SB SECURITY

Shift Protocol

Security Review

Aug 23, 2025

Contents

1. About SBSecurity 3
2. Disclaimer 3
3. Risk classification 3
   - 3.1. Impact 3
   - 3.2. Likelihood 3
   - 3.3. Action required for severity levels 3
4. Executive Summary 4
5. Findings 5
   - 5.1. High severity 5
     - 5.1.1. Buffer will be locked in the vault 5
     - 5.1.2. Buffer must be based on the actual balance 6
   - 5.2. Medium severity 7
     - 5.2.1. Fees not collected before percentage is updated 7
     - 5.2.2. Fees cannot be re-enabled 8
     - 5.2.3. performance fees can be gamed 8
     - 5.2.4. Deposits not utilized efficiently and can lead to insolvency of the vault 9
     - 5.2.5. After fees claims and deposits, updateTvL of feed must be called 9
     - 5.2.6. withdraw() should not be pausable 10
   - 5.3. Low severity 11
     - 5.3.1. performanceFee must be based on only the tvl - snapshot tvl 11
     - 5.3.2. maintenance fee per second will have precision loss 11
     - 5.3.3. performance fee cannot be updated while contracts are paused 11
     - 5.3.4. delete the mappings instead of manually defaulting the properties 12
     - 5.3.5. In retrieveLiquidity add 0 liquidity check 13
     - 5.3.6. fees must be immediately withdrawable 13
     - 5.3.7. whitelisting can be bypassed with transfer 13
     - 5.3.8. dust accumulating in the withdrawals 14
     - 5.3.9. extract vault ctor arguments into 2 structs 14
     - 5.3.10. Unused storage variables 14
     - 5.3.11. Extract whitelist toggle into a separate function 15
     - 5.3.12. in deposit, simplify the _calcSharesFromToken arguments 15
     - 5.3.13. stale comment(s) 16
     - 5.3.14. getWithdrawStatus should use the userState's timelock, not the global property 16
     - 5.3.15. active users counter tracking can be gamed 16
     - 5.3.16. decimals function of the tvl vault should follow the vault's base token decimals 17
     - 5.3.17. Excessive balance check 17
     - 5.3.18. feeCollector doesn't increase activeUsers 17
   - 5.4. Governance severity 17
     - 5.4.1. Executor and Oracle are causing centralization risks 17
   - 5.5. Notes 18

1. About SBSecurity

SBSecurity is a duo of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

Book a Security Review with us at sbsecurity.net or reach out on Twitter @Slavcheww.

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

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

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

4. Executive Summary

Overview

Project	Shift Protocol
Repository	Private
Commit Hash	700f51b4a2274af82c280937c31748ae69b8717d
Resolution	8034c49b47a8b11dd52e6ea60f9b953b51c826af
Timeline	August 15 - August 23, 2025

Scope

ShiftVault.sol
ShiftTvlFeed.sol
ShiftManager.sol
ShiftAccessControl.sol
AccessModifier.sol
Constants.sol

Issues Found

Critical Risk	0
High Risk	2
Medium Risk	6
Low/Info Risk	18
Governance Risk	1

5. Findings

5.1. High severity

5.1.1. Buffer will be locked in the vault

Severity: High Risk

Description: There's a buffer which is used to keep a % of the entire tvl in the vault, so it can serve as a offset from the losses or simply to back up the executor when batch withdrawal is resolved. Currently, buffer is allocated correctly, but the tokens remain in the vault and aren't utilized as available to be withdrawn.

The result will be that these buffered tokens will be locked in the vault and won't serve any purpose.

The only way to withdraw them is to reset the buffer percentage to 0 and the executor to request them as liquidity.

Recommendation: After fixing [H-02], in `resolveWithdraw`, make sure to add the buffered amount to `availableToWithdraw`:

1. We have to be extra careful and ensure the buffer only for the particular batch, otherwise, after the 1st withdrawal, all the buffer goes to it, then the 2nd batch will mistakenly assume again the entire batch is for it, and `availableForWithdraw` will account for the same tokens multiple times.

2. If the `availableForWithdraw` is greater than the `requiredTokens`, you should take only `availableForWithdraw`, otherwise, you lock more than needed and will impact the executor's liquidity.

Another possible, easier solution is the buffer to be fulfilled directly in the deposit function, then when a withdrawal is resolved, it gets depleted and filled again. This removes all the possibilities of introducing variables that double-account the balance of the vault.

Resolution: Fixed

5.1.2. Buffer must be based on the actual balance

Severity: High Risk

Description: Currently, the buffer calculations are based on the reported TvL value, but this is fundamentally wrong, because it will include the balance in the vault + unrealized profit + the balance of the executor.

This will at least cause the percentage of the buffered amount that isn't available for the executor to be bigger than the one defined.

For `buffer18pt` = 0.2e18 (20%), `baseToken.balanceOf(vault)` = 100e18, `TvL` = 200e18, `_calcBufferValue` will return 40e18, instead of 20e18.

```
function _calcBufferValue() internal view returns (uint256) {
    IShiftTvlFeed.TvlData memory lastTvl = tvlFeed.getLastTvl();
    require(block.timestamp - lastTvl.timestamp < FRESHNESS_VALIDITY, "ShiftVault: stale TVL data");
    (uint256 tvl18pt,) = _normalize(lastTvl.value, tvlFeed.decimals());
    (, uint8 baseTokenScaleFactor) = _normalize(1, baseToken.decimals()); // Only to retrieve missing
decimals from base token

    if (bufferBps == 0) return 0; // No buffer, return TVL only

    UD60x18 buffer = ud(tvl18pt).mul(ud(buffer18pt)).div(ud(1e18));
    return baseTokenScaleFactor == 0 ? buffer.unwrap() : buffer.unwrap() / 10 ** baseTokenScaleFactor;
}
```

In more severe scenarios, the buffer calculated will be bigger than the balance of the vault, indicating 0 liquidity, while in reality, it must be a percentage of the balance.

Recommendation: Base the buffer calculation on the vault's balance, not the TVL, because these tokens aren't in the vault and they're not actually being buffered.

```
function _calcBufferValue() internal view returns (uint256) {
      IShiftTvlFeed.TvlData memory lastTvl = tvlFeed.getLastTvl();
      require(block.timestamp - lastTvl.timestamp < FRESHNESS_VALIDITY, "ShiftVault: stale TVL data");
-     (uint256 tvl18pt,) = _normalize(lastTvl.value, tvlFeed.decimals());
-     (, uint8 baseTokenScaleFactor) = _normalize(1, baseToken.decimals()); // Only to retrieve missing
decimals from base token
+      uint256 netBalanceOfVault18pt, uint8 baseTokenScaleFactor) =
_normalize(baseToken.balanceOf(address(this), baseToken.decimals());

      if (bufferBps == 0) return 0; // No buffer, return TVL only

-     UD60x18 buffer = ud(tvl18pt).mul(ud(buffer18pt)).div(ud(1e18));
+     UD60x18 buffer = ud(netBalanceOfVault18pt).mul(ud(buffer18pt)).div(ud(1e18));
      return baseTokenScaleFactor == 0 ? buffer.unwrap() : buffer.unwrap() / 10 ** baseTokenScaleFactor;
  }
```

Resolution: Fixed

5.2. Medium severity

5.2.1. Fees not collected before percentage is updated

Severity: Medium Risk

Description: Both maintenance and performance fee setters are missing collect invocation, before setting the new values. The exact issue is that the old vault performance will be used with the new vault parameters.

Furthermore, if the fees are disabled (their Fee18Pt variables are set to 0) the previously accumulated fees will be lost.

Recommendation: Claim the fees before updating the fee percentages. It can be done by batching claim → update functions together or add claim execution inside the update function.

Resolution: Fixed

5.2.2. Fees cannot be re-enabled

Severity: Medium Risk

Description: The current implementation should be possible to re-enable the fees if they're 0. But there's an excessive checks in the update functions, that prevent the admin from being able to do so.

```
function updateMaintenanceFee(uint16 _annualFeeBps) public virtual onlyAdmin {
    require(_annualFeeBps > 0, "ShiftManager: zero maintenance fee");
```

```
function updatePerformanceFee(uint16 _feeBps) external onlyAdmin {
    require(paused, "ShiftManager: contract not paused");
    require(_feeBps > 0, "ShiftManager: zero performance fee");
```

Recommendation: Remove the require checks, preventing the admin to pass 0s.

Resolution: Fixed

5.2.3. performance fees can be gamed

Severity: Medium Risk

Description: The current mechanism of tracking whether the vault is in profit is prone to inflation. Gains calculation: tvl - snapshot + cumulative deposit - cumulative withdraw can be gamed from any adversary by creating a huge deposit, inflating both tvl and cumulative deposit, fee claimed, then the deposit withdrawn before even being issued from the executor.

This is more feasible towards the end of the week, when batch withdrawals are being settled.

However, there's a risk for the depositor in case the share price decreases, so practicality is under question.

Recommendation: The current implementation of the perf. fee makes it impossible to be mitigated and anyone willing to take the risk can execute the attack if he wants to harm the other users, as the shares minted as part of the fee issuance dillute the share price.

Resolution: Fixed - After extensive discussion with the protocol team, we agreed to change the gains calculation to tvl - (snapshot + cumulative deposit - cumulative withdraw).

5.2.4. Deposits not utilized efficiently and can lead to insolvency of the vault

Severity: Medium Risk

Description: Depositors can earn yield even without contributing to it. The reason is because their funds aren't forwarded directly to the executor, rather than they have to be pulled in a given moment. In the same time they can immediately earn or lose if the share price changes.

Another more impactful scenario is when the idling deposits are more than what the executor invested in the underlying strategy and the idling deposits want to be withdrawn.

Then the profit from the early depositors can be eaten by the new depositors withdrawals or if there isn't buffer, the executor must take the deposits and calculate the rate based on idling deposits + withdrawn from the strategy. If there's a buffer, there can be loss if the profit from the strategy is less than the total withdrawals. Even if the profit is more, the depositor, whose funds actually were used will receive less profit than the actual.

Recommendation: Ideally the funds, collected from deposits must be immediately forwarded to the underlying strategy and used to support the positive movement of the share price.

Resolution: Fixed

5.2.5. After fees claims and deposits, updateTvL of feed must be called

Severity: Medium Risk

Description: When fees are claimed, new shares are being minted. In a normal scenario, this isn't an issue because no user action can be performed without a new TvL record to be pushed.

However, there are 2 examples, where the order of operations can break this trust assumption:

1. `requestDeposit` → `allowDeposit` → `claim` → `deposit` (while request is still considered "valid").

For example, a deposit is allowed on totalSupply = 100, fees are claimed and 10 are minted in total. From this point, the state of vault, returned from `getTvlEntry`, is stale as there are 110 shares, not 100 and the user will receive more shares, since the TvL is the denominator.

2. `requestDeposit` for Bob → `requestDeposit` for Alice → `allowDeposit` Bob → `allowDeposit` Alice → from here on, no matter whose is, the 2nd deposit's shares will be based on stale TvL state.

Less impact, since both `totalSupply` and `totalAssets` grow in proportion, but again the rounding will cause the share price to change.

Recommendation: There are no other actions that increase the share price, also oracle and admin have superior control, make sure to not allow these 2 orders to happen.

Resolution: Acknowledged

5.2.6. `withdraw()` should not be pausable

Severity: Medium Risk

Description: The withdrawal process has request → claim steps and therefore, once the request is executed and X amount of tokens are locked for the current withdrawal batch, withdrawals must be open for users to receive their tokens. But `withdraw()` has a pause check and when the system is paused, they will not be able to receive their tokens.

```
function withdraw() external nonReentrant notPaused {
```

Recommendation: Remove the `notPaused` from `withdraw()`. Deposits and withdrawals can also be made pauseable separately.

Resolution: Fixed

5.3. Low severity

5.3.1. performanceFee must be based on only the tvl - snapshot tvl

Severity: Low Risk

Description: Currently, deposits and withdraws will be double accounted in the tvl and cumulativeDeposit/cumulativeWithdraw. For example, when new deposit for X amount enters the vault, oracle will report new TvL = old TvL + X, `cumulativeDeposit` will be also increased with X. Rendering the cumulative numbers redundant. Furthermore, as explained in M-03 they can be used to falsely make the vault appear in profit.

Recommendation: Base the `performanceFee` only on the `TvL - snapshotTvL`.

Resolution: Fixed

5.3.2. maintenance fee per second will have precision loss

Severity: Low Risk

Description: Due to dividing to seconds per year, the annual maintenance fee will be slightly lower than intended:

```
20%: 0.2e18 / 31536000 = 6341958396.7529....
6341958396 * 0.2e18 will be ~199999999976256000
```

Recommendation: Instead of dividing by seconds per year, you can use a pre-calculated constant that represents the inverse of seconds per year with higher precision.

Resolution: Acknowledged

5.3.3. performance fee cannot be updated while contracts are paused

Severity: Low Risk

Description: Performance fee cannot be updated when contract is paused, but the maintenance fee can, which creates and inconsistency.

Recommendation: Either add the paused check to `updateMaintenanceFee` or remove it from `updatePerformanceFee`.

Resolution: Fixed

5.3.4. delete the mappings instead of manually defaulting the properties

Severity: Low Risk

Description: The issue is the user state data from storage after operations isn't deleted, but it's property values defaulted, which could lead to unnecessary gas costs and potential state inconsistencies when users interact with the vault multiple times.

Recommendation:

deposit: Delete the userDepositStates entry but save the `requestIndex` as a memory variable first

withdraw: Delete the userWithdrawStates entry

cancelWithdraw: Delete the `userWithdrawStates` entry and also fix the `timelock` property not being defaulted

reqDeposit: When `_isExpired` is true, delete the `userDepositStates` entry

Resolution: Fixed

5.3.5. In retrieveLiquidity add 0 liquidity check

Severity: Low Risk

Description: `retrieveLiquidity` will do transfer regardless there's liquidity or no. For certain asset, if they revert on 0-value transfers this will revert the execution.

```
function retrieveLiquidity() external onlyExecutor nonReentrant notPaused {
    uint256 liquidity = _calcResolverLiquidity(_calcBufferValue());
    baseToken.safeTransfer(msg.sender, liquidity);
}
```

Recommendation:

```
function retrieveLiquidity() external onlyExecutor nonReentrant notPaused {
    uint256 liquidity = _calcResolverLiquidity(_calcBufferValue());
+   if(liquidity == 0) revert("No liquidity");
    baseToken.safeTransfer(msg.sender, liquidity);
}
```

Resolution: Fixed

5.3.6. fees must be immediately withdrawable

Severity: Low Risk

Description: Currently, the shares from fees should also follow the normal path of withdrawal, meaning there's an opportunity cost before being exercised and they can be subject of loss or profit.

This shouldn't be the case as the fees have to be immediately available for withdrawal.

Recommendation: Add logic to make it possible for the `feeRecipient` to materialize his fee shares.

Resolution: Acknowledged

5.3.7. whitelisting can be bypassed with transfer

Severity: Low Risk

Description: whitelisting can be bypassed by utilizing an address which is already whitelisted and then transfer to any other account. The reason is that the check is applied only on deposit/ withdraw.

Recommendation: `_update` can be overriden to check the `to` address, but care should be taken the vault to be exception from the check.

Resolution: Acknowledged

5.3.8. dust accumulating in the withdrawals

Severity: Low Risk

Description: requiredTokens in `resolveWithdraw` are all the `shares * rate`, but each individual withdrawal is `user shares * rate`, the sum of all tokens actually withdrawn <= required tokens, most likely roundings will cause dust to accumulate in the vault under the `availableForWithdraw`. These tokens will be forever locked in the vault, as they cannot be withdrawn from users and from the executor.

Recommendation: You can add sweep functionality to be able to claim the residue tokens.

Resolution: Fixed

5.3.9. extract vault ctor arguments into 2 structs

Severity: Informational Risk

Description: A lot of arguments are being provided when deploying a new vault. There's a risk of passing them in the wrong order as they're the same types consecutively:

```
constructor(
    address _accessControlContract,
    address _tokenContract,
    address _tvlFeedContract,
    address _feeCollector,
    string memory _shareName,
    string memory _shareSymbol,
    uint256 _minDeposit,
    uint256 _maxTvl,
    uint32 _timeLock
)
```

Recommendation: Create new structs, so you can manage it easier in case there's a need of arguments modification. Also, this will make it harder to mess up when passing the values, as you'll 1st need to create the struct and match the property name with the value.

Resolution: Fixed

5.3.10. Unused storage variables

Severity: Informational Risk

Description: The following storage variables aren't used anywhere in the code:

- maintenanceFeeBpsAnnual

- performanceFeeBps

- bufferBps (used only for the require, can be replaced with buffer18pt)

Recommendation: Modify the code to by removing them and relying on their 18pt representations.

Resolution: Fixed

5.3.11. Extract whitelist toggle into a separate function

Severity: Informational Risk

Description: Currently, `manageWhitelist` does 2 different actions at once, it can toggle the global whitelist and per-user whitelist. It can be confusing for management.

```
function manageWhitelist(address[] calldata _user) external onlyAdmin {
    if (_user[0] == address(0)) {
        whitelistEnabled = !whitelistEnabled;
    } else {
        for (uint256 i = 0; i < _user.length; i++) {
            isWhitelisted[_user[i]] = !isWhitelisted[_user[i]];
        }
    }
}
```

Recommendation: Extract whitelisting toggle and individuals whitelist into a separate functions so you follow the SOA concept

Resolution: Fixed

5.3.12. in deposit, simplify the `_calcSharesFromToken` arguments

Severity: Informational Risk

Description: Now the `_calcSharesFromToken` doesn't reuse already variables, already initialized in outer deposit function and do the tvl fetch and amounts normalization again.

```
function _calcSharesFromToken(uint256 _tokenAmount, uint256 _tvlIndex) internal view returns (uint256) {
    IShiftTvlFeed.TvlData memory lastTvl = tvlFeed.getTvlEntry(_tvlIndex);
    (uint256 baseToken18pt,) = _normalize(_tokenAmount, baseToken.decimals());
    if (lastTvl.supplySnapshot == 0) {
        return baseToken18pt; // If no supply, return token amount as shares
    }
    (uint256 tvl18pt,) = _normalize(lastTvl.value, tvlFeed.decimals());
    return _calcShare(baseToken18pt, tvl18pt, lastTvl.supplySnapshot);
}
```

Recommendation:

1. Instead of `_tokenAmount` pass the `baseToken18pt` and avoid the 2nd `_normalize` call for the same asset

2. Pass the `tvl18pt` as an argument

3. Pass the supplySnapshot as an argument

Resolution: Fixed

5.3.13. stale comment(s)

Severity: Informational Risk

Description: `_calcTokenFromShares` has a wrong comment, indicating the `_rate` must be in 6 decimals, this is wrong as the calling functions always make sure to scale the number to 18 decimals.

Recommendation: Change the comment to say that rate must be in ~~18 decimals~~ feed's decimals.

Resolution: Fixed

5.3.14. `getWithdrawStatus` should use the userState's timelock, not the global property

Severity: Informational Risk

Description: The global `timelock` variable, used to check whether withdrawal has matured, can be inaccurate if it gets modified after user has requested withdrawal.

```
function getWithdrawStatus()
        external
        view
        returns (uint8 status, uint256 shareAmount, uint256 tokenAmount, uint256 unlockTime)
    {
        WithdrawState storage userState = userWithdrawStates[msg.sender];
        uint256 shares = userState.sharesAmount;
        if (shares == 0) return (0, 0, 0, 0); // No withdrawal requested

        uint256 batchId = userState.batchId;
        uint256 rate = batchWithdrawStates[batchId].rate;
        uint256 unlkTime = userState.requestedAt + uint256(timelock);
```

Recommendation: Use `userState.timelock` instead.

Resolution: Fixed

5.3.15. active users counter tracking can be gamed

Severity: Informational Risk

Description: Since `activeUsers` check is only done on deposit and withdraw, it will miss the scenarios when the shares are being transferred. The counter can be left out of sync if user transfers his shares to someone else, who's also already holder. When the recipient withdraws all his shares counter will be decremented once, while it was incremented twice.

Recommendation: Override the `_update` function from `ERC20` and delegate the functionality of tracking the holders counter, or consider removing the tracker completely, as the block explorers offer it natively.

Resolution: Fixed

5.3.16. decimals function of the tvl vault should follow the vault's base token decimals

Severity: Informational Risk

Description: TvL decimals must match the decimals of the `baseToken` in the vault, as they ultimately track the same asset. However, there's a risk of introducing an missmatch, as there's nothing in the code that prevents it.

Recommendation: Refactor the function in TvLFeed to reuse the decimals of the `baseToken`.

Resolution: Fixed

5.3.17. Excessive balance check

Severity: Informational Risk

Description: `ShiftVault::reqWithdraw` does a check, whether the sender has enough share balance. This check is excessive, since the transfer which is done below will revert in case user has insufficient balance.

Recommendation: Remove it.

Resolution: Fixed

5.3.18. feeCollector doesn't increase activeUsers

Severity: Informational Risk

Description: Current mechanism mint shares to `feeCollector` as part of the fee issuance, however it doesn't increment the `activeUsers` counter. As a result, after when totalSupply becomes 0, the counter will falsely indicate there's 1 active user.

Recommendation: Override the `_update` function from `ERC20` and delegate the functionality of tracking the holders counter, or consider removing the tracker completely, as the block explorers offer it natively.

Resolution: Fixed

5.4. Governance severity

5.4.1. Executor and Oracle are causing centralization risks

Severity: Governance Risk

Description: Vault is centralized around the oracle and executor. This limitation is caused by the external strategy the team is willing to use and the fact that there's no native onchain solution for achieving it. That means the funds should be managed manually, on a best-effort basis.

Therefore, there's a risk of miscalculations of the share rate for both deposit and withdrawal batches. Essentially, the executor is the single point of failure.

Recommendation: Due to the strategy of the team to use Starknet perps, there's no other option to achieve it, which is based solely on onchain solutions, in a decentralized manner.

Resolution: Acknowledged

5.5. Notes

Description:

1. Make `REQUEST_VALIDITY` and `FRESHNESS_VALIDITY` upgradable.

2. Consider having a withdrawal deadline: Currently, there's no deadline for claiming them, while the allocated tokens are locked in the vault w/o being used in the strategy. For example, executor function that can pull expired withdrawal requests.

3. maxTvL scaling in deposit is unnecessary, because both tvl and base are originally in the same decimals

4. If vault curators fail to detect, a user can pull an inflation attack by donating assets and increasing the tvl to cause share rounding for the next users. - It's almost impossible to be done because the operations are not in a specific order and can be prevented by the Admin.

5. Shorten the request validity and tvl data freshness

6. In `getSharePrice`, instead of reverting when TVL data isn't fresh, return boolean. It'll be easier for integrators to handle.

7. Rebasing tokens not supported, since the `amountForWithdraw` is fixed, if later on token rebases negatively, it will be impossible for the last user to retrieve his withdrawal, due to insufficient balance.

8. Replace all the string errors with custom reverts.

9. Add event emissions in the setters.

Resolution: Fixed - 1, 3, 6, 8, 9