Description: The `initialize` function in `DSTokenClassSwap` contract does not validate that the input addresses `_sourceDSToken` and `_targetDSToken` are non-zero addresses.

```
DSTokenClassSwap.sol
    function initialize(address _sourceDSToken, address _targetDSToken) public override onlyProxy initializer {
        __BaseDSContract_init();
        sourceDSToken = IDSToken(_sourceDSToken);//@audit-issue INFO check zero address
        sourceServiceConsumer = IDSServiceConsumer(_sourceDSToken);
        targetDSToken = IDSToken(_targetDSToken);
        targetServiceConsumer = IDSServiceConsumer(_targetDSToken);
    }
```

Recommended Mitigation: Add zero address validation checks.

Securitize: Fixed in commit b26a16.

Cyfrin: Verified.