Securitize Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Hans

December 17, 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

The audited part of the Securitize system consists of four components: the NAV Provider, the Redemption contract, the Swap contract, and the Vault contract. These components work together to allow investors to convert the digital security tokens into stable tokens and vice versa.

4.1 Architecture

4.1.1 Securitize NAV Provider

- Defines a common interface (`ISecuritizeNavProvider`) to get the current NAV rate for redeem/swap/liquidation operations involving DSToken and stable coins.
- The rate is interpreted as the number of stable coins that can be exchanged for one DSToken. (asset:liquidity ratio)
- The rate is supposed to be expressed in the same decimals as the stable coin.
- UUPS upgradeable contract pattern is used.

4.1.2 Securitize Redemption Protocol

- Allows investors to redeem their digital securities for stable coins.
- Designed to support multiple types of liquidity providers for supplying stable coins. The current implementation supports two types of liquidity providers: `AllowanceLiquidityProvider` and `CollateralLiquidityProvider`. `AllowanceLiquidityProvider` allows the provider wallet to supply stable coins directly by approving the provider contract to transfer stable coins. `CollateralLiquidityProvider` interacts with an external collateral redemption contract to supply stable coins.
- Future extensions can be made by extending the `ILiquidityProvider` interface.
- UUPS upgradeable contract pattern is used.

4.1.3 Securitize Swap Contract

- Investors can swap their ERC20 tokens for stable coins.
- There are two swap modes defined by `SwapMode` enum: `ExternalCollateralRedemption`, `DirectStableCoin`.
- There is no swap rate mechanism in the contract and all the amounts are provided by the caller.
- Only issuer or master can call the swap function.

Token Flow ExternalCollateralRedemption Mode

```
Investor -> ERC20 -> Issuer
LiquidityProvider -> Collateral -> Contract -> ExternalRedemption -> Stablecoin -> Investor
```

DirectStableCoin Mode

```
Investor -> ERC20 -> Issuer
LiquidityProvider -> Stablecoin -> Investor
```

4.1.4 Securitize Vault Contract

The SecuritizeVault is an ERC4626-compliant tokenized vault system that manages assets with the following key features:

- Asset deposit and redemption mechanisms
- Liquidation functionality with configurable public access
- NAV (Net Asset Value) based share price calculation
- Role-based access control
- Upgradeable contract architecture

4.2 Actors

- **NAV Provider Owner**
  - Sets the rate.
  - Pauses and unpauses the contract.
  - Upgrades the contract.
- **Redemption Owner**
  - Sets the liquidity provider and NAV rate provider.
  - Pauses and unpauses the contract.
  - Upgrades the contract.
- **Swap**
  - Owner role
    - Pauses and unpauses the contract.
  - Master role
    - Upgrades the contract.
  - Issuer role

        - Calls the swap function.
- **Vault**
    - AccessControl Admin role
        - Pauses and unpauses the contract.
        - Upgrades the contract.
        - Adds and removes roles.
    - Vault Operator Role
        - Interacts with the vault via `deposit` and `redeem` functions.
    - Liquidator Role
        - Interacts with the vault via `liquidate` function especially when the liquidation is not public.
- **Investor**
    - Engages in redemption/swap/liquidate operations.
    - Holds and manages digital securities and stable coins.

5 Audit Scope

- securitize_dev-bc-nav-provider All Solidity contracts in the `contracts` directory are included in the audit scope.
- securitize_dev-bc-redemption All Solidity contracts in the `contracts` directory except the `mocks` are included in the audit scope.
- securitize_dev-bc-securitize-vault All Solidity contracts in the `contracts` directory except the `mocks` are included in the audit scope.
- securitize_dev-securitize-swap All Solidity contracts in the `contracts` directory except the `mocks` and `non-auditable` are included in the audit scope.

6 Executive Summary

Over the course of 10 days, the Cyfrin team conducted an audit on the Securitize smart contracts provided by Securitize. In this period, a total of 13 issues were found.

The audit revealed a critical issue with SecuritizeVault's share token having incorrect decimals, alongside several medium-risk findings including ambiguous owner terminology in access control, inconsistent token amounts in external redemption, and lack of slippage protection in the liquidation function. Multiple low-risk issues were identified, such as insufficient storage gap in BaseSecuritizeSwap contract, and missing input validation in privileged functions. The codebase had potential security concerns regarding replay attacks due to static DOMAIN_SEPARATOR and missing PausableUpgradeable initialization in the SecuritizeSwap contract. Several informational findings were noted regarding incorrect/misleading comments, naming improvements, and an unused state variable in BaseSecuritizeSwap contract. The findings demonstrate a need for particular attention to decimal handling, access control mechanisms, and proper initialization of upgradeable contracts.

Summary

Project Name	Securitize
Repository	bc-redemption-sc
Commit	3977ca8ffb25...
Fix Commit	8254e84a8bd2...
Repository 2	securitize-swap
Commit	c58f74611a97...
Fix Commit	8254e84a8bd2...
Repository 3	bc-securitize-vault-sc
Commit	5ddd6f995c94...
Fix Commit	8254e84a8bd2...
Repository 3	bc-nav-provider-sc
Commit	f86f85cf520a...
Fix Commit	8254e84a8bd2...
Audit Timeline	Dec 3rd - Dec 16th
Methods	Manual Review, Stateful Fuzzing

Issues Found

Critical Risk	1
High Risk	0
Medium Risk	3
Low Risk	5
Informational	4
Gas Optimizations	0
Total Issues	13

Summary of Findings

[C-1] SecuritizeVault's share token is in wrong decimals	Resolved
[M-1] Lack of slippage protection in the SecuritizeVault's liquidation function	Resolved
[M-2] Inconsistent token amount in external redemption process of SecuritizeVault's liquidate function	Resolved
[M-3] Ambiguous owner terminology creates confusion in access control of SecuritizeVault	Resolved
[L-1] Missing input validation in privileged functions	Resolved
[L-2] Insufficient storage gap in BaseSecuritizeSwap contract	Resolved

[L-3] Missing PausableUpgradable initialization in SecuritizeSwap contract initialization	Resolved
[L-4] Potential replay attacks due to static DOMAIN$_{SEPARATOR}$	Resolved
[L-5] Share tokens can be transferred	Acknowledged
[I-1] Incorrect/misleading comments	Resolved
[I-2] Naming improvements	Resolved
[I-3] Incorrect/misleading comments (2) - SecuritizeVault	Resolved
[I-4] Unused state variable in BaseSecuritizeSwap contract	Resolved

7 Findings

7.1 Critical Risk

7.1.1 SecuritizeVault's share token is in wrong decimals

Description: The `SecuritizeVault` inherits OpenZeppelin's `ERC4626Upgradeable` where share tokens maintain the same decimals as the underlying asset token by default. However, the vault overrides `_convertToShares()` and `_convertToAssets()` to integrate NAV provider rates, which are assumed to be in liquidation token decimals. This creates an inconsistency in decimal handling when external redemption is used and the liquidation token differs from the asset token.

In `_convertToShares()`, the returned share amount is calculated using the NAV rate and maintains the decimals of the liquidation token instead of the expected share token decimals. Similarly in `_convertToAssets()`, one branch of calculation assumes the input shares are in liquidation token decimals while the other assumes asset token decimals.

```
SecuritizeVault.sol
310:     function _convertToShares(uint256 _assets, Math.Rounding rounding) internal view virtual
↪  override(ERC4626Upgradeable) returns (uint256) {
311:         uint256 rate = navProvider.rate();//@audit-info D{liquidationToken}
312:         uint256 decimalsFactor = 10 ** decimals();
313:         uint256 totalSharesAfterDeposit = rate.mulDiv(totalAssets() + _assets, decimalsFactor,
↪  rounding);//@audit-info D{liquidationToken} * D{asset} / D{asset} -> D{liquidationToken}
314:         if (totalSharesAfterDeposit >= totalSupply()) {
315:             return totalSharesAfterDeposit - totalSupply();//@audit-issue the return amount must
↪  be in the decimals of the share token but it is in liquidationToken's decimals
316:         }
317:         return 0;
318:     }

333:     function _convertToAssets(uint256 _shares, Math.Rounding rounding) internal view virtual
↪  override(ERC4626Upgradeable) returns (uint256) {
334:         if (totalSupply() == 0) {
335:             return 0;
336:         }
337:         uint256 rate = navProvider.rate();//@audit-info D{liquidationToken}
338:         uint256 decimalsFactor = 10 ** decimals();
339:         return Math.min(_shares.mulDiv(decimalsFactor, rate, rounding),
↪  _shares.mulDiv(totalAssets(), totalSupply(), rounding));//@audit-info D{share} * D{asset} /
↪  D{liquidationToken}, D{share} * D{asset} / D{share}
340:     }
```

Impact: Users will receive incorrect share token amounts during minting/burning operations due to decimal mismatches between liquidation tokens and share tokens, leading to significant value loss when the vault share token is integrated to external DeFi protocols.

Proof Of Concept: Below is a test case written in Foundry.

```
    function testShareDecimalsWrong() public {
        navProvider.setRate(1e6);// nav provider rate is 1 in the decimals of liquidationToken

        // deposit
        vm.startPrank(owner);
        uint256 depositAmount = 1000e18;// 1000 asset tokens
        assetToken.mint(owner, depositAmount);
        assetToken.approve(address(vault), depositAmount);
        vault.deposit(depositAmount, owner);
        uint256 redeemShares = vault.balanceOf(owner); // 100% redeem
        emit log_named_uint("Share token decimals", vault.decimals()); // 18
        emit log_named_uint("Share amount", redeemShares); // 1000 * 1e6
```

```
        vm.stopPrank();
    }
```

The output is as below.

```
forge test -vvv --match-test testShareDecimalsWrong

Ran 1 test for test/forge/SecuritizeVaultTest_R.t.sol:SecuritizeVaultTest_R
[PASS] testShareDecimalsWrong() (gas: 159913)
Logs:
  Share token decimals: 18
  Share amount: 1000000000
```

Recommended Mitigation: There can be two solutions.

1. Override `_decimalsOffset()` or `decimals()` so that they give the decimals of liquidation token. This can be a more straightforward fix but it is not recommended because. When the external redemption is used, the liquidation token will be a stable coin with 6 decimals. Hence the share tokens will be in 6 decimals as well and it will incur precision loss in numerous places. Also it should be checked if the other parts of the OZ's ERC4626Upgradeable is not affected.
2. Change the implementation so that the `_convertToShares` and `_convertToAssets` return values in the correct decimals.

It is also recommended to use more accurate naming in varibles (e.g. use `assetDecimalFactor` or `shareDecimalFactor` rather than `decimalFactor`) and document the decimals of values and formulas in all places.

Securitize: Fixed in commit 9788de.

Cyfrin: Verified.

7.2 Medium Risk

7.2.1 Lack of slippage protection in the SecuritizeVault's liquidation function

Description: The SecuritizeVault's `liquidate()` function converts share tokens to asset tokens using the NAV provider's rate without allowing users to specify a minimum output amount. Since the conversion rate is dynamic and external redemption may be involved, users are exposed to potential value loss from rate changes between transaction submission and execution. This is particularly concerning in volatile market conditions or when there is high latency in transaction processing.

Impact: Users may receive fewer assets than expected during liquidation due to rate changes, leading to direct financial losses.

Recommended Mitigation:

1. Add a `minOutputAmount` parameter to the liquidate function:

```
function liquidate(uint256 shares, uint256 minOutputAmount) public override(ISecuritizeVault)
↪  whenNotPaused {
...
    require(assets >= minOutputAmount, "Insufficient output amount");
...
}
```

2. Consider implementing a time limit parameter to protect against long-pending transactions
3. Add events to track actual output amounts for monitoring purposes

Securitize: Fixed in commit 42e651.

Cyfrin: Verified.

7.2.2 Inconsistent token amount in external redemption process of SecuritizeVault's liquidate function

Description: In SecuritizeVault's liquidation process with external redemption, there's likely a mismatch between the actual received stable coins from the redemption contract and the amount calculated to send to the liquidator. The vault calculates the output amount using `navProvider.rate()` after the external redemption, but this calculated amount may differ from the actual stable coins received due to rounding discrepancies and rate variations during the redemption process.

```
SecuritizeVault.sol
261:         if (address(0) != address(redemption)) {
262:             IERC20(asset()).approve(address(redemption), assets);
263:             redemption.redeem(assets);//@audit-info this sends underlying token to the redemption,
↪  receives stablecoin into this contract
264:             uint256 rate = navProvider.rate();
265:             uint256 decimalsFactor = 10 ** decimals();
266:             // after external redemption, vault gets liquidity to supply msg.sender (assets * nav)
267:             // liquidationToken === stableCoin
268:             liquidationToken.safeTransfer(msg.sender, assets.mulDiv(rate, decimalsFactor,
↪  Math.Rounding.Floor));//@audit-issue possible inconsistency in the amoutns. consider sending the
↪  balance delta instead
269:         }
```

Impact: Users either receive less stable coins than they should (value loss) or the transaction reverts due to insufficient balance when the calculated amount exceeds received tokens, making the liquidation functionality unreliable.

Recommended Mitigation: Track actual received stable coins from redemption contract and transfer them.

```
    if (address(0) != address(redemption)) {
        IERC20 liquidityToken = IERC20(redemption.liquidity());
        uint256 balanceBefore = liquidityToken.balanceOf(address(this));
        redemption.redeem(assets);
        uint256 receivedAmount = liquidityToken.balanceOf(address(this)) - balanceBefore;
        liquidityToken.transfer(msg.sender, receivedAmount);
    }
```

Securitize: Fixed in commit ef761a.

Cyfrin: Verified.

7.2.3 Ambiguous owner terminology creates confusion in access control of SecuritizeVault

Description: SecuritizeVault implements two different ownership concepts that create confusion in access control. It uses a custom `OWNER_ROLE` for vault operations (deposit/redeem) while inheriting `OwnableUpgradeable` which provides its own `owner()` function. This creates a situation where the term "owner" refers to different addresses depending on the context - the `OWNER_ROLE` holder can perform vault operations but cannot execute `onlyOwner` functions like `pause()`, while the Ownable owner can pause but cannot perform vault operations.

Impact: This ambiguity could lead to security issues if developers or users misunderstand which "owner" has what permissions, potentially resulting in failed operations or incorrect access control implementations in dependent contracts.

Proof Of Concept:

```
function testPause() public {
    // admin (the deployer) is Ownable owner - can pause/unpause
    vm.startPrank(admin);
    vault.pause();
    assertTrue(vault.paused());
    vault.unpause();
    vm.stopPrank();

    // OWNER_ROLE holder cannot pause
    vm.startPrank(owner);
    vm.expectRevert(abi.encodeWithSelector(OwnableUpgradeable.OwnableUnauthorizedAccount.selector,
    ↪  owner));
    vault.pause();
    vm.stopPrank();

    // Demonstrating the confusion:
    assertTrue(vault.isOwner(owner));     // True: has OWNER_ROLE
    assertFalse(vault.isOwner(admin));    // False: doesn't have OWNER_ROLE
    assertTrue(vault.owner() == admin);    // True: is Ownable owner
}
```

Recommended Mitigation:

1. Consolidate ownership model to use either roles or Ownable pattern exclusively
2. If both are needed, rename functions/roles to be more explicit:

```
// Instead of OWNER_ROLE
bytes32 public constant VAULT_OPERATOR_ROLE = keccak256("VAULT_OPERATOR_ROLE");

// Instead of isOwner()
function isVaultOperator(address account) public view returns (bool) {
    return hasRole(VAULT_OPERATOR_ROLE, account);
```

```
}
```

3. Add clear documentation explaining the distinction between different types of ownership

Securitize: Fixed in commit 887554.

Cyfrin: Verified.

7.3 Low Risk

7.3.1 Missing input validation in privileged functions

Description: Admin functions lack proper parameter validation, which could lead to unintended state changes if incorrect values are provided. While admin users are expected to act correctly, human error remains possible.

```
SecuritizeRedemption.sol
75:     function initialize(address _asset, address _navProvider) public onlyProxy initializer
↪  navProviderNonZero(_navProvider) {
76:         __BaseDSContract_init();
77:         asset = IERC20(_asset);//@audit-issue INFO non zero check
78:         navProvider = ISecuritizeNavProvider(_navProvider);
79:     }
80:

SecuritizeRedemption.sol
81:     function updateLiquidityProvider(address _liquidityProvider) onlyOwner external {
82:         address oldProvider = address(liquidityProvider);
83:         liquidityProvider = ILiquidityProvider(_liquidityProvider);//@audit-issue INFO non zero
↪  check
84:         emit LiquidityProviderUpdated(oldProvider, address(liquidityProvider));
85:     }

SecuritizeRedemption.sol
87:     function updateNavProvider(address _navProvider) onlyOwner navProviderNonZero(_navProvider)
↪  external {
88:         address oldProvider = address(navProvider);
89:         navProvider = ISecuritizeNavProvider(_navProvider);//@audit-issue INFO non zero check
90:         emit NavProviderUpdated(oldProvider, address(navProvider));
91:     }
```

```
CollateralLiquidityProvider.sol
66:     function initialize(address _recipient, address _liquidityToken, address _securitizeRedemption)
↪  public onlyProxy initializer {
67:         __BaseDSContract_init();
68:         recipient = _recipient;//@audit-issue LOW sanity check
69:         liquidityToken = IERC20(_liquidityToken);
70:         securitizeRedemption = ISecuritizeRedemption(_securitizeRedemption);
71:     }

98:
99:     function setCollateralProvider(address _collateralProvider) external onlyOwner {
100:         address oldAddress = address(collateralProvider);
101:         collateralProvider = _collateralProvider;//@audit-issue LOW sanity check
102:         emit CollateralProviderUpdated(oldAddress, address(collateralProvider));
103:     }
```

```
AllowanceLiquidityProvider.sol
65:     function initialize(address _recipient, address _liquidityToken, address _securitizeRedemption)
↪  public onlyProxy initializer {
66:         __BaseDSContract_init();
67:         recipient = _recipient;//@audit-issue LOW sanity check
68:         liquidityToken = IERC20(_liquidityToken);
69:         securitizeRedemption = ISecuritizeRedemption(_securitizeRedemption);
70:     }

85:     function setAllowanceProviderWallet(address _liquidityProviderWallet) external onlyOwner {
86:         address oldAddress = liquidityProviderWallet;
87:         liquidityProviderWallet = _liquidityProviderWallet;//@audit-issue sanity check
```

```
88:         emit AllowanceLiquidityProviderWalletUpdated(oldAddress, liquidityProviderWallet);
89:     }
```

Securitize: Fixed in commit 8254e8.

Cyfrin: Verified.

7.3.2 Insufficient storage gap in BaseSecuritizeSwap contract

Description: The BaseSecuritizeSwap contract implements a storage gap of 40 slots (`uint256[40] __gap`), which doesn't follow OpenZeppelin's practice of reserving 50 storage slots in total for upgradeable contracts. This deviation from the standard could potentially cause storage collision issues in future upgrades.

```
abstract contract BaseSecuritizeSwap is BaseDSContract, PausableUpgradeable {
    // ... other contract code ...
    uint256[40] __gap; // @audit-issue LOW -> uint256[43]
}
```

Below shows the current storage layout.

```
forge inspect BaseSecuritizeSwap storage --pretty
| Name                         | Type                              | Slot | Offset | Bytes | Contract
↪                                                  |
|------------------------------|-----------------------------------|------|--------|-------|------------
↪  ----------------------------------------------|
| services                     | mapping(uint256 => address)       | 0    | 0      | 32    |
↪  BaseSecuritizeSwap |
| __gap                        | uint256[49]                       | 1    | 0      | 1568  |
↪  BaseSecuritizeSwap |
| dsToken                      | contract IDSToken                 | 50   | 0      | 20    |
↪  BaseSecuritizeSwap |
| stableCoinToken              | contract IERC20                   | 51   | 0      | 20    |
↪  BaseSecuritizeSwap |
| erc20Token                   | contract IERC20                   | 52   | 0      | 20    |
↪  BaseSecuritizeSwap |
| collateralToken              | contract IERC20                   | 53   | 0      | 20    |
↪  BaseSecuritizeSwap |
| issuerWallet                 | address                           | 54   | 0      | 20    |
↪  BaseSecuritizeSwap |
| liquidityProviderWallet      | address                           | 55   | 0      | 20    |
↪  BaseSecuritizeSwap |
| externalCollateralRedemption | contract IRedemption              | 56   | 0      | 20    |
↪  BaseSecuritizeSwap |
| swapMode                     | enum BaseSecuritizeSwap.SwapMode  | 56   | 20     | 1     |
↪  BaseSecuritizeSwap |
| __gap                        | uint256[40]                       | 57   | 0      | 1280  |
↪  BaseSecuritizeSwap |
```

The storage of `BaseSecuritizeSwap` starts from `dstoken` and we can see 7 slots are used before the `__gap` variable. Following the OpenZeppelin's general practice, we should add 43 slots gap at the end.

Recommended Mitigation: Increase the storage gap to 43 slots to align with OpenZeppelin's best practices:

```
abstract contract BaseSecuritizeSwap is BaseDSContract, PausableUpgradeable {
--     uint256[40] __gap;
++     uint256[43] __gap;
}
```

Securitize: Fixed in commit 0c2a6e.

Cyfrin: Verified.

7.3.3 Missing PausableUpgradable initialization in SecuritizeSwap contract initialization

Description: The `SecuritizeSwap` contract inherits from `PausableUpgradeable` (through `BaseSecuritizeSwap`) but doesn't call `__Pausable_init()` during initialization. While this currently doesn't affect functionality since the default state (paused = false) matches the initialization state, it's a deviation from best practices.

Recommended Mitigation: For completeness and following best practices, add the Pausable initialization:

```
function initialize(...) public override initializer onlyProxy {
    BaseSecuritizeSwap.initialize(
        _dsToken,
        _stableCoin,
        _erc20Token,
        _issuerWallet,
        _liquidityProvider,
        _externalCollateralRedemption,
        _collateralToken,
        _swapMode
    );
    __BaseDSContract_init();
    __Pausable_init();
}
```

Securitize: Fixed in commit 637bcc.

Cyfrin: Verified.

7.3.4 Potential replay attacks due to static DOMAIN_SEPARATOR

Description: The DOMAIN_SEPARATOR is set only once during initialization and includes `block.chainid`. This could theoretically enable cross-chain replay attacks in the event of a hard fork. Although the contract implements a nonce system (`mapping(string => uint256) internal noncePerInvestor`) that effectively prevents such attacks, it is recommended to follow the best practice to ensure the domain separator always have a correct value.

```
function initialize(...) public override initializer onlyProxy {
    // ... other initialization code ...

    DOMAIN_SEPARATOR = keccak256(
        abi.encode(
            EIP712_DOMAIN_TYPE_HASH,
            NAME_HASH,
            VERSION_HASH,
            block.chainid,  //@audit-issue LOW this can change in the case of hard fork
            this,
            SALT
        )
    );
}
```

Recommended Mitigation: While not strictly necessary due to the nonce protection, following best practices, consider making DOMAIN_SEPARATOR dynamic. OpenZeppelin's EIP712 can be used instead as well.

```
function DOMAIN_SEPARATOR() public view returns (bytes32) {
    return keccak256(
        abi.encode(
            EIP712_DOMAIN_TYPE_HASH,
```

```
            NAME_HASH,
            VERSION_HASH,
            block.chainid,
            this,
            SALT
        )
    );
}
```

Securitize: Fixed in commit e31353.

Cyfrin: Verified.

7.3.5 Share tokens can be transferred

Description: The `SecuritizeVault` implements ERC4626's `deposit` function with an additional restriction requiring the depositor to be the same as the receiver (`_msgSender() == receiver`). While this was intended to ensure vault tokens are only owned by the designated investor, this restriction is ineffective since the share tokens implement ERC20 functionality and can be freely transferred after minting.

```
SecuritizeVault.sol
205:     function deposit(uint256 assets, address receiver)
206:         public
207:         override(ERC4626Upgradeable, ISecuritizeVault)
208:         whenNotPaused
209:         onlyRole(OWNER_ROLE)
210:         returns (uint256)
211:     {
212:         require(_msgSender() == receiver, "Sender should be equal than receiver");//@audit-issue
↪  not meaningful because share tokens can be transferred
213:         return super.deposit(assets, receiver);
214:     }
215:
```

Impact: The restriction provides a false sense of security and creates unnecessary friction for legitimate use cases where a depositor may want to directly deposit to another address, while failing to achieve the intended access control since tokens remain transferable.

Recommended Mitigation:

1. Remove the `_msgSender() == receiver` check since it provides no meaningful benefit.
2. If strict ownership control is required, consider:
    - Implementing transfer restrictions on the share tokens
    - Using a non-transferable token standard
    - Adding an allowlist mechanism for valid token holders

Securitize: Acknowledged.

Cyfrin: Acknowledged.

7.4 Informational

7.4.1 Incorrect/misleading comments

Description: Comments that do not accurately reflect the code's functionality can lead to misinterpretation during development, code review, and future maintenance.

```
ISecuritizeNavProvider.sol
20: /**
21:  * @title ISecuritizeNavProvider
22:  * @dev Defines a common interface to get NAV (Native Asset Value) Rate to  //@audit-issue INFO
↪  incomplete comment
23:  */
```

In the instance below, it is also recommended to write the amount conversion formula.

```
ISecuritizeNavProvider.sol
38:
39:     /**
40:      * @dev Set rate. It is expressed with the same decimal numbers as stable coin//@audit-issue
↪  INFO it is called liquidity token in other places, not "stable coin"
41:      */
42:     function setRate(uint256 _rate) external;//@audit-info same decimal to liquidity token
43:
44:     /**
45:      * @dev The asset:liquidity rate.//@audit-info INFO liquidityAmount = assetAmount * rate() /
↪  assetDecimals
46:      * @return The asset:liquidity rate.
47:      */
48:     function rate() external view returns (uint256);
49: }
```

```
SecuritizeRedemption.sol
64: /**
65:     * @dev Throws if called by any account other than the owner.//@audit-issue INFO incorrect
↪  comment
66:     */
67:     modifier navProviderNonZero(address _address) {
68:         require(_address != address(0) , "NAV rate provider address can not be zero");
69:         _;
70:     }
```

```
SecuritizeRedemption.sol
72:     /**
73:     * @dev Throws if called by any account other than the owner.//@audit-issue INFO incorrect
↪  comment
74:     */
75:     function initialize(address _asset, address _navProvider) public onlyProxy initializer
↪  navProviderNonZero(_navProvider) {
76:         __BaseDSContract_init();
77:         asset = IERC20(_asset);
78:         navProvider = ISecuritizeNavProvider(_navProvider);
79:     }
```

```
ISecuritizeRedemption.sol
60:     /**
61:      * @dev The NAV rate provider implementation.//@audit-issue INFO misleading comment, it's not
↪  necessarily an implementation.
```

```
62:      * @return The address of the NAV rate provider.
63:      */
64:     function navProvider() external view returns (ISecuritizeNavProvider);
65:
66:     /**
67:      * @dev Update the NAV rate provider implementation.//@audit-issue INFO misleading comment,
↪  it's not necessarily an implementation.
68:      * @param _navProvider The NAV rate provider implementation address//@audit-issue INFO
↪  misleading comment, it's not necessarily an implementation.
69:      */
70:     function updateNavProvider(address _navProvider) external;
```

ISecuritizeRedemption.sol

```
42:     /**
43:      * @dev The liquidity provider implementation.//@audit-issue INFO misleading comment, it's not
↪  necessarily an implementation.
44:      * @return The address of the liquidity provider.
45:      */
46:     function liquidityProvider() external view returns (ILiquidityProvider);
47:
48:     /**
49:      * @dev Update the liquidity provider implementation.//@audit-issue INFO misleading comment,
↪  it's not necessarily an implementation.
50:      * @param _liquidityProvider The liquidity provider implementation address//@audit-issue INFO
↪  misleading comment, it's not necessarily an implementation.
51:      */
52:     function updateLiquidityProvider(address _liquidityProvider) external;
```

CollateralLiquidityProvider.sol

```
56:     /**
57:      * @dev Throws if called by any account other than the owner.//@audit-issue INFO misleading
↪  comment, it's not the owner
58:      */
59:     modifier onlySecuritizeRedemption() {
60:         if (address(securitizeRedemption) != _msgSender()) {
61:             revert RedemptionUnauthorizedAccount(_msgSender());
62:         }
63:         _;
64:     }
```

AllowanceLiquidityProvider.sol

```
55:     /**
56:      * @dev Throws if called by any account other than the owner.//@audit-issue INFO wrong comment
57:      */
58:     modifier onlySecuritizeRedemption() {
59:         if (address(securitizeRedemption) != _msgSender()) {
60:             revert RedemptionUnauthorizedAccount(_msgSender());
61:         }
62:         _;
63:     }
```

Securitize: Fixed in commit 8254e8.

Cyfrin: Verified.

7.4.2 Naming improvements

Description: While functionally correct, certain function/modifier names could be more descriptive to better reflect their purpose and implementation.

```
SecuritizeRedemption.sol
64:     /**
65:      * @dev Throws if called by any account other than the owner.
66:      */
67:     modifier navProviderNonZero(address _address) {//@audit-issue INFO better naming addressNonZero
68:         require(_address != address(0) , "NAV rate provider address can not be zero");
69:         _;
70:     }
```

Securitize: Fixed in commit 8254e8.

Cyfrin: Verified.

7.4.3 Incorrect/misleading comments (2) - SecuritizeVault

Description: Comments that do not accurately reflect the code's functionality can lead to misinterpretation during development, code review, and future maintenance.

```
ISecuritizeVault.sol
72:     /**
73:      * @dev Grants the Redeemer role to an account. Emits a OwnerAdded event.//@audit-issue INFO
↪  Redeemer role is not defined in the interface. Must be Owner.
74:      *
75:      * @param _account The address to which the Owner role will be granted.
76:      */
77:     function addOwner(address _account) external;

86:     /**
87:      * @dev Checks if an account has the Redeemer role.//@audit-issue INFO Redeemer role is not
↪  defined in the interface. Must be Owner.
88:      *
89:      * @param _account The address to check for the Redeemer role.
90:      * @return bool Returns true if the account has the Redeemer role, false otherwise.
91:      */
92:     function isOwner(address _account) external view returns (bool);
```

```
SecuritizeVault.sol
120:     /**
121:      * @dev Grants the owner role to an account. Emits a RedeemerAdded event.//@audit-issue INFO
↪  wrong comment, should be Owner
122:      * Owners can deposit and redeem. Emits OwnerAdded event
123:      *
124:      * @param _account The address to which the Redeemer role will be granted.
125:      */
126:     function addOwner(address _account) external addressNotZero(_account)
↪  onlyRole(DEFAULT_ADMIN_ROLE) {
127:         grantRole(OWNER_ROLE, _account);
128:         emit OwnerAdded(_account);
129:     }
130:
131:     /**
132:      * @dev Revokes the Owner role from an account. Emits a OwnerRevoked event.
133:      *
134:      * @param _account The address from which the Redeemer role will be revoked.
135:      */
```

```
136:     function revokeOwner(address _account) external addressNotZero(_account)
↪  onlyRole(DEFAULT_ADMIN_ROLE) {
137:         revokeRole(OWNER_ROLE, _account);
138:         emit OwnerRevoked(_account);
139:     }
```

SecuritizeVault.sol

```
320:     /**
321:      * @dev Internal conversion function (from shares to assets) with support for rounding
↪  direction.
322:      *
323:      * This function overrides the default behavior in ERC4626Upgradeable
324:      * to ensure a conversion using NavProvider rate between assets and shares.
325:      *
326:      * min (xsSHARE / NAV, xsSHARE * tSHARE / tsSHARE) // <- xsSHARE & tASSSET / tsSHARE
327:      *
328:      * For more details, view ERC4626Upgradeable documentation.
329:      *
330:      * @param _shares The amount of shares to convert to assets.
331:      * @return uint256 The equivalent amount of assets.
332:      */
```

Securitize: Fixed in commit 172aed.

Cyfrin: Verified.

7.4.4 Unused state variable in BaseSecuritizeSwap contract

Description: The `BaseSecuritizeSwap` contract contains an unused state variable `dsToken`. This variable is declared but never referenced in any function throughout the contract or its inheriting contracts. While this doesn't pose any security risks, it unnecessarily increases gas costs during deployment and may cause confusion for developers maintaining the codebase.

Securitize: Fixed in commit ab46d0.

Cyfrin: Verified.